Master's Program in
Computational Logic (MCL)

Fakultät Informatik
Technische Universität Dresden

Automatic and Interactive Search in Flexible Dispute Derivations for Assumption-Based Argumentation:

Analysis, Implementation, Evaluation.

June 2022

Author
Piotr Jerzy Gorczyca
MNr.: 4879011

Supervisor
Dr. Sarah Alice Gaggl

Co-supervisor
Dr. Martin Diller

Reviewer
Dr. habil. Hannes Strass

Declaration of Authorship

Author: Piotr Jerzy Gorczyca
Immatriculation number: 4879011
Title: Automatic and Interactive Search in Flexible Dispute Derivations for Assumption-Based Argumentation: Analysis, Implementation, Evaluation.

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part to obtain a degree or any other qualification neither in this nor in any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text. No other resources apart from the references and auxiliary means indicated in the bibliography were used in the development of the presented work.

Dresden, July 5, 2022

(Author's signature)

Acknowledgements

During the 3-year period of my studies at the TU Dresden and, particularly, while working on this thesis, I have received a great deal of support for which I would like to express my dearest gratitude.

I am deeply indebted to Dr. Sarah Alice Gaggl for the invaluable guidance. Thank you for agreeing to supervise my thesis and for the insightful feedback I have been continuously receiving. It helped me greatly to stay on track and determine on what I should focus and how things could be approached. I could always count on your assistance and motivation.

I could not have managed to complete this work without its co-supervisor, Dr. Martin Diller, whom I would like to particularly thank for his major contributions. His expertise in the related field helped me greatly, especially in developing the methodology applied in this work and forming its shape. Thank you for all the suggestions, the continuous support and the guidance you provided.

I am very grateful to Dr. Gaggl and Dr. Diller for their readiness for the regular and frequent meetings and the flexibility in arranging them. Thank you for always having the time for me.

I would also like to express my dearest appreciation to Dr. habil. Hannes Strass, who agreed to review my thesis and provided me with great dose of enlightening feedback in very short time. Thank you for taking the time to explain the ambiguities very clearly.

Special gratitude is dedicated to the both supervisors and the reviewer for their patience, understanding and flexibility, whenever things did not go as planned.

Moreover, I would like to thank Ms. Romy Thieme from the TUD's Service Center For International Students for the assistance in explaining many different procedures related to the thesis as well as during my entire study program. Thank you for your patience and kindness. Your help was invaluable, especially during the turbulent period of the pandemic, when many new procedures were imposed.

Finally, I would like to thank my family and friends.

I am deeply grateful to my parents Justyna and Jerzy Gorczyca for everything they have done for me. Thank you for your emotional and financial support and your continuous encouragement. I would not be where I am if it were not for you and I could not wish for better and more supportive parents.

I would also like to thank my sister Hanna Gorczyca. Thank you for advising about living in Germany and with the translations of formal documents. Thank you also for planting the idea of trying my luck at a foreign, German university in my mind and particularly suggesting TU Dresden. Your motivation has kept pushing me forward throughout the recent years.

I am also very grateful to my parents and my sister for helping me move to Dresden.

I am particularly thankful to my girlfriend, Martyna Adamiec. I could always count on your support and encouragement. Thank you for your patience, whenever I was busy with my studies and your frequent visits to Dresden, thanks to which we managed to withstand being separated.

I spent one of the most interesting and entertaining periods of my life in Dresden. I would like to thank those who were part of it, my friends Camilla Salve and Szymon Chrost, my fellow students Islam Hamada and Mohamed Nadeem, and those, with whom I had the pleasure to share our dorm: Majed Al Medawar, Leo Proshkin, and Hassan Irfan. Thank you for all the fascinating conversations and enjoyable moments we shared.

Abstract

Formal argumentation (FA) is a branch of knowledge representation and reasoning in artificial intelligence, which offers ways of resolving potential conflicts within a knowledge base and inferring which claims can be trusted. FA can be further divided between abstract- and structured argumentation, with the first treating arguments as atomic entities and the latter allowing to further investigate their internal structure. Assumption-based argumentation (ABA) is one of the main general structured argumentation frameworks, in which dispute derivations (DDs) are methods for determining the acceptance of claims in dialectical manner. With DDs arguments and counter-arguments are constructed interchangeably between, as can be conceived, two fictitious players – the proponent and the opponent of a set of claims under scrutiny.

Among the many formalisations of DDs for ABA, which have been proposed throughout the past decades, flexible dispute derivations (FlexDDs) are the latest. FlexDDs offer a number of solutions unprecedented in the former versions, including reusing one player's arguments by the other player or support for complete and stable argumentation semantics to name a few. Most notably however, alongside the regular, backward, top-down reasoning from claims to premises they also allow for construction of forward, bottom-up arguments, from premises to claims. Moreover, FlexDDs comes in two versions – a more high-level and abstract argument-based version as well as a more concrete and implementation-focused rule-based version.

In this thesis we focus on automatized search of successful dispute derivations for flexible dispute derivations. We devise procedures and examine the properties of strategies tailored for specific semantics, with the aim of isolating features impacting the efficiency, among other concerns. Furthermore, we investigate the influence of forward reasoning on the performance of the search procedures.

We have performed a thorough empirical evaluation, the results of which are presented and interpreted, to back up our claims and hypotheses. Moreover, our implementation for FlexDDs has been significantly extended, currently capable of following the defined search strategies in an automatic search for successful DDs as well as having rich support for interactive reasoning, besides many other notable features which we report on in this work.

Keywords. Formal Argumentation · Structured Argumentation · Assumption-Based Argumentation · Dispute derivations.

Contents

1 Introduction

1.1 Motivation

Formal Argumentation (FA) offers various reasoning mechanisms for representing and evaluating arguments, conflicts between them, as well as determining sets of arguments being coherent together in some specific manner. Two main forms of argumentation have been established over the last decades of thorough research in FA [1, 2, 16]: *Abstract Argumentation* (AA) [3, 6] and *Structured Argumentation* (SA) [5]. While the first treats arguments as atomic entities, the latter further investigates their internal structure. In SA, within rule-based argumentation [22], we can in turn distinguish between *ASPIC* [22, 23] and *Assumption-Based Argumentation* (ABA) [10, 12, 28]. Regardless of the form of argumentation chosen, arguments are always defined in terms of so called frameworks. In AA arguments are given directly by the framework's underlying directed graph structure, in which nodes represent arguments and edges attack relations between them. In SA, frameworks are tuples of various entities used to construct the arguments and attacks between them.

For ABA, frameworks are tuples $(\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ with $\mathcal{L}$ being the alphabet (with elements called statements); $\mathcal{R}$ a set of rules of the form $h \leftarrow B, h \in \mathcal{L}, B \subseteq \mathcal{L}$ with h and B called the *head* and the *body* of the rule respectively; $\mathcal{A} \subseteq \mathcal{L}$ the set called assumptions and $^- : \mathcal{A} \to \mathcal{L}$ a total mapping of assumptions to alphabet elements, where for each assumption $a \in \mathcal{A}$, $\bar{a}$ is called the contrary of a.

Example 1.1.1 (ABA framework)**.** Consider an ABA framework $F' = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, where $\mathcal{A} = \{a, b, c, d, e, f, g, h, i\}$, $^-(u) = \bar{u}$ for $u \in \mathcal{A}$ and $\mathcal{L} = \mathcal{A} \cup \{\{h\} \cup B \mid h \leftarrow B \in \mathcal{R}\} \cup \{\bar{u} \mid u \in \mathcal{A}\}$ where:

$$\mathcal{R} = \{p \leftarrow a, b, q;\ q \leftarrow \bar{e}, r;\ q \leftarrow i;\ r \leftarrow c;\ r \leftarrow d;\ \bar{e} \leftarrow c;\ \bar{f} \leftarrow \bar{e};\ \bar{i} \leftarrow g;\\ \bar{a} \leftarrow e, v;\ \bar{a} \leftarrow z;\ v \leftarrow h;\ v \leftarrow i;\ z \leftarrow f\}.$$

Intuitively, each (complete) argument Arg in ABA, given a framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ can be seen as a rooted tree, where the root is labelled with the main claim of the argument and with branches from a node labelled with h to each node labelled with $b \in B$ given that there is a rule $h \leftarrow B \in \mathcal{R}$. Every leaf of an argument must either be labelled with an assumption $a \in \mathcal{A}$ or with an element h if there is a rule $h \leftarrow \varnothing \in \mathcal{R}$, otherwise the argument is called incomplete. Furthermore, an argument Arg attacks an argument Arg' if there is a node labeled with a in Arg' where $a \in \mathcal{A}$ and a node labelled with $\bar{a}$ in Arg. We also say that an argument Arg defends argument Arg'' (possibly $Arg = Arg''$) from argument Arg' if it holds that Arg' attacks Arg'' and Arg attacks Arg'.

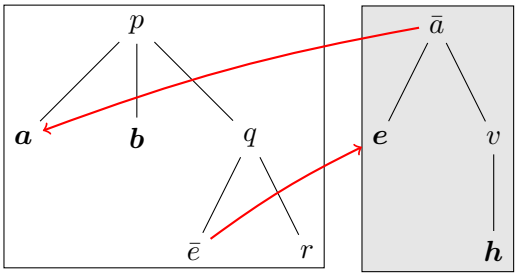


Figure 1.1: An incomplete argument Arg_p for claim p (left) and a complete one, $Arg_{\bar{a}}$, for $\bar{a}$ (right) from the framework from Example 1.1.1. Letters labeling nodes are elements of the alphabet, with those in boldface indicating assumptions. Black edges denote support with the upper tip labelled with a head of a rule and bottom tip labelled with a statement from a body of a rule. Red pointy arrows indicate attacks. Arg_p having no color filling symbolizes its incompleteness, in contrast to $Arg_{\bar{a}}$.

In Figure 1.1 two arguments Arg_p and $Arg_{\bar{a}}$ constructible from the framework from Example 1.1.1 are presented. Note that whenever branching from a parent node labeled with h to its children labeled with $b_1, \ldots, b_n$ occurs, there must exist a rule $h \leftarrow b_1, \ldots, b_n \in \mathcal{R}$. For example for the direct children labeled with a, b and q of the root node labeled with p of Arg_p there is a rule $p \leftarrow a, b, q \in \mathcal{R}$. Also, the argument for $\bar{a}$ is complete as all its leaves are labeled with assumptions (e and h), whereas the argument for p is incomplete as the leaves $\bar{e}$ and r are not assumptions. Moreover Arg_p attacks $Arg_{\bar{a}}$ as it has a node labelled with $\bar{e}$ and one of the two leaves of $Arg_{\bar{a}}$ is labeled with e. The opposite also holds, with $Arg_{\bar{a}}$ attacking the assumption a of Arg_p with the claim of its root node. Simultaneously Arg_p and $Arg_{\bar{a}}$ defend themselves from one another.

One of the key tasks in Formal Argumentation is that of determining (credulously) acceptable arguments and claims. In ABA this problem, given a claim p, asks whether there exists a set of complete arguments $S = \{Arg_1, \ldots, Arg_n\}$ (called an extension) containing an argument for that claim p, such that some criteria are met. Those criteria are defined by so called *argumentation semantics*, with the admissible semantics being the most basic one. Admissibility imposes two requirements: i) S must be conflict free, meaning that there are no two arguments in S such that one attacks the other, and ii) S must defend itself, meaning that for each complete argument Arg' outside of S attacking some argument in S (and therefore attacking the entire set S), there must be some argument in S attacking Arg'.

The green arguments from Figure 1.2 satisfy the condition of an admissible extension for the claim p, because i) they are conflict-free (there are no attack arrows between any green arguments) and they counter-attack all their attackers (yellow arguments). Note that there are no more complete arguments which could be constructed in order to attack the extension. This extension is a proof that p can be accepted under admissible semantics.

There are a few approaches to deciding acceptance of claims in ABA [7, 8, 9, 11, 13, 14, 18, 19, 20, 21, 27]. One of them is based on so called dispute-derivations, which mimics a dispute between two fictional players, called proponent and opponent of a claim. Although other, reduction-based methods [19, 20, 21] have proved to be much more efficient than dispute derivations, the latter are still worth of consideration on various grounds, the main one being that they provide justi-

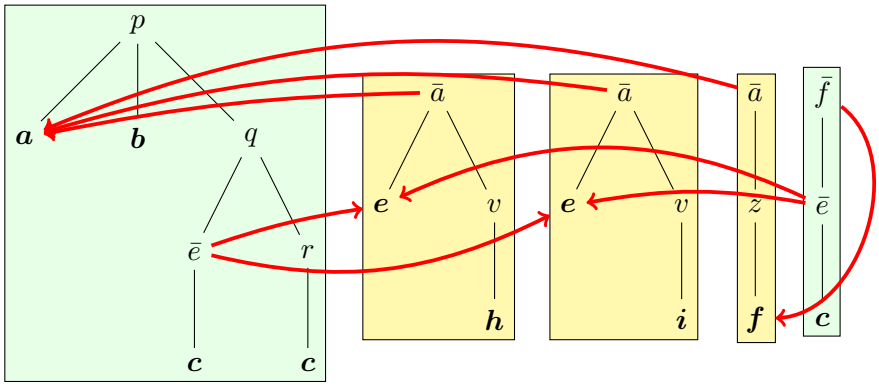


Figure 1.2: Complete arguments for claims p, $\bar{a}$ and $\bar{f}$ constructed from framework from Example 1.1.1. Arguments with green fillings are those that the extension consists of, whereas the ones with yellow are those attacking the extension. Red arrows indicate the attacks.

fications of the claims in questions in form of arguments and counter-arguments resembling an actual dispute.

In this approach players perform moves in the pursuit of forming arguments. The proponent has to construct an argument for the claim under scrutiny, meanwhile the opponent has to assemble arguments attacking assumptions of the proponent's constructed argument. This is in turn followed by the proponent counter-attacking opponent's arguments by forming arguments attacking their assumptions. This process continues until one player is unable to proceed. The winner determines the outcome, meaning that if there exists a way for the proponent to win, then the claim can be accepted, otherwise it cannot.

Over the last few years many variations of dispute derivations have been proposed [7, 8, 9, 11, 13, 14, 18, 27], among others with *Graphical Dispute Derivation* [7, 9] as one of the most prominent examples. Each newly introduced variation has improved on their predecessor's shortcomings, and it was no different with the *Flexible Dispute Derivations* (FlexDDs) [11] variant defined most recently. Among other refinements FlexDDs introduced, it has defined a more effective procedure minimizing the number of redundant moves.

Before a quick introductory example is presented, let us first become acquainted with some key features of FlexDDs. We call the assumptions present in the proponent's arguments *defences* and the assumptions which the proponent attacks *culprits*. For instance, assuming that the arguments from Figure 1.2 are a final dispute state in FlexDDs, we would have the set of defences consisting of a, b, c and the set of culprits consisting of e, f. As has been stated previously, at each stage of the dispute derivation both players decide to either continue to backward expand their existing incomplete arguments or to start creating new ones to attack the other player. The proponent however is constrained by the requirement of not using more than one rule for each statement[1]. This alone causes the entire procedure to be non-deterministic, as the proponent has to check every possibility (branch) whenever several rules are applicable. This does not hold for

[1] FlexDDs adopt this behaviour from *Graphical Dispute Derivations* [7, 9], whose authors convincingly argue against having more than one rule with the same head in the set of proponent's arguments for both conceptual and computational reasons.

the opponent, who, on the contrary, has to attack the proponent using all possible means. There is an exception to that rule, i.e. the opponent cannot use rules containing culprits in their bodies (naturally neither can the proponent as their arguments have to remain conflict-free, as required by the admissibility).

Now, how could one approach constructing arguments when trying to find a successful FlexDD given a goal? Since the moves of the proponent assume branching, whereas those of the opponent do not, and given that the opponent has to construct all possible counter-arguments anyways, allowing the opponent to perform all their moves before the proponent starts to perform theirs seems justifiable. Such a dispute derivation could then look as in the Figure 1.3, where the tree represents the search of all successful FlexDDs justifying the claim p. Note that the tree branches at every non-deterministic choice point (as e.g. when there is more than one rule of a certain head for the proponent to use). If one would be interested in obtaining a single positive solution only, the search would then terminate at the first successful leaf node. If all branches are unsuccessful, the search returns a negative answer, meaning that a claim (claims) cannot be credulously accepted under certain semantics. In our examples (Figs. 1.3 and 1.5) we will show full search of successful DDs, with the purpose of presenting how different choices would impact the outcome and how the search would be resumed after a potential failure (i.e. from the most recent branching-point).

Step 0. of the search tree depicted in Figure 1.3 symbolizes that at this stage only the goal p is a part of the current dispute state. Since it is not an assumption it cannot be attacked by the opponent yet. At the next step (1.) the proponent uses rule $p \leftarrow \boldsymbol{a}, \boldsymbol{b}, q$ to backward expand it. Because there are no other rules supporting p this choice is deterministic – the proponent has no other rule to use. But as soon as Step 1. is performed and $\boldsymbol{a}$ and $\boldsymbol{b}$ are introduced as the new defences the opponent can start constructing arguments against them. They do so in Steps 2-6. where the following arguments attacking $\boldsymbol{a}$ are constructed: $\bar{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{h}]$; $\bar{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{i}]$ and $\bar{a} \leftarrow [z \leftarrow \boldsymbol{f}]$. At this point, the opponent cannot perform any other moves, so the turn goes back to the proponent who chooses to complete its argument for p by trying to justify q. There are however two rules applicable at this point, namely $q \leftarrow \bar{e}, r$ and $q \leftarrow \boldsymbol{i}$, hence the proponent has to branch to consider them both. Note that at the sub-tree rooted at node 7, due to $\mathbf{e}$ becoming a culprit, the opponent arguments $\bar{a} \leftarrow \mathbf{e}, [v \leftarrow \mathbf{h}]$ and $\bar{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{i}]$ become counter-attacked. After Step 7. the proponent has to again make a choice between rules $r \leftarrow \mathbf{c}$ and $r \leftarrow \boldsymbol{d}$ resulting in node 8 (and 11). In Steps 9., 10. (and 12., 13.) the argument for p gets completed by expanding the statement $\bar{e}$ with rule $\bar{e} \leftarrow \mathbf{c}$ and a counter argument against the assumption $\mathbf{f}$ in the opponents argument $\bar{a} \leftarrow [z \leftarrow \boldsymbol{f}]$ is constructed, respectively. Note that after Step 10. the proponent does not have to backward expand $\bar{e}$ as it has already been justified in the previous moves[2].

Step 10. is a successful state as the proponent was able to construct a complete argument for p and counter attack all the opponent's arguments. The set of arguments formed by both players are the same as in Figure 1.2. The state at Step 13. is also a successful one, with the only difference that there, the statement r is justified by the assumption $\boldsymbol{d}$. The search terminates at Step 15. with its last branch being unsuccessful, as the opponent has managed to construct an argument attacking a defence $\boldsymbol{i}$, namely $\bar{i} \leftarrow \boldsymbol{g}$, which the proponent has no means to counter-attack.

We will refer to this strategy as "patient" since it makes the proponent wait patiently until the opponent finishes attacking their arguments. The "patient" strategy allowed to find a successful

[2] This behaviour is enforced by FlexDDs as an optimization and will be explained in the following chapter.

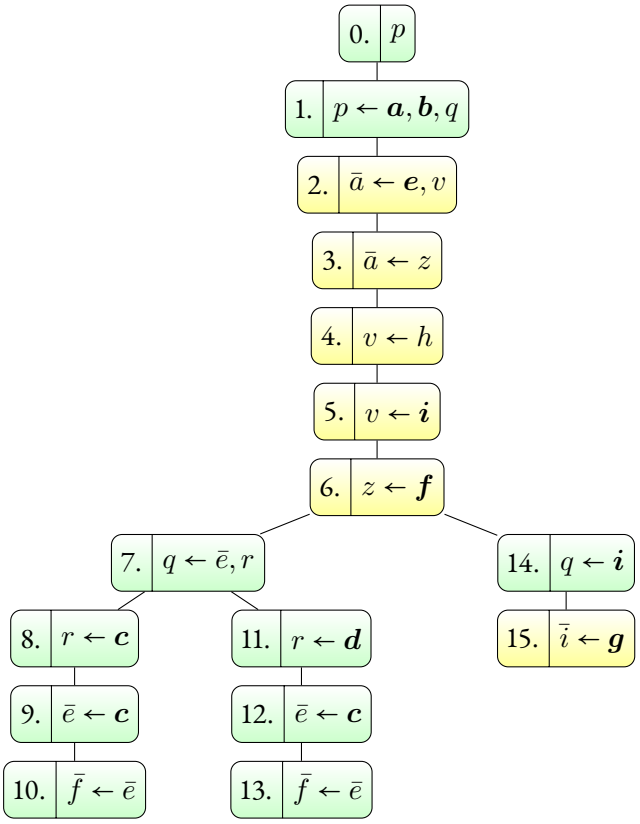


Figure 1.3: A "patient" strategy search tree for finding a successful dispute derivation. Nodes in green colour indicate proponent moves, whereas those in yellow indicate opponent's. Numbers labelling nodes specify the order in which the moves are performed. Assumptions are in boldface. Note that the tree branches whenever the proponent has more than one rule to use, i.e. nodes 7 and 14 for a rule with q in the head or nodes 8 and 11 for a rule with r in the head.

dispute derivation in 10 steps. It is of interest to see whether this could have been done in a smaller number of steps by considering a more "eager" strategy as shown in Figure 1.4.

The "eager" strategy presented in Figure 1.4 behaves conversely to the "patient" one, in that it makes the proponent move all they can before allowing the opponent to move. Note that in the sub-tree rooted at Step 2. the opponent does not use the rule $\bar{a} \leftarrow \boldsymbol{e}, v$. This is due to the constraint that the opponent must not use rules containing culprits, and since $\boldsymbol{e}$ becomes a culprit at Step 2., such rules will never appear in the sub-tree rooted at Step 2. As a consequence, the opponent does not have to backward expand v and neither $v \leftarrow \boldsymbol{h}$ nor $v \leftarrow \boldsymbol{i}$ are used in these sub-trees. On the whole, a successful dispute derivation is found at Step 7. and 12. Once again a fruitless node has been encountered, but this time one step earlier (at Step 14.).

For this particular example the "eager" strategy was successful 3 steps earlier than the "patient" one. This example however should not lead the reader to believe that the latter strategy will be superior in all cases. It is not difficult to think about a potential, slightly modified version of the framework from Example 1.1.1, in which the branch rooted at Step 3. of Figure 1.4 fails and the one

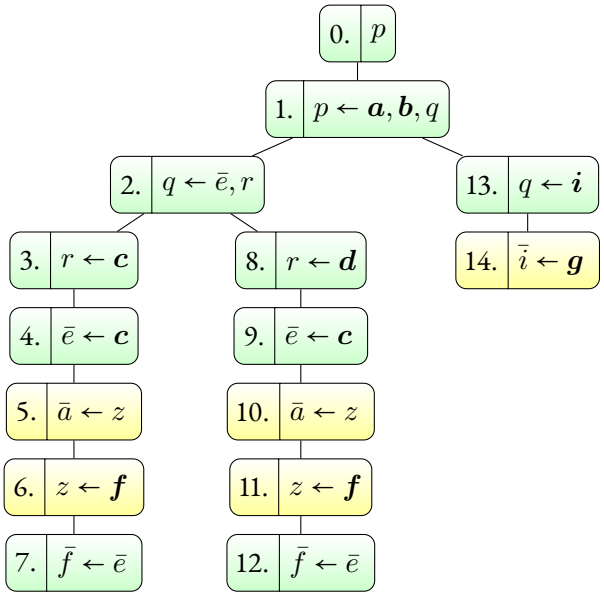


Figure 1.4: An "eager" strategy search tree for finding a successful dispute derivation.

rooted at Step 8. succeeds. Now, additionally assume that in this modified version the argument constructed at Steps 5-6. (and 10-11.), namely $\bar{a} \leftarrow [z \leftarrow \boldsymbol{f}]$ would take more intermediate steps ($\bar{a} \leftarrow [z \leftarrow \ldots \leftarrow \boldsymbol{f}]$). The computation of this argument would initially happen in the first, fruitless branch, and then would be repeated in the latter. In such a situation in which the opponent's argument construction is computationally demanding, the "patient" strategy can be more useful. On the other hand, the "eager" strategy can also reduce the number of rules for the opponent via the above-mentioned culprit constraint which filters out rules as shown in Figure 1.4. Therefore it is not so obvious which approach is better in general, if any.

FlexDDs additionally introduce another mechanism, called (conservative and non-conservative) forward reasoning which can also highly impact the efficiency of the search for a successful dispute derivation. Non-conservative forward reasoning allows the proponent to use assumptions which are not related to the current dispute state and its conservative counterpart allows them to use rules whose body consists solely of statements for which the proponent has generated (complete) arguments. The rule choice is no longer non-deterministic when using forward moves, as the proponent does not introduce any new statements he might later be unable to prove or defend. This follows from the fact that the rule body of rules introduced by forward moves consists of only justified statements and the rule head, being the only newly introduced statement, is being justified when using the rule. Search utilizing forward moves is illustrated in Figure 1.5.

In Step 1. of the dispute derivation search depicted in Figure 1.5 the proponent adds the assumption $\boldsymbol{c}$ to their argument set, which is not related to the dispute state at that stage. Because it is an assumption, and therefore a complete argument, the proponent later uses rules whose body contains only $\boldsymbol{c}$ (Steps 2. and 3.) thus constructing complete arguments for statements entailed by these rules. This is later repeated, but with rules whose body consists of the new statements they had created arguments for (Steps 4. and 5.). Ultimately they arrive at Step 6., where the goal

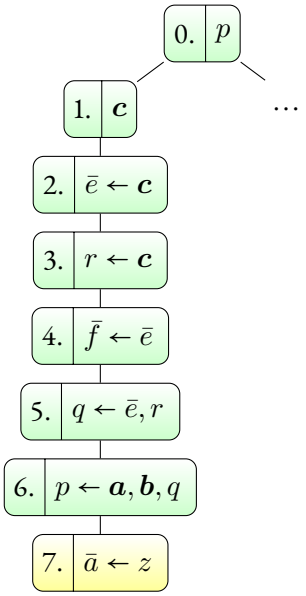


Figure 1.5: A strategy search tree for finding a successful dispute derivation utilizing forward reasoning.

p is derived. In the next step the opponent tries to attack the proponent arguments, but fails to construct a complete argument due to all rules relevant to the dispute being blocked (because of culprits in bodies).

In the presented example, the "forward" approach has found a successful dispute derivation in the same number of steps as the "eager" approach, but was able to do so with a reduced amount of branching. The benefits of that approach should be clear, as deterministic choices in general lead to faster computation. However, such non-deterministic assumption choices as the one at Step 1. (non-conservative forward moves), given that a framework contains numerous assumptions, might cause the search space to "explode " in practice.

We have seen three approaches for finding successful dispute derivations. Not only is that a negligibly small portion of all possible approaches, but the approaches themselves can be defined more precisely. E.g. it can be specified which statement s should be backward extended first, i.e. the one for which there are the most rules with s in the head or the fewest. Consider for example the search from Figure 1.3 again. Had the statement $\bar{e}$ been chosen to be backward expanded next after Step 7 (using rule $\bar{e} \leftarrow \boldsymbol{c}$), the branching caused with the choice between $r \leftarrow \boldsymbol{c}$ and $r \leftarrow \boldsymbol{d}$ would have been delayed by one step, causing the second successful dispute derivation to be found one step earlier.

When choosing between several possible rules $h \leftarrow B$ with the same head h, one could for example decide between those with the smallest or the largest cardinality of the body $|B|$. Assuming the smallest body approach in the search from Figure 1.3 again, the order of visiting nodes 7. and 14. would have been swapped, leading to finding the successful dispute state a few steps later (after considering the entire unsuccessful branch).

What is also interesting is to learn of the impact of the choice between *depth-first-search* (DFS) or *breadth-first-search* (BFS). The considered examples are all ordered according to DFS, and the

change of the search algorithm to BFS would result in the tree traversal by the steps ⟪0, 1, 23, 4, 5, 6, 7, 14, 8, 11, 15, 9, 12, 10, 13⟫ and ⟪0, 1, 2, 13, 3, 8, 14, 4, 9, 5, 10, 6, 11, 7, 12⟫ for Figure 1.3 and Figure 1.4, respectively, making the search significantly longer. It is not so difficult however to think of examples of situations where BFS search would come in handy.

Finally, the gain of efficiency of the intuitively major optimization of conservative forward reasoning is of interest, i.e. what is it compared to FlexDDs strategies not taking advantage of it and compared to current state-of-the-art systems for dispute derivations which do not employ it.

In this thesis we will formalise the search approaches described above as strategies in FlexDDs setup, devise search algorithms employing those strategies and evaluate them empirically, hopefully addressing issues posed above regarding various approaches to automatic search. We will also present our interactive system `flexABle`, which, among other notable features, implements said search procedures and compare its performance to the main state-of-the-art system for DDs in ABA, called `abagraph`.

1.2 Contributions and Thesis Structure

The contributions of this thesis are manifold. Firstly, related literature regarding dispute derivation variants and potential search strategies was reviewed, the results being described in the following section. Secondly, a small modification correcting the termination condition of FlexDDs for the complete semantics is shown in Chapter 3. Thirdly, search algorithms and strategies for FlexDDs have been devised and are presented in Chapter 4. Fourthly, these findings were later used when implementing a system supporting FlexDDs, called `flexABle`, capable of not only automatically searching for a successful dispute derivation, but also offering many other features related to FlexDDs. In particular, several mechanisms for interactive exploration of FlexDDs are incorporated in `flexABle`. The system and its most important features are discussed in Chapter 5. Fifthly, a series of experiments was carried out resulting in an evaluation of FlexDDs performance when used in `flexABle`, performance of several strategies for FlexDDs, as well as a comparison with the main competing state-of-the-art dispute derivation system. This evaluation is the subject of Chapter 6.

Remaining chapters discuss necessary formal background (Chapter 2), conclusions (Chapter 7) and possible directions of further research in this area (Chapter 8).

1.3 Related Work

As mentioned earlier, there have been several variants of dispute derivations to date. They differ between one another mostly in the representation of dispute states, which often has direct impact on the efficiency of the variant. In other cases they also offer support for different argumentation semantics.

Dispute derivations for ABA have first been introduced by Dung et al. [13], solely for the admissible semantics. They are defined as finite sequences of quadruples $(\mathcal{P}_i, \mathcal{O}_i, \mathcal{A}_i, \mathcal{C}_i)$, where P_i (O_i) denotes the set of (sets of) statements held by the proponent (opponent), $\mathcal{A}_i$ is the set of assumptions used by the proponent to justify their statements called defences and C_i the set of assumptions the proponent decided to attack. The latter are called culprits. Statements in P_i

and O_i can be seen as leaf nodes of argument trees built during the dispute which still need to be examined – potentially attacked by the opposing player if such a statement is an assumption or otherwise backward extended with a rule. A new dispute state is created in the first case by adding the contrary of the now attacked assumption to the attacking player's set. In the latter case a statement is backward expanded, i.e. replaced with the body of a rule with this statement in the head. Ultimately, if the derivation is successful the defences, being the set of assumptions supporting the initial goal claim, are returned.

The aforementioned dispute derivation machinery has been further extended resulting in *AB-*, *GB-*, and *IB-Dispute Derivations* [14] tailored for admissible, grounded and ideal semantics, respectively, which have also been implemented in Prolog as part of the `CaSAPI` [17] system. AB-Dispute Derivations were subject to yet another augmentation by Gaertner and Toni [18], giving rise to *Structured AB-Dispute Derivations*, which additionally record the set of arguments $Args$ implicitly computed by both players together with the attack relation Att between them. Moreover, the players keep track of their states no longer with just sets of statements but rather with "potential arguments" – tree structures representing incomplete arguments of the players which once completed are moved to $Args$. Interestingly, Gaertner and Toni identify five choice points which have to be defined in any system employing Structured AB-Dispute Derivations. Those choice points are the choice of the player (proponent or opponent), the choice of potential argument given a player, the choice of a premise given a potential argument (returning one statement belonging to the premises of the argument), the choice of rule given a statement (if more than one are applicable) and the choice to attack an assumption or not (when the opponent is selected and an assumption is picked as a premise of one of its arguments, the proponent can choose between attacking or ignoring it). The authors reveal some of their concrete choices when implementing the new version of `CeSAPI` (3.0) thus developing a search strategy, i.e. prioritizing proponent (choice of player), prioritizing arguments which still have non-assumptions in their premises (choice of argument) and prioritizing rules by the order in which they appear in the input ABA framework file (choice of rule).

AB-Dispute Derivations have undergone two more modifications resulting from the requirements of the setup in which these dispute derivations have been used by Craven et al. [8]. Here the authors attempt utilizing dispute derivations in support of medical decision-making, more specifically regarding treatments for early-stage breast cancer. When representing arguments solely as their pending premises (as is the case with P_i and O_i of AB-Dispute Derivations) the chain of reasoning from the goal statement to said premises through intermediate sub-arguments gets lost. This information is of crucial importance for clinicians, therefore the first modification enforces that each argument, apart from identifying its set of premises, is additionally represented by a tree. Moreover, as the second modification, a preference ordering can be imposed on the set of rules in an ABA framework. This is used to rank rules based on clinical trials according to the reliability level of those trials. A system implementing such a modified version of AB-Dispute Derivations, named `sxdd` has been developed in C++. Craven et al. have also pointed out a few choice points the parameters in their modified version of AB-Dispute Derivations (player choice, argument choice for both players, node choice given an argument for both players) and investigated the possible domains for these parameters. They have also evaluated all definable combinations using a single input framework. The presented results showed that only a small fraction of the strategies have been successful given a certain cut-off time. The authors argue that in practi-

cal setups spawning a large number of variants in parallel and relying on the assumption that at least some of them would terminate in reasonable time is a justifiable approach, but they do not attempt to find strategies being exceptionally superior on average.

In the meantime, Toni [27] has proposed *Structured X-Dispute Derivations*, generalising AB-, GB- and IB-dispute derivations into one parametrized procedure and simultaneously recording the complete arguments and attacks between them constructed in the process as in Structured AB-Dispute Derivations.

Novel notions of *rule-minimal arguments* and *argument graphs* have enabled Craven and Toni [9] and Craven et al. [7] to devise *Graphical Dispute Derivations* (Graph-DDs). This variant keeps track of justifications of claims for both players obtained during the derivation, which can later be re-used if claims again appear in the players sets. This avoids re-computation of claims already argued for and ensures that obtained arguments are rule-minimal, i.e. constructed from no two different rules with the same head, which is desirable from the perspective of performance as well as for conceptual reasons. Graph-DDs support admissible and grounded semantics. The authors have developed Prolog based systems for Graph-DDs and Structured X-Dispute Derivations, as defined by Toni [27] (named `abagraph` and `proxdd`, respectively) in order to obtain an evaluation of both approaches, which proved superiority of the first machinery over the latter. The authors define a number of possible strategies resembling those mentioned before and state that any of those can be manually selected in their system. For experiments they randomly selected a few of them and configured both evaluated systems to follow the same strategies for the same problems. This was however used to assess the implementations, rather than find out about more meaningful or optimal search strategy.

Finally, *Flexible Dispute Derivations* (FlexDDs) have been introduced by Diller et al. [11]. Each dispute state in FlexDDs contains "a closure" of arguments for both players, i.e. all rule-minimal arguments constructible from rules and assumptions used by players throughout the derivation. Moreover, FlexDDs optimizes the procedure by e.g. allowing the opponent to use the proponent arguments as well as introducing forward reasoning to dispute derivations. Additionally, apart from supporting admissible semantics, FlexDDs offer procedures for finding defence sets for semantics never addressed in dispute derivations for ABA: stable and complete.

2 Foundations

In this chapter we will formally introduce notions, starting off with the most general ones from ABA and then considering relevant concepts from flexible dispute derivations necessary to understand the issues investigated in this thesis.

2.1 Assumption-Based Argumentation

Definition 2.1.1 (Assumption-Based Argumentation (ABA) Framework)**.** [28] An ABA framework is a tuple $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ where:

- $(\mathcal{L}, \mathcal{R})$ is a deductive system, with the set $\mathcal{L}$ called the *alphabet* and $\mathcal{R}$ a set of inference rules, each of the form $h \leftarrow B$ with $\{h\}, B \subseteq \mathcal{L}$. Elements h and B are often referred to as the rule's *head* and *body*, respectively, whereas the elements of $\mathcal{L}$ are called statements (or claims),
- $\mathcal{A} \subseteq \mathcal{L}$ is a (non-empty) subset of the alphabet, whose elements are referred to as *assumptions*
- $^-$ is a total mapping from $\mathcal{A}$ into $\mathcal{L}$, where, for an assumption $a \in \mathcal{A}$, $\overline{a}$ is the contrary of a.

Additionally, for a set of statements $S \subseteq \mathcal{L}$ we define its contraries as $\overline{S} = \{\overline{u} \in \mathcal{L} \mid u \in (S \cap \mathcal{A})\}$. In all of this thesis (unless explicitly stated otherwise) we will assume the ABA framework to be fixed and thus not define notions relative to an ABA framework. Furthermore, the fixed ABA framework will be assumed to be flat, as defined in Definition 2.1.2.

Definition 2.1.2 (Flatness)**.** An ABA Framework is said to be *flat*, if it holds that no assumption appears in the head of any rule, formally $\{h \mid h \leftarrow B \in \mathcal{R}\} \cap \mathcal{A} = \varnothing$.

The concept of an argument in ABA has been defined in several different ways. To remain consistent with the notions which flexible dispute derivations operate on, we will stick to the definition used in FlexDDs [11], which, in turn, borrow from ASPIC+ [22].

Definition 2.1.3 (ABA Argument)**.** Given a flat ABA framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, we define a potential argument recursively as follows:

- $A = s$ if $s \in \mathcal{L}$. Then we have that $Conc(A) = s$, $Prem(A) = s$, $Asm(A) = \{s\} \cap \mathcal{A}$, $Sub(A) = s$.

- $A = s \leftarrow A_1, \ldots, A_n$ if $s \in \mathcal{L}$ and $A_1, \ldots, A_n$ are arguments such that there exists a rule $s \leftarrow Conc(A_1), \ldots, Conc(A_n) \in \mathcal{R}$. Then we have that $Conc(A) = s$, $Prem(A) = Prem(A_1) \cup \ldots \cup Prem(A_n)$, $Asm(A) = Asm(A_1) \cup \ldots \cup Asm(A_n)$, $Sub(A) = \{A\} \cup Sub(A_1) \cup \ldots \cup Sub(A_n)$.

Functions $Conc$,$Prem$,Asm, and Sub return the conclusion, set of premises, assumptions, and sub-arguments of a given argument, respectively. Given an argument A we say that the set of statements $Prem(A)$ supports A and we call A an argument for claim s if $s = Conc(A)$.

As an example, consider the argument $A = p \leftarrow \boldsymbol{a}, \boldsymbol{b}, [q \leftarrow \overline{e}, r]$ constructible from the ABA framework from Example 1.1.1. Then $Conc(A) = p$, $Prem(A) = \{\boldsymbol{a}, \boldsymbol{b}, \overline{e}, r\}$, $Asm(A) = \{\boldsymbol{a}, \boldsymbol{b}\}$, and $Sub(A) = \{A; \boldsymbol{a}; \boldsymbol{b}; q \leftarrow \overline{e}, r; \overline{e}; r\}$. A is an argument for the claim p.

Additionally, we call an argument A' complete if it holds that $Prem(A') \subseteq \mathcal{A}$, otherwise A' is a potential argument, which holds for the above-considered argument A as it has two non-assumption premises, namely $\overline{e}$ and r. Usually, only the complete arguments are actually called arguments in ABA. However, dispute derivations operate on their "potential" variant and hence the need to extend the definition. We note in particular that stand-alone claims and rules also are thus (simple one-step potential) arguments.

We extend the set of functions $F = \{Conc, Prem, Asm, Sub\}$ for sets of arguments $Args$ in the obvious manner, i.e. $f(Args) = \bigcup_{A \in Args} f(A)$ for $f \in F$.

Definition 2.1.4 (Rule-minimality)**.** [7] An argument A is said to be rule-minimal if there are no two sub-arguments $h \leftarrow B, h' \leftarrow B' \in Sub(A)$ s.t. $h = h'$ and $B \neq B'$.

We will extend this notion to a set of arguments $Args$ in the following manner: $Args$ is said to be rule-minimal if there are no two sub-arguments $h \leftarrow B, h' \leftarrow B' \in Sub(Args)$ s.t. $h = h'$ and $B \neq B'$. Therefore not only every argument $A \in Args$ is rule-minimal, but also every claim is justified by exactly the same rule in each sub-argument contained in $Args$.

An argument A is said to *attack* another argument A' if its conclusion is the contrary of one of the assumptions of A', formally if $Conc(A) = \overline{u}$ for some $u \in Asm(A')$ [28]. Furthermore, if A' attacks another argument A'', then A is said to *defend* A'' from A'.

Attack and defense have been further equivalently redefined w.r.t. sets of assumptions. A set of assumptions U attacks another set of assumptions U' if an argument supported by U attacks an argument supported by U'. Naturally, if another set of assumptions U'' is attacked by U', then U defends it from U'.

Finally, the relations of attack and defence can also be defined in a hybrid way, between sets of assumptions and arguments as in Definition 2.1.5.

Definition 2.1.5 (Hybrid attack relations)**.** Given two sets of assumptions U and an argument A, we say that:

- A attacks U if $Conc(A) = \overline{u}$ and $u \in U$,
- U attacks A if there exists an argument A_U s.t. $Prem(A_U) \subseteq U$ and A_U attacks A.

Corresponding hybrid notions of defense can easily be obtained from Definition 2.1.5. Those relations are also often extended to sets of entities, i.e. sets of arguments and sets of sets of assumptions if a property holds for single elements of the sets. E.g. a set of arguments $Args$ attacks

another set of arguments $Args'$ if there is an argument in $Args$ which attacks an argument in $Args'$.

Semantics are the central notion in formal argumentation, determining the way to choose arguments coherent with each other. Definition 2.1.6 describes semantics relevant to the thesis in terms of sets of assumptions. Other, equivalent versions utilizing the arguments-based or the hybrid view can be obtained by using the argument-based or hybrid notions of attack and defense, respectively.

Definition 2.1.6 (Considered argumentation semantics)**.** [28] A set of assumptions $U \subseteq \mathcal{A}$ is said to be a σ-extension, where σ is called:

- *admissible* if U does not attack itself (is conflict-free) and defends itself from all sets of assumptions attacking it,
- *complete* if U is admissible and contains all assumptions that it defends,
- *stable* if U is admissible and contains all assumptions it does not attack,
- *preferred* if U is $\subseteq$-maximal admissible, i.e. there is no admissible extension U' s.t. $U \subset U'$,
- *grounded* if U is $\subseteq$-minimal complete, i.e. there is no complete extension U' s.t. $U \supset U'$.

Sticking to the convention of flexibility in operating on either sets of arguments or assumptions, we will use the term σ-extension sometimes to refer to sets of arguments and sometimes to sets of assumptions. We will only state exactly which we mean when this cannot clearly be inferred from the context.

2.2 Flexible Dispute Derivations

In this section we will introduce notions related to flexible dispute derivations. All of them are taken from [11].

Definition 2.2.1 (Argument expansion)**.** An expansion of a set of arguments $Args = \{A_1, \ldots, A_n\}$ w.r.t an argument A' with $Conc(A_1) \cup \ldots \cup Conc(A_n) \subseteq Prem(A')$ is obtained from A' by replacing at least one $s_i \in Prem(A')$ for which $s_i = Conc(A_i)$ with A_i for each $1 \leq i \leq n$. We denote it $A' \lessdot Args$. When $n = 1$, we will often denote the expansion as $A' \lessdot A_1$.

In line with Definition 2.2.1 we can further distinguish between a *forward-* and a *backward expansion*. The *forward expansion* of a set of arguments $Args$ w.r.t. the set of rules $\mathcal{R}$ (treated as 1-step arguments) is of the form $h \leftarrow B \lessdot Args$, with $h \leftarrow B \in Args$. On the other hand, the *backward expansion* of an argument A w.r.t. $\mathcal{R}$ is an expansion of the form $A \lessdot h \leftarrow B$, given that $h \leftarrow B \in \mathcal{R}$.

As an example consider the framework from Example 1.1.1 and a potential (incomplete) argument of the form $A = p \leftarrow \boldsymbol{a}, \boldsymbol{b}, q$ with $Prem(A) = \{\boldsymbol{a}, \boldsymbol{b}, q\}$ Given that there exists a rule $q \leftarrow \bar{e}, r \in \mathcal{R}$ a backward expansion $A' = A \lessdot q \leftarrow \bar{e}, r$ yields another incomplete argument $A' = p \leftarrow \boldsymbol{a}, \boldsymbol{b}, [q \leftarrow \bar{e}, r]$. Alternatively, assuming that there exists a singleton set $Args = \{\overline{e} \leftarrow \boldsymbol{c}\}$ containing a complete argument for $\overline{e}$ and given a rule $f \leftarrow \overline{e} \in \mathcal{R}$, a forward expansion $A'' = f \leftarrow \overline{e} \lessdot Args$ yields an argument $A'' = f \leftarrow [\overline{e} \leftarrow \boldsymbol{c}]$.

Definition 2.2.2 (Argument set closure)**.** Let $Args$ be a set of arguments and A a single argument. Then $Args \bowtie A$ is the closure of $Args \cup \{A\}$ under sub-arguments and argument expansions, where a set of arguments $Args'$ is said to be:

- closed under sub-arguments if $Args' = Sub(Args')$ and
- closed under argument expansions if every argument $A' = A'' \lessdot Args''$ obtainable by argument expansion from any $A'' \in Args'$, $Args'' \subseteq Args'$ is in $Args'$.

$Args \rtimes A$ denotes a rule-minimal closure of $Args \cup \{A\}$. Then assuming that $Args$ is closed under sub-arguments, closed under argument expansions and rule minimal, $Args \rtimes A$ is the closure under sub-arguments and arguments expansions of $Args \cup \{A\}$ if this closure is also rule minimal, while otherwise $Args \rtimes A = Args$ (i.e. expansions making the argument set not rule-minimal are disallowed).

Flexible dispute derivations (FlexDDs), which we described informally in the introduction as a game between two fictional players, namely the proponent and the opponent of a set of claims γ in question, consist of a sequence of tuples $(\mathcal{B}, \mathcal{P})$. Each tuple denotes a dispute state, where $\mathcal{B}$ and $\mathcal{P}$ represent the opponent's and proponent's sets of arguments, respectively. Note that one of the refinements of FlexDDs was enabling the opponent to re-use the proponent's arguments and therefore given a sequence of dispute states of a FlexDDs $(\mathcal{B}_1, \mathcal{P}_1), \ldots, (\mathcal{B}_n, \mathcal{P}_n)$ it holds that $\mathcal{P}_i \subseteq \mathcal{B}_i$ for all $1 \leq i \leq n$. That is also why the opponent's set is not denoted with $\mathcal{O}$ (as was in previous iterations of DDs for ABA), but instead with $\mathcal{B}$ (which stands for *Both*). Goals are assumed to be consistent, i.e. $\gamma \cap \overline{\gamma} = \varnothing$. Each next dispute state is a result of a dispute advancement carried out by either player and the initial dispute state is of the form (γ, γ), i.e. contains goals only. Prior to defining an advancement, let us state the relevant auxiliary notations defined by Diller et al. [11] in Table 2.1.

Notation	Description
$\mathcal{D} = Asm(\mathcal{P})$	Defences
$\mathcal{C} = \{u \in \mathcal{A} \mid \overline{u} \in Conc(\mathcal{P})\}$	Culprits
$\mathcal{R}^- = \{h \leftarrow B \in \mathcal{R} \mid B \cap \mathcal{C} \neq \varnothing\}$	Blocked rules (culprits in bodies)
$\mathcal{R}^{\sim} = \{h \leftarrow B \in \mathcal{R} \mid (\{h\} \cup B) \cap (\overline{B} \cup \mathcal{C} \cup \overline{\mathcal{D}}) \neq \varnothing\}$	Rules blocked for the proponent (either inconsistent or containing culprits or contraries of defences in head or body)
$\mathcal{P}^* = \{a \in \mathcal{P} \mid Prem(a) \subseteq \mathcal{A}\}$	Proponent's "complete" arguments
$\mathcal{B}^{*\setminus^-} = \{a \in \mathcal{B} \mid Prem(a) \subseteq (\mathcal{A} \setminus \mathcal{C})\}$	Opponent's "complete" unblocked arguments
$\mathcal{P}^+ = \{a \in \mathcal{P} \setminus \mathcal{P}^* \mid \neg\exists a' \neq a \in \mathcal{P}$ s.t. $Conc(a') = Conc(a)$ and $a' \in \mathcal{P}^* or\ a \in Sub(a')\}$	Maximal incomplete proponent arguments
$\mathcal{P}^{\#}_{\gamma\cup\overline{\mathcal{C}}} = \{a \in \mathcal{P}^+ \mid Conc(a) \in \gamma \cup \overline{\mathcal{C}}\}$	Maximal incomplete proponent arguments for goals and contraries of culprits
$\mathcal{B}^{!\setminus^-}_S = \{a \in \mathcal{B} \mid Asm(a) \cap \mathcal{C} = \varnothing, Conc(a) \in S\}$	Unblocked arguments with conclusions in $S \subseteq \mathcal{L}$
$\mathcal{A}^! = \{u \in \mathcal{A} \mid u \in Asm(\mathcal{B}^{!\setminus^-}_{\mathcal{D}})\}$	Candidates for culprits
$\mathcal{J} = \{u \in \mathcal{A} \setminus \mathcal{C} \mid \overline{u} \notin Conc(\mathcal{B}^{*\setminus^-})\}$	Assumptions defended at the dispute state
$\mathcal{H} = \{h \mid h \leftarrow B \lessdot A, h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^{\sim}, A \subseteq \mathcal{P}^*\}$	Conclusions of arguments obtainable from $\mathcal{P}^*$ by forward expansion

Table 2.1: Auxiliary notation for FlexDDs. All defined w.r.t. a dispute state $(\mathcal{B}, \mathcal{P})$.

In FlexDDs a proponent dispute state advancement from a dispute state $(\mathcal{B}, \mathcal{P})$ is a dispute state $(\mathcal{B}', \mathcal{P}')$ with $\mathcal{P}' \rtimes \{a\} \neq \mathcal{P}$, $\mathcal{B}' = \mathcal{B} \bowtie \{a\}$, $X_1 \subseteq \overline{\mathcal{A}}$, $X_2 \subseteq \mathcal{A}$, where a is defined as in Table 2.2.

Move type	V.	a definition
PB$-(\overline{\mathcal{A}^!} \cup X_1)$	1	$a = a' \lessdot h \leftarrow B$ for $h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^{\sim}, a' \in \mathcal{P}^{\#}_{\gamma\cup\overline{\mathcal{C}}}$
	2	$a = h \leftarrow B$ for $h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^{\sim}$ with $h \in (\overline{\mathcal{A}^!} \cup X_1) \setminus \overline{\mathcal{D}}$
PF$-((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup X_2)$	1	$a = h \leftarrow B \lessdot A$ for $h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^{\sim}, A \subseteq \mathcal{P}^*$
	2	$a = u$ for $u \in ((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup X_2) \setminus (\overline{u} \cup \mathcal{C} \cup \overline{\mathcal{D}})$

Table 2.2: Possible proponent moves in FlexDDs. Column "Move type" denotes the type of move and defines variable parameters w.r.t. which moves can be defined. Each of the moves comes in two variants, denoted in the column "V.". Finally, each of the moves produces a different (potential) argument a, described in column "a definition", upon which the proponent dispute state advancement depends.

On the other hand, an opponent dispute state advancement from a dispute state $(\mathcal{B}, \mathcal{P})$ is a dispute state $(\mathcal{B}', \mathcal{P})$ with $\mathcal{B}' \bowtie \{a\}$, $Y_1 \subseteq \overline{\mathcal{A}}$, $Y_2 \subseteq \mathcal{A}$, where a is defined as in Table 2.3.

Move type	V.	a definition
OB-$(\overline{\mathcal{D}} \cup Y_1)$	1	$a = a' \ltimes h \leftarrow B$ for $a' \in \mathcal{B}^{!\,\setminus^-}_{\overline{\mathcal{D}} \cup Y_1}, h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^-$
	2	$a = h \leftarrow B$ for $h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^-$ with $h \in \overline{\mathcal{D}} \cup Y_1$
OF-$((\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2)$	1	$a = h \leftarrow B \ltimes A$ for $A \subseteq \mathcal{B}^{*\,\setminus^-}, h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^-$
	2	$a = u$ for $u \in (\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2 \setminus \mathcal{C}$

Table 2.3: Possible opponent moves in FlexDDs. Column "Move type" denotes the type of move and the move's variable parameters, and "V." the variant. "a definition" defines the new argument constructed within the move.

Note that both players have two types of moves (proponent having PB-$(\overline{\mathcal{A}^!} \cup X_1)$ and PF-$((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup X_2)$ whereas opponent having OB-$(\overline{\mathcal{D}} \cup Y_1)$ and OF-$((\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2)$), each of which have, in turn, two variants. This yields four ways for either player to construct the new argument a and create a new dispute state. Furthermore, observe that the moves are parametrized – their behaviour can be altered by modifying parameters X_1 and X_2 for the proponent and Y_1 and Y_2 for the opponent. Table 2.4 presents *advancement types* – predefined ways of constraining players' moves, by specifying types of moves which can be used together with the parameters. Note that the DF advancement type is the "least constraining" variant, whereas DAB is the "most constraining" one, which manifests itself two ways. Firstly, contrary to DAB, DF allows for PF1 and OF1 moves. Secondly, in DAB PF and OF moves are defined w.r.t. a subset of all assumptions, whereas PB and OB are defined w.r.t. a subset of assumptions' contraries. Opposed to that, DF defines its moves w.r.t. to all assumptions and all assumptions' contraries, respectively. Other advancement types, i.e. DABF, DS are less constrained than DAB and more than DF. With slight abuse of the $\subseteq$ notation, we distinguish between more and less constrained variants, i.e. given two advancement types AT, AT' we denote $AT \subseteq AT'$ if AT' is less constrained than AT (or equivalently all A moves are A' moves), yielding DAB $\subseteq$ DABF, DABF $\subseteq$ DS, and DS $\subseteq$ DF.

Advancement	Proponent	Opponent
DAB	PB-$(\overline{\mathcal{A}^!})$, PF-$(\overline{\mathcal{A}^!} \cap \mathcal{A})$-2	OB-$(\overline{\mathcal{D}})$, OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2
DABF	PB-$(\overline{\mathcal{A}^!})$, PF-$(\overline{\mathcal{A}^!} \cap \mathcal{A})$	OB-$(\overline{\mathcal{D}})$, OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2
DS	PB-$(\overline{\mathcal{A}^!})$, PF-$(\mathcal{A})$	OB-$(\overline{\mathcal{D}})$, OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2
DF	PB-$(\overline{\mathcal{A}})$, PF-$(\mathcal{A})$	OB-$(\overline{\mathcal{A}})$, OF-$(\mathcal{A})$

Table 2.4: Advancement types

Finally, FlexDDs define so called *termination criteria* for argumentation semantics determining the outcome of the dispute derivation, where a certain criterion being satisfied by an obtained dispute state $(\mathcal{B}, \mathcal{P})$ guarantees that the arguments in $\mathcal{P}$ (or equivalently the set of defences $\mathcal{D}$) constitute an extension of the semantics associated with the criterion. Table 2.5 presents the termination criteria TA and TS for the admissible and stable semantics, respectively. We introduce

a slight correction to the termination criteria TC proposed by Diller et al. [11] for the complete semantics; this issue we will consider in detail in the next chapter.

C	Prop. Winning	Opp. Cannot Move	Prop. Cannot Move
TA	$\gamma \cup \overline{\mathcal{C}} \subseteq Conc(\mathcal{P}^*)$, $\mathcal{B}^{!\setminus^-}_{\overline{\mathcal{D}}} \cap \mathcal{B}^{*\setminus^-} = \varnothing$	OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 or OF-$(\mathcal{A})$	PB-$(\overline{\mathcal{A}^!}) \cup$ PF-$(\overline{\mathcal{A}^!} \cap \mathcal{A})$-2 or PF-$(\mathcal{A})$
TS	$\gamma \cup \overline{\mathcal{C}} \subseteq Conc(\mathcal{P}^*)$, $\mathcal{B}^{!\setminus^-}_{\overline{\mathcal{D}}} \cap \mathcal{B}^{*\setminus^-} = \varnothing$, $\mathcal{D} \cup \mathcal{C} = \mathcal{A}$	OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 or OF-$(\mathcal{A})$	PB-$(\overline{\mathcal{A}^!}) \cup$ PF-$(\mathcal{A})$

Table 2.5: Termination criteria for admissible and stable semantics

The meaning of the criteria is as follows: if an obtained dispute state $(\mathcal{B}, \mathcal{P})$ satisfies the condition defined in the "Prop. Winning" column and the opponent cannot further advance the dispute state in at least one of the two ways defined in the disjuncts of "Opp. Cannot Move" column, then the proponent wins and $\mathcal{P}$ (and hence $\mathcal{D}$) is an extension associated with the criterion. Otherwise, if the condition in the "Prop. Winning" is not satisfied and the proponent cannot advance in at least one of the two ways defined in "Prop. Cannot Move", then the dispute derivation terminates with a negative answer. If neither of the two cases hold, the dispute derivation does not terminate and shall be continued.

Given an advancement type AT and a termination criterion C we refer to a flexible dispute derivation variant in which each advancement is in accordance with AT and which is terminated when C is satisfied as a AT+C FlexDD. It has been shown by Diller et al. [11] that DF+TA FlexDDs are sound for the admissible semantics, meaning that if there is a DF + TA FlexDD ending with a dispute state $(\mathcal{B}, \mathcal{P})$ and the proponent as the winner, then the set of defences $\mathcal{D}$ is an admissible extension w.r.t. which the set of goals γ is acceptable. Naturally, this holds for all other "more constraining" advancement types. Furthermore, similar results were obtained for DF+ TS FlexDDs and the stable semantics.

It has also been shown that if $\mathcal{L}$ is finite, then DAB + TA FlexDDs are complete for credulous acceptance w.r.t. the admissible semantics, meaning that if the set of goals γ is acceptable for some admissible extension then there is a DAB + TA FlexDD ending with a dispute state $(\mathcal{B}, \mathcal{P})$ and the proponent as winner s.t. $\mathcal{D}$ is an admissible extension w.r.t. which γ is acceptable. Again, naturally this holds for all other "less restricting" advancement types and similar results were obtained for DS + TS FlexDDs and the stable semantics.

Table 2.6 presents an example of a successful DAB + TA FlexDD for the framework from Example 1.1.1. It terminates at Step 10. with a positive answer as TA is satisfied: a complete argument has been constructed for the only goal p and the culprits $\boldsymbol{f}$ and $\boldsymbol{e}$, hence fulfilling the first condition of "Prop. winning".

Step and move type	$\mathcal{P}$	$\mathcal{B} \setminus \mathcal{P}$
0.	$\{p\}$	$\{\}$
1. (PB1, $p \leftarrow \boldsymbol{a}, \boldsymbol{b}, q$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, q\}$	$\{\}$
2. (OB2, $\overline{a} \leftarrow \boldsymbol{e}, v$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, q\}$	$\{\overline{a} \leftarrow \boldsymbol{e}, v\}$
3. (OB2, $\overline{a} \leftarrow z$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, q\}$	$\{\overline{a} \leftarrow \boldsymbol{e}, v; \overline{a} \leftarrow z\}$
4. (OB1, $v \leftarrow \boldsymbol{h}$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, q\}$	$\{\overline{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow z\}$
5. (OB1, $v \leftarrow \boldsymbol{i}$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, q\}$	$\{\overline{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{i}]; \overline{a} \leftarrow z\}$
6. (OB1, $z \leftarrow \boldsymbol{f}$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, q\}$	$\{\overline{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow \boldsymbol{e}, [v \leftarrow \boldsymbol{i}]; \overline{a} \leftarrow [z \leftarrow \boldsymbol{f}]\}$
7. (PB1, $q \leftarrow \overline{e}, r$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, [q \leftarrow \overline{e}, r]\}$	$\{\overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{i}]; \overline{a} \leftarrow [z \leftarrow \boldsymbol{f}]\}$
8. (PB1, $r \leftarrow \boldsymbol{c}$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, [q \leftarrow \overline{e}, [r \leftarrow \boldsymbol{c}]]\}$	$\{\overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{i}]; \overline{a} \leftarrow [z \leftarrow \boldsymbol{f}]\}$
9. (PB1, $\overline{e} \leftarrow \boldsymbol{c}$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, [q \leftarrow [\overline{e} \leftarrow \boldsymbol{c}], [r \leftarrow \boldsymbol{c}]]\}$	$\{\overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{i}]; \overline{a} \leftarrow [z \leftarrow \boldsymbol{f}]\}$
10. (PB1, $\overline{f} \leftarrow \overline{e}$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, [q \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}], [r \leftarrow \boldsymbol{c}^{\mathcal{D}}]]; \overline{f} \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}]\}$	$\{\overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{h}]; \overline{a} \leftarrow \boldsymbol{e}^{\mathcal{C}}, [v \leftarrow \boldsymbol{i}]; \overline{a} \leftarrow [z \leftarrow \boldsymbol{f}^{\mathcal{C}}]\}$

Table 2.6: A DAB + TA FlexDD for Example 1.1.1. Only maximal arguments (i.e. arguments which are not sub-arguments of other arguments within a players set) are shown. A statement s is marked as $s^{\mathcal{D}}$ if s is a defence or as $s^{\mathcal{C}}$ if it is a culprit.

Although the opponent has managed to create three complete arguments against a defence a, all of them contain a culprit in their premises, and therefore are blocked. This way the second "proponent winning" condition is satisfied. Finally, note that the opponent cannot advance in the OB-($\overline{\mathcal{D}}$) manner: there are no unblocked arguments against defences which could be backward expanded (OB-($\overline{\mathcal{D}}$)-1), neither are there any more rules with heads as contraries of defences (OB-($\overline{\mathcal{D}}$)-2). What is more, no assumption is a contrary of any defence (OF-($\overline{\mathcal{D}}$)-2). All this means the fulfillment of the "Opp. Cannot Move" column condition, collectively causing the FlexDD to terminate with an affirmative answer. Note that the performed moves correspond to the leftmost branch of Figure 1.3.

Table 2.7 presents another example of a FlexDD for the same framework as before, only this time it is of type DS + TA (it could also be a DF + TA). At step 1. a PF-$((\overline{\mathcal{A}^{\mathrm{I}}} \cap \mathcal{A}) \cup \mathcal{A})$-2 move is performed, yielding the new argument – assumption c – being a part of the set $\mathcal{J}$, which would not be possible in DAB + TA. Furthermore, at Steps 2-5. moves of types PF1 are performed which again are disallowed in DAB.

At Step 7. the dispute terminates as once again the opponent is unable to continue according to the selected advancement type: argument $\overline{a} \leftarrow z$ is not complete and cannot be backward expanded with OB-$(\overline{\mathcal{D}})$-1, because the only applicable rule $z \leftarrow \boldsymbol{f}$ contains a culprit $\boldsymbol{f}$. Rule $\overline{a} \leftarrow \boldsymbol{e}, v$ cannot be used either, because it contains a culprit $\boldsymbol{e}$. On the other hand, the proponent has managed to construct complete arguments for the goal as well as culprit contraries and therefore wins the dispute derivation. Note that the performed moves from this example are those of Figure 1.5.

Step and move type	$\mathcal{P}$	$\mathcal{B} \setminus \mathcal{P}$
0.	$\{p\}$	$\{\}$
1. (PF2, c)	$\{p;\ \boldsymbol{c}^{\mathcal{D}}\}$	$\{\}$
2. (PF1, $\overline{e} \leftarrow \boldsymbol{c}$)	$\{p;\ \overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}\}$	$\{\}$
3. (PF1, $r \leftarrow \boldsymbol{c}$)	$\{p;\ r \leftarrow \boldsymbol{c}^{\mathcal{D}}, \overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}\}$	$\{\}$
4. (PF1, $\overline{f} \leftarrow \overline{e}$)	$\{p;\ r \leftarrow \boldsymbol{c}^{\mathcal{D}}, \overline{f} \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}]\}$	$\{\}$
5. (PF1, $q \leftarrow \overline{e}, r$)	$\{p;\ q \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}], [r \leftarrow \boldsymbol{c}^{\mathcal{D}}]; \overline{f} \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}]\}$	$\{\}$
6. (PB1, $p \leftarrow \boldsymbol{a}, \boldsymbol{b}, q$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, [q \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}], [r \leftarrow \boldsymbol{c}^{\mathcal{D}}]]; \overline{f} \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}]\}$	$\{\}$
7. (OB2, $\overline{a} \leftarrow z$)	$\{p \leftarrow \boldsymbol{a}^{\mathcal{D}}, \boldsymbol{b}^{\mathcal{D}}, [q \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}], [r \leftarrow \boldsymbol{c}^{\mathcal{D}}]]; \overline{f} \leftarrow [\overline{e} \leftarrow \boldsymbol{c}^{\mathcal{D}}]\}$	$\{\overline{a} \leftarrow z\}$

Table 2.7: A DS + TA FlexDD for Example 1.1.1

2.3 Flexible Dispute Derivations – Rule-Based Variant

An alternative representation of FlexDDs, namely rule-based flexible dispute derivations (RlFlexDDs) has also been proposed, in which both players put forward statements and rules, rather than arguments. Similarly to regular FlexDDs, RlFlexDDs are sequences of dispute states of the form $(\mathbb{B}, \mathbb{P})$, where $\mathbb{B} \subseteq (\mathcal{L} \cup \mathcal{R})$ and $\mathbb{P} \subseteq \mathbb{B}$, thus rather than storing arguments they record elements of the framework. Again, a consistent set of goals γ is assumed, and the initial dispute state is of the form (γ, γ) with the goals viewed as elements of the alphabet, rather than arguments as in regular FlexDDs.

This kind of formalization of flexible dispute derivations leads to a more efficient implementation by avoiding the necessity of computing arguments and closures w.r.t. arguments. Instead, as stated above, one can keep track of the dispute derivation by simply storing framework elements.

In order to define the rule-based advancements, in Table 2.8 we first define auxiliary notations closely related to those from the regular FlexDDs, but now in the rule-based context.

Notation	Description
$\mathcal{D} = \mathbb{P} \cap \mathcal{A}$	Defences
$\mathcal{C} = \{u \in \mathcal{A} \mid \overline{u} \in \mathbb{P}\}$	Culprits
$\mathcal{J}_{\mathbb{B}} = \mathcal{R} \setminus \mathbb{B}$	Remaining rules for the opponent
$\mathcal{J}_{\mathbb{P}} = \mathcal{R} \setminus \mathbb{P}$	Remaining rules for the proponent
$\mathcal{J}_{\mathbb{B}}^{-} = \{h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \mid B \cap \mathcal{C} \neq \varnothing\}$	Remaining rules blocked for the opponent
$\mathcal{J}_{\mathbb{P}}^{\sim} = \{h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \mid (\{h\} \cup B) \cap (\overline{B} \cup \mathcal{C} \cup \overline{\mathcal{D}}) \neq \varnothing\}$	Remaining rules blocked for the proponent
$(\mathbb{P} \cap \mathcal{L})^{\downarrow} = \{s \in \mathbb{P} \cap \mathcal{L} \mid \neg\exists h \leftarrow B \in \mathbb{P} \text{ with } h = s\}$	Played unexpanded statements of the proponent
$(\mathbb{B} \cap \mathcal{L})^{\uparrow\uparrow} = \{s \in (\mathbb{B} \cap \mathcal{L}) \mid \neg\exists h \leftarrow B \in (\mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{-} \text{ with } h = s)\}$	Played fully expanded statements
$\mathbb{B}^{-} = (\mathbb{B} \cap \mathcal{C}) \cup \{s \in (\mathbb{B} \cap \mathcal{L})^{\uparrow\uparrow} \setminus \mathcal{A} \mid \neg\exists h \leftarrow B \in (\mathbb{B} \cap \mathcal{R}) \setminus \mathbb{B}^{-} \text{ with } h = s\} \cup \{h \leftarrow B \in \mathbb{B} \cap \mathcal{R} \mid B \cap \mathbb{B}^{-} \neq \varnothing\}$	Played blocked pieces
$\mathbb{P}^{*} = (\mathbb{P} \cap \mathcal{A}) \cup \{h \leftarrow B \in (\mathbb{P} \cap \mathcal{R}) \mid B \subseteq \mathbb{P}^{*}\} \cup \{s \in (\mathbb{P} \cap (\mathcal{L} \setminus \mathcal{A})) \mid \exists h \leftarrow B \in \mathbb{P}^{*} \text{ with } h = s\}$	"Complete" played pieces of the proponent
$\mathbb{B}^{*\setminus -} = (\mathbb{B} \cap (\mathcal{A} \setminus \mathcal{C})) \cup \{h \leftarrow B \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap \mathcal{R} \mid B \subseteq \mathbb{B}^{*\setminus -}\} \cup \{s \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap (\mathcal{L} \setminus \mathcal{A}) \mid \exists h \leftarrow B \in \mathbb{B}^{*\setminus -} \text{ with } h = s\}$	Unblocked complete played pieces of the opponent
$\mathbb{B}_{S}^{!\setminus -} = ((\mathbb{B} \setminus \mathbb{B}^{-}) \cap S) \cup \{s \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap \mathcal{L} \mid \exists h \leftarrow B \in \mathbb{B}_{S}^{!\setminus -} \cap \mathcal{R} \text{ with } s \in B\} \cup \{h \leftarrow B \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap \mathcal{R} \mid h \in \mathbb{B}_{S}^{!\setminus -}\}$	Unblocked pieces supporting statements in $S \subseteq \mathcal{L}$
$\mathcal{A}^{!} = \mathcal{A} \cap \mathbb{B}_{\overline{\mathcal{D}}}^{!\setminus -}$	Culprit candidates
$\mathcal{J} = \{u \in \mathcal{A} \setminus \mathcal{C} \mid \overline{u} \notin \mathbb{B}^{*\setminus -}\}$	Currently defended assumptions
$\mathcal{H} = \{h \mid h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}, B \subseteq \mathbb{P}^{*}\}$	Statements derivable from $\mathbb{P}^{*}$

Table 2.8: Auxiliary notations for the rule-based approach. All notions defined w.r.t. a dispute state $(\mathbb{B}, \mathbb{P})$.

Then, in RlFlexDDs, a proponent dispute state advancement from a dispute state $(\mathbb{B}, \mathbb{P})$ is a dispute state $(\mathbb{B}', \mathbb{P}')$ with $\mathbb{P}' = \mathbb{P} \cup T$, $\mathbb{B}' = \mathbb{B} \cup T$, $X_1 \subseteq \overline{\mathcal{A}}$, $X_1 \subseteq \mathcal{A}$, where T is defined as in Table 2.9.

Move type	V.	T definition
PB$-(\overline{\mathcal{A}^!} \cup X_1)$	1	$T = \{h \leftarrow B\} \cup B$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}$ with $h \in (\mathbb{P} \cap \mathcal{L})^{\downarrow}$
	2	$T = \{h\} \cup \{h \leftarrow B\} \cup B$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}$ with $h \in (\overline{\mathcal{A}^!} \cup X_1) \setminus (\mathbb{P} \cup \overline{\mathcal{D}})$
PF$-(\overline{(\mathcal{A}^! \cap \mathcal{A})} \cup X_2)$	1	$T = \{h\} \cup \{h \leftarrow B\}$ with $h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}$, with $h \notin \mathbb{P}$ or $h \in (\mathbb{P} \cap \mathcal{L})^{\downarrow}$, $B \subseteq \mathbb{P}^*$
	2	$T = \{u\}$ for $u \in ((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup X_2) \setminus (\mathbb{P} \cup \{\overline{u}\} \cup \mathcal{C} \cup \overline{\mathcal{D}})$

Table 2.9: Possible proponent moves in RlFlexDDs. Column "Move type" denotes the type of move and move's variable parameters, and "V." the variant. "T definition" defines the new set of pieces that the move adds to $\mathbb{B}$ and $\mathbb{P}$.

On the other hand, a rule-based version of the opponent dispute state advancement from a dispute state $(\mathbb{B}, \mathbb{P})$ is a dispute state $(\mathbb{B}', \mathbb{P})$ with $\mathbb{B}' = \mathbb{B} \cup T$, $Y_1 \subseteq \overline{\mathcal{A}}$, $Y_2 \subseteq \mathcal{A}$, where T is defined as in Table 2.10.

Move type	V.	T definition
OB$-(\overline{\mathcal{D}} \cup Y_1)$	1	$T = \{h \leftarrow B\} \cup B$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{\sim}$ with $h \in \mathbb{B}^{!\setminus -}_{\overline{\mathcal{D}} \cup Y_1} \cap \mathcal{L}$
	2	$T = \{h\} \cup \{h \leftarrow B\} \cup B$ for for $h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{\sim}$ with $h \in \overline{\mathcal{D}} \cup Y_1$
OF$-((\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2)$	1	$T = \{h\} \cup \{h \leftarrow B\}$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{\sim}$ with $\mathbb{B}^{*\setminus -}$ for each $b \in B$
	2	$T = \{u\}$ for $u \in ((\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2) \setminus (\mathcal{A} \cap \mathbb{B})$

Table 2.10: Possible opponent moves in RlFlexDDs. Column "Move type" denotes the type of move and move's variable parameters, and "V." the variant. "T definition" defines the new set of pieces that move adds to $\mathbb{B}$.

Advancement types and termination criteria remain the same as for the regular FlexDDs version, i.e. as presented in Table 2.4 and Table 2.5 with a small exception for the latter. Namely, due to the fact that RlFlexDDs operate on dispute states in a different form – sets of statements and rules rather than arguments – then the "Proponent winning" conditions have to be modified accordingly as shown in Table 2.11.

C	Prop. Winning
TA	$\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*$, $\overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -} = \varnothing$
TS	$\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*$, $\overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -} = \varnothing$, $\mathcal{D} \cup \mathcal{C} = \mathcal{A}$

Table 2.11: "Proponent winning" conditions in termination criteria for RlFlexDDs.

Table 2.12 presents a DS + TA RlFlexDD for the ABA framework from Example 1.1.1, corresponding to the FlexDD from Table 2.7. At Step 2. the statement $\boldsymbol{c}$ is added to $\mathbb{P}$. Because it is an assumption, it automatically becomes a member of $\mathbb{P}^*$ – the set of complete pieces played by the proponent. In the next steps the proponent keeps putting forward rules, whose bodies are contained in $\mathbb{P}^*$ and hence adding their heads to $\mathbb{P}^*$. At Step 6. rule $p \leftarrow \boldsymbol{a}, \boldsymbol{b}, q$ is uttered by the proponent. Note that at this point q is a "complete" statement and since $\boldsymbol{a}$ and $\boldsymbol{b}$ are assumptions then the goal p becomes "complete" as well. Therefore at Step 6. $\mathbb{P}^*$ contains the goal p as well as both culprit contraries $\overline{e}$ and $\overline{f}$, hence satisfying the first condition for "proponent winning" of TA in the rule-based setup, namely $\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*$.

Step and move type	$\mathbb{P}$	$\mathbb{B}\backslash\mathbb{P}$
0.	$\{p\}$	$\{\}$
1. (PF2, c)	$\{p;\ \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\}$
2. (PF1, $\overline{e} \leftarrow \boldsymbol{c}$)	$\{p;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\}$
3. (PF1, $r \leftarrow \boldsymbol{c}$)	$\{p;\ r^* \leftarrow \boldsymbol{c}^{\mathcal{D}*};\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\}$
4. (PF1, $\overline{f} \leftarrow \overline{e}$)	$\{p;\ r^* \leftarrow \boldsymbol{c}^{\mathcal{D}*};\ \overline{f}^* \leftarrow \overline{e}^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\}$
5. (PF1, $q \leftarrow \overline{e}, r$)	$\{p;\ q^* \leftarrow \overline{e}^*, r^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*};\ r^* \leftarrow \boldsymbol{c}^{\mathcal{D}*};\ \overline{f}^* \leftarrow \overline{e}^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\}$
6. (PB1, $p \leftarrow \boldsymbol{a}, \boldsymbol{b}, q$)	$\{p^* \leftarrow \boldsymbol{a}^{\mathcal{D}*}, \boldsymbol{b}^{\mathcal{D}*}, q^*;\ q^* \leftarrow \overline{e}^*, r^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}, r^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}; \overline{f}^* \leftarrow \overline{e}^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\}$
7. (OB2, $\overline{a} \leftarrow z$)	$\{p^* \leftarrow \boldsymbol{a}^{\mathcal{D}*}, \boldsymbol{b}^{\mathcal{D}*}, q^*;\ q^* \leftarrow \overline{e}^*, r^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}, r^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}; \overline{f}^* \leftarrow \overline{e}^*;\ \overline{e}^* \leftarrow \boldsymbol{c}^{\mathcal{D}*}\}$	$\{\overline{a}^- \leftarrow z^-\}$

Table 2.12: A DS + TA RlFlexDD for framework from Example 1.1.1. For brevity reasons each statement s being a part of a player's set (either $\mathbb{B}$ or $\mathbb{P}$) is represented by a rule present in this set containing s in either the head or the body. E.g. at Step 2., statements $\overline{e}$ and $\boldsymbol{c}$ even though they are a part of $\mathbb{P}$ are not explicitly listed and are supposed to be both represented by the rule $\overline{e} \leftarrow \boldsymbol{c}$. Since $\mathbb{P} \subseteq \mathbb{B}$ we only show the contents of $\mathbb{B}\backslash\mathbb{P}$ in the right-most column. Statements being a part of $\mathbb{P}^*$ are marked with $*$, those of $\mathbb{B}^-$ with $-$ and defences with $\mathcal{D}$.

In the last step of Table 2.12 the opponent puts forward the rule $\overline{a} \leftarrow z$ to attack the defence a. Note that the only rule with z in the head in $\mathcal{R}$ is $z \leftarrow \boldsymbol{f}$, which due to the fact that $\boldsymbol{f}$ became a culprit at Step 4. becomes an element of the set of blocked rules $\mathcal{J}^-_{\mathbb{B}}$. This in turn causes z to become a part of $(\mathbb{B} \cap \mathcal{L})^{\uparrow\uparrow}$ and in consequence also of $\mathbb{B}^-$. Also, $\overline{a}$ is a part of $(\mathbb{B} \cap \mathcal{L})^{\uparrow\uparrow}$, because all other rules with $\overline{a}$ in the head are blocked due to $\boldsymbol{e}$ becoming a culprit at Step 2. As a consequence $\overline{a}$ also becomes an element of $\mathbb{B}^-$.

Note that no defence contrary at Step 7. is in $\mathbb{B}^{*\backslash -}$, fulfilling condition $\overline{\mathcal{D}} \cap \mathbb{B}^{*\backslash -} = \varnothing$ of TA. Since there are no other means for the opponent to advance further, the dispute derivation terminates with the proponent as the winner.

2.4 Most Important Refinements Introduced By Flexible Dispute Derivations

In flexible dispute derivations arguments of both players are represented by a graph, rather than only those of the proponent as in graphical dispute derivations [7, 9]. Furthermore, in FlexDDs the opponent makes use of the full argument graph obtained with the advancement of both players (as opposed to the proponent who only has a sub-graph of arguments created solely by them at their disposal), enabling them to utilize arguments created by the proponent and releasing him from the obligation to again derive arguments for already established claims.

Rule-based flexible dispute derivations offer a less abstract alternative to FlexDDs, in which arguments are represented by rules and statements. While still enjoying the properties of FlexDDs, RlFlexDDs are easier to implement.

Forward reasoning is an even more significant feature of FlexDDs, unused in previous variants of dispute derivations. We have seen two versions of such reasoning: (1) the conservative forward moves – deriving further conclusions from justified statements – and (2) the non-conservative forward moves – allowing making use of assumptions not directly related with the current dispute state. This kind of reasoning combined with the traditional backward approach allows to introduce dispute derivations for semantics not considered in previous work, i.e. stable and complete, of which the latter one will be described in full in the following chapter.

3 A Modification in Flexible Dispute Derivations for the Complete Semantics

Flexible dispute derivations, as defined by Diller et al. [11], are capable of supporting the complete semantics by defining an advancement type DC and termination criterion TC, specified for both, FlexDDs and RlFlexDDs, in Table 3.1 and Table 3.2, respectively.

Proponent	Opponent
PB-$(\overline{\mathcal{A}^!})$, PF-$((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup \mathcal{J})$	OB-$(\overline{\mathcal{D}} \cup \overline{\mathcal{J}})$, OF-$((\overline{\mathcal{D}} \cup \overline{\mathcal{J}}) \cap \mathcal{A})$-2

Table 3.1: DC– advancement type for FlexDDs and the complete semantics.

Prop. Winning			
FlexDDs	RlFlexDDs	Opp. Cannot Move	Prop. Cannot Move
$\gamma \cup \overline{\mathcal{C}} \subseteq Conc(\mathcal{P}^*)$, $\mathcal{B}^{!\setminus^-}_{\overline{\mathcal{D}}} \cap \mathcal{B}^{*\setminus^-} = \varnothing$, $\mathcal{J} \subseteq \mathcal{D}$, $\mathcal{H} \subseteq Conc(\mathcal{P}^*)$	$\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*$, $\overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus^-} = \varnothing$, $\mathcal{J} \subseteq \mathcal{D}$, $\mathcal{H} \subseteq \mathbb{P}^*$	OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 or OF-$(\mathcal{A})$	PB-$(\overline{\mathcal{A}^!}) \cup$ PF-$((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup \mathcal{J})$ or PF-$(\mathcal{A})$

Table 3.2: Termination criteria TC for the complete semantics. Note that the proponent winning condition is specified for both versions, regular FlexDDs and RlFlexDDs, whereas the conditions regarding the ability of each player to move are common for both versions as for the other semantics. Conditions marked with blue colour in the "Prop. Winning" were not present in the original work on FlexDDs [11] and have been additionally specified to guarantee the soundness.

However, a small but important modification is present in the termination criteria TC presented in Table 3.2 relative to the original formalization in form of an additional condition $\mathcal{H} \subseteq Conc(\mathcal{P}^*)$ for FlexDDs (and $\mathcal{H} \subseteq \mathbb{P}^*$ for RlFlexDDs). By providing a counter-example in Table 3.3 for the framework from Example 3.0.1, we will show why a DC + TC FlexDD without this additional condition is not guaranteed to be sound for the complete semantics.

Example 3.0.1 (ABA framework)**.** Consider an ABA framework $F' = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, where $\mathcal{A} = \{a, b, c, d\}$, $\mathcal{R} = \{\overline{c} \leftarrow d, \overline{b} \leftarrow c\}$, $^-(u) = \overline{u}$ for $u \in \mathcal{A}$ and $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$.

St. & m. type	$\mathcal{P}$	$\mathcal{B} \setminus \mathcal{P}$	$\mathcal{J} \setminus \mathcal{D}$
0.	$\{\boldsymbol{a}\}$	$\{\}$	$\{\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}\}$
1. (OB2, $\overline{b} \leftarrow \boldsymbol{c}$)	$\{\boldsymbol{a}\}$	$\{\overline{b} \leftarrow \boldsymbol{c}\}$	$\{\boldsymbol{c}, \boldsymbol{d}\}$
2. (OB2, $\overline{c} \leftarrow \boldsymbol{d}$)	$\{\boldsymbol{a}\}$	$\{\overline{b} \leftarrow \boldsymbol{c}; \overline{c} \leftarrow \boldsymbol{d}\}$	$\{\boldsymbol{d}\}$
3. (PF2, $\boldsymbol{d}$)	$\{\boldsymbol{a}^{\mathcal{D}}, \boldsymbol{d}^{\mathcal{D}}\}$	$\{\overline{b} \leftarrow \boldsymbol{c}; \overline{c} \leftarrow \boldsymbol{d}^{\mathcal{D}}\}$	$\{\}$
4. (PF1, $\overline{c} \leftarrow \boldsymbol{d}$)	$\{\boldsymbol{a}, \overline{c} \leftarrow \boldsymbol{d}^{\mathcal{D}}\}$	$\{\overline{b} \leftarrow \boldsymbol{c}^{\mathcal{C}}\}$	$\{\boldsymbol{b}\}$
5. (PF2, $\boldsymbol{b}$)	$\{\boldsymbol{a}, \overline{c} \leftarrow \boldsymbol{d}^{\mathcal{D}}; \boldsymbol{b}^{\mathcal{D}}\}$	$\{\overline{b} \leftarrow \boldsymbol{c}^{\mathcal{C}}\}$	$\{\}$

Table 3.3: Counterexample of a DC + TC FlexDD for goals $\gamma = \{\boldsymbol{a}\}$ and the ABA framework from Example 3.0.1. Contrary to previous examples of FlexDDs, this time a new column $\mathcal{J} \setminus \mathcal{D}$ is introduced in order to keep track of the assumptions defended by the proponent at a dispute state which are not defences.

Assuming the additional condition is not present in TC, the dispute derivation would have terminated at Step 3. with a positive answer, claiming that the set of assumptions $\{\boldsymbol{a}, \boldsymbol{d}\}$ is a complete extension. However, as seen in Figure 3.1 this is not true: there exists an argument A s.t. $Prem(A) \subseteq \{\boldsymbol{a}, \boldsymbol{d}\}$ which defends $\boldsymbol{b}$, namely $A = \overline{c} \leftarrow \boldsymbol{d}$. Once this argument gets constructed by the proponent (Step 4.) the opponent's argument $\overline{b} \leftarrow \boldsymbol{c}$ gets defeated due to $\boldsymbol{c}$ becoming a culprit. At this point $\boldsymbol{b}$ re-appears in $\mathcal{J} \setminus \mathcal{D}$, and the proponent has to claim it in order to make the extension complete. This shows, that the notion $\mathcal{J}$ of "currently defended assumptions by the proponent" is limited only to those assumptions defended by arguments constructed by the proponent, rather than the defences.

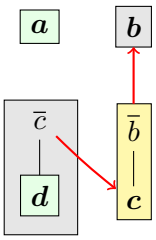


Figure 3.1: Argument representation of the dispute state at Step 3. from Table 3.3.

In the remainder of this chapter, we will prove that the additional adjustment to TC suffices to guarantee the soundness of DC + TC FlexDDs for the complete semantics. Following the original proofs [11] we will do it indirectly, by showing the soundness of DF + TC FlexDDs, from which also follows the soundness for the more restricted advancement types. Moreover, we will only prove it for the FlexDDs version, relying on the correspondence that exists between FlexDDs and RlFlexDDs (for this see [11]) that the same holds for RlFlexDDs.

Theorem 3.0.1. DF + TC FlexDDs are sound for the complete semantics, meaning that given a set of goals γ if there is a DF + TC FlexDD ending with a dispute state $(\mathcal{B}, \mathcal{P})$ and the proponent as winner, then $\mathcal{D}$ is a complete extension w.r.t. which γ is acceptable.

Proof. Let $(\mathcal{B}, \mathcal{P})$ be a flexible dispute derivation state obtained by following the rules of DF advancement type and satisfying TC for a set of conflict-free goals γ. By showing that $\mathcal{D}$ (and conse-

quently $\mathcal{P}$) constitute an admissible extension, we will first prove soundness of DF + TC FlexDDs for the admissible semantics before turning to their complete counterpart. Since soundness for the admissible semantics has already been proven by Diller et al. [11], this part of the proof will have the form of a sketch.

(Admissible semantics.) First of all, at step $(\mathcal{B}, \mathcal{P})$ the goals γ must have been justified by construction of complete arguments for each of them as indicated by the satisfied TC condition $\gamma \cup \overline{\mathcal{C}} \subseteq \mathcal{P}^*$. As defined previously, an admissible extension is a conflict-free set of assumptions, that defends itself from all its attackers. By construction (definition of proponent's moves) $\mathcal{D}$ must always remain conflict-free as required by the proponent's moves restrictions to not use any rules nor assumptions containing culprits or attacking $\mathcal{D}$. Now what remains to be shown is that all attackers of $\mathcal{D}$ have been counter-attacked. By contradiction assume it is not the case. Then either (1) there is some complete argument $A' \in \mathcal{B}^{!\,\setminus^-}_{\overline{\mathcal{D}}}$ attacking a defence $u \in \mathcal{D}$ which is not counter-attacked by a proponent's complete argument or (2) such argument can still be constructed or completed. In case (1) we already encounter a contradiction as TC condition $\mathcal{B}^{!\,\setminus^-}_{\overline{\mathcal{D}}} \cap \mathcal{B}^{*\,\setminus^-} = \varnothing$ results in $A' \notin \mathcal{B}^{*\,\setminus^-}$. Then either (i) $Asm(A') \cap \mathcal{C} \neq \varnothing$ or (ii) A' is not a complete argument, which is an immediate contradiction of our previous assumption. Case (i) on the other hand indicates that A' is indeed counter-attacked by some proponent's argument A s.t. $Conc(A) \in (\mathcal{C} \cap Prem(A'))$. Again, the $\gamma \cup \overline{\mathcal{C}} \subseteq \mathcal{P}^*$ condition ensures that A is complete yielding contradiction in this case as well. In case (2) A' (i) has not yet been constructed by the opponent or (ii) is not yet complete. If (i) is the case the construction of A' can be commenced using OB-$(\overline{\mathcal{D}})$-2 (then $A' = h \leftarrow B$, given a rule $h \leftarrow B \in \mathcal{R}$ s.t. $h = \overline{u}$) or even completed within one step with OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 (then $A = \overline{u}$, given there is an assumption $\overline{u} \in \mathcal{A}$). In (ii) case A' is incomplete and can be backward-extended using OB-$(\overline{\mathcal{D}})$-1 creating A'' (then $A'' = A \lessdot h \leftarrow B$, given a rule $h \leftarrow B \in \mathcal{R}$ s.t. $h \in Prem(A')$). However, neither of those moves is applicable as required by the TC condition stating that the opponent cannot move using neither OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 nor OF-$(\mathcal{A})$. Note that only one of those two disjuncts is enough to satisfy the condition, as the second one can be seen as a stronger condition than the first one in that every complete argument can be constructed by it (opponent can use any assumption and then propagate forward using them as premises for new arguments effectively enabling opponent to yield any complete argument constructible). Since only complete arguments attacking defences are of interest and the second condition being satisfied means that no such argument can be constructed we have a contradiction in either case.

(Complete semantics.) Now, knowing that $\mathcal{D}$ is an admissible extension, let us assume that it is not a complete one. From the definition of complete semantics it follows that in such a case there must exist an assumption u s.t. $u \notin \mathcal{D}$ and $\mathcal{D}$ defends u. The latter means that if there is a set of arguments $Args'$ s.t. $Conc(Args') = \overline{u}$ and a set of assumptions $U' \subseteq Asm(Args')$ s.t. $\forall A' \in Args'\ \exists u' \in Asm(A')$ s.t. $u' \in U'$ then there also exists a set of conflict-free arguments $Args = \{A \mid Prem(A) \subseteq \mathcal{D}\}$ s.t. $\overline{U'} \subseteq Conc(Args)$. Assuming no such set $Args'$ exists (meaning that u is not attacked) we have that $u \in \mathcal{J}$ and by the $\mathcal{J} \subseteq \mathcal{D}$ condition of TC it must consequently hold that $u \in \mathcal{D}$ leading to a contradiction with our assumption that $u \notin \mathcal{D}$. Therefore $Args'$ must exist. Now let us assume that $Args \subseteq \mathcal{P}^*$, and therefore $Conc(Args) \subseteq Conc(\mathcal{P}^*)$. In such case $Args'$ would be blocked, i.e. $Args' \cap \mathcal{B}^{*\,\setminus^-} = \varnothing$ and therefore $\overline{u} \notin Conc(\mathcal{B}^{*\,\setminus^-})$ again resulting the contradictory conclusion that $u \in \mathcal{J}$. Therefore $Conc(Args) \nsubseteq Conc(\mathcal{P}^*)$

and consequently $Args \notin \mathcal{P}^*$. Note that since $\mathcal{D} \subseteq \mathcal{P}^*$ and $\mathcal{P}^*$ and $Args$ are conflict-free (by admissibility and assumption respectively), then it holds that $Args \subseteq (\mathcal{P}^* \cup \{h \leftarrow B \ltimes A \mid h \leftarrow B \in \mathcal{R} \setminus \mathcal{R}^{\sim}, Args \in \mathcal{P}^*\})$. Then $Conc(Args) \subseteq \mathcal{H}$. From the additional TC condition we know that $\mathcal{H} \subseteq Conc(\mathcal{P}^*)$. By transitivity we then have that $Conc(Args) \subseteq Conc(\mathcal{P}^*)$ which again is a contradiction. □

4 Automatic Search

4.1 Move-Type-Preference Based Strategies

As shown previously, flexible dispute derivations are defined in terms of several types of moves for each player. At each dispute state a number of moves of possibly different types are applicable. Therefore, in the pursuit of meaningful ways to carry out the search for a successful dispute derivation, it seems only reasonable to define preference orderings for types of move and, additionally, specify what should happen when more than one move of a certain type can be executed.

In the previous chapters move types have been defined parametrically containing variables according to the chosen advancement types. E.g. for the DAB advancement type we have defined the move PF-$(\overline{\mathcal{A}^!} \cap \mathcal{A})$-2, whereas the corresponding move type in DS is PF-$(\mathcal{A})$-2. In what follows, whenever the advancement type (and therefore the parameter value) is of no interest we will denote the moves simply without parameters, e.g. PF2. If both moves of some kind are considered, the number will simply be dropped, i.e. PF would stand for both PF1 and PF2. We will also use symbols $\langle\!\langle\, \rangle\!\rangle$ to denote sequences, i.e. collections in which the order of elements appearing in them is of significance.

The methods described in this chapter serve as theoretical underpinning of the system supporting FlexDDs we developed, hence the presented definitions will be relative to the more implementation-focused, rule-based variant of flexible dispute derivations.

Definition 4.1.1 (Potential move)**.** Given an ABA framework $(\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ a potential move m is defined as a pair $m = (type, piece)$ where:

$$piece = \begin{cases} h \leftarrow B \in \mathcal{R} & \text{if } type \in \{\texttt{PB}, \texttt{PF1}, \texttt{OB}\} \\ a \in \mathcal{A} & \text{if } type \in \{\texttt{PF2}, \texttt{OF2}\} \end{cases}$$

A potential move, as defined in Definition 4.1.1, contains the type of the move and a related framework piece, i.e. a rule for moves of types utilizing rules (rule moves) or an assumption for moves exploiting uttering of a single assumption (assumption moves).

Definition 4.1.2 (Preference ordering of move types)**.** A preference ordering of move types $M = \{\texttt{PB1}, \texttt{PB2}, \texttt{PF1}, \texttt{PF2}, \texttt{OB1}, \texttt{OB2}, \texttt{OF2}\}$ is a strict total order $<_M$ on the move types. For two move types $t, t' \in M$ we say that t is preferred to t' if $t <_M t'$. Given a non-empty set of possible moves $O = \{(t_1, p_1), \ldots, (t_n, p_n)\}$ we define as $<_M (O) = \{p \mid (min_{<_M}\{t_1, \ldots, t_n\}, p) \in O\}$ the set of moves of type chosen from O by $<_M$.

Definition 4.1.2 explains a rather intuitive notion of preference ordering, i.e. given an ordering $<_M$ and a set of possible moves O, an ordering will select the most preferred move type according to $<_M$ and pieces from possible moves from O of this type.

Definition 4.1.3 (Rule head choice)**.** Given a set of rules $R \subseteq \mathcal{R}$, we denote by $R_{h'}$ the subset of R consisting only of rules with head h', formally $R_{h'} = \{h \leftarrow B \in R \mid h = h'\}$. Furthermore, we define two families of rule head choice functions $2^{\mathcal{R}} \mapsto 2^{\mathcal{R}}$, namely `MostRules` and `LeastRules`. For every function $\texttt{mostRules} \in \texttt{MostRules}$ it holds for every $R \subseteq \mathcal{R}$ that $\texttt{mostRules}(R) \in \{R_{h'} \mid \neg\exists h''\ with\ |R_{h''}| > |R_{h'}|\}$. Similarly, for every function $\texttt{leastRules} \in \texttt{LeastRules}$ it holds for every $R \subseteq \mathcal{R}$ that $\texttt{leastRules}(R) \in \{R_{h'} \mid \neg\exists h''\ with\ |R_{h'}| > |R_{h''}|\}$.

When applied to a set of rules, a rule choice function r from Definition 4.1.3 will simply return a subset of rules having the same head. The returned subset will be of greatest cardinality among such subsets if $r \in \texttt{MostRules}$, otherwise of least cardinality (if $r \in \texttt{LeastRules}$).

Definition 4.1.4 (State distance)**.** Let us denote by $\mathcal{S}$ the set of all possible dispute states. Given a dispute state $(\mathbb{B}, \mathbb{P})$, set of goals γ and a termination criterion $c \in \{\texttt{TA}, \texttt{TC}, \texttt{TS}\}$ `stateDistance` is a function $\{\texttt{TA}, \texttt{TC}, \texttt{TS}\} \times 2^{\mathcal{L}} \times \mathcal{S} \mapsto \mathbb{N}$ defined as follows:

$$\texttt{stateDistance}(c, \gamma, (\mathbb{B}, \mathbb{P})) = \begin{cases} |\overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -}| + |(\gamma \cup \overline{\mathcal{C}}) \setminus \mathbb{P}^*| & \text{if } c = \texttt{TA} \\ |\overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -}| + |(\gamma \cup \overline{\mathcal{C}}) \setminus \mathbb{P}^*| + |\mathcal{J} \setminus \mathcal{D}| & \text{if } c = \texttt{TC} \\ |\overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -}| + |(\gamma \cup \overline{\mathcal{C}}) \setminus \mathbb{P}^*| + |\mathcal{A} \setminus (\mathcal{D} \cup \mathcal{C})| & \text{if } c = \texttt{TS} \end{cases}$$

Informally, the function `stateDistance` from Definition 4.1.4 measures how close the current dispute state is to one satisfying the given termination criterion.

Definition 4.1.5 (The set of statements in the dispute state after application of a rule)**.** Given a dispute state $(\mathbb{B}, \mathbb{P})$, a rule $h \leftarrow B$ and a player $p \in \{\texttt{proponent}, \texttt{opponent}\}$ the set of statements in the dispute state after application of a rule is defined as:

$$(\mathbb{B}, \mathbb{P})_{(h \leftarrow B, p)} = \begin{cases} \big((\mathbb{B} \cap \mathcal{L}) \cup \{h\} \cup B,\ \mathbb{P} \cap \mathcal{L}\big) & \text{if } p = \texttt{opponent} \\ \big((\mathbb{B} \cap \mathcal{L}) \cup \{h\} \cup B,\ (\mathbb{P} \cap \mathcal{L}) \cup \{h\} \cup B\big) & \text{if } p = \texttt{proponent} \end{cases}$$

Definition 4.1.6 (Rule comparison)**.** Rule comparison functions (at a dispute state) are functions of the form $\mathcal{R} \times \mathcal{S} \times 2^{\mathcal{L}} \times \{\texttt{proponent}, \texttt{opponent}\} \times \{\texttt{TA}, \texttt{TC}, \texttt{TS}\} \mapsto \mathbb{N}$. We define the following rule comparison functions: `bodySize`, `newStatementsSize` and `lookaheadSize`. Given a rule $r = h \leftarrow B$, a dispute state $(\mathbb{B}, \mathbb{P})$, a set of goals γ, a player $p \in \{\texttt{proponent}, \texttt{opponent}\}$ and a termination criterion $c \in \{\texttt{TA}, \texttt{TC}, \texttt{TS}\}$, we have:

- $\texttt{bodySize}(r, (\mathbb{B}, \mathbb{P}), \gamma, p, c) = |B|$
- $\texttt{lookaheadSize}(r, (\mathbb{B}, \mathbb{P}), \gamma, p, c) = \texttt{stateDistance}(c, \gamma, (\mathbb{B}, \mathbb{P})_{(r,p)})$
- $\texttt{newStatementsSize}(h \leftarrow B, (\mathbb{B}, \mathbb{P}), \gamma, p, c) =$

$$|(\{h\} \cup B) \setminus \mathbb{X}| \text{ where } \mathbb{X} = \begin{cases} \mathbb{B} & \text{if } p = \texttt{opponent} \\ \mathbb{P} & \text{if } p = \texttt{proponent} \end{cases}$$

Definition 4.1.7 (Rule sorting)**.** Let us again denote by $\mathcal{S}$ the set of all possible dispute states. Given a rule comparison function $dist$ and $asc \in \{\top, \bot\}$ sorting functions w.r.t. $dist$ and asc

are functions $sort_{dist,asc}$: $2^{\mathcal{R}} \times 2^{\mathcal{L}} \times S \times \{\texttt{proponent}, \texttt{opponent}\} \times \{\texttt{TA}, \texttt{TC}, \texttt{TS}\}$ satisfying for $R \subseteq \mathcal{R}$, a dispute state $(\mathbb{B}, \mathbb{P})$, a player $p \in \{\texttt{proponent}, \texttt{opponent}\}$ and a termination criteria $c \in \{\texttt{TA}, \texttt{TC}, \texttt{TS}\}$:

i) $sort_{dist,asc}(R, \gamma, (\mathbb{B}, \mathbb{P}), p, c) = \langle\!\langle r_1, \ldots, r_n \rangle\!\rangle$ with $r_i \in R$ iff $r_i \in \langle\!\langle r_1, \ldots, r_n \rangle\!\rangle$ for each $1 \leq i \leq n$,

ii) if $sort_{dist,\top}(R, \gamma, (\mathbb{B}, \mathbb{P}), p, c) = \langle\!\langle r_1, \ldots, r_n \rangle\!\rangle$ then $dist(r_i, \gamma, (\mathbb{B}, \mathbb{P}), p, c) \leq dist(r_j, \gamma, (\mathbb{B}, \mathbb{P}), p, c)$ whenever $i \leq j$ for each $1 \leq i, j \leq n$,

iii) if $sort_{dist,\bot}(R, \gamma, (\mathbb{B}, \mathbb{P}), p, c) = \langle\!\langle r_1, \ldots, r_n \rangle\!\rangle$ then $dist(r_i, \gamma, (\mathbb{B}, \mathbb{P}), p, c) \geq dist(r_j, \gamma, (\mathbb{B}, \mathbb{P}), p, c)$ whenever $i \leq j$ for each $1 \leq i, j \leq n$,

Functions defined in Definition 4.1.6 provide ways of ranking rules by assigning natural numbers to them. Then, sorting functions from Definition 4.1.7 enables to sort the rules in either ascending or descending order by their assigned number. This allows us to formally define the order in which rules should be considered in the automatic search for dispute derivations. E.g. a sorting function $sort_{\texttt{bodySize},\top}$ would return a sequence of rules sorted in ascending order by the body size.

Definition 4.1.8 (Strategy)**.** A strategy $Strat$ is a tuple $(<_M, H, R, A)$, where $<_M$ is a preference ordering of move types, $H \in \{\texttt{mostRules}, \texttt{leastRules}\}$ is a head choice function, $R \in \{\texttt{bodySize}, \texttt{newStatementsSize}, \texttt{lookaheadSize}\}$ is a rule comparison function and $A \in \{\top, \bot\}$. Given a set of possible moves $O = \{(t_1, m_1), \ldots (t_n, m_n)\}$ at a dispute state $(\mathbb{B}, \mathbb{P})$ and a termination criterion $c \in \{\texttt{TA}, \texttt{TC}, \texttt{TS}\}$, moves chosen by the strategy are defined as:

$$Strat(O) = \begin{cases} (min_{<_M}\{t_1, \ldots, t_n\}, <_M (O)) & \text{if } min_{<_M}\{t_1, \ldots, t_n\} \in \{\texttt{PF2}, \texttt{OF2}\} \\ (min_{<_M}\{t_1, \ldots, t_n\}, sort_{R,A}(H(<_M (O)), (\mathbb{B}, \mathbb{P}), \texttt{proponent}, c)) & \\ & \text{if } min_{<_M}\{t_1, \ldots, t_n\} \in \{\texttt{PB1}, \texttt{PB2}, \texttt{PF1}\} \\ (min_{<_M}\{t_1, \ldots, t_n\}, sort_{R,A}(H(<_M (O)), (\mathbb{B}, \mathbb{P}), \texttt{opponent}, c)) & \\ & \text{if } min_{<_M}\{t_1, \ldots, t_n\} \in \{\texttt{OB1}, \texttt{OB2}\} \end{cases}$$

As defined in Definition 4.1.8, assuming a set of possible moves, a strategy returns a pair, of which the first element is the type of the chosen moves and the second is the sequence of actual moves of this type, filtered and ordered according to the chosen strategy parameters. Note that because we focused on moves using rules, then if the move type chosen by the strategy is an assumption move (moves PF2 or OF2) all moves of this type are simply returned.

Naturally, the H, R, and A elements of $Strat$ could be defined individually for both players. In such case each player could examine rule moves in their own way. However, since that approach would significantly increase the number of possible definable strategies, we decided to force both players to proceed in the same manner[1].

[1] However, our implementation described in Chapter 5 allows to specify the parameters H, R, and A independently for each player.

4.2 Search Algorithms

We now turn to defining search algorithms for finding successful dispute derivations. Towards these, we first need to extend the notion of dispute derivation state to obtain a data structure aiding in optimizing the search or enabling to support grounded and preferred semantics.

Definition 4.2.1 (Extended dispute derivation state)**.** Given an ABA framework $(\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ and a (rule-based) dispute derivation state $(\mathbb{B}, \mathbb{P})$ an extended dispute derivation state is defined as a tuple $(\mathbb{B}, \mathbb{P}, I_a, I_c, Ctr)$ where $I_a, I_c, Ctr \subseteq \mathcal{A}$ we call the *ignored proponent's assumptions*, *ignored culprit candidates* and *constraints*, respectively.

The intuitive meaning of the additional elements in extended dispute derivation states as in Definition 4.2.1 is as follows:

- I_a must not become defences, i.e. the proponent cannot put forward these assumptions nor any rule containing these assumptions in its body during a dispute derivation,
- I_c must not become culprits, i.e. if $u \in I_c$, then the proponent cannot let $\overline{u}$ become a part of $\mathbb{P}$, neither by putting forward the statement $\overline{u}$ if $\overline{u}$ is an assumption, nor by putting forward a rule having $\overline{u}$ in its head or body.
- Ctr again, must not become defences. Although in this sense these assumptions satisfy the same constraint as those in I_a, they come to have this role for a different reason in the algorithms we put forward in what follows, thus the need to distinguish between them.

Definition 4.2.2 introduces the *filter* function, which removes possible moves violating the above conditions. We will explain how filtering of such moves impact the search in what comes after.

Definition 4.2.2 (Filtering moves)**.** Given a set of possible moves $O = \{(t_1, m_1), \ldots (t_n, m_n)\}$, and an extended dispute state $S = (\mathbb{B}, \mathbb{P}, I_a, I_c, Ctr)$ we define the following auxiliary functions:

- $filterP_{\mathcal{R}}(O, S) = \Big\{(t, h \leftarrow B) \in O) \;\Big|\; h \leftarrow B \in \mathcal{R}, B \cap (Ctr \cup I_a) = \varnothing, \{u \in \mathcal{A} \mid \overline{u} \cap (\{h\} \cup B) \neq \varnothing\} \cap I_c = \varnothing, t \in \{\texttt{PB}, \texttt{PF1}\}\Big\}$
- $filterP_{\mathcal{A}}(O, S) = \Big\{(t, a) \in O \;\Big|\; a \in \mathcal{A}, a \notin (Ctr \cup I_a), \{u \in \mathcal{A} \mid \overline{u} = a\} \cap I_c = \varnothing, t = \texttt{PF2}\Big\}$
- $filterO(O, S) = \Big\{(t, m) \in O \;\Big|\; t \in \{\texttt{OB}, \texttt{OF}\}\Big\}$

Then, we define the function, which filters possible moves to be performed as:

$$filter(O, S) = filterP_{\mathcal{R}}(O, S) \cup filterP_{\mathcal{A}}(O, S) \cup filterO(O, S)$$

I_a are those assumptions, which during a dispute derivation could be added to $\mathbb{P}$ using PF2 moves. In such cases our non-deterministic search algorithm branches and creates two dispute states: one which adds the assumption to $\mathbb{P}$ and the other which adds it to I_a, blocking it also for the future dispute states of this branch. The motivation behind such behaviour is that if an

assumption has not been added to the defence on the first occasion, then it seems reasonable to not consider adding it in the later stages of the same branch of the search.

Similarly, I_c are those culprit candidates, which during a dispute derivation could have been attacked by the proponent (using moves PB2 or PF2), who decided not to do so. When encountering a culprit candidate, automatic search creates new branches for state(s) attacking said assumption using all possible means as well as a state ignoring (not-attacking) it. On the other hand, Ctr is used to support the grounded and preferred argumentation semantics and will be regarded later. Algorithm 1 formally defines the behaviour of the search algorithm imposed by I_a and I_c (cases PF2 and PB2). For PB1 moves states for all possible rules are created, preserving the order imposed by the strategy. For PF1 moves as well as for all opponent moves a state related to the first move in the ordered moves sequence is created, since there is no non-determinism involved.

Function GetNextStates from Algorithm 1 has the current dispute state, the chosen strategy and a set of moves which are possible to be performed at the current dispute state as input data and returns the sequence of dispute states to be examined next. In line 2 the strategy is applied to the set of possible moves, returning the type of the chosen moves as well as the sequence of the moves of that type chosen by the strategy. Furthermore, in a **switch** statement the new states are prepared appropriately to the move type. Note that whenever choices are deterministic (as in cases PF1, OB1, OB2 or OF2) a singleton sequence is simply returned with the next dispute state. In other cases a sequence of next states representing all non-deterministic choices the algorithm has is returned.

In case PB1, in which the proponent has to backward expand a statement, the algorithm returns dispute states for each of the rules in the order specified by the strategy. Case PB2 assumes attacking a culprit candidate (or culprit candidates, if heads of chosen rules attack more than one assumption) by the proponent. However, the proponent can also choose not to attack. Therefore, $attackingStates$ contains the sequence of "attacking" dispute states for each applicable rule, where the the order of rules enforced by the strategy is taken into account. Additionally, an $ignoringState$ is created, in which the proponent decides to give up on attacking the culprit candidate by adding it to I_c.

In case PF2 the proponent can put forward an assumption. If the chosen advancement type is DAB or DABF, then the assumption is a contrary of some culprit candidate. However, if the advancement type is beyond the aforementioned ones, then the assumption can be "unrelated" to the the current dispute (a non-conservative move). Therefore, a dispute state which accepts adding the assumption to the players' sets is created ($takingState$) as well as another one, which ignores it ($ignoringState$). In the latter case, there is a distinction: if the assumption attacks some culprit candidates, then these culprit candidates are added to I_c. This represents the similar case as with PB2, where the proponent could decide not to attack particular culprit candidates. However, if the assumption is not attacking any culprit candidates (non-conservative move), the proponent can simply choose to not put it forward, namely add it to I_a.

Algorithm 1: `GetNextStates` function.

input : $S = (\mathbb{B}, \mathbb{P}, I_a, I_c, Ctr)$ /* current dispute state */,
$Strat = (<_M, H, R, A)$ /* strategy */,
$O = \{(t_1, m_1), \ldots, (t_n, m_n)\}$ /* a set of possible moves at the current state S w.r.t. to the chosen advancement type and termination criteria */

output: $newStates = \langle\!\langle (\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_l, \mathbb{P}_l, I_{a_l}, I_{c_l}, Ctr_l) \rangle\!\rangle$ /* chosen sequence states to be examined next */

---

function `GetNextStates`(S, *Strat*, O)
- $(chosenMoveType,\ chosenMovesSequence) \longleftarrow Strat(O)$ /* get chosen move type and moves sequence using strategy *Strat* */
- $newStates \longleftarrow \langle\!\langle \rangle\!\rangle$ /* initialize *newStates* */
- **switch** *chosenMoveType* **do**
  - **case** `PB1` **do** /* *chosenMovesSequence* is a sequence of rules */
    - $newStates \longleftarrow \langle\!\langle (\mathbb{B} \cup (\{h\} \cup B_1), \mathbb{P} \cup (\{h\} \cup B_1), I_a, I_c, Ctr), \ldots, (\mathbb{B} \cup (\{h\} \cup B_n), \mathbb{P} \cup (\{h\} \cup B_n), I_a, I_c, Ctr) \rangle\!\rangle$ for $h \leftarrow B_i \in chosenMovesSequence$ /* create a sequence of new dispute states using rules in *chosenMovesSequence*, preserve the order */
  - **case** `PB2` **do** /* *chosenMovesSequence* is a sequence of rules */
    - $attackingStates \longleftarrow \langle\!\langle (\mathbb{B} \cup (\{h\} \cup B_1), \mathbb{P} \cup (\{h\} \cup B_1), I_a, I_c, Ctr), \ldots, (\mathbb{B} \cup (\{h\} \cup B_n), \mathbb{P} \cup (\{h\} \cup B_n), I_a, I_c, Ctr) \rangle\!\rangle$ for $h \leftarrow B_i \in chosenMovesSequence$ /* create a sequence of new dispute states in which prop. attacks some culprit candidate using rules in *chosenMovesSequence*, preserve the order */
    - $attackedAssumptions \longleftarrow \{a \in \mathcal{A} \mid \overline{a} \leftarrow B \in chosenMovesSequence\}$ /* get the set of assumptions attacked by head of rules from *chosenMovesSequence* */
    - $ignoringState \longleftarrow (\mathbb{B}, \mathbb{P}, I_a, I_c \cup attackedAssumptions, Ctr)$ /* get new state in which the attacked assumptions get ignored */
    - $newStates \longleftarrow attackingStates.append(ignoringState)$ /* append *ignoringState* to *attackingStates* */
  - **case** `PF1` **do** /* *chosenMovesSequence* is a sequence of rules */
    - $h \leftarrow B \longleftarrow chosenMovesSequence[0]$ /* take first rule, does not matter which */
    - $newStates \longleftarrow \langle\!\langle (\mathbb{B} \cup (\{h\} \cup B), \mathbb{P} \cup (\{h\} \cup B), I_a, I_c, Ctr) \rangle\!\rangle$ /* get new state by adding rule to $\mathbb{B}$ and $\mathbb{P}$ */
  - **case** `PF2` **do** /* *chosenMovesSequence* is a sequence of assumptions */
    - $b \longleftarrow chosenMovesSequence[0]$ /* take first assumption b */
    - $takingState \longleftarrow (\mathbb{B} \cup \{a\}, \mathbb{P} \cup \{a\}, I_a, I_c, Ctr)$ /* get new state by adding b to $\mathbb{B}$ and $\mathbb{P}$ */
    - $attackedAssumptions \longleftarrow \{a \in \mathcal{A} \mid \overline{a} = b\}$ /* get set of assumptions attacked by b */
    - $ignoringState \longleftarrow Nil$ /* initialize *ignoringState* */
    - **if** $attackedAssumptions \neq \varnothing$ **then** /* if b attacks some assumptions */
      - $ignoringState \longleftarrow (\mathbb{B}, \mathbb{P}, I_a, I_c \cup attackedAssumptions, Ctr)$ /* get new *ignoringState* by adding attacked assumptions to ignored culprit candidates I_c */
    - **else** /* otherwise, get new *ignoringState* by adding b to ignored assumptions I_a */
      - $ignoringState \longleftarrow (\mathbb{B}, \mathbb{P}, I_a \cup \{b\}, I_c, Ctr)$
    - $newStates \longleftarrow \langle\!\langle takingState, ignoringState \rangle\!\rangle$ /* return both states */
  - **case** `OB1` or `OB2` or `OF1` **do** /* *chosenMovesSequence* is a sequence of rules */
    - $h \leftarrow B \longleftarrow chosenMovesSequence[0]$ /* take first rule */
    - $newStates \longleftarrow \langle\!\langle (\mathbb{B} \cup (\{h\} \cup B), \mathbb{P}, I_a, I_c, Ctr) \rangle\!\rangle$ /* get new state by adding rule to $\mathbb{B}$ */
  - **case** `OF2` **do** /* *chosenMovesSequence* is a sequence of assumptions */
    - $a \longleftarrow chosenMovesSequence[0]$ /* take first assumption a */
    - $newStates \longleftarrow \langle\!\langle (\mathbb{B} \cup \{a\}, \mathbb{P}, I_a, I_c, Ctr) \rangle\!\rangle$ /* get new state by adding a to $\mathbb{B}$ */
- **return** *newStates*

The recursive function `GetSuccessfulDDs` from Algorithm 2 is the main automatic-search procedure, responsible for constructing and operating on the dispute derivation search tree. It starts with checking the guard condition – if no more dispute states are remaining – and returns the successful dispute states found in the process if that is the case. Otherwise, it removes the first dispute state from the collection $States$ and generates the set of possible moves at this state according to the advancement type A. This is represented by the function $getPossibleMoves$, which is not explicitly listed here, but returns the moves applicable for dispute state advancement according to Table 2.9 and Table 2.10, as well as advancement types defined in Table 2.4 or Table 3.1 from previous chapters. The moves are further filtered using the function from Definition 4.2.2. In line 9 another implicitly defined function is used, namely $isOver$, which checks whether the termination criterion C is satisfied as defined in Table 2.11 or Table 3.2, using the current state $currState$ to verify if the "proponent winning" condition is fulfilled and $possibleMoves$ to check each player's advancement possibilities. This function can return one of the following outcomes: the proponent has won, the opponent has won, or the dispute derivation is still ongoing. In the first case the obtained dispute state is appended to the set of found successful states. In the latter case, the dispute state is simply abandoned. Finally, if the dispute state has not finished yet, further advancement alternatives are obtained using the function `GetNextStates` defined in Algorithm 1. The obtained sequence of the next possible search steps is then either appended at the beginning or the end of the remaining dispute state collection, depending on the search type St. In case of depth-first search (DFS), the first option applies, and the collection of states $states'$ can be seen as a stack. In the latter breadth-first search (BFS) case, $states'$ resembles a queue.

Algorithm 2: `GetSuccessfulDDs` recursive function. Assuming it is called at the beginning of a dispute derivation and a set of goals γ, fixed strategy $Strat$, termination criteria C, advancement type A and search type St the initial function call is `GetSuccessfulDDs`$(\langle\!\langle(\gamma, \gamma, \varnothing, \varnothing, \varnothing)\rangle\!\rangle, \varnothing, Strat, C, A, St)$. Here, as well as in the following listings method $pop()$ called on a sequence returns the first element of it and removes it from the sequence. If called on a set it simply returns a single element from the set and removes it from the set.

input : $States = \langle\!\langle(\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_n, \mathbb{P}_n, I_{a_n}, I_{c_n}, Ctr_n)\rangle\!\rangle$, /* current dispute states */
$SuccStates = \{(\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_l, \mathbb{P}_l, I_{a_l}, I_{c_l}, Ctr_l)\}$ /* currently found successful states */,
$Strat = (<_M, H, R, A)$, /* strategy */
$C \in \{\texttt{TA}, \texttt{TC}, \texttt{TS}\}$ /* termination criteria */,
$A \in \{\texttt{DAB}, \texttt{DABF}, \texttt{DC}, \texttt{DS}\}$ /* advancement type */,
$St \in \{\texttt{DFS}, \texttt{BFS}\}$ /* search type, depth-first-search or breadth-first search */
output: $succStates = \{(\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_m, \mathbb{P}_m, I_{a_m}, I_{c_m}, Ctr_m)\}$ /* successful states found */

```
1  function GetSuccessfulDDs(States, SuccStates, Strat, C, A, St)
2      if States = ⟨⟨⟩⟩ then                                   /* if there are no more states to examine */
3          return SuccStates                                   /* return found successful states */
4      currState ⟵ States.pop()                                /* take first state from States */
5      possibleMoves ⟵ getPossibleMoves(currState, A)          /* get possible moves at the current state
         according to advancement A */
6      possibleMoves ⟵ filter(possibleMoves, currState)        /* filter possible moves */
7      states′ ⟵ States                                        /* initialize states′ with remaining states */
8      succStates′ ⟵ SuccStates                                /* initialize succStates′ with currently found successful states */
9      switch isOver(currState, possibleMoves, C) do           /* check if dispute over according to termination
         criteria C */
10         case ProponentWon do                                /* C indicates that proponent has won */
11             succStates′.append(currState)
12         case OpponentWon do                                 /* C indicates that opponent has won */
               // do nothing…
13         case NotOver do                                     /* C indicates that dispute has not terminated */
14             newStates ⟵ GetNextStates(currState, Strat, possibleMoves)
15             if searchType = DFS then                        /* if DFS prepend newStates to the beginning of the list */
16                 states′ ⟵ newStates.join(states′)
17             else // else, if BFS append newStates at the end
18                 states′ ⟵ states′.join(newStates)
19     GetSuccessfulDDs(states′, succStates′, Strat, C, A, St) /* continue searching using new states
         states′ */
```

We have additionally decided to re-use the procedure from Algorithm 2 for the complete semantics in order to define algorithms searching for grounded and preferred extensions, utilizing the mechanism of constraints (Ctr) from Definition 4.2.1.

Algorithm 3: `GetSuccessfulGrdDDs` recursive function. For a set of goals γ, a fixed strategy $Strat$ and a search type St the initial function call is `GetSuccessfulGrdDDs`$(0, \gamma, \{\varnothing\}, Strat, St)$.

input : N, /* an integer, current assumption set size */
$\gamma \subseteq \mathcal{L}$, /* goals */
$Asms = \{U_1, \ldots, U_n\}, U_i \subseteq \mathcal{A}$ for $1 \leq i \leq n$, /* set of sets of assumptions */
$Strat = (<_M, H, R, A)$, /* strategy */
$St \in \{\texttt{DFS}, \texttt{BFS}\}$ /* search type */
output: $succStates = \{(\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_l, \mathbb{P}_l, I_{a_l}, I_{c_l}, Ctr_l)\}$ /* success. states found */

```
1  function GetSuccessfulGrdDDs(N, γ, Asms, Strat, St)
2      if Asms = ∅ then                                   /* if all assumption sets in Asms checked */
3          n' ⟵ N + 1                                     /* increase the cardinality of next assumption sets */
4          asms' ⟵ {x ⊆ A | |x| = n'}                     /* get new, larger assumption sets */
5          GetSuccessfulGrdDDs(n', γ, asms', Strat, St)   /* search for grounded extensions using new
             assumption sets asms' */
6      else
7          aSet ⟵ Asms.pop()                              /* take any assumption set from Asms */
8          ctrs ⟵ A \ aSet                                /* fix the contraries as the complement of aSet */
9          succGrd ⟵ GetSuccessfulDDs({(aSet, aSet, ∅, ∅, Ctr)}, ∅, Strat, TC, DC, St)
             /* find all successful grounded dispute states using aSet */
10         if succGrd = ∅ then                            /* if no successful found */
11             GetSuccessfulGrdDDs(N, γ, Asms, Strat, St) /* check remaining sets in Asms */
12         else                                           /* otherwise, return those successful states which justify goals γ */
13             return {(B, P, I_a, I_c, Ctr) ∈ succGrd | γ ⊆ P}
```

As has been previously defined, a grounded extension S is a $\subseteq$-minimal complete extension, meaning that no strict subset of a grounded extension is a complete extension. Hence, in the procedure from Algorithm 3 we search for such an extension, by starting from the minimal set of assumptions (the empty set) and gradually adding new assumptions (line 4) until a complete extension is found. If a current set of assumptions $aSet$ is considered, the rest of the assumptions $\mathcal{A} \setminus aSet$ become constraints (lines 7 and 8). Then (line 9) the `GetSuccessfulDDs` procedure is used to check if $aSet$ is a complete extension (note the use of `DC` and `TC`). If that is the case, then the extension is returned, provided that it justifies the goals γ. Otherwise, the search continues.

Similarly to its grounded counterpart, the algorithm for the preferred semantics also utilizes constraints and the procedure `GetSuccessfulDDs` to find successful dispute derivations satisfying completeness. However, in case of the preferred semantics, the search starts from the maximal set of assumptions (simply all assumptions, $\mathcal{A}$) which gets gradually reduced. The procedure is presented in Algorithm 4.

Algorithm 4: `GetSuccessfulPrefDDs` recursive function. For a set of goals γ, a fixed strategy $Strat$ and a search type St the initial function call is `GetSuccessfulPrefDDs`(γ, $\varnothing$, $\{\mathcal{A}\}$, $\varnothing$, $\varnothing$, $Strat$, St).

input : $\gamma \subseteq \mathcal{L}$, /* goals */
$SuccStates = \{(\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_n, \mathbb{P}_n, I_{a_n}, I_{c_n}, Ctr_n)\}$, /* currently found successful states */
$Asms = \{U_1, \ldots, U_l\}, U_i \subseteq \mathcal{A}$ for $1 \leq i \leq l$, /* set of sets of assumptions */
$SuccAsms = \{U_1, \ldots, U_j\}, U_i \subseteq \mathcal{A}$ for $1 \leq i \leq j$, /* set of successful assumption sets */
$FailAsms = \{U_1, \ldots, U_k\}, U_i \subseteq \mathcal{A}$ for $1 \leq i \leq k$, /* set of failed assumption sets */
$Strat = (<_M, H, R, A)$, /* strategy */
$St \in \{\texttt{DFS}, \texttt{BFS}\}$ /* search type */
output: $succStates = \{(\mathbb{B}_1, \mathbb{P}_1, I_{a_1}, I_{c_1}, Ctr_1), \ldots, (\mathbb{B}_m, \mathbb{P}_m, I_{a_m}, I_{c_m}, Ctr_m)\}$ /* successful states found */

```
1  function GetSuccessfulPrefDDs(γ, Asms, SuccAsms, FailAsms, Strat, St)
2      if Asms = ∅ then                                   /* if all assumption sets in Asms checked */
3          asms' ⟵ {newASet ⊂ fail | fail ∈ FailAsms, |newASet| = |fail| − 1, newASet ∉
             FailAsms, ¬∃aSet ∈ SuccAsms s.t. newASet ⊆ aSet}   /* get new set of assumption sets */
4          if asms' = ∅ then                              /* if no new assumption set created, return from function */
5              return {(B, P, I_a, I_c, Ctr) ∈ SuccStates | γ ⊆ P}   /* return succ. states justifying γ */
6          else                                           /* otherwise search new pref. extensions using the new set */
7              GetSuccessfulPrefDDs(γ, SuccStates, asms', SuccAsms, FailAsms, Strat, St)
8      else
9          aSet ⟵ Asms.pop()                              /* take any assumption set from Asms */
10         ctrs ⟵ A \ aSet                                /* fix the contraries as the complement of aSet */
11         succPref ⟵ GetSuccessfulDDs(({(aSet, aSet, ∅, ∅, ctrs)}, ∅, Strat, TC, DC, St)
             /* find all successful preferred dispute states using aSet */
12         if succPref = ∅ then                           /* if no successful found */
13             GetSuccessfulPrefDDs(γ, SuccStates, Asms, SuccAsms, FailAsms ∪ {aSet},
                 Strat, St)   /* add aSet to set of failed assumption sets, check remaining sets in Asms */
14         else                                           /* otherwise add aSet to successful assumptions and succPref to successful states */
15             GetSuccessfulPrefDDs(γ, SuccStates ∪ succPref, Asms, SuccAsms ∪ {aSet},
                 FailAsms, Strat, St)
```

4.3 Approximate reasoning

When computing successful dispute derivations becomes too inefficient (especially, in case of large frameworks), approximate methods can offer a trade-off between faster computation and reduced accuracy. The main motivation behind using approximate methods is the hypothesis that it allows to obtain the answer more efficiently, when some simplifications are applied.

There has been some work in the use of approximate algorithms in formal argumentation. In structured argumentation alone one can distinguish between the techniques proposed by Thimm and Rienstra [26] and Sun [25]. Since the first one is defined in terms of the ASPIC+ structured argumentation framework we will not consider it in this work and instead focus on a simplification of an approach defined in the latter work, which is specifically tailored for dispute derivations in ABA, called “rule sampling”.

Rule sampling allows for conducting dispute derivations on parts of the original framework, which reduces the number of rules in the framework. Rules are crucial to ABA, since they allow to construct arguments, hence this approach assumes that many arguments will not have to be constructed or counter-attacked if a subset of rules is removed. Additionally, note that the presence of many same-headed rules is one of the most significant sources of non-determinism and inefficiency in our algorithms.

We can however distinguish at least two approaches to this kind of approximation, which we will refer to as "static" and "dynamic" rule sampling. The first one removes a number of rules from the framework at the beginning of the dispute derivation, banning them from being used throughout the entire dispute derivation. On the other hand, the latter one does so during the dispute derivation, i.e. when a number of possible rule-moves is applicable to the current dispute state, some of them might be discarded. Although both approaches may seem very similar, there are differences: first of all, the "static" approach may filter out rules, which would not be used during the dispute derivation anyway, whereas the "dynamic" approach will remove rules relevant to the current dispute state. Moreover, in the "dynamic" rule-sampling, some ruled-out moves may again become available in further steps of the dispute derivation, assuming that they will not be randomly chosen to be removed again. "Static" sampling behaves differently, i.e. once a rule is randomly removed from the search at the beginning, it will never become available again.

Following the approach by Sun [25], sampling in this work is carried out for both players independently, i.e. each player gets assigned a value denoting the probability of a rule to not be removed, namely p_{prop} and p_{opp} for the proponent and the opponent, respectively.

In terms of the credulous acceptance problem, independent sampling has the advantage over "general" sampling (single reduction of rules' set for both players) in that, while it is an approximation it still can guarantee some correctness for certain cases. Note that with no sampling ($p_{prop} = 1, p_{opp} = 1$) the obtained answers are guaranteed to be true. With sampling from the opponent side ($p_{prop} = 1, p_{opp} \in [0, 1]$), the negative answers are guaranteed to be true. This is a simple consequence of the fact that if the proponent cannot successfully defend from a subset of all opponent's constructible arguments, then certainly he cannot defend from all of them either. The positive answers are not guaranteed to be true because, as mentioned before, there might be arguments from which the proponent might be unable to defend himself, which have not been constructed due to the reduction of the opponent's rule set. The exact opposite holds for sampling from the proponent, i.e. if $p_{prop} \in [0, 1]$, $p_{opp} = 1$, where only the positive answers are guaranteed to be correct. In case of bi-directional sampling ($p_{prop} \in [0, 1], p_{opp} \in [0, 1]$) no answers are guaranteed to be correct. This shows that for large frameworks one could use approximation to obtain an answer more quickly and still be certain about its correctness for certain cases.

Given that these kinds of distinctions between various sampling combinations have been defined, it is of interest to examine not only the accuracy (probability, that an obtained answer is correct) obtained by an approximation, but other indicators such as *sensitivity* (the probability that an obtained positive answer is correct) and *specificity* (probability that an obtained negative answer is correct). According to what has been stated before, we expect the results of sampling from the proponent side to have specificity of 1.0 and sampling from the opponent side to have sensitivity of 1.0.

5 Implementation

We have implemented the rule-based approach to FlexDDs in Scala version 2.13.5 as the system which we call `flexABle`. The choice of programming language for this project was not accidental. Given that most of the algorithms defined in Section 4 include recursive calls or the fact that ABA-arguments are defined as trees (which cannot be operated on without recursion), many other (non-functional) programming languages would require to optimize those functions and rule out the recursion in order to avoid stack overflowing. However, recursive calls are the preferred way of handling repetitive operations in functional programming languages (such as Scala), which are converted to regular loops "under the hood" (tail recursion). This way the efficiency is not affected and simultaneously, code written in Scala can remain simple, elegant and easy to understand. Additionally, Scala offers a built-in support of numerous operations which enable implementing many operations directly from their formal definitions. Finally, functional programming's pattern matching technique is of great use whenever one needs to differentiate between divergent entities be that, in our case, the recognition between single statements s or rules $h \leftarrow B$ for instance within the set of elements put forward by each player ($\mathbb{B}$ or $\mathbb{P}$) or between moves of different types.

The system `flexABle` supports a variety of tasks related to (flexible) dispute derivations in ABA. In the following section some of the main features will be described. The program and some examples are available at

`https://github.com/gorczyca/aba-dd-rule-based`

5.1 Interactive Reasoning

The main objective of `flexABle` is to support the execution of rule-based flexible dispute derivations. Users can simulate moves of both players according to any advancement type or termination criterion. At each step all possible applicable moves can be viewed from which one can be chosen to be performed next. The system prints the current dispute state and, whenever some termination criterion is met, this is indicated to the user. Additionally, the user can backtrack in order to e.g. pick a different move at a certain point. At any stage the termination criterion and advancement type can be changed. Derivations can additionally be restored to any past state.

The depicted Dispute States 2. and 3. on the left side of Figure 5.1 correspond to States 14. and 15. of the dispute-search from Figure 1.4. The user interface of `flexABle` starts with the system informing about the last move performed and the current dispute state by showing the contents of the following sets: the proponent's set $\mathbb{P}$ (`P`), the opponent's set without proponent's pieces $\mathbb{B} \setminus \mathbb{P}$ (`B\P`), defences $\mathcal{D}$ (`D`) and culprits $\mathcal{C}$ (`C`). This is followed by showing all applicable moves at that stage upon the user's request (command "`?`" in the figure). Next the user chooses the move

to be performed by specifying its type and index as presented under “Possible moves”. At Step 3. the termination of the derivation is announced together with its outcome.

```
2: PB1: q ← i:
P:
 *q ← i; *a; *p ← a,b,q;
 *i; *p; *q; *b
B\P:

D:
 a,i,b
C:

?
Possible moves:
OB2:
 0: OB2: xa ← e,v
 1: OB2: xa ← z
 2: OB2: xi ← g
ob2 2
3: OB2: xi ← g:
P:
 *q ← i; *a; *p ← a,b,q;
 *i; *p; *q; *b
B\P:
 *g; *xi; *xi ← g
D:
 a,i,b
C:

Game over. Opponent won.
```

```
6: OB1: z ← f:
 ...
struct
[0] Proponent Inside
[1] Proponent Attacking
[2] Opponent Inside
[3] Opponent Attacking
1
Goal view:
 UNBLOCKED DEFENCES CONTRARIES:
 [0] xa
0
Argument view:
 [0] xa ← [z ← [f]]
 [1] xa ← [e,v ← [i]]
 [2] xa ← [e,v ← [h]]
1
Statement view:
 [0] e
 [1] i
0
Rule/assumption view:
 [0] PB2: xe ← c
0
7: PB2: xe ← c:
 ...
```

Figure 5.1: User interface of `flexABle`. Here, as well as in all other examples from this section, the used input framework is the one from Example 1.1.1. The user input is highlighted with green colour. For each assumption $u \in \mathcal{A}$, its contrary $\bar{u}$ is denoted as xu. The left-side listing presents the regular support of rule-based FlexDDs execution. The right-side listing illustrates the “interactive-structured” view.

Interactive, Structured Reasoning

Our system `flexABle` offers additional support for exploring dispute derivations, which we call the “interactive-structured” view. This view provides four approaches, namely *proponent’s inside / attacking* and *opponent’s inside / attacking*, allowing to explore the current dispute state from

the most general perspective to the most specific one and choose the next move to perform. E.g. given a player, the *inside* views first present statements for which the player constructs arguments and allows to choose one of them. Next the incomplete arguments of the player for this statement are presented, from which the user can again choose one, followed by `flexABle` presenting non-assumption premises of the argument and again allowing the user to choose one. Finally, once a statement has been chosen, the system proposes all possible moves (rules) for the player to expand it.

The *attacking* views behave in a similar manner, but with a small difference. Given a player they present the opposing player's arguments and allow the user to choose how they should be attacked. As an example consider the right-side listing of Figure 5.1. The starting state is Step 6. from the dispute derivation from Figure 1.3 when the user switches to the *proponent's attacking* view. First (goal view) $\bar{a}$ (`xa`) is shown as the only statement for which the opponent builds their arguments. After choosing the only option all arguments for $\bar{a}$ that the opponent has constructed by that time, and which have not yet been counter-attacked are presented (argument view), from which the user chooses $\bar{a} \leftarrow [\boldsymbol{e}, v \leftarrow [\boldsymbol{i}]]$. This argument's assumptions $\boldsymbol{e}$ and $\boldsymbol{i}$ are later presented as the only two possible points of attack of this argument (statement view), out of which $\boldsymbol{e}$ gets chosen. In the last step (rule/assumption view) the only move attacking the chosen assumption is presented and selected by the user resulting in the creation of the next dispute state.

The "interactive-structured" view thus provides a user of `flexABle` with an insight of the internal structure of arguments and allows them to choose the next move from the "argument-based" viewpoint of dispute derivations.

5.2 Automatic Search

The system `flexABle` is also capable of finding a successful (rule-based FlexDD) dispute derivation given an input ABA framework provided that there exists one supporting the goals. It supports the semantics mentioned in this work i.e. admissible, complete, stable – by choosing the appropriate advancement type and termination criteria – as well as grounded and preferred – by making use of the algorithms from Chapter 4. The search strategy can be set to any combination of parameters expressible by Definition 4.1.8. A Prolog-inspired feature allows to either stop or continue the search once a successful dispute derivation has been found.

The left-side listing of Figure 5.2 presents the result of the automatic search of a successful dispute derivation given the framework from Example 1.1.1 and goal p and with `DAB` and `TA` as advancement type and termination criteria, respectively. The user keeps searching for the next successful dispute derivation until the entire search tree has been traversed. Note that the search corresponds precisely to that from Figure 1.3. The search strategy producing this output, according to the Definition 4.1.8, can be defined as follows: $Str' = (<'_M, \texttt{mostRules}, \texttt{bodySize}, \texttt{T})$ where $<'_M = \langle\!\langle \texttt{PF1}, \texttt{OB2}, \texttt{OF2}, \texttt{OB1}, \texttt{PB1}, \texttt{PB2}, \texttt{PF2} \rangle\!\rangle$.

The right-side listing of Figure 5.2 shows the use of a similar feature, namely performing n moves forward according to specified strategy. The dispute derivation after performing 8 moves arrives at Step 8. of the dispute derivation from Figure 1.3.

Additionally, `flexABle` supports approximate reasoning in both variants – "static" and "dynamic" described in the previous chapter. For each player, the user can also set the associated p

value in order to obtain an approximated answer using the desired sampling direction and rule's selection probability.

```
auto 1
Finding a successful derivation.
This can take a moment...
Successful derivation found in
0.0157521s.
1: [PB1: p ← a,b,q]
2: [OB2: xa ← e,v]
3: [OB2: xa ← z]
4: [OB1: v ← h]
5: [OB1: v ← i]
6: [OB1: z ← f]
7: [PB1: q ← xe,r]
8: [PB1: r ← c]
9: [PF1: xe ← c]
10: [PF1: xf ← xe]
Press ENTER to finish, ; to find
another one
;
Successful derivation found in
0.003243s.
1: [PB1: p ← a,b,q]
2: [OB2: xa ← e,v]
3: [OB2: xa ← z]
4: [OB1: v ← h]
5: [OB1: v ← i]
6: [OB1: z ← f]
7: [PB1: q ← xe,r]
8: [PB1: r ← d]
9: [PB1: xe ← c]
10: [PF1: xf ← xe]
Press ENTER to finish, ; to find
another one
;
No successful derivations found
```

```
f 8
8 moves performed.
1: [PB1: p ← a,b,q]
2: [OB2: xa ← e,v]
3: [OB2: xa ← z]
4: [OB1: v ← h]
5: [OB1: v ← i]
6: [OB1: z ← f]
7: [PB1: q ← xe,r]
8: [PB1: r ← c]

8: PB1: r ← c:
P:
 *r ← c; xe; *a; p ← a,b,q; p; *r;
 q; *b; *c; q ← xe,r
B\P:
 -e; -xa ← e,v; *v ← i; *f; *i; *v;
 *z ← f; *xa ← z; *h; *v ← h; *xa;
 *z
D:
 a,b,c
C:
 e
```

Figure 5.2: Automatic successful dispute derivation search in `flexABle`. Listing on the left side presents two full successful dispute derivations found. Listing on the right side presents the outcome of performing 8 moves forward.

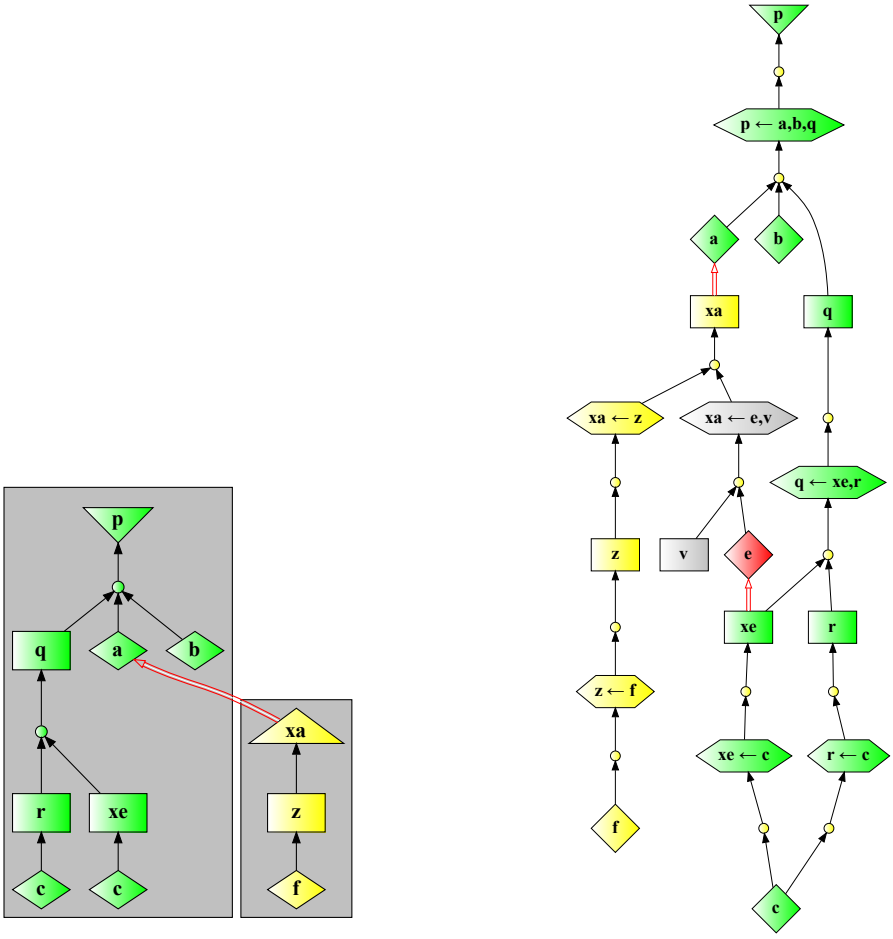


Figure 5.3: Graphical representation of Step 6. of the dispute derivation from Figure 1.3 in both representations, "argument-based" (left) and "rule-based" (right). In the left sub-figure the solid grey background indicates the arguments' completeness as before. The owner of the argument is represented with the arguments' nodes colour, where green-node arguments belong to the proponent and yellow-node ones to the opponent. The shapes of the argument tree nodes hint at the trait of the statement labelling the feature, with the triangular and diamond shapes marking arguments' main claims and assumptions respectively. Other ordinary statements have rectangular shapes. Again black arrows indicate rule-support whereas red ones indicate attack. In the right sub-figure each element of $\mathbb{B}$ is presented. Green elements belong to $\mathbb{P}$ and yellow ones to $\mathbb{B}\backslash\mathbb{P}$. As before, red arrows and black arrows indicate attack and support relation, respectively. Assumptions have diamond, rules hexagonal, goals triangular and regular statements rectangular shapes. Culprits have red background. Note that the grey-coloured nodes are not elements of $\mathbb{B}$, but they indicate that such a rule or statement is blocked.

5.3 Graphical Output

A vital feature of `flexABle` is the generation of a graphical representation given a rule-based dispute derivation state $(\mathbb{B}, \mathbb{P})$. The system converts the sets of rules and assumptions from players' sets and outputs the corresponding argument sets, i.e. the closure of arguments under sub-arguments and argument expansions constructible from $\mathbb{B}$ and $\mathbb{P}$ for the opponent and proponent, respectively. Then only maximal arguments for goals and contraries of culprits and contraries of defences are shown. The argument representation is encoded in the graph description language DOT, which can be later visualised by tools such as `Graphviz`.

Additionally, `flexABle` offers the option of generating a graphical representation of the "rule-based" dispute state $(\mathbb{B}, \mathbb{P})$. In this view, each element of $\mathbb{B}$ is shown and its properties are indicated with certain colours or shapes. Additionally, support and attack relations are shown.

The left sub-figure of Figure 5.3 presents the result of such a conversion from sets of rules and assumptions to arguments obtained by `flexABle`, where Step 6. of the dispute derivation from Figure 1.3 is shown. Note that the system indicates whether an argument is complete. Moreover, `flexABle` is capable of detecting and notifying about self-conflicting (attacking own assumptions) or circular (containing branches with directed paths leading from a node labelled with some statement s to another node labelled with the same statement s) arguments. The latter ensures termination of argument generation. The right sub-figure of Figure 5.3 presents the "rule-based" graphical representation of the same dispute state.

5.4 Interactive, Argument-Based Reasoning

A final, worth mentioning feature of `flexABle` is the "interactive, argument-based" reasoning mode. In this mode the user simply keeps choosing between the players and the system constructs one random argument for the chosen player. This enables the user to traverse through the dispute derivation much faster by putting forward complete arguments one-by-one. The system also indicates when the the dispute derivation has terminated.

Figure 5.4 depicts three successive states using the "interactive, argument-based" reasoning mode. The first, left state is the initial one for the framework from Example 1.1.1 with goal p. The second, middle one is after creating a random argument for the incomplete proponent's statement p. Finally, the last, right state is the result of creating a random argument for the opponent. At this point the dispute derivation terminates unsuccessfully, which is indicated with the red background colour.

Integration of Different Features

All of the features described in this chapter can be freely used at any stage of the dispute derivation and mixed with other features. E.g. it is possible to begin the dispute derivation by moving n moves forward, then, at the n-th step check possible moves and perform a chosen one. This can be followed by switching into "interactive, structured" or "interactive argument-based" view. From that state the user can decide to let the system find a successful state using automatic search. At any point, the dispute can be restored to any previous point, the termination criterion and

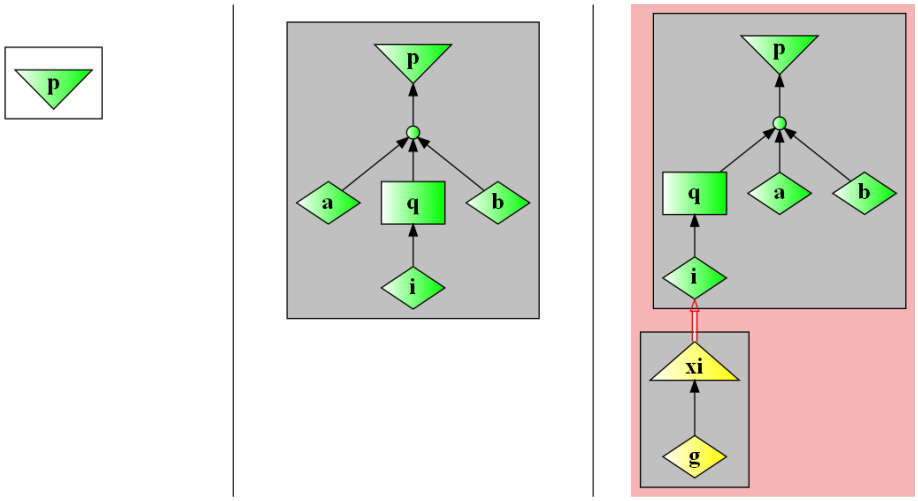


Figure 5.4: “Interactive, argument-based” reasoning.

advancement type can be modified and a graphical output (“argument” or “rule”-based) can be generated. Furthermore, the graphical representation can be generated automatically after each user’s action enabling users to observe dispute states as they grow.

6 Evaluation

6.1 Experimental Setup

The number of strategies, definable as in Section 4, is too large to be evaluated in its entirety. All possible orderings alone amount to 7! (5040) and this number would have to be multiplied by the magnitude of parameters for head- and rule choice, and for both proponent and the opponent. Therefore only some fraction of definable strategies can be used to conduct experiments on.

The idea behind the choice of meaningful strategies was to manually pick a couple of them, whose behaviour should be as diversified as possible. At the same time, strategies should only differ between one another in as few parameters as possible in order to enable us to draw conclusions from their behaviour and link these to said parameters. Table 6.1 presents the strategies we chose to focus on in our experiments.

The chosen strategies' intuitive behaviour is as follows:

- S_1 – prioritizing the opponent's moves, "patient" strategy – starts with the opponent constructing all possible arguments attacking the proponent's defences. Only if all of their applicable moves have been performed, can the proponent continue to construct their arguments or counter-attack the opponent. This strategy should maximally delay branching and aim to avoid the situation in which the opponent constructs all arguments in an unsuccessful branch. Additionally, delaying proponent's branching reduces the probability of constructing opponent's arguments against defences which anyway cannot be defended by the proponent.
- S_2 – prioritizing the proponent's moves, "eager" strategy – starts with the proponent constructing all necessary arguments for his yet unjustified claims and, as soon there is some

Strat.	$<_M$	H	R	A
S_1	⟪ PF1, OB1, OB2, OF2, PB1, PB2, PF2 ⟫	leastRules	newStatementsSize	⊤
S_{1a}		mostRules	newStatementsSize	⊥
S_{1b}	⟪ OB1, OB2 , OF2, PB1, PB2, PF2, PF1 ⟫	leastRules	newStatementsSize	⊤
S_2	⟪ PF1, PB2, PF2, PB1, OB2, OF2, OB1 ⟫	leastRules	newStatementsSize	⊤
S_{2a}		mostRules	newStatementsSize	⊥
S_{2b}	⟪ PB2, PF2, PB1, OB2, OF2, OB1, PF1 ⟫	leastRules	newStatementsSize	⊤
S_{AG}	⟪ PF1, PB1, OB2, OF2, OB1, PB2, PF2 ⟫	*random*	lookaheadSize	⊤

Table 6.1: Strategies selected for experimental evaluation. The H, R, and A choices in each strategy indicate the selected values for both players (both, the proponent and the opponent).

candidate for a culprit, then the proponent constructs complete arguments attacking it. Only when none of the proponent's moves are applicable, can the opponent perform their moves. This strategy allows for branching at the very early stages of the dispute derivation (which can be computationally inefficient), but at the same time, each of those "eager" branches restricts opponent moves by creating culprits.

In general, S_1 aims at finding long dispute derivations with few non-deterministic choices, whereas S_2 branches a lot but each dispute derivation branch should be substantially shorter than in the strategy S_1.

When choosing a rule head, both players in strategies S_1 and S_2 will always choose the one with the fewest rules (H = `leastRules`). Given a set of rules of a certain head both players will choose the one that introduces the fewest statements (R = `newStatementsSize`, $A = \top$). For these two strategies additional variants, namely a and b, have been defined. Strategy S_{xa} only differs from S_x in that it would choose a rule head with the greatest amount of rules and a rule introducing the greatest number of new statements (H = `mostRules`, R = `newStatementsSize`, $A = \bot$). Strategy S_{xb} on the other hand differs from S_x only in that the move type `PF1` is moved to the end of the $<_M$. These subtle changes are designed to help in isolating the influence of the single parameters on the performance of the strategies. In S_{xa} the influence of different head- and rule choice is of interest, whereas in S_{xb} we want to investigate how beneficial conservative forward moves are.

Additionally, a strategy S_{AG} has been defined, which is interesting for a number of reasons. First of all, it has been devised to "resemble" the system `abagraph`'s default strategy (the explanation of the similarities will be described in the next subsection devoted to `abagraph` and its strategies). Moreover, it is interesting to see, whether it pays off to use a more sophisticated (and computationally demanding) rule choice function (`lookaheadSize`).

Other evaluated systems

In order to compare the performance of `flexABle` with current state-of-the-art solutions, in our experiments we also consider the other most relevant system offering support for similar ABA-related problems called `abagraph`. It is an open-source system implementing graphical dispute derivations [7, 9] in SICStus Prolog, and the most recent version of Prolog-based ABA dispute derivation systems, upgrading its predecessors `proxdd` and `grapharg` [9]. It supports enumeration of successful dispute derivations for ABA frameworks w.r.t. the admissible and grounded semantics. For our experiments `abagraph` will be used with its default strategy and four more custom strategies defined in Table 6.2 and will operate on SICStus Prolog version 4.7.1.

We will give a brief informal explanation of the strategies from Table 6.2 and refer to the work of Craven and Toni [7] or `http://robertcraven.org/proarg/abagraph.html` for details. We have decided to include the default strategy in our evaluation, assuming that it might have been purposely chosen by the authors as notable, based on e.g. experiments they internally conducted or their intuition. We additionally wanted to check a few more `abagraph` strategies in order to at least slightly increase the chance of finding an optimal one for that system. To this end we fixed a number of parameters for each value, i.e. $Turn \in \{Proponent, Opponent\}$, $Sentence \in \{Patient, Eager\}$, $OppSet = Smallest$, and $Rule = Smallest\ body$. Strategies 1-4 are all resulting combinations of those parameters' values.

Str.	*Turn*	*OppSet*	*Sentence*	*Rule*
Default	*Proponent*	*Smallest*	*Patient*	*Look-ahead 1-step*
1	*Proponent*	*Smallest*	*Patient*	*Smallest body*
2	*Opponent*	*Smallest*	*Patient*	*Smallest body*
3	*Proponent*	*Smallest*	*Eager*	*Smallest body*
4	*Opponent*	*Smallest*	*Eager*	*Smallest body*

Table 6.2: Chosen `abagraph` strategies. *Sentence* and *Rule* parameters are the same for both players, proponent and opponent, for each strategy.

In strategies' definitions from Table 6.2 *Turn* indicates which player will be prioritized. Setting the *OppSet* parameter determines which opponent's justification set will be chosen. Furthermore, *Sentence* denotes which sentence for the proponent (or given an opponent's justification set) will be chosen, with *Patient* meaning non-assumption if possible and *Eager* meaning the contrary. Furthermore, if a sentence is chosen for backward extension then the *Rule* choice offers two alternatives: *Look-ahead 1-step* and selecting a rule with the smallest body.

In `abagraph` attacks of one player's argument happen at the turn of that player, i.e. when a player is chosen and an assumption from his set is selected then the opposing player can commence creating an attack against that assumption. This seems counter-intuitive from the perspective of FlexDDs definitions. It can however be mimicked to some degree in `flexABle`. Note that the default `abagraph` strategy will prioritize the proponent (*Turn* = *Proponent*). Then a non-assumption will be selected if possible (*Statement* = *Patient*) to be backward expanded using a rule. This should be resembled by `PB1` moves appearing second in the ordering $<_M$ of the strategy S_{AG} from Table 6.1. Otherwise, if no non-assumption is available, an assumption will be picked and the opponent will start constructing an argument against it. This in turn will be resembled in S_{AG} with moves `OB2` and `OF2` appearing at positions 3. and 4. in $<_M$. If no proponent moves are possible, the turn switches to the opponent who by sticking to the *Patient* strategy will select a non-assumption and try to backward extend it with a rule. This also will happen in S_{AG} due to `OB1` at position 5. in $<_M$. Otherwise, if an assumption is selected then the proponent has the chance to attack it, and it is no different in $<_M$ of S_{AG} where `PB2` and `PF2` occupy the last two positions. The `PF1` moves, having no counterpart in graphical dispute derivations, have been placed at the beginning of $<_M$ to check its influence. However, for our experiments we will also check strategies which do not use `PF1` moves when evaluating the `DAB` advancement type. H has been set to *random* due to there not being any equivalent choice point in `abagraph`. Finally, rule choices in both strategies are set to 1-step look-ahead.

Input frameworks

Following Lehtonen et al. [19] we have used the 680 ABA frameworks[1] used earlier in experiments by Craven et al. [9] or Lehtonen et al. [20] and likewise similarly for each framework 10 sentences were randomly chosen, forming 6800 benchmarks as framework-sentence pairs. Lehtonen et al. [19] have filtered the benchmarks obtained this way, removing the ones established as trivial in previous work [20] for decision problems. The trivial instances were those in which the sentence

[1] `http://robertcraven.org/proarg/experiments.html`

was either derivable from the empty assumption set or not derivable at all. In our experiments we decided that the output obtained in terms of dispute derivations might be insightful even for trivial problems, therefore all 6800 benchmarks were used for all our experiments. Nevertheless, results restricted to the non-trivial instances still indicate similar tendencies. Readers interested in those results are referred to the appendix.

Experimental setup

All experiments have been performed in parallel on a cluster operating on Red Hat Enterprise Linux Server version 7.9, kernel version 3.10.0-1127.19.1.el7.x86_64. Each task has been allocated with a 2.5Ghz Intel Xeon E5-2680 v3 CPU with 16GB RAM and has been given a 600-second timeout for tasks involving admissible semantics and a 1200-second timeout for other semantics.

6.2 Results

In Tables 6.3, 6.4 and 6.7 to 6.9 the leftmost "Str" column indicates a strategy, while columns `DAB`, `DABF`, `DC`, and `DS` advancement types. Sub-columns indicate search type, either depth- (DFS) or breadth-first search (BFS). The "#timeout." row shows for how many instances computation time exceeded the given cut-off value, "total time" and "95% time" denote, respectively, the combined times for all instances and for 95% of instances with the fastest solving time in hours for a given setup (consisting of a triple – strategy, advancement type and search type). The latter gives an insight in how fast a strategy is if one decides to filter out a small portion of the most bothersome instances.

Rows "min", "median", "mean", and "max" present information about instances which did not time out for a given setup in seconds. Their meaning is self-explanatory, i.e. "min" and "max" stand for minimal and maximal time needed to solve an instance within a setup. "Mean" and "median" denote mean and median of all time spans needed to solve all instances within a setup. As mentioned before, timed-out instances are not included in this analysis, which favours "mean" and "median" in those setups for which the time needed often exceeded given bounds. This was however done intentionally, as otherwise "max" value would denote the timeout value in almost every case. The table rows with green background denote globally best obtained results (best values for the given parameter among all setups) whereas those with red background hint at the globally worst.

We acknowledge that all experimental results and therefore all drawn conclusions are based on a single benchmark set, which may not be the most representative one. This choice was however a consequence of the fact that said benchmarks served as input frameworks in numerous evaluations of previous systems for ABA, hence we decided to follow this convention.

6.2.1 Admissible semantics

Table 6.3 presents results for the credulous admissible acceptance problem obtained by rule-based FlexDDs as implemented in `flexABle`. In what follows we will examine them trying to answer research questions posed in the introduction by isolating the impact of single features on the outcome.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	193	179	0	2	13	13	36	39
	total time [h]	38.39	35.70	3.73	4.05	7.37	7.84	11.84	12.56
	95% time [h]	2.79	2.83	2.67	2.77	2.96	3.27	2.73	2.84
	min [s]	1.15	1.21	1.15	1.19	1.15	1.28	1.19	1.21
	median [s]	1.45	1.48	1.43	1.49	1.57	1.73	1.45	1.53
	mean [s]	3.39	3.18	1.97	1.97	2.76	3.01	3.11	3.22
	max [s]	595.72	587.20	468.25	497.25	501.04	461.19	488.27	554.67
S_{1a}	#timeout.	868	369	9	11	17	22	82	76
	total time [h]	153.97	68.50	6.37	6.32	9.29	9.25	21.24	20.52
	95% time [h]	97.29	11.80	3.07	3.07	2.96	3.06	2.77	2.94
	min [s]	1.22	1.11	1.21	1.24	1.20	1.20	1.14	1.21
	median [s]	1.54	1.47	1.64	1.66	1.53	1.60	1.44	1.53
	mean [s]	5.63	3.90	2.58	2.38	3.42	2.97	4.06	4.20
	max [s]	563.50	586.91	541.11	471.05	530.01	394.96	536.21	588.57
S_{1b}	#timeout.	193	179	195	180	272	236	300	257
	total time [h]	38.24	35.79	38.55	35.69	52.23	45.33	56.28	48.84
	95% time [h]	2.67	2.92	2.79	2.72	3.50	3.08	3.52	3.32
	min [s]	1.16	1.17	1.15	1.14	1.19	1.16	1.18	1.18
	median [s]	1.38	1.56	1.46	1.43	1.52	1.42	1.51	1.63
	mean [s]	3.31	3.23	3.30	3.08	3.80	3.28	3.47	3.29
	max [s]	577.92	586.83	591.18	523.41	561.56	584.93	594.66	573.06
S_2	#timeout.	133	115	0	0	180	260	167	240
	total time [h]	28.64	26.08	3.06	3.12	40.82	56.41	39.12	53.04
	95% time [h]	2.98	3.00	2.68	2.73	3.92	6.48	3.77	5.46
	min [s]	1.18	1.24	1.13	1.19	1.22	1.21	1.18	1.15
	median [s]	1.63	1.64	1.47	1.49	1.58	1.64	1.57	1.49
	mean [s]	3.49	3.72	1.62	1.65	5.88	7.19	6.12	7.15
	max [s]	595.30	577.70	142.26	145.30	599.98	599.69	596.26	598.55
S_{2a}	#timeout.	760	337	1	4	181	262	173	252
	total time [h]	136.43	61.98	4.46	4.88	41.10	56.48	39.80	54.46
	95% time [h]	79.75	5.36	2.73	2.88	3.93	6.46	3.68	5.84
	min [s]	1.14	1.15	1.19	1.18	1.17	1.17	1.17	1.19
	median [s]	1.45	1.49	1.47	1.56	1.57	1.59	1.49	1.63
	mean [s]	5.80	3.22	2.27	2.23	5.95	7.04	5.96	6.84
	max [s]	592.45	506.79	556.86	550.73	585.76	590.76	589.11	596.52
S_{2b}	#timeout.	133	115	133	115	416	838	416	800
	total time [h]	28.55	26.16	28.61	25.84	81.07	166.78	80.48	159.82
	95% time [h]	2.95	3.05	2.93	2.65	24.39	110.09	23.80	103.13
	min [s]	1.21	1.28	1.22	1.13	1.18	1.15	1.21	1.21
	median [s]	1.60	1.67	1.58	1.43	1.50	1.60	1.62	1.55
	mean [s]	3.44	3.76	3.47	3.59	6.61	16.34	6.28	15.86
	max [s]	584.23	579.89	588.73	586.65	566.89	571.98	550.97	598.96
S_{AG}	#timeout.	566	331	1	2	11	15	54	59
	total time [h]	102.05	62.26	4.51	4.65	7.54	7.49	15.16	15.33
	95% time [h]	45.37	5.91	2.95	2.95	3.20	2.67	2.89	2.80
	min [s]	1.19	1.15	1.22	1.21	1.22	1.12	1.14	1.14
	median [s]	1.61	1.50	1.58	1.57	1.71	1.38	1.54	1.46
	mean [s]	4.45	3.93	2.30	2.29	3.03	2.65	3.29	2.93
	max [s]	588.04	590.52	543.61	575.64	517.40	509.57	588.91	519.54

Table 6.3: Results for the admissible semantics using `flexABle`

Impact of Conservative Forward Moves (PF1)

In all setups the DABF advancement type dominates its counterparts: it has the least number of timeouts and requires the least amount of time to solve all instances. Note that the only exceptions to that are the S_{1b} and S_{2b} strategies, which are those strategies which do not take advantage of the conservative forward moves (PF1 move is at the end of $<_M$ of those strategies, effectively resulting in those strategies being almost identical for both DAB and DABF which is confirmed by the data in Table 6.3). Strategy S_2 is of particular interest in DABF as it not only was able to solve all instances, but also did that in the least amount of time (3.06h for its DFS variant). Additionally, it holds the least mean of solving time (1.62s for DFS). This is surprising as none of the other setups, which due to having some instances timed out and therefore ruled out for the computation of this statistics, was able to beat the result of strategy S_2 when used with DABF. Moreover, out of all instances, strategy S_2 with DABF needed only 142.26s to solve the most demanding instance. Note that for almost all other setups this value is close to the timeout value of 600s.

Note that for DAB strategies $S_{xb} = S_x$ for $x \in \{1, 2\}$ as there are no PF1 moves in DAB anyway. For all other advancement types (DABF, DC, and DS) the S_{xb} strategies perform significantly worse than S_x, which again indicates the advantageous nature of conservative forward moves.

The above indicates that the "non-branching" nature of the conservative forward moves is of great value in terms of performance of dispute derivations.

Impact of Strategy

As discussed previously, strategy S_2 when combined with DABF produced the best results. However, used with DABF all major strategies (S_1, S_2, S_{AG}) perform similarly well. One could therefore conclude, that such good performance is the result of the use of conservative forward moves rather than the strategy.

Nevertheless, S_2 has obtained the best results among setups utilizing DAB. This suggest that when there are no non-conservative moves available, the strategy with an "eager" proponent (favourising proponent moves over the opponent) is more efficient. This does not hold in the other scenario however (with DC and DS advancement types). The reason is obvious, as an "eager" proponent more often has to guess an assumption (using non-conservative forward moves) which substantially increases the search space.

Observe that S_{xa} performs worse than S_x for $x \in \{1, 2\}$, particularly when used with DAB. This is an indication that when choosing rule heads it is better to take those with fewest rules applicable and when selecting a rule it is more reasonable to take those that introduce the least number of new statements, regardless of advancement type, search type or strategy.

Finally, the S_{AG} strategy is among the worst-performing ones for DAB. The reason for that might be that often a head with many applicable rules is selected when this choice is random (this would be suggested by the fact that S_{AG} performs slightly better than S_{1a} and S_{2a}, which choose heads having maximal number of rules). Additionally, the `lookaheadSize` computation of S_{AG} might cause additional inefficiencies. Note that for other advancement types S_{AG} produces results comparable with the best strategies for those advancement types. This, in turn, might be caused by taking advantage of the conservative forward moves. Additionally, the `lookaheadSize` for TA might behave similarly to choosing rules which introduce the least number of new statements which is a desired behaviour as discussed before. What is more, the non-conservative for-

ward moves, causing DC and DS setups to be less efficient, are second to last in $<_M$ of S_{AG}. All of the above possibly results in S_{AG} performing slightly worse than S_1 for DC and DS.

Impact of Non-Conservative Forward Moves

As discussed previously, our results indicate, that the non-conservative forward moves do not offer any improvement in terms of efficiency. Note that no setup employing DC or DS managed to outperform the results of its DABF counterpart. This is particularly visible in S_2 where non-conservative moves are used the most and obtained results are significantly worse than in setups with advancements without those kinds of moves. However, we think that for certain cases the non-conservative forward moves might be beneficial. Particularly, when some "prior" knowledge, especially regarding assumptions present in the successful dispute derivations' defences is available, then this kind of moves could lead to finding such dispute derivations more efficiently. For most of the cases however, the non-conservative forward moves cause additional inefficiencies.

Impact of Search Type

The results indicate that BFS leads to slightly better results when used with DAB in general, but particularly better with DAB and strategies S_{1a}, S_{2a}, and S_{AG}. These strategies tend to choose rules introducing many new statements, which, as argued before, is inefficient. While in such situations DFS would be stuck in those inefficient branches of the search tree, BFS helps to escape them by jumping to a sibling branch. Efficiency gain in other strategies in DAB is negligibly small.

For DABF it is difficult to find any strong correlations between efficiency and search type, as this advancement type is very efficient regardless.

BFS should however especially be avoided when non-conservative moves are frequently used (see strategies S_2, S_{2a} and S_{2b}). In such setups search often branches between unrelated assumptions. Many of such branches are fruitless, but BFS examines all of them equally.

Number of Moves in Successful Dispute Derivations – Analysis

While performing experiments we recorded a number of parameters to further investigate the characteristics of the obtained dispute derivations. For each successful dispute state found we saved the number of defences, culprits, proponent / opponent rules and statements as well as the number of moves required to obtain the state. We will only consider the last parameter here, whereas the rest can be found in the appendix.

Table 6.4 presents the statistics regarding the number of moves needed to obtain a successful state for instances solved in all setups. Furthermore, we only include non-trivial instances (as described by Lehtonen et. al [19, 20]) in order to emphasize the differences and avoid data contamination.

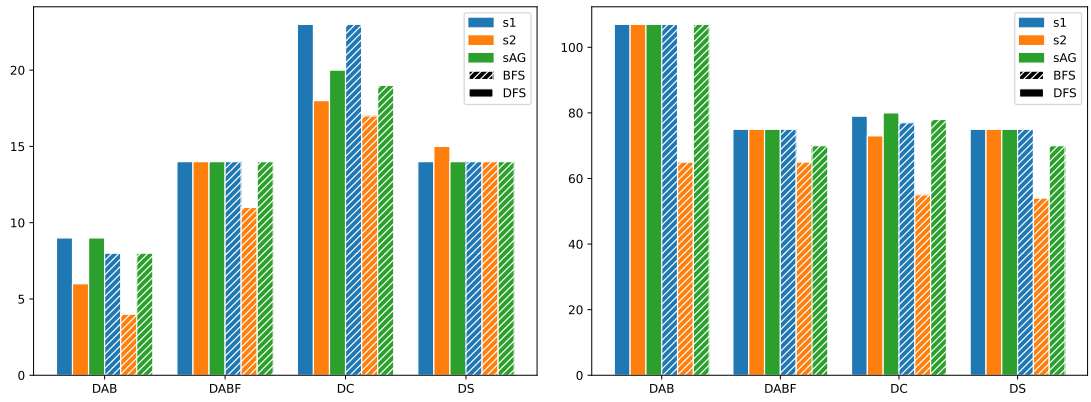
Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	1.00	1.00	2.00	2.00	2.00	2.00	2.00	2.00
	median	9.00	8.00	14.00	14.00	23.00	23.00	14.00	14.00
	mean	19.29	18.91	20.67	20.43	29.17	29.17	20.63	20.40
	max	107.00	107.00	75.00	75.00	79.00	77.00	75.00	75.00
S_{1a}	min	1.00	1.00	2.00	2.00	3.00	3.00	2.00	2.00
	median	8.00	6.00	13.00	13.00	24.00	24.00	13.00	13.00
	mean	19.50	18.12	19.96	19.64	29.16	29.13	19.91	19.60
	max	107.00	107.00	75.00	70.00	79.00	77.00	75.00	70.00
S_{1b}	min	1.00	1.00	1.00	1.00	2.00	2.00	1.00	1.00
	median	9.00	8.00	9.00	8.00	21.00	20.00	9.00	9.00
	mean	19.29	18.91	19.29	18.91	30.85	30.84	19.23	18.80
	max	107.00	107.00	107.00	107.00	107.00	107.00	107.00	107.00
S_2	min	1.00	1.00	2.00	2.00	2.00	1.00	2.00	1.00
	median	6.00	4.00	14.00	11.00	18.00	17.00	15.00	14.00
	mean	14.19	5.72	18.93	11.90	23.12	19.78	20.31	16.91
	max	107.00	65.00	75.00	65.00	73.00	55.00	75.00	54.00
S_{2a}	min	1.00	1.00	2.00	2.00	3.00	1.00	2.00	1.00
	median	6.00	4.00	13.00	11.00	18.00	16.00	14.00	13.00
	mean	14.77	5.79	18.16	11.59	22.92	19.79	19.68	16.54
	max	107.00	65.00	75.00	65.00	74.00	59.00	71.00	54.00
S_{2b}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	6.00	4.00	6.00	4.00	13.00	11.00	6.00	6.00
	mean	14.19	5.72	14.19	5.72	18.98	12.85	15.18	8.94
	max	107.00	65.00	107.00	65.00	109.00	49.00	107.00	49.00
S_{AG}	min	1.00	1.00	1.00	2.00	2.00	1.00	2.00	2.00
	median	9.00	8.00	14.00	14.00	20.00	19.00	14.00	14.00
	mean	19.14	18.72	20.38	20.25	25.63	25.37	20.36	20.22
	max	107.00	107.00	75.00	70.00	80.00	78.00	75.00	70.00

Table 6.4: Statistics regarding the number of moves of successful dispute derivations for the admissible semantics using `flexABle`.

First of all, note that in no setup does the value for a BFS variant exceed its DFS counterpart[2]. In most cases both values are the same or very similar but in some variants they vary significantly (observe the "max" values in setups with DC and S_{2b} where DFS holds the globally maximal value and BFS the globally minimal one). This is an obvious consequence of the virtue of BFS which guarantees to find the shortest path if one exists. Therefore, if one requires dispute derivation to be brief, they should use BFS.

The maximal number of moves needed to find a successful dispute derivation was 109 in a setup employing DC and strategy S_{2b} with DFS. Almost all setups with DAB or with strategy S_{1b} or S_{2b} obtained a similarly large number of 107. Such results suggest that avoiding conservative forward moves can lead to dispute derivations being longer. Note that for some of the aforementioned setups, when the search type was BFS and the strategy was S_{2b}, the maximal number of moves is significantly lower. In those cases an "eager" strategy might have caused the opponent's moves to be blocked what in turn leads to dispute derivations being shorter and, as argued before, shorter paths are more likely to be found in a search tree when using BFS.

Data also implies that setups not taking advantage of the conservative forward moves usually produce dispute derivations with the least number of steps (see "min", "median" and "mean" in setups with DAB). The reason behind this is that in those setups no unrelated moves are performed, contrary to when forward moves are applicable. In the latter situation rules are automatically used if their bodies are justified. Even longer paths are produced when non-conservative moves can be applied, where random assumptions, and therefore complete pieces are added to the dispute states resulting in even more rules being available for PF1 moves. This can be seen by DC setups yielding maximal values for all parameters ("min", "median", "mean", and "max").



(a) Median

(b) Maximal number

Figure 6.1: Bar plots indicating the number of moves in successful dispute derivations for admissible semantics using flexABle.

[2]There is an exception to that in statistics "min" for strategy S_{AG} with DABF, namely the BFS variant provides a higher value than DFS. This is however due to the fact that S_{AG} chooses rule heads randomly, hence DFS and BFS trees for this strategy can differ. Note that "mean", "median", "max" as well as "min" for other advancement types within this strategy for BFS never exceeds the DFS variant meaning that usually BFS provides the shortest path even when the rule choice is random.

The two aforementioned results show that, although restricting moves to only backward ones usually produces shorter dispute derivations, in some cases forward moves significantly shorten the length of obtained dispute derivations. Figure 6.1 visualises this by representing the median and the maximal number of moves for the three major strategies (S_1, S_2, and S_{AG}) with all advancement- and search types.

Comparison with abagraph

To compare our system and the novel form of dispute derivations we have performed the same set of experiments using abagraph– implementing graphical dispute derivations.

	abagraph strategy				
	Default	1	2	3	4
#timeout.	142	139	258	139	764
total time [h]	62.53	76.73	120.86	80.52	256.42
95% time [h]	34.41	48.72	74.27	52.70	199.57
min [s]	0.66	0.67	0.65	0.74	0.65
median [s]	17.76	17.42	21.51	18.54	69.89
mean [s]	21.00	28.94	42.82	30.99	76.84
max [s]	544.24	542.47	562.49	461.95	589.76

Table 6.5: Results for the admissible semantics using abagraph

As stated before and backed by the results, the non-conservative moves do not seem to improve the performance of flexABle. Therefore when comparing its efficiency with that of abagraph we will focus only on those FlexDD variants which employ DAB or DABF advancement types. This will allow us to learn about how the FlexDD procedure performs when either restricted to backward moves only (as in graphical dispute derivations which abagraph implements) or when the most impactful optimization of conservative forward moves is compared to abagraph.

The results obtained for the same benchmarks using the latter system are presented in Table 6.5. When juxtaposed with the results for our system the number of timeouts is the most conspicuous statistic. Note that among all setups which take advantage of the conservative forward moves the highest number of timeouts is 11 (DABF, BFS, S_{1a}), whereas the best result amid abagraph strategies is 139 (for strategies 1 and 3). Furthermore, again from setups utilizing PF1 moves, strategy S_{1a} with DFS required the greatest amount of time to solve[3] all instances (6.37h) and 95% of fastest instances (3.07h). On the other hand, the fastest (Default) strategy of abagraph required around 10 times more time (62.53h and 34.41h for all and 95% of instances respectively). Those statistics clearly imply that flexABle with PF1 moves dominates abagraph, and consequently some variants of FlexDDs are superior to Graph-DDs in terms of performance.

The situation is different when FlexDDs, as implemented in flexABle, are deprived of PF1 moves. Note that the least number of timeouts among abagraph strategies is substantially lower than in most DAB setups for flexABle, proving that the greatest optimization of FlexDDs are

[3] Note that timeouts are also included in solving times, i.e. for each instance that timed out the full cut-off value is added to total and 95% times.

those very moves. Furthermore, `abagraph` runs on a system called SICStus, built around a high-performance Prolog engine as opposed to the comparably naive tree-traversal algorithms `flexABle` implements. Nevertheless, strategy S_2 was still able to slightly outperform all `abagraph` strategies in terms of both: timeouts and time needed to solve all instances, further showing that other optimizations of FlexDDs also play a role in terms of the efficiency of `flexABle`. This role however should not be a major one, since `abagraph` outperformed `flexABle` with `DAB` for the only comparable pair of strategies, i.e. the Default `abagraph` strategy and S_{AG} strategy of `flexABle`.

By observing "mean" and "median" statistics we can conclude that an average instance should be solved with `flexABle` several times faster than when using `abagraph`. We can see that also in Figure 6.2, in which the performance of a selection of setups for each system is presented. We have chosen the best and the worst variants in terms of timeouts for each: `flexABle` with `DAB`, `flexABle` with `DABF` and `abagraph`. Additionally, we have included the only comparable setups i.e. those with strategy S_{AG} for `flexABle` and the Default strategy for `abagraph`. The effectiveness of `flexABle` for a single instance is visible from the fact that there is a straight vertical line for each `flexABle` setup in the plot indicating that a set of instances is solved in negligible amount of time compared to the maximal available cut-off time value. That set contains a vast majority (at least 5600 out of 6800, giving 82%) of instances. As can be seen in the figure, this does not hold for `abagraph`. Furthermore, the figure indicates that `DABF` setups are capable of solving virtually all problems and that the results of `abagraph`'s best strategy (1) performs comparably to `flexABle` with S_2.

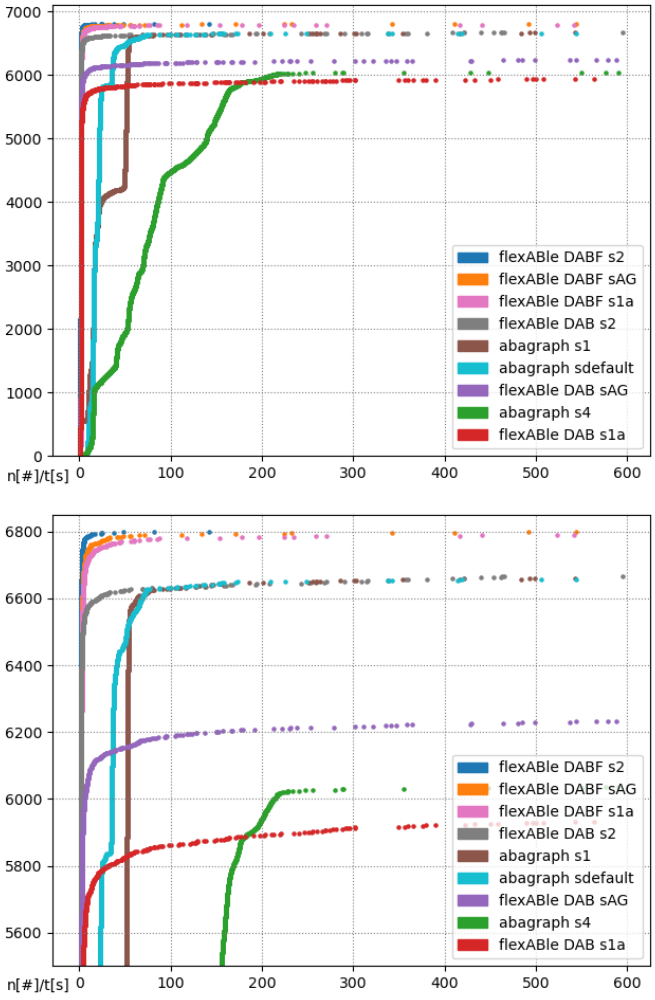


Figure 6.2: Inverted "cactus plots" comparing the performance of `flexABle` and `abagraph`. The lower plot is a "zoomed-in" version of the upper one, in which the performance of the best solvers and their setups is more visible. In both plots the instances are sorted in ascending order by the time required, where x-axes indicate the time (in seconds) and y-axes instance ordinals.

Approximate reasoning

We also performed an evaluation of approximate algorithms as described in Chapter 4 in both the "static" and "dynamic" variants. For the p parameter we chose the following range of values $range(p) = \{0.5, 0.6, 0.7, 0.8, 0.9\}$. We restricted combinations of p_{prop} and p_{opp} to the following 3 setups: (1) $p_{prop} \in range(p)$, $p_{opp} = 1$, (2) $p_{prop} = 1$, $p_{opp} \in range(p)$ and (3) $p_{prop} \in range(p), p_{prop} = p_{opp}$. This way we were able to substantially reduce the number of experiments and simultaneously investigate the properties of proponent side sampling, opponent side sampling and bi-directional sampling.

p_{opp}		p_{prop} 0.5	0.6	0.7	0.8	0.9	1
0.5	acc.	0.925					0.977
	spec.	0.975	–	–	–	–	0.97
	sens.	0.733					1.0
0.6	acc.		0.943				0.982
	spec.	–	0.981	–	–	–	0.978
	sens.		0.796				1.0
0.7	acc.			0.959			0.988
	spec.	–	–	0.986	–	–	0.984
	sens.			0.854			1.0
0.8	acc.				0.973		0.992
	spec.	–	–	–	0.991	–	0.99
	sens.				0.907		1.0
0.9	acc.					0.987	0.996
	spec.	–	–	–	–	0.996	0.995
	sens.					0.955	1.0
1	acc.	0.939	0.954	0.967	0.98	0.99	
	spec.	1.0	1.0	1.0	1.0	1.0	–
	sens.	0.702	0.773	0.84	0.901	0.953	

(a) "Static" sampling

p_{opp}		p_{prop} 0.5	0.6	0.7	0.8	0.9	1
0.5	acc.	0.926					0.976
	spec.	0.974	–	–	–	–	0.97
	sens.	0.741					1.0
0.6	acc.		0.944				0.982
	spec.	—	0.98	–	–	–	0.978
	sens.		0.804				1.0
0.7	acc.			0.961			0.987
	spec.	–	–	0.986	–	–	0.984
	sens.			0.865			1.0
0.8	acc.				0.975		0.992
	spec.	–	–	–	0.991	–	0.99
	sens.				0.918		1.0
0.9	acc.					0.988	0.996
	spec.	–	–	–	–	0.995	0.995
	sens.					0.962	1.0
1	acc.	0.945	0.959	0.971	0.983	0.992	
	spec.	1.0	1.0	1.0	1.0	1.0	–
	sens.	0.732	0.802	0.862	0.916	0.962	

(b) "Dynamic" sampling

Table 6.6: Results regarding accuracy, specificity and sensitivity of both "static" and "dynamic" sampling for various values of the p parameter. The greyed-out fields indicate combinations for which we did not perform experiments.

Table 6.6 presents the results we obtained when evaluating the approximate methods. In order to obtain these we performed experiments for each pair (p_{prop}, p_{opp}) in accordance to the above-mentioned possible combinations of p-values and the dispute derivation variant setups from Table 6.3 (thus giving us new approximate dispute derivation setups). Then we summed up the number of all true positive-, true negative-, false positive- and false negative- answers and calculated accuracy, specificity and sensitivity.

Additionally, note that indeed some guarantees are ensured, i.e. guarantee of true positive answers when $p_{opp} = 1$ or that of true negative answers when $p_{prop} = 1$. This is reflected by the 1.0 results for specificity and sensitivity for setups involving these p-values. These kinds of certainties can potentially be useful in situations when one needs to be sure that an obtained result is correct, e.g. to confirm or rule out a treatment method in medical decision-making support.

Furthermore, one can observe that the results are very similar for both sampling types, hence accuracy (and likewise sensitivity and specificity) of sampling type should not be a factor when choosing one of the sampling types over another. The choice could rather be based upon performance of sampling types, these being presented in Table 6.7 and Table 6.8 for the combinations of (p_{prop}, p_{opp}) involving the value 0.7. We have chosen this value to generally represent the behaviour of approximate reasoning, because, while still being relatively high to use in a real-life setting (hoping to obtain results of reasonable accuracy), it was low enough to indicate the tendencies of various setups of approximations. Nevertheless, the tendencies were sustained for different values of p, as can be seen in the appendix.

First let us focus on the "static" sampling method. Table 6.7a and Table 6.7c indicate that restricting the set of rules available for the proponent, when non-conservative forward moves are in use (setups with DC and DS), results in a decrease of efficiency of approximate reasoning. Such reasoning suffers from similar problems as strategies S_{1b} and S_{2b} as seen in Table 6.3, which are

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-86) 107	(-84) 95	(+1) 1	(-2) 0	(-7) 6	(-1) 12	(+29) 65	(+23) 62
	total time [h]	(-11.91) 26.48	(-11.30) 24.40	(+0.65) 4.38	(+0.03) 4.08	(+0.75) 8.12	(-0.09) 7.75	(+6.29) 18.13	(+5.33) 17.89
S_{1a}	#timeout.	(-278) 590	(-85) 284	(-7) 2	(-8) 3	(-6) 11	(-4) 18	(+42) 124	(+56) 132
	total time [h]	(-41.0) 112.97	(-10.29) 58.21	(-1.56) 4.81	(-0.77) 5.55	(+0.36) 9.65	(+1.80) 11.05	(+8.44) 29.68	(+10.96) 31.48
S_{1b}	#timeout.	(-70) 123	(-75) 104	(-85) 110	(-69) 111	(-56) 216	(-48) 188	(-5) 295	(+12) 269
	total time [h]	(-9.92) 28.32	(-10.80) 24.99	(-11.66) 26.89	(-7.90) 27.79	(-5.87) 46.36	(-4.28) 41.05	(+2.80) 59.08	(+4.80) 53.64
S_2	#timeout.	(-74) 59	(-63) 52	(+0) 0	(+0) 0	(+89) 269	(+78) 338	(+96) 263	(+102) 342
	total time [h]	(-11.61) 17.03	(-10.83) 15.25	(+0.26) 3.32	(+0.35) 3.47	(+15.39) 56.21	(+13.73) 70.14	(+17.40) 56.52	(+17.85) 70.89
S_{2a}	#timeout.	(-273) 487	(-112) 225	(+0) 1	(-3) 1	(+101) 282	(+83) 345	(+84) 257	(+75) 327
	total time [h]	(-43.28) 93.15	(-14.05) 47.93	(-0.19) 4.27	(-1.16) 3.72	(+15.84) 56.94	(+13.47) 69.95	(+14.24) 54.04	(+12.83) 67.29
S_{2b}	#timeout.	(-62) 71	(-74) 41	(-74) 59	(-61) 54	(+49) 465	(-96) 742	(+46) 462	(-78) 722
	total time [h]	(-9.58) 18.97	(-12.54) 13.62	(-12.65) 15.96	(-10.89) 14.95	(+10.27) 91.34	(-13.31) 153.47	(+8.92) 89.40	(-13.16) 146.66
S_{AG}	#timeout.	(-182) 384	(-89) 242	(+0) 1	(-2) 0	(-2) 9	(-1) 14	(+42) 96	(+32) 91
	total time [h]	(-26.49) 75.56	(-10.56) 51.70	(-0.24) 4.27	(-0.55) 4.10	(+0.55) 8.09	(+0.95) 8.44	(+7.28) 22.44	(+5.93) 21.26

(a) $p_{prop} = 0.7, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+8) 201	(+4) 183	(+0) 0	(-2) 0	(-5) 8	(+0) 13	(-4) 32	(-3) 36
	total time [h]	(+1.10) 39.49	(+1.48) 37.18	(+1.09) 4.82	(-0.22) 3.83	(-0.97) 6.40	(-0.47) 7.37	(-0.22) 11.62	(+0.01) 12.57
S_{1a}	#timeout.	(-43) 825	(-46) 323	(+0) 9	(-4) 7	(+5) 22	(-3) 19	(-13) 69	(-16) 60
	total time [h]	(-6.89) 147.08	(-7.76) 60.74	(+0.45) 6.82	(-0.45) 5.87	(+1.36) 10.65	(-0.46) 8.79	(-1.75) 19.49	(-2.71) 17.81
S_{1b}	#timeout.	(+8) 201	(-5) 174	(-1) 194	(+4) 184	(+4) 276	(-10) 226	(-21) 279	(-9) 248
	total time [h]	(+2.82) 41.06	(-0.49) 35.30	(+1.01) 39.56	(+1.18) 36.87	(+0.99) 53.22	(-1.82) 43.51	(-2.92) 53.36	(-1.38) 47.46
S_2	#timeout.	(+17) 150	(+15) 130	(+0) 0	(+0) 0	(-26) 154	(-12) 248	(-10) 157	(-9) 231
	total time [h]	(+2.12) 30.76	(+1.92) 28.0	(+0.37) 3.43	(+0.22) 3.34	(-3.34) 37.48	(-1.98) 54.43	(-2.43) 36.69	(-1.31) 51.73
S_{2a}	#timeout.	(-47) 713	(-40) 297	(+0) 1	(-1) 3	(-22) 159	(-15) 247	(-20) 153	(-13) 239
	total time [h]	(-7.81) 128.62	(-4.64) 57.34	(-0.10) 4.36	(-0.26) 4.62	(-4.50) 36.60	(-1.79) 54.69	(-2.57) 37.23	(-1.15) 53.31
S_{2b}	#timeout.	(+14) 147	(+13) 128	(+1) 134	(+14) 129	(+23) 439	(+67) 905	(+25) 441	(+62) 862
	total time [h]	(+2.13) 30.68	(+1.55) 27.71	(+2.35) 30.96	(+1.67) 27.51	(+4.24) 85.31	(+11.35) 178.13	(+3.48) 83.96	(+11.19) 171.01
S_{AG}	#timeout.	(-65) 501	(-53) 278	(+0) 1	(-1) 1	(+2) 13	(-4) 11	(-19) 35	(-21) 38
	total time [h]	(-11.06) 90.99	(-9.14) 53.12	(+2.09) 6.60	(-0.12) 4.53	(-0.36) 7.18	(-0.47) 7.02	(-2.28) 12.88	(-3.24) 12.09

(b) $p_{prop} = 1, p_{opp} = 0.7$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-67) 126	(-68) 111	(+0) 0	(-2) 0	(+7) 20	(+7) 20	(+20) 56	(+17) 56
	total time [h]	(-7.32) 31.07	(-9.62) 26.08	(+0.20) 3.93	(-0.18) 3.87	(+2.17) 9.54	(+2.07) 9.91	(+3.45) 15.29	(+3.11) 15.67
S_{1a}	#timeout.	(-296) 572	(-112) 257	(-6) 3	(-9) 2	(-2) 15	(-1) 21	(+12) 94	(+7) 83
	total time [h]	(-46.05) 107.92	(-16.41) 52.09	(-0.93) 5.44	(-2.0) 4.32	(+0.18) 9.47	(+0.28) 9.53	(+2.47) 23.71	(+1.30) 21.82
S_{1b}	#timeout.	(-52) 141	(-72) 107	(-61) 134	(-61) 119	(-34) 238	(-37) 199	(+0) 300	(+0) 257
	total time [h]	(-6.41) 31.83	(-9.74) 26.05	(-8.53) 30.02	(-7.81) 27.88	(-2.85) 49.38	(-4.14) 41.19	(+1.95) 58.23	(+2.38) 51.22
S_2	#timeout.	(-41) 92	(-53) 62	(+0) 0	(+0) 0	(+78) 258	(+77) 337	(+71) 238	(+83) 323
	total time [h]	(-6.98) 21.66	(-9.54) 16.54	(+0.33) 3.39	(+0.44) 3.56	(+13.02) 53.84	(+11.72) 68.13	(+12.58) 51.70	(+12.50) 65.54
S_{2a}	#timeout.	(-291) 469	(-137) 200	(+2) 3	(-3) 1	(+77) 258	(+83) 345	(+75) 248	(+81) 333
	total time [h]	(-47.34) 89.09	(-21.16) 40.82	(+0.31) 4.77	(-0.67) 4.21	(+13.90) 55.0	(+12.30) 68.78	(+11.90) 51.70	(+12.99) 67.45
S_{2b}	#timeout.	(-58) 75	(-54) 61	(-50) 83	(-32) 83	(+68) 484	(-40) 798	(+64) 480	(-44) 756
	total time [h]	(-9.63) 18.92	(-9.01) 17.15	(-8.82) 19.79	(-5.67) 20.17	(+11.99) 93.06	(-5.38) 161.40	(+13.26) 93.74	(-7.52) 152.30
S_{AG}	#timeout.	(-223) 343	(-115) 216	(-1) 0	(-2) 0	(+7) 18	(-1) 14	(+21) 75	(+7) 66
	total time [h]	(-34.14) 67.91	(-17.74) 44.52	(-0.64) 3.87	(-0.52) 4.13	(+1.04) 8.58	(+0.56) 8.05	(+2.87) 18.03	(+1.20) 16.53

(c) $p_{prop} = 0.7, p_{opp} = 0.7$

Table 6.7: "Static" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.7. Rows with green, red and yellow background indicate an improvement, deterioration or no change with respect to the exact (not-approximate) method in the same setup, respectively. A value in parentheses next to the result denotes the difference between the exact and approximate method.

inefficient due to the use of non-conservative forward moves when simultaneously not taking advantage of forward inferences using rules (PF1 moves), when the set of rules is limited for the proponent. At the same time note that the non-deterministic choice of a rule is one of the main causes of decreased efficiency among DAB setups which result in the number of timeouts and amount of time required significantly reduced when $p_{prop} = 0.7$.

The behaviour of the approximate method with $p_{prop} = 1$ and $p_{opp} = 0.7$ from Table 6.7b is not very consistent, hindering us from drawing any meaningful conclusions. Note that for some setups the efficiency is improved, whereas for others it is worse, but the influence does not seem to be of major significance in any direction. This might imply that opponent moves involving rules do not contribute as much to the procedures' efficiency as those of the proponent due to the lack of non-determinism in the opponent's choices. At the same time, if some opponent's arguments need not be generated, then they do not have to be counter-attacked by the proponent. On the other hand the solving time of some negative instances (those, for which the answer is negative) might increase if the opponent is not able to quickly win due to having less rules at his disposal.

Sampling from the opponent side only leads to similar results for the "dynamic" variant (see Table 6.8b). However, sampling from the proponent- or both sides produces a more congruent outcome as presented in Table 6.8a and Table 6.8c. Note that the results obtained for setups with DAB as well as those with DC and DS for proponent- and bi-directional sampling are improved. A possible explanation could be that this kind of sampling reduces branching of proponent moves utilising rules (PB1 and PB2 moves), because when more than one rule is applicable, there is a chance that some of them will be ruled out. Rules enabling conservative forward moves can also be filtered out at some points of the search, but there is a chance that they will be available for use at later stages. This way "dynamic" sampling combines the best of both worlds: reducing number of options for non-deterministic choices, while simultaneously taking advantage of conservative forward moves.

A slight improvement can also be observed for most of the setups with DABF. However, in no configuration was sampling able to improve the most efficient exact setup, i.e. (DABF, S_2, DFS). This suggests that these kind of approximations might be more useful when one decides on the less optimal setups (i.e. those with advancement types other than DABF).

Note that, the only 3 setups, in which the "static" sampling is more efficient than the "dynamic" sampling are those with the DAB advancement type and strategies S_{1a}, S_{2a} and S_{AG}. The reason behind such outcome might be similar to why in general those strategies perform poorly (see 6.3), which is the choice of rules introducing many new statements. The "dynamic" sampling's nature might work to its disadvantage for these strategies, as such rules, even if removed at one step, can again become problematic in the longer run. On the other hand, in "static" sampling, once a problematic rule gets removed, it does not participate in the dispute derivation until its end.

Figure 6.3 presents the results of "static" and "dynamic" bi-directional sampling for all considered values of p, as well as the "exact" reasoning for the worst- and best performing strategies for advancement types DAB and DABF and the search type DFS. As indicated by the data in the Tables 6.7 and 6.8, there is no improvement when strategies employ the PF1 moves (in setups employing DABF). However, significant improvement can be seen in setups without the conservative forward moves (employing DAB). As argued before, the inefficient S_{1a} strategy performs much better with "static" sampling applied. On the other hand, the latter, "dynamic" form of sampling provides greater efficiency-improvements when using the best-performing strategy S_2.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-114) 79	(-117) 62	(+0) 0	(-2) 0	(-9) 4	(-8) 5	(-13) 23	(-17) 22
	total time [h]	(-16.88) 21.51	(-16.91) 18.79	(+0.21) 3.94	(-0.09) 3.96	(-2.10) 5.27	(-2.52) 5.32	(-3.35) 8.49	(-3.20) 9.36
S_{1a}	#timeout.	(-137) 731	(-8) 361	(-9) 0	(-11) 0	(-9) 8	(-13) 9	(-33) 49	(-20) 56
	total time [h]	(-20.60) 133.37	(+0.25) 68.75	(-1.99) 4.38	(-2.41) 3.91	(-3.04) 6.25	(-3.47) 5.78	(-5.40) 15.84	(-4.48) 16.04
S_{1b}	#timeout.	(-119) 74	(-121) 58	(-129) 66	(-122) 58	(-153) 119	(-134) 102	(-33) 267	(-45) 212
	total time [h]	(-16.57) 21.67	(-17.85) 17.94	(-18.86) 19.69	(-18.04) 17.65	(-23.96) 28.27	(-20.53) 24.80	(-3.24) 53.04	(-5.13) 43.71
S_2	#timeout.	(-110) 23	(-100) 15	(+0) 0	(+0) 0	(-48) 132	(-61) 199	(-47) 120	(-56) 184
	total time [h]	(-17.42) 11.22	(-18.16) 7.92	(+1.27) 4.33	(+0.26) 3.38	(-9.09) 31.73	(-11.69) 44.72	(-9.48) 29.64	(-10.02) 43.02
S_{2a}	#timeout.	(-141) 619	(-20) 317	(-1) 0	(-4) 0	(-52) 129	(-69) 193	(-49) 124	(-54) 198
	total time [h]	(-22.40) 114.03	(-0.63) 61.35	(-0.92) 3.54	(-1.30) 3.58	(-10.28) 30.82	(-13.12) 43.36	(-8.76) 31.04	(-8.45) 46.01
S_{2b}	#timeout.	(-114) 19	(-97) 18	(-105) 28	(-94) 21	(-32) 384	(-174) 664	(-39) 377	(-175) 625
	total time [h]	(-19.67) 8.88	(-18.27) 7.89	(-17.57) 11.04	(-17.75) 8.09	(-4.13) 76.94	(-26.70) 140.08	(-3.64) 76.84	(-30.28) 129.54
S_{AG}	#timeout.	(-118) 448	(-50) 281	(-1) 0	(-2) 0	(-3) 8	(-5) 10	(-8) 46	(-13) 46
	total time [h]	(-17.69) 84.36	(-5.96) 56.30	(-0.72) 3.79	(-1.21) 3.44	(-0.74) 6.80	(-1.30) 6.19	(-2.10) 13.06	(-2.22) 13.11

(a) $p_{prop} = 0.7, p_{opp} = 1$

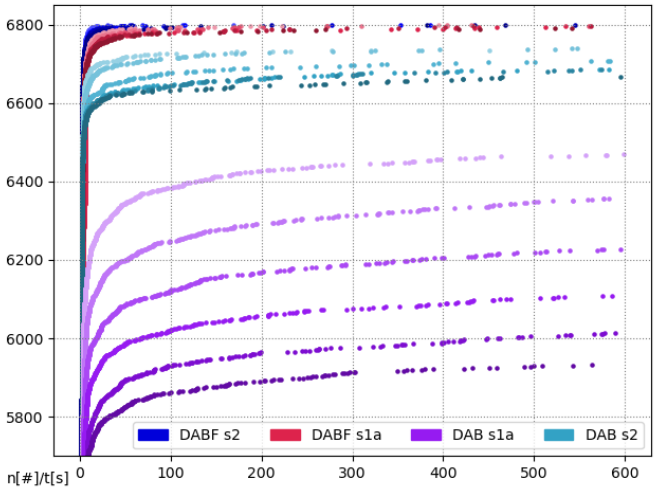
Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+36) 229	(+30) 209	(+0) 0	(-1) 1	(+1) 14	(-2) 11	(-1) 35	(-1) 38
	total time [h]	(+6.37) 44.76	(+5.35) 41.05	(+0.27) 4.0	(+1.0) 5.05	(+0.23) 7.60	(-1.21) 6.63	(+1.01) 12.85	(+0.21) 12.77
S_{1a}	#timeout.	(-4) 864	(-5) 364	(+0) 9	(-1) 10	(-2) 15	(-4) 18	(-6) 76	(-4) 72
	total time [h]	(+0.23) 154.20	(-0.38) 68.12	(-0.08) 6.29	(-0.10) 6.22	(-0.49) 8.80	(-0.41) 8.84	(+0.64) 21.88	(-0.84) 19.68
S_{1b}	#timeout.	(+38) 231	(+23) 202	(+42) 237	(+35) 215	(+16) 288	(+18) 254	(+34) 334	(+26) 283
	total time [h]	(+6.42) 44.66	(+4.27) 40.06	(+8.0) 46.55	(+6.73) 42.42	(+2.62) 54.85	(+4.26) 49.59	(+7.07) 63.35	(+4.74) 53.58
S_2	#timeout.	(+9) 142	(+13) 128	(+0) 0	(+0) 0	(-59) 121	(-86) 174	(-43) 124	(-84) 156
	total time [h]	(+1.94) 30.58	(+3.92) 30.0	(+0.48) 3.54	(+0.25) 3.37	(-9.95) 30.87	(-16.81) 39.60	(-8.27) 30.85	(-15.37) 37.67
S_{2a}	#timeout.	(-16) 744	(-48) 289	(+0) 1	(+1) 5	(-54) 127	(-92) 170	(-53) 120	(-94) 158
	total time [h]	(-2.08) 134.35	(-7.20) 54.78	(+0.92) 5.38	(+0.06) 4.94	(-8.95) 32.15	(-17.27) 39.21	(-9.06) 30.74	(-17.15) 37.31
S_{2b}	#timeout.	(+17) 150	(+11) 126	(+8) 141	(+22) 137	(+25) 441	(+18) 856	(+36) 452	(+13) 813
	total time [h]	(+2.65) 31.20	(+2.05) 28.21	(+2.47) 31.08	(+3.99) 29.83	(+1.75) 82.82	(+1.55) 168.33	(+4.60) 85.08	(-0.14) 159.68
S_{AG}	#timeout.	(-7) 559	(-5) 326	(+0) 1	(+1) 3	(-2) 9	(-3) 12	(-8) 46	(-10) 49
	total time [h]	(-1.16) 100.89	(-0.67) 61.59	(+0.21) 4.72	(+0.85) 5.50	(-0.90) 6.64	(+0.15) 7.64	(-1.12) 14.04	(-0.93) 14.40

(b) $p_{prop} = 1, p_{opp} = 0.7$

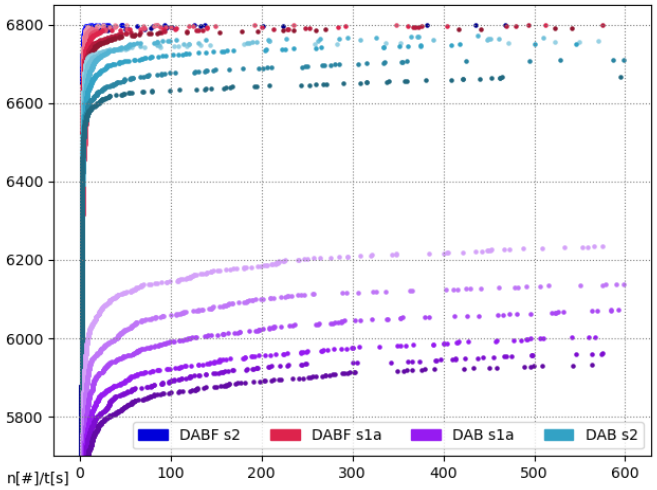
Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-117) 76	(-100) 79	(+0) 0	(-2) 0	(-12) 1	(-11) 2	(-14) 22	(-16) 23
	total time [h]	(-16.91) 21.48	(-12.12) 23.58	(-0.48) 3.25	(-0.47) 3.58	(-3.11) 4.26	(-3.17) 4.67	(-2.64) 9.20	(-2.20) 10.36
S_{1a}	#timeout.	(-141) 727	(+3) 372	(-9) 0	(-11) 0	(-11) 6	(-14) 8	(-40) 42	(-27) 49
	total time [h]	(-22.02) 131.95	(+0.97) 69.47	(-1.76) 4.61	(-2.31) 4.01	(-3.80) 5.49	(-2.39) 6.86	(-7.62) 13.62	(-6.10) 14.42
S_{1b}	#timeout.	(-96) 97	(-91) 88	(-110) 85	(-106) 74	(-131) 141	(-115) 121	(-9) 291	(-13) 244
	total time [h]	(-13.83) 24.41	(-13.14) 22.65	(-15.20) 23.35	(-15.19) 20.50	(-20.50) 31.73	(-16.82) 28.51	(+0.55) 56.83	(+0.36) 49.20
S_2	#timeout.	(-104) 29	(-97) 18	(+0) 0	(+0) 0	(-115) 65	(-146) 114	(-95) 72	(-146) 94
	total time [h]	(-18.70) 9.94	(-18.24) 7.84	(+0.63) 3.69	(+0.36) 3.48	(-21.0) 19.82	(-29.34) 27.07	(-18.43) 20.69	(-27.74) 25.30
S_{2a}	#timeout.	(-160) 600	(-75) 262	(-1) 0	(-4) 0	(-108) 73	(-154) 108	(-102) 71	(-145) 107
	total time [h]	(-25.51) 110.92	(-9.67) 52.31	(-1.05) 3.41	(-1.48) 3.40	(-19.79) 21.31	(-29.17) 27.31	(-19.54) 20.26	(-27.0) 27.46
S_{2b}	#timeout.	(-100) 33	(-99) 16	(-103) 30	(-92) 23	(-38) 378	(-164) 674	(-35) 381	(-165) 635
	total time [h]	(-16.60) 11.95	(-18.02) 8.14	(-17.68) 10.93	(-16.80) 9.04	(-4.33) 76.74	(-29.80) 136.98	(-3.65) 76.83	(-30.24) 129.58
S_{AG}	#timeout.	(-146) 420	(-55) 276	(-1) 0	(-2) 0	(-6) 5	(-9) 6	(-19) 35	(-24) 35
	total time [h]	(-22.23) 79.82	(-7.0) 55.26	(-1.16) 3.35	(-0.89) 3.76	(-2.17) 5.37	(-1.48) 6.01	(-4.34) 10.82	(-4.76) 10.57

(c) $p_{prop} = 0.7, p_{opp} = 0.7$

Table 6.8: "Dynamic" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.7. As before, rows with green, red and yellow background indicate an improvement, deterioration or no change with respect to the exact (not-approximate) method in the same setup, respectively, and a value in parentheses next to the result denotes the difference between the exact and approximate method.



(a) "Static" sampling.



(b) "Dynamic" sampling.

Figure 6.3: Inverted "cactus plots" presenting the results of "static" and "dynamic" sampling for approximate reasoning using $p_{prop} = p_{opp} = p$, $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ and exact reasoning ($p = 1.0$) for the best (S_2) and worst (S_{1a}) performing strategies for DAB and DABF and DFS. Results are grouped by their advancement type and strategy into similarly coloured groups, with the darkest plot indicating the exact reasoning ($p = 1.0$). The brighter the colour of a plot, the smaller the value of the p parameter, with the brightest colour indicating $p = 0.5$. In both plots the instances are sorted in ascending order by the time required, where x-axes indicate the time (in seconds) and y-axes instance ordinals.

6.2.2 Complete & Stable Semantics

We further investigated the performance of search strategies for semantics beyond admissible supported by FlexDDs, i.e. complete and stable. To this end we employed the DC + TC dispute variants

for the first and DS + TS for the latter case. We used the same set of benchmarks (framework-statement pairs) as in the previous experiments, but now queried flexABle about acceptance of the goal w.r.t. the complete and stable semantics. Because those problems are more complicated the timeout value was doubled to 20 minutes (1200 seconds).

Knowing that each complete extension must also be an admissible one, we hypothesized that starting with the task of finding an admissible extension first and then trying to extend it to a complete one might produce better results, than when looking for a complete extension from scratch. Hence in our experiments we compare the performance of two approaches: first, one that restricts the moves to those of DABF until TA is satisfied and then allows for moves of DC trying to find a state satisfying TC, and secondly the approach allowing for DC moves right from the very beginning. Since each stable extension must necessarily be admissible as well, we also used that approach for the stable semantics. Results for both semantics are presented in Table 6.9.

Note that first of all that starting with DABF + TA provides better results than when directly starting with the advancement type and termination criteria of the desired semantics. Furthermore, strategies not utilizing conservative forward moves (S_{1b} and S_{2b}) result in very poor performance.

The patient strategy (S_1) performs much better for the complete semantics than the eager one (S_2). This can be explained, because, as we have seen before in Table 6.3, eagerly guessing assumptions with non-conservative moves leads to inefficiency.

As we said before, each stable and complete extension must be admissible. Moreover each admissible one is a subset of some complete extension. However, the latter property does not hold for the stable semantics. Therefore, even though the approach with first searching for an admissible extension performs better for the stable semantics than the direct approach, which admissible extension is found before starting to extend it into a stable one is crucial. Remember that by TS each assumption must either be a defence or a culprit. Hence, the proponent, when searching for an admissible extension, should attack and commit to as many assumptions as possible which is how strategy S_2 works. Note that interestingly S_{2a} has slightly less timeouts than S_2 for the stable semantics when starting with finding an admissible extension first, which is likely to be happening due to S_{2a} choosing rules which introduce more new statements to $\mathbb{P}$, resulting also in more new culprits and defences.

Strategy S_{AG} performs moderately worse than S_1 for both semantics. Note that this strategy chooses rules based on the termination criteria, so it is tailored for each semantics individually. However, this approach does not lead to improvement, possibly because of the additional computations needed for each available rule.

6.2.3 Grounded & Preferred Semantics

As a final set of experiments, we evaluated flexABle for the grounded and preferred semantics for the same set of benchmarks as in previous experiments. To this end we used the algorithms described in Chapter 4 for those semantics, which use FlexDDs as a sub-routine to check if an assumption set constitutes a complete extension. For the latter we used the same configurations as those from Table 6.9. The timeout value remained the same as in the previous set of experiments, namely 1200s. However, the results clearly indicated the inferiority of this approach with regard to abagraph's capabilities. Interested reader is referred to the appendix for detailed results.

Str.		**complete** semantics				**stable** semantics			
		Start w/ DABF+TA		Start w/ DC+ TC		Start w/ DABF+TA		Start w/ DS+ TS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	20	34	73	101	168	163	526	549
	total time [h]	18.70	20.67	51.84	61.85	71.28	72.12	230.24	237.71
	95% time [h]	3.63	3.38	8.50	8.87	5.32	5.34	116.89	124.34
	min [s]	1.23	1.24	1.37	1.37	1.50	1.49	1.29	1.23
	median [s]	1.71	1.56	3.18	3.17	1.94	2.00	7.33	6.35
	mean [s]	6.39	4.96	14.72	15.14	8.29	9.64	31.49	31.48
	max [s]	1043.18	1027.22	1150.74	1191.46	1067.98	1191.26	1147.57	1159.78
S_{1a}	#timeout.	26	45	87	120	171	174	533	557
	total time [h]	21.85	25.62	58.06	69.32	74.76	75.38	231.30	241.24
	95% time [h]	3.30	3.63	8.87	10.16	5.76	5.65	117.95	127.87
	min [s]	1.15	1.23	1.29	1.39	1.47	1.50	1.32	1.45
	median [s]	1.46	1.60	3.22	3.60	2.00	1.96	6.75	6.67
	mean [s]	7.00	5.66	15.58	15.79	9.64	9.43	30.80	32.02
	max [s]	1082.85	1181.78	1179.61	1140.69	1116.41	1138.50	1195.62	1198.85
S_{1b}	#timeout.	275	682	663	1155	892	869	–	–
	total time [h]	110.71	246.23	263.77	420.20	330.54	318.26		
	95% time [h]	8.49	132.85	150.43	306.81	217.19	204.88		
	min [s]	1.14	1.13	1.37	1.34	1.22	1.21	–	–
	median [s]	1.61	1.49	4.59	4.73	1.68	1.66		
	mean [s]	10.50	11.07	25.08	22.34	20.21	17.29		
	max [s]	1193.69	1178.69	1187.64	1183.59	1195.85	1191.52		
S_2	#timeout.	212	230	887	1004	106	137	377	442
	total time [h]	93.49	99.24	503.27	551.97	54.02	60.29	178.07	195.34
	95% time [h]	7.03	7.50	389.93	438.60	4.68	4.76	64.73	81.97
	min [s]	1.13	1.18	1.28	1.24	1.53	1.54	1.63	1.28
	median [s]	1.44	1.53	16.29	14.95	1.98	2.02	5.70	5.46
	mean [s]	12.47	12.36	126.38	134.92	10.05	7.89	29.37	27.16
	max [s]	1103.37	1165.62	1194.59	1198.39	1188.77	1192.53	1195.66	1161.73
S_{2a}	#timeout.	211	233	895	1022	104	133	382	443
	total time [h]	96.35	102.98	516.70	559.69	55.00	61.52	178.67	195.20
	95% time [h]	8.12	9.65	403.36	446.33	4.67	4.96	65.32	81.84
	min [s]	1.17	1.16	1.32	1.27	1.48	1.51	1.67	1.33
	median [s]	1.58	1.81	16.92	16.04	1.94	2.04	5.77	5.53
	mean [s]	14.21	13.87	133.11	136.41	10.93	9.28	28.79	26.90
	max [s]	1198.18	1156.81	1194.80	1199.87	1192.13	1197.52	1199.80	1199.09
S_{2b}	#timeout.	853	931	–	–	891	896	–	–
	total time [h]	328.05	350.38			337.54	332.56		
	95% time [h]	214.70	236.99			224.19	219.18		
	min [s]	1.15	1.19	–	–	1.22	1.13	–	–
	median [s]	1.56	1.60			1.73	1.59		
	mean [s]	26.44	24.50			24.68	20.60		
	max [s]	1190.36	1147.35			1195.93	1194.30		
S_{AG}	#timeout.	21	34	93	125	175	176	557	580
	total time [h]	19.97	21.69	62.92	71.38	76.14	76.54	250.66	259.80
	95% time [h]	3.71	3.39	10.69	10.35	6.01	5.92	137.31	146.43
	min [s]	1.22	1.20	1.35	1.29	1.48	1.45	1.29	1.30
	median [s]	1.67	1.50	3.72	3.42	1.94	1.97	7.40	7.34
	mean [s]	6.89	5.51	17.13	16.02	9.67	9.70	37.46	38.44
	max [s]	1123.00	1194.65	1187.25	1196.56	1148.11	1143.22	1161.28	1196.50

Table 6.9: Results for the complete and stable semantics using `flexABle`. As before, green and red cell backgrounds indicate best and worst results obtained globally. Greyed-out cells represent setups which exceeded the total available running time.

7 Conclusions

We were able to formally define automatic-search strategies for RlFlexDDs, based on the ordering of the move types as well as functions determining the order in which rules should be considered. We have carefully chosen a few of such definable strategies for empirical evaluation and identified the possible influence that some features of the strategies may have on the performance of the search.

We mostly focused on credulous acceptance w.r.t. admissible semantics, for which we conducted experiments on the largest scale and also analysed the results in the greatest depth. Most importantly, we have established the crucial influence of conservative forward moves, which allow for bottom-up moves from a set of justified premises. We have seen that mostly due to this kind of moves the procedure became very efficient, significantly overcoming the performance results of the previous state-of-the art ABA DD system. Moreover, we discovered the impact of other features, such as that of the strategy or the search type, and although its significance has been eclipsed by the conservative forward moves, they also played a role, particularly when forward moves are not used. We notice a decrease of efficiency when the non-conservative forward moves were used, suggesting that on average moves of this kind would not improve the performance. Finally, we incorporated two variants of approximate reasoning into the "credulous admissible acceptance" experiments, which showed that for some cases the "dynamic" version might be more effective than the "static" one, which actually can be even less effective than the "exact" (unapproximated) reasoning. The results on evaluation of directional sampling has confirmed, that we can be positive about the outcome even when using approximation, which could have potential applications in cases when a claim needs to be ruled in or ruled out respectively. Lastly, we checked the efficiency of FlexDDs for semantics beyond admissible: stable, complete, grounded and preferred. While the proposed algorithms and strategies for the last two semantics did not perform well, we have still obtained reasonable results for the first two.

For all of the most remarkable results, we provided hypotheses trying to explain how strategies' design choices affecting the search resulted in the obtained outcome. Additionally, we presented an implementation supporting automatic search of dispute derivations as defined within this work, which was used to perform the experiments. The implementation however provides much more than merely automatic search. It also offers aid in interactively searching for a successful dispute derivation including a rich graphical interface.

8 Future Work

There are a number of ways that one could continue the current work, e.g. regarding future experiments. First of all, the input benchmarks could be more diversified. Although, when deciding for the chosen benchmark set, we followed the methodology of evaluation from previous dispute derivation systems, we have already encountered an indication that this benchmark set might not be very representative (recall that rule sampling even for small p values produced results of high accuracy). Hence, the input frameworks should originate from more than a single source.

Nevertheless, the current benchmarks source, which is a Prolog-based generator implemented by Craven and Toni [7], can still be used in further evaluation. Following the experiments from [19] more complicated benchmarks could be gradually generated, until they become too complicated for the most efficient search setups (such as the strategy S_2 with `DABF`). This way the limitations of our system and FlexDDs could be more clearly found.

One could also aim at a different approach at selecting strategies for evaluation. In our case we chose strategies based on our intuition, to which an alternative could be using more formal methods. E.g. one could use some clustering algorithms to group similarly-behaving strategies together and then choose one representative from each group, hoping that this would make selected strategies more likely diverse. Alternatively, one could try to simply evaluate a larger number of strategies, or even all of them.

The implementation would certainly also benefit from some further modifications, e.g. the branching algorithms could be optimized to somehow identify parts of the dispute state which led to an unsuccessful state in the previously considered branches, in order to sooner recognize fruitless dispute states in the following search stages. Creating an actual graphical user interface (GUI) instead of the current command-line interface (CLI) would definitely improve the user's experience. Ideally, such a GUI system could e.g. represent the current dispute state in form of a directed graph of nodes, seen as "abstract" arguments. Then zooming in would reveal arguments' internal structure, enabling users to choose arguments' elements which e.g. should be backward extended or attacked.

Further advancements described above are regarding the implementation and evaluation parts of this work. There are however many new directions on the more abstract conceptual level in which this work could proceed. E.g. grounded- or preferred-based search algorithms could be defined, in which the proponent during the dispute derivation would either try to use as few as possible new assumptions in the former case, or as many as possible in the latter. Dispute derivations obtained this way would not be guaranteed to represent grounded or preferred extensions, but would be inspired by those semantics.

Even though in our experiments the non-conservative forward moves have not improved the efficiency (and in most cases even worsened it), we hypothesised that in some very specific cases it could be useful. These cases include situations when there is some "knowledge" about the

successful dispute state available "from the outside", i.e. it is known that some assumptions will be included in the fruitful dispute state. To this end, one could utilize different, more efficient, reduction-based ABA solvers, such as `aspforaba` [19] to first obtain the defences and then control the amount of the defences being fed to FlexDDs as the "knowledge from the outside", hence evaluating the use of non-conservative forward moves when further knowledge is available about the domain on which a dispute is carried out.

Finally, the FlexDDs (and consequently our system) could be modified to support non-flat ABA frameworks (where rules with assumption occurring in the heads is allowed). The behaviour of such procedures could then be further evaluated to also take into account the influence of a framework's flatness in the search of successful DDs.

Bibliography

1. K. Atkinson, P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G. R. Simari, M. Thimm, and S. Villata. "Towards Artificial Argumentation". *AI Mag.* 38:3, 2017, pp. 25–36.
2. P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre. *Handbook of Formal Argumentation*. College Publications, 2018. ISBN: 9781848902756.
3. P. Baroni, M. Caminada, and M. Giacomin. "Abstract Argumentation Frameworks and Their Semantics". In: *Handbook of Formal Argumentation*. Ed. by P. Baroni, D. Gabbay, and M. Giacomin. College Publications, 2018, pp. 159–236.
4. T. Bench-Capon and P. E. Dunne. "Argumentation in artificial intelligence". *Artificial Intelligence* 171:10, 2007. Argumentation in Artificial Intelligence, pp. 619–641. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2007.05.001. URL: https://www.sciencedirect.com/science/article/pii/S0004370207000793.
5. P. Besnard, A. Garcia, A. Hunter, S. Modgil, H. Prakken, G. Simari, and F. Toni. "Introduction to structured argumentation". *Argument & Computation* 5, 2014. DOI: 10.1080/19462166.2013.869764.
6. G. Brewka, S. Polberg, and S. Woltran. "Generalizations of Dung Frameworks and Their Role in Formal Argumentation". *IEEE Intell. Syst.* 29:1, 2014, pp. 30–38.
7. R. Craven and F. Toni. "Argument graphs and assumption-based argumentation". *Artificial Intelligence* 233, 2016, pp. 1–59. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2015.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0004370215001800.
8. R. Craven, F. Toni, C. Cadar, A. Hadad, and M. Williams. "Efficient Argumentation for Medical Decision-Making". *13th International Conference on the Principles of Knowledge Representation and Reasoning, KR 2012*, 2012.
9. R. Craven, F. Toni, and M. Williams. "Graph-Based Dispute Derivations in Assumption-Based Argumentation". In: *Theory and Applications of Formal Argumentation*. Ed. by E. Black, S. Modgil, and N. Oren. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 46–62. ISBN: 978-3-642-54373-9.
10. K. Cyras, X. Fan, C. Schulz, and F. Toni. "Assumption-Based Argumentation: Disputes, Explanations, Preferences". In: *Handbook of Formal Argumentation*. Ed. by P. Baroni, D. Gabbay, and M. Giacomin. College Publications, 2018, pp. 365–408.

11. M. Diller, S. A. Gaggl, and P. Gorczyca. "Flexible Dispute Derivations with Forward and Backward Arguments for Assumption-Based Argumentation". In: *Logic and Argumentation*. Ed. by P. Baroni, C. Benzmüller, and Y. N. Wáng. Springer International Publishing, Cham, 2021, pp. 147–168. ISBN: 978-3-030-89391-0.

12. P. Dung, R. Kowalski, and F. Toni. "Assumption-based argumentation". *Argumentation in AI*, 2009. cited By 7, pp. 25–44.

13. P. Dung, R. Kowalski, and F. Toni. "Dialectic proof procedures for assumption-based, admissible argumentation". *Artificial Intelligence* 170:2, 2006, pp. 114–159. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2005.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0004370205001256.

14. P. Dung, P. Mancarella, and F. Toni. "Computing ideal sceptical argumentation". *Artificial Intelligence* 171:10, 2007. Argumentation in Artificial Intelligence, pp. 642–674. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2007.05.003. URL: https://www.sciencedirect.com/science/article/pii/S000437020700080X.

15. P. M. Dung. "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games". *Artificial Intelligence* 77:2, 1995, pp. 321–357. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(94)00041-X. URL: https://www.sciencedirect.com/science/article/pii/000437029400041X.

16. D. Gabbay, M. Giacomin, and G. Simari. *Handbook of Formal Argumentation, Volume 2*. v. 2. College Publications, 2021. ISBN: 9781848903364.

17. D. Gaertner and F. Toni. "CaSAPI: A system for credulous and sceptical argumentation". *Proc. of ArgNMR*, 2009.

18. D. Gaertner and F. Toni. "Computing Arguments and Attacks in Assumption-Based Argumentation". *IEEE Intelligent Systems* 22:6, 2007, pp. 24–33. DOI: 10.1109/MIS.2007.105.

19. T. Lehtonen, J. P. Wallner, and M. Järvisalo. "Declarative Algorithms and Complexity Results for Assumption-Based Argumentation". *J. Artif. Int. Res.* 71, 2021, pp. 265–318. ISSN: 1076-9757. DOI: 10.1613/jair.1.12479. URL: https://doi.org/10.1613/jair.1.12479.

20. T. Lehtonen, J. P. Wallner, and M. Järvisalo. "From Structured to Abstract Argumentation: Assumption-Based Acceptance via AF Reasoning". In: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Ed. by A. Antonucci, L. Cholvy, and O. Papini. Springer International Publishing, Cham, 2017, pp. 57–68. ISBN: 978-3-319-61581-3.

21. T. Lehtonen, J. P. Wallner, and M. Järvisalo. "Reasoning over Assumption-Based Argumentation Frameworks via Direct Answer Set Programming Encodings". *Proceedings of the AAAI Conference on Artificial Intelligence* 33:01, 2019, pp. 2938–2945. DOI: 10.1609/aaai.v33i01.33012938. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4149.

22. S. Modgil and H. Prakken. "Abstract Rule-Based Argumentation". In: *Handbook of Formal Argumentation*. Ed. by P. Baroni, D. Gabbay, and M. Giacomin. College Publications, 2018, pp. 287–364.

23. S. Modgil and H. Prakken. "The ASPIC + framework for structured argumentation: A tutorial". *Argument & Computation* 5, 2014. DOI: 10.1080/19462166.2013.869766.

24. G. Simari and I. Rahwan. *Argumentation in Artificial Intelligence*. 2009. ISBN: 978-0-387-98196-3. DOI: 10.1007/978-0-387-98197-0.

25. C. Sun. "Approximate Inference for Assumption-based Argumentation in AI". MA thesis. Koblenz: Universität Koblenz - Landau, 2021.

26. M. Thimm and T. Rienstra. "Approximate Reasoning with ASPIC+ by Argument Sampling". In: 2020.

27. F. Toni. "A generalised framework for dispute derivations in assumption-based argumentation". *Artificial Intelligence* 195, 2013, pp. 1–43. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2012.09.010. URL: https://www.sciencedirect.com/science/article/pii/S0004370212001233.

28. F. Toni. "A tutorial on assumption-based argumentation". *Argument & Computation* 5, 2014. DOI: 10.1080/19462166.2013.869878.

Appendix

Non-trivial Instances Results

	abagraph strategy				
	1	2	3	4	5
#timeout.	93	191	93	400	97
total time [h]	24.89	43.87	24.84	82.38	23.08
95% time [h]	15.22	34.18	15.16	72.67	13.40
min [s]	0.76	0.67	0.76	0.65	0.66
median [s]	17.04	21.95	18.10	62.99	17.09
mean [s]	31.99	45.09	31.80	74.90	23.60
max [s]	542.47	562.49	461.95	589.76	418.82

Table 1: Results for admissible semantics using `abagraph` and the set of input benchmarks restricted to the non-trivial ones.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	118	108	0	2	12	12	32	34
	total time [h]	22.23	20.17	1.32	1.54	4.38	4.51	7.92	8.34
	95% time [h]	12.56	10.50	0.58	0.60	0.80	0.84	0.94	0.98
	min [s]	1.20	1.26	1.22	1.28	1.28	1.41	1.31	1.28
	median [s]	1.71	1.76	1.85	1.90	2.19	2.41	1.91	2.03
	mean [s]	8.94	7.47	4.13	3.80	7.53	7.94	8.34	8.63
	max [s]	595.72	510.72	468.25	497.25	501.04	461.19	488.27	554.67
S_{1a}	#timeout.	394	285	8	9	14	19	71	65
	total time [h]	69.33	51.16	2.94	2.81	5.82	5.54	16.20	15.15
	95% time [h]	59.66	41.49	0.77	0.76	1.08	0.98	6.53	5.48
	min [s]	1.27	1.20	1.36	1.36	1.29	1.31	1.21	1.38
	median [s]	2.23	1.97	2.15	2.14	2.30	2.38	2.02	2.14
	mean [s]	17.42	15.15	5.08	4.15	11.06	7.57	14.59	14.32
	max [s]	563.50	586.91	541.11	318.78	530.01	394.96	536.21	588.57
S_{1b}	#timeout.	118	108	120	108	181	148	199	163
	total time [h]	22.20	20.19	22.32	20.21	33.22	27.52	36.43	30.03
	95% time [h]	12.53	10.51	12.65	10.53	23.55	17.84	26.76	20.36
	min [s]	1.21	1.23	1.22	1.19	1.29	1.21	1.25	1.24
	median [s]	1.65	1.81	1.73	1.69	2.08	2.02	1.80	1.97
	mean [s]	8.82	7.53	8.11	7.60	11.35	10.20	12.34	10.41
	max [s]	577.92	507.17	591.18	523.41	536.70	584.93	594.66	573.06
S_2	#timeout.	89	73	0	0	140	219	130	199
	total time [h]	17.72	15.57	0.73	0.75	29.63	44.78	28.12	41.86
	95% time [h]	8.05	5.90	0.56	0.56	19.96	35.11	18.45	32.18
	min [s]	1.24	1.36	1.21	1.27	1.43	1.44	1.31	1.23
	median [s]	1.89	1.85	1.82	1.85	3.22	5.86	3.12	5.17
	mean [s]	9.79	11.38	2.30	2.34	22.45	32.01	22.77	32.88
	max [s]	595.30	577.70	142.26	145.30	599.98	599.69	596.26	598.55
S_{2a}	#timeout.	356	276	1	4	140	221	135	211
	total time [h]	63.20	49.05	1.80	2.02	29.92	44.98	28.89	42.99
	95% time [h]	53.53	39.38	0.60	0.63	20.25	35.31	19.22	33.32
	min [s]	1.20	1.20	1.28	1.29	1.33	1.27	1.26	1.35
	median [s]	1.89	1.87	1.88	1.97	3.36	6.24	3.15	5.72
	mean [s]	17.48	12.48	5.13	4.24	23.47	31.55	22.67	29.96
	max [s]	582.90	506.79	556.86	425.75	585.76	590.76	576.10	596.52
S_{2b}	#timeout.	89	73	89	73	309	423	308	400
	total time [h]	17.67	15.61	17.72	15.61	58.56	80.98	57.94	75.78
	95% time [h]	8	5.94	8.05	5.94	48.89	71.31	48.27	66.11
	min [s]	1.26	1.33	1.30	1.17	1.41	1.55	1.43	1.61
	median [s]	1.84	1.87	1.83	1.67	6.71	28.17	6.58	23.43
	mean [s]	9.64	11.51	9.80	11.49	30.22	51.85	28.23	43.68
	max [s]	584.23	579.89	588.73	586.65	566.89	568.17	550.97	598.96
S_{AG}	#timeout.	325	272	1	2	11	15	54	59
	total time [h]	57.44	49.07	1.85	2.02	4.73	5.14	12.60	12.84
	95% time [h]	47.77	39.39	0.74	0.75	0.98	0.90	2.98	3.17
	min [s]	1.27	1.22	1.33	1.37	1.34	1.21	1.30	1.27
	median [s]	2.23	2.15	2.14	2.12	2.48	2.13	2.13	2.14
	mean [s]	14.25	15.24	5.29	5.31	9.16	8.36	11.82	9.90
	max [s]	588.04	517.74	543.61	575.64	517.40	509.57	588.91	519.54

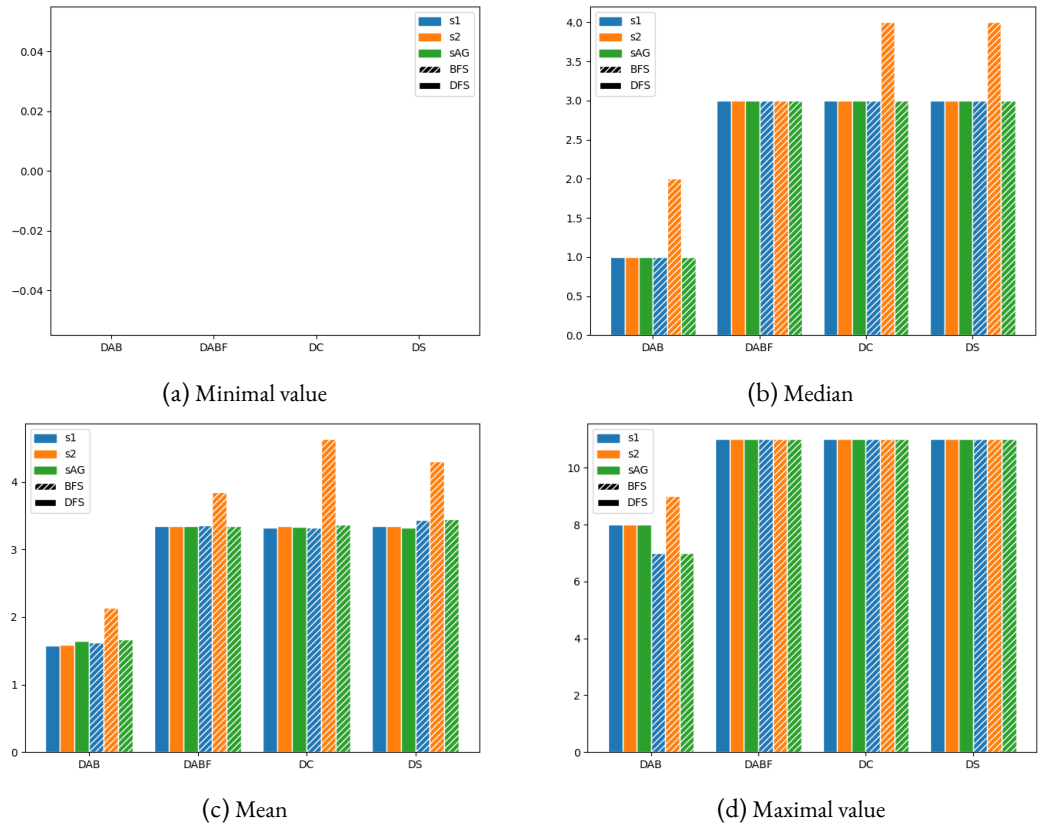
Table 2: Results for admissible semantics using `flexABle` and the set of input benchmarks restricted to the non-trivial ones.

Analysis of Various Statistics in Admissible Semantics

Culprits

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	3.00	3.00	3.00	3.00	3.00	3.00
	mean	1.58	1.62	3.35	3.35	3.32	3.33	3.34	3.44
	max	8.00	7.00	11.00	11.00	11.00	11.00	11.00	11.00
S_{1a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	3.00	3.00	3.00	3.00	3.00	3.00
	mean	1.77	1.67	3.34	3.36	3.31	3.32	3.34	3.44
	max	9.00	7.00	11.00	11.00	11.00	11.00	11.00	11.00
S_{1b}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	1.58	1.62	1.58	1.62	1.58	1.63	1.57	1.76
	max	8.00	7.00	8.00	7.00	8.00	7.00	8.00	9.00
S_2	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	2.00	3.00	3.00	3.00	4.00	3.00	4.00
	mean	1.59	2.13	3.34	3.85	3.35	4.63	3.35	4.31
	max	8.00	9.00	11.00	11.00	11.00	11.00	11.00	11.00
S_{2a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	2.00	3.00	3.00	3.00	4.00	3.00	4.00
	mean	1.79	2.21	3.34	3.88	3.34	4.61	3.34	4.28
	max	9.00	9.00	11.00	11.00	11.00	11.00	11.00	11.00
S_{2b}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	2.00	1.00	2.00	1.00	3.00	1.00	2.00
	mean	1.59	2.13	1.59	2.13	1.55	2.92	1.55	2.59
	max	8.00	9.00	8.00	9.00	8.00	9.00	8.00	9.00
S_{AG}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	3.00	3.00	3.00	3.00	3.00	3.00
	mean	1.65	1.67	3.34	3.34	3.33	3.37	3.32	3.45
	max	8.00	7.00	11.00	11.00	11.00	11.00	11.00	11.00

Table 3: Statistics regarding the number of culprits of successful dispute derivations for the admissible semantics using `flexABle`.



(a) Minimal value

(b) Median

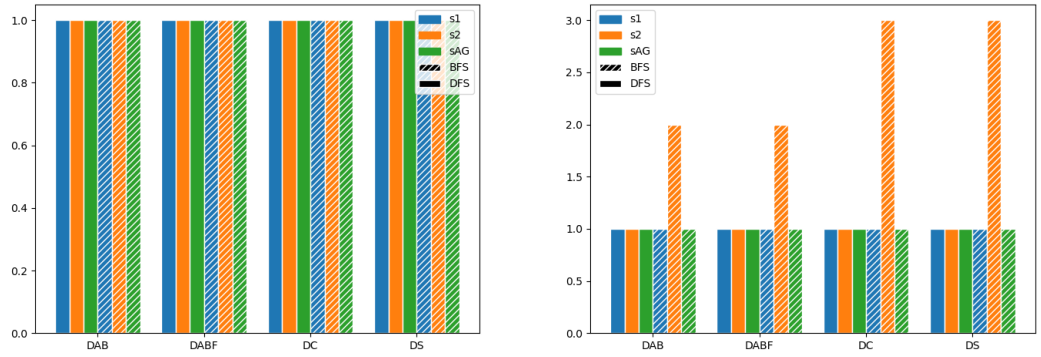
(c) Mean

(d) Maximal value

Figure 1: Bar plots indicating the number of culprits in successful dispute derivations for admissible semantics using `flexABle`.

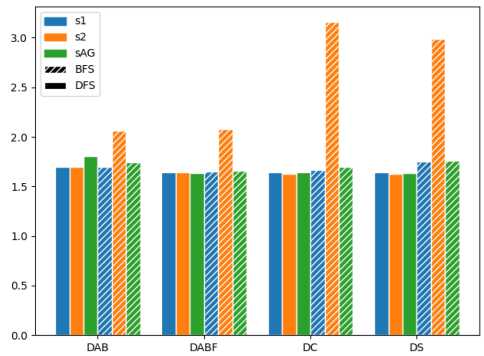
Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	1.70	1.70	1.64	1.65	1.64	1.67	1.64	1.75
	max	8.00	9.00	8.00	9.00	8.00	9.00	8.00	9.00
S_{1a}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	1.94	1.76	1.68	1.66	1.68	1.69	1.68	1.76
	max	8.00	9.00	8.00	9.00	8.00	9.00	8.00	9.00
S_{1b}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	1.70	1.70	1.70	1.70	1.70	1.73	1.70	1.87
	max	8.00	9.00	8.00	9.00	8.00	9.00	8.00	9.00
S_2	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	2.00	1.00	2.00	1.00	3.00	1.00	3.00
	mean	1.70	2.07	1.64	2.08	1.63	3.16	1.63	2.99
	max	8.00	9.00	8.00	9.00	8.00	10.00	8.00	10.00
S_{2a}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	2.00	1.00	2.00	1.00	3.00	1.00	2.00
	mean	1.95	2.16	1.68	2.11	1.63	3.09	1.63	2.91
	max	8.00	9.00	8.00	9.00	8.00	10.00	8.00	10.00
S_{2b}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	2.00	1.00	2.00	1.00	3.00	1.00	3.00
	mean	1.70	2.07	1.70	2.07	1.63	3.18	1.63	3.00
	max	8.00	9.00	8.00	9.00	8.00	10.00	8.00	10.00
S_{AG}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	1.80	1.74	1.64	1.65	1.64	1.70	1.64	1.76
	max	8.00	9.00	8.00	9.00	8.00	9.00	8.00	9.00

Table 4: Statistics regarding the number of defences of successful dispute derivations for the admissible semantics using `flexABle`.

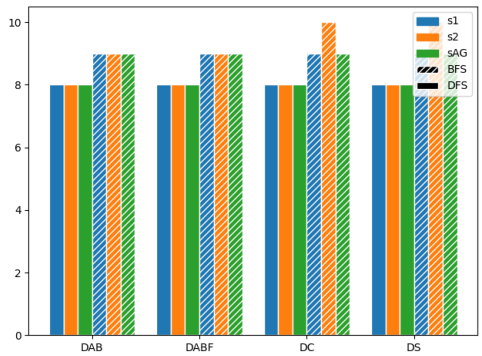


(a) Minimal value

(b) Median



(c) Mean



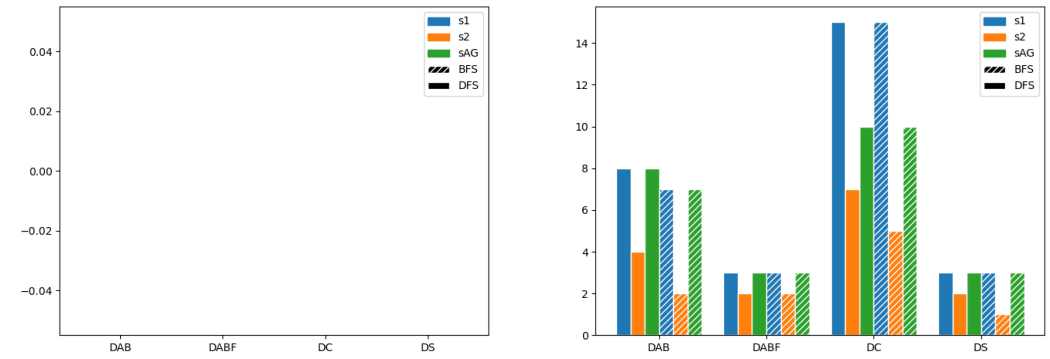
(d) Maximal value

Figure 2: Bar plots indicating the number of defences in successful dispute derivations for admissible semantics using `flexABle`.

Opponent's rules

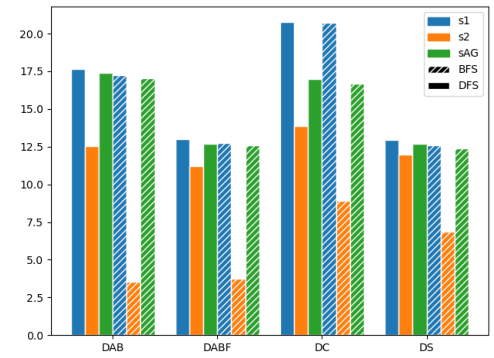
Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	8.00	7.00	3.00	3.00	15.00	15.00	3.00	3.00
	mean	17.67	17.24	12.99	12.76	20.75	20.74	12.94	12.59
	max	107.00	107.00	75.00	75.00	75.00	75.00	75.00	75.00
S_{1a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	7.00	6.00	3.00	3.00	14.00	14.00	3.00	3.00
	mean	17.55	16.44	12.24	11.96	20.92	20.89	12.20	11.79
	max	107.00	107.00	70.00	70.00	74.00	74.00	70.00	70.00
S_{1b}	min	0.00	0.00	0.00	0.00	1.00	1.00	0.00	0.00
	median	8.00	7.00	8.00	7.00	19.00	19.00	8.00	7.00
	mean	17.67	17.24	17.67	17.24	28.54	28.49	17.62	16.91
	max	107.00	107.00	107.00	107.00	107.00	107.00	107.00	107.00
S_2	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	4.00	2.00	2.00	2.00	7.00	5.00	2.00	1.00
	mean	12.53	3.55	11.22	3.73	13.87	8.89	11.97	6.88
	max	107.00	65.00	75.00	65.00	72.00	43.00	75.00	43.00
S_{2a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	4.00	2.00	2.00	2.00	7.00	5.00	2.00	2.00
	mean	12.72	3.57	10.42	3.40	13.71	8.97	11.36	6.61
	max	107.00	65.00	71.00	65.00	73.00	43.00	71.00	43.00
S_{2b}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	4.00	2.00	4.00	2.00	10.00	6.00	3.00	2.00
	mean	12.53	3.55	12.53	3.55	16.14	8.27	12.92	4.99
	max	107.00	65.00	107.00	65.00	107.00	39.00	107.00	39.00
S_{AG}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	8.00	7.00	3.00	3.00	10.00	10.00	3.00	3.00
	mean	17.42	17.02	12.69	12.58	16.96	16.68	12.68	12.37
	max	107.00	107.00	73.00	70.00	73.00	75.00	73.00	70.00

Table 5: Statistics regarding the number of opponent's rules of successful dispute derivations for the admissible semantics using `flexABle`.

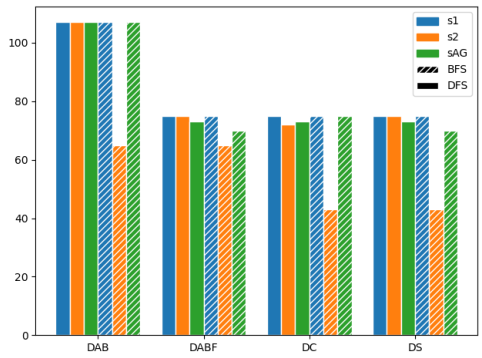


(a) Minimal value

(b) Median



(c) Mean



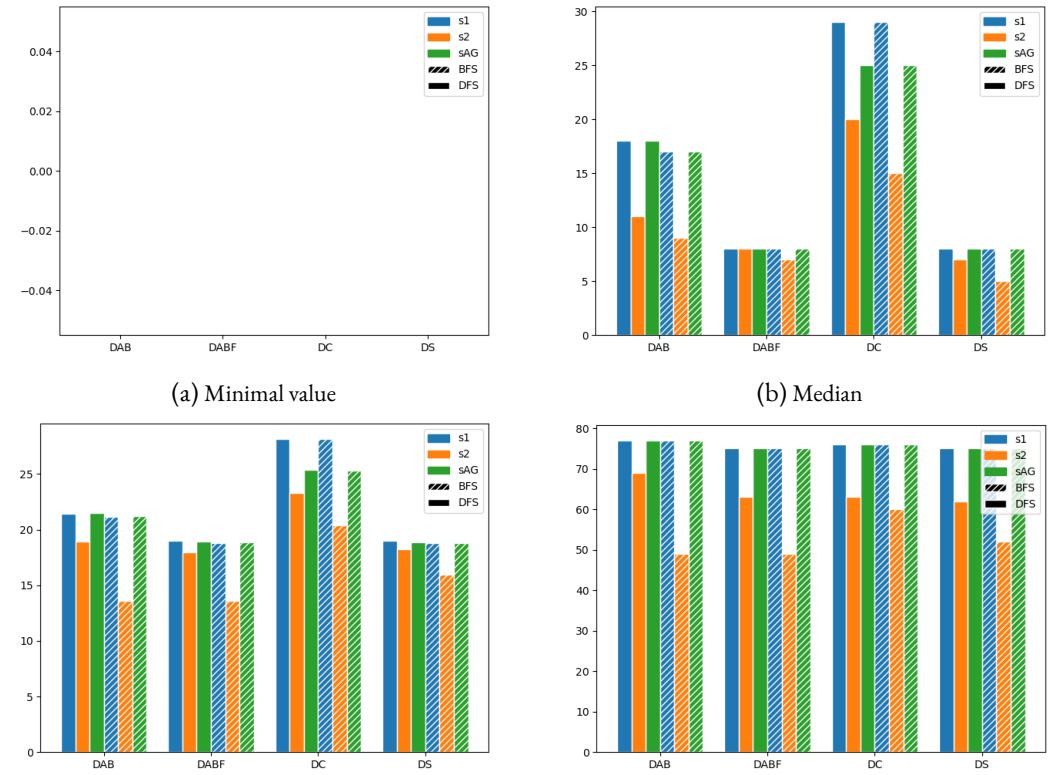
(d) Maximal value

Figure 3: Bar plots indicating the number of opponent's rules in successful dispute derivations for admissible semantics using `flexABle`.

Opponent's statements

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	18.00	17.00	8.00	8.00	29.00	29.00	8.00	8.00
	mean	21.44	21.18	19.02	18.82	28.13	28.13	18.99	18.78
	max	77.00	77.00	75.00	75.00	76.00	76.00	75.00	75.00
S_{1a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	20.00	17.00	8.00	8.00	31.00	31.00	8.00	8.00
	mean	22.12	21.22	19.14	18.87	29.93	29.93	19.11	18.82
	max	77.00	77.00	75.00	75.00	76.00	76.00	75.00	75.00
S_{1b}	min	0.00	0.00	0.00	0.00	2.00	2.00	0.00	0.00
	median	18.00	17.00	18.00	17.00	33.00	33.00	18.00	17.00
	mean	21.44	21.18	21.44	21.18	31.87	31.87	21.43	21.09
	max	77.00	77.00	77.00	77.00	77.00	77.00	77.00	77.00
S_2	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	11.00	9.00	8.00	7.00	20.00	15.00	7.00	5.00
	mean	18.96	13.63	17.94	13.61	23.29	20.36	18.24	15.97
	max	69.00	49.00	63.00	49.00	63.00	60.00	62.00	52.00
S_{2a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	13.00	9.00	8.00	7.00	24.00	16.00	7.00	5.00
	mean	19.67	13.86	17.79	13.47	24.36	21.29	18.26	16.02
	max	65.00	49.00	55.00	49.00	63.00	62.00	62.00	52.00
S_{2b}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	11.00	9.00	11.00	9.00	21.00	17.00	10.00	7.00
	mean	18.96	13.63	18.96	13.63	25.18	20.88	18.91	14.76
	max	69.00	49.00	69.00	49.00	68.00	59.00	68.00	52.00
S_{AG}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	18.00	17.00	8.00	8.00	25.00	25.00	8.00	8.00
	mean	21.53	21.19	18.91	18.84	25.36	25.28	18.88	18.79
	max	77.00	77.00	75.00	75.00	76.00	76.00	75.00	75.00

Table 6: Statistics regarding the number of opponent's statements of successful dispute derivations for the admissible semantics using `flexABle`.



(a) Minimal value

(b) Median

(c) Mean

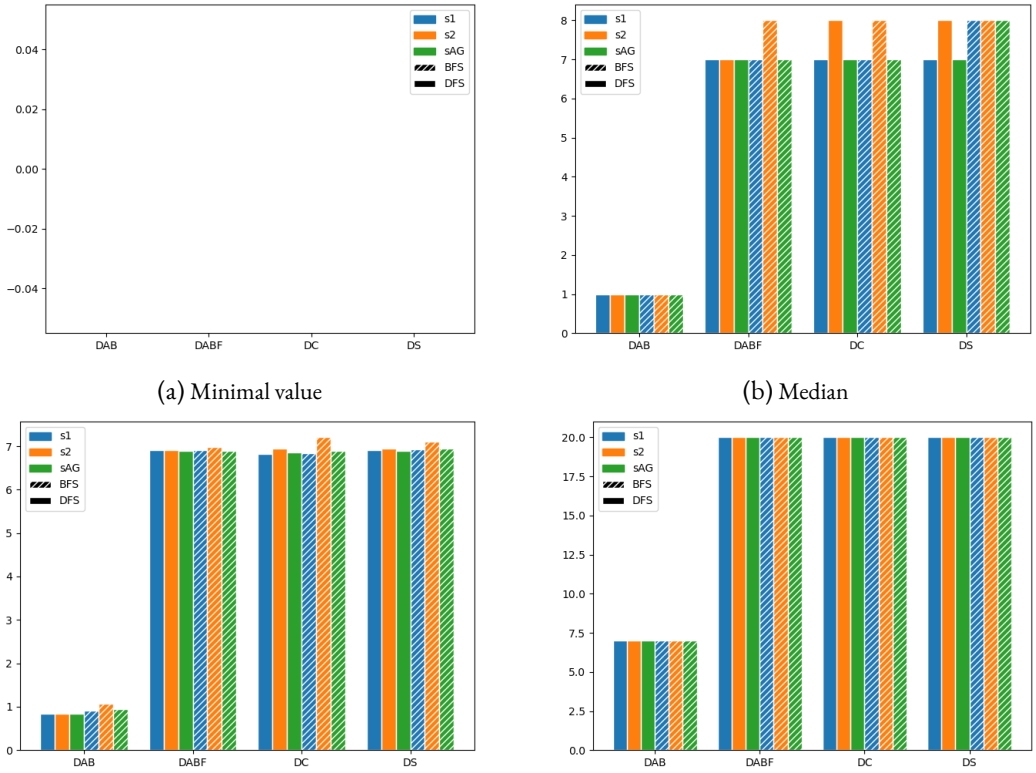
(d) Maximal value

Figure 4: Bar plots indicating the number of opponent's statements in successful dispute derivations for admissible semantics using `flexABle`.

Proponent's rules

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	7.00	7.00	7.00	7.00	7.00	8.00
	mean	0.84	0.92	6.91	6.91	6.83	6.84	6.91	6.93
	max	7.00	7.00	20.00	20.00	20.00	20.00	20.00	20.00
S_{1a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	7.00	7.00	7.00	7.00	7.00	7.00
	mean	1.05	0.93	6.93	6.90	6.83	6.83	6.93	6.92
	max	11.00	7.00	20.00	20.00	20.00	20.00	20.00	20.00
S_{1b}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	0.84	0.92	0.84	0.92	0.84	0.91	0.84	0.98
	max	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00
S_2	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	7.00	8.00	8.00	8.00	8.00	8.00
	mean	0.84	1.07	6.92	6.98	6.94	7.21	6.94	7.11
	max	7.00	7.00	20.00	20.00	20.00	20.00	20.00	20.00
S_{2a}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	7.00	8.00	8.00	8.00	8.00	8.00
	mean	1.08	1.06	6.93	6.97	6.93	7.22	6.93	7.10
	max	11.00	8.00	20.00	20.00	20.00	20.00	20.00	20.00
S_{2b}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	mean	0.84	1.07	0.84	1.07	0.88	1.20	0.88	1.07
	max	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00
S_{AG}	min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	median	1.00	1.00	7.00	7.00	7.00	7.00	7.00	8.00
	mean	0.84	0.94	6.90	6.89	6.87	6.90	6.89	6.95
	max	7.00	7.00	20.00	20.00	20.00	20.00	20.00	20.00

Table 7: Statistics regarding the number of proponent's rules of successful dispute derivations for the admissible semantics using `flexABle`.



(a) Minimal value

(b) Median

(c) Mean

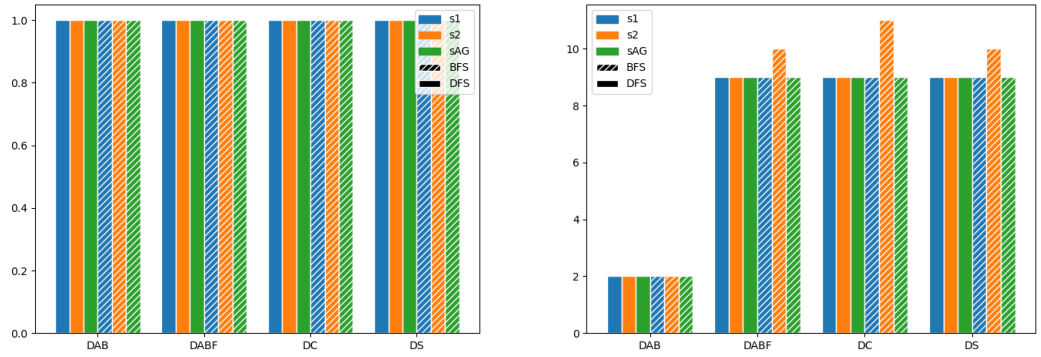
(d) Maximal value

Figure 5: Bar plots indicating the number of proponent's rules in successful dispute derivations for admissible semantics using `flexABle`.

Proponent's statements

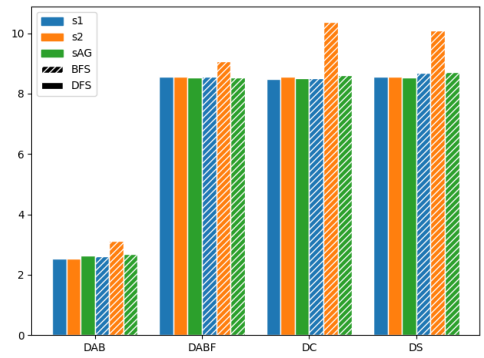
Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	9.00	9.00	9.00	9.00	9.00	9.00
	mean	2.54	2.62	8.55	8.55	8.48	8.51	8.55	8.68
	max	9.00	10.00	23.00	23.00	23.00	23.00	23.00	23.00
S_{1a}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	9.00	9.00	9.00	9.00	9.00	9.00
	mean	2.99	2.69	8.61	8.57	8.52	8.53	8.61	8.68
	max	14.00	11.00	23.00	23.00	23.00	23.00	23.00	23.00
S_{1b}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
	mean	2.54	2.62	2.54	2.62	2.54	2.63	2.54	2.85
	max	9.00	10.00	9.00	10.00	9.00	10.00	9.00	11.00
S_2	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	9.00	10.00	9.00	11.00	9.00	10.00
	mean	2.54	3.13	8.56	9.06	8.57	10.36	8.57	10.10
	max	9.00	11.00	23.00	23.00	23.00	24.00	23.00	23.00
S_{2a}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	9.00	10.00	9.00	11.00	9.00	10.00
	mean	3.04	3.22	8.62	9.08	8.56	10.31	8.56	10.01
	max	14.00	12.00	23.00	23.00	23.00	24.00	23.00	23.00
S_{2b}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	2.00	2.00	2.00	4.00	2.00	3.00
	mean	2.54	3.13	2.54	3.13	2.51	4.38	2.51	4.06
	max	9.00	11.00	9.00	11.00	9.00	12.00	9.00	12.00
S_{AG}	min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	median	2.00	2.00	9.00	9.00	9.00	9.00	9.00	9.00
	mean	2.65	2.68	8.54	8.54	8.51	8.60	8.53	8.71
	max	11.00	10.00	23.00	23.00	23.00	23.00	23.00	23.00

Table 8: Statistics regarding the number of proponent's statements of successful dispute derivations for the admissible semantics using `flexABle`.

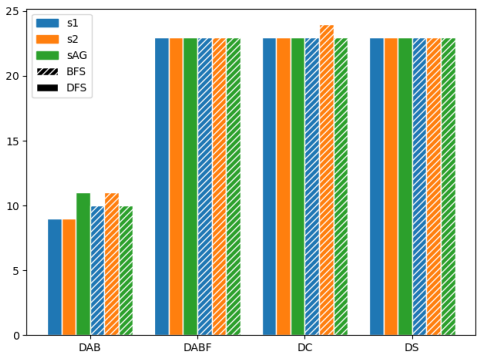


(a) Minimal value

(b) Median



(c) Mean



(d) Maximal value

Figure 6: Bar plots indicating the number of proponent's statements in successful dispute derivations for admissible semantics using `flexABle`.

"Static" sampling

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-131) 62	(-132) 47	(+0) 0	(-1) 1	(-9) 4	(+6) 19	(+54) 90	(+47) 86
	total time [h]	(-21.48) 16.91	(-21.38) 14.32	(+0.09) 3.82	(-0.34) 3.71	(+0.35) 7.72	(+2.89) 10.73	(+10.95) 22.79	(+8.88) 21.44
S_{1a}	#timeout.	(-512) 356	(-200) 169	(-7) 2	(-9) 2	(-7) 10	(-10) 12	(+61) 143	(+61) 137
	total time [h]	(-82.36) 71.61	(-31.11) 37.39	(-0.51) 5.86	(-2.08) 4.24	(-0.54) 8.75	(+0.0) 9.25	(+15.25) 36.49	(+13.44) 33.96
S_{1b}	#timeout.	(-127) 66	(-117) 62	(-128) 67	(-134) 46	(-157) 115	(-114) 122	(-45) 255	(-30) 227
	total time [h]	(-19.51) 18.73	(-18.87) 16.92	(-19.79) 18.76	(-21.37) 14.32	(-21.77) 30.46	(-14.78) 30.55	(-4.09) 52.19	(-2.04) 46.80
S_2	#timeout.	(-83) 50	(-91) 24	(+0) 0	(+1) 1	(+153) 333	(+110) 370	(+135) 302	(+125) 365
	total time [h]	(-15.07) 13.57	(-17.53) 8.55	(+0.11) 3.17	(+0.31) 3.43	(+25.57) 66.39	(+18.01) 74.42	(+21.57) 60.69	(+20.60) 73.64
S_{2a}	#timeout.	(-471) 289	(-213) 124	(-1) 0	(-2) 2	(+141) 322	(+108) 370	(+138) 311	(+120) 372
	total time [h]	(-78.52) 57.91	(-32.66) 29.32	(-1.05) 3.41	(-0.77) 4.11	(+23.06) 64.16	(+21.14) 77.62	(+23.07) 62.87	(+19.10) 73.56
S_{2b}	#timeout.	(-95) 38	(-91) 24	(-104) 29	(-85) 30	(+71) 487	(-150) 688	(+78) 494	(-141) 659
	total time [h]	(-17.46) 11.09	(-17.23) 8.93	(-18.93) 9.68	(-15.60) 10.24	(+15.57) 96.64	(-24.62) 142.16	(+15.57) 96.05	(-23.41) 136.41
S_{AG}	#timeout.	(-320) 246	(-182) 149	(-1) 0	(-1) 1	(-5) 6	(-5) 10	(+54) 108	(+72) 131
	total time [h]	(-51.78) 50.27	(-28.49) 33.77	(+0.51) 5.02	(+1.73) 6.38	(+0.04) 7.58	(+0.62) 8.11	(+9.63) 24.79	(+13.21) 28.54

(a) $p_{prop} = 0.5, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-8) 185	(-16) 163	(+0) 0	(-2) 0	(-3) 10	(-3) 10	(-2) 34	(-9) 30
	total time [h]	(-1.06) 37.33	(-2.50) 33.20	(+0.71) 4.44	(-0.01) 4.04	(-0.32) 7.05	(-1.29) 6.55	(+2.46) 14.30	(-1.94) 10.62
S_{1a}	#timeout.	(-74) 794	(-91) 278	(-4) 5	(-6) 5	(+4) 21	(-8) 14	(-22) 60	(-31) 45
	total time [h]	(-11.90) 142.07	(-16.75) 51.75	(-0.92) 5.45	(-1.24) 5.08	(+0.25) 9.54	(-1.20) 8.05	(-3.50) 17.74	(-6.88) 13.64
S_{1b}	#timeout.	(-16) 177	(-4) 175	(+6) 201	(-17) 163	(-16) 256	(-34) 202	(-29) 271	(-27) 230
	total time [h]	(-2.54) 35.70	(+0.28) 36.07	(+0.53) 39.08	(-2.90) 32.79	(-1.40) 50.83	(-5.25) 40.08	(-4.33) 51.95	(-5.30) 43.54
S_2	#timeout.	(-1) 132	(+6) 121	(+0) 0	(+0) 0	(-45) 135	(-25) 235	(-32) 135	(-27) 213
	total time [h]	(-0.98) 27.66	(+0.29) 26.37	(+0.50) 3.56	(+0.52) 3.64	(-7.70) 33.12	(-4.89) 51.52	(-6.62) 32.50	(-4.88) 48.16
S_{2a}	#timeout.	(-100) 660	(-91) 246	(+0) 1	(-1) 3	(-40) 141	(-35) 227	(-28) 145	(-36) 216
	total time [h]	(-16.62) 119.81	(-15.25) 46.73	(-0.14) 4.32	(-0.11) 4.77	(-7.22) 33.88	(-6.40) 50.08	(-4.45) 35.35	(-6.08) 48.38
S_{2b}	#timeout.	(+1) 134	(+3) 118	(+0) 133	(+13) 128	(-12) 404	(+71) 909	(-5) 411	(+54) 854
	total time [h]	(+0.17) 28.72	(-0.13) 26.03	(-0.45) 28.16	(+1.28) 27.12	(-3.55) 77.52	(+10.91) 177.69	(-1.64) 78.84	(+7.75) 167.57
S_{AG}	#timeout.	(-122) 444	(-95) 236	(-1) 0	(-2) 0	(+1) 12	(-8) 7	(-33) 21	(-38) 21
	total time [h]	(-21.08) 80.97	(-17.04) 45.22	(-0.93) 3.58	(-0.62) 4.03	(-0.90) 6.64	(-1.65) 5.84	(-6.51) 8.65	(-7.02) 8.31

(b) $p_{prop} = 1, p_{opp} = 0.5$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-101) 92	(-106) 73	(+1) 1	(-2) 0	(-3) 10	(-1) 12	(+24) 60	(+13) 52
	total time [h]	(-16.0) 22.39	(-17.23) 18.47	(-0.25) 3.48	(-0.14) 3.91	(+0.02) 7.39	(+0.60) 8.44	(+4.50) 16.34	(+2.34) 14.90
S_{1a}	#timeout.	(-537) 331	(-221) 148	(-7) 2	(-10) 1	(+4) 21	(+5) 27	(+6) 88	(+16) 92
	total time [h]	(-88.22) 65.75	(-36.75) 31.75	(-1.64) 4.73	(-2.16) 4.16	(+1.32) 10.61	(+1.09) 10.34	(+1.30) 22.54	(+2.25) 22.77
S_{1b}	#timeout.	(-98) 95	(-105) 74	(-99) 96	(-118) 62	(-87) 185	(-105) 131	(-68) 232	(-37) 220
	total time [h]	(-16.47) 21.77	(-17.56) 18.23	(-15.31) 23.24	(-19.33) 16.36	(-8.51) 43.72	(-13.12) 32.21	(-8.54) 47.74	(-3.89) 44.95
S_2	#timeout.	(-68) 65	(-76) 39	(+0) 0	(+1) 1	(+75) 255	(+92) 352	(+91) 258	(+116) 356
	total time [h]	(-13.66) 14.98	(-14.15) 11.93	(+0.10) 3.16	(+0.09) 3.21	(+11.73) 52.55	(+14.07) 70.48	(+14.28) 53.40	(+17.16) 70.20
S_{2a}	#timeout.	(-491) 269	(-228) 109	(+0) 1	(-3) 1	(+112) 293	(+93) 355	(+78) 251	(+99) 351
	total time [h]	(-81.41) 55.02	(-33.90) 28.08	(-0.71) 3.75	(-1.36) 3.52	(+19.42) 60.52	(+14.37) 70.85	(+12.28) 52.08	(+14.63) 69.09
S_{2b}	#timeout.	(-62) 71	(-68) 47	(-76) 57	(-55) 60	(+72) 488	(-96) 742	(+56) 472	(-102) 698
	total time [h]	(-11.37) 17.18	(-13.30) 12.86	(-14.52) 14.09	(-11.33) 14.51	(+13.50) 94.57	(-17.35) 149.43	(+10.25) 90.73	(-20.83) 138.99
S_{AG}	#timeout.	(-350) 216	(-212) 119	(-1) 0	(-2) 0	(+0) 11	(+0) 15	(-3) 51	(+8) 67
	total time [h]	(-58.59) 43.46	(-35.05) 27.21	(-0.23) 4.28	(-0.62) 4.03	(-0.82) 6.72	(+2.73) 10.22	(-0.67) 14.49	(+1.83) 17.16

(c) $p_{prop} = 0.5, p_{opp} = 0.5$

Table 9: "Static" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.5.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-103) 90	(-106) 73	(+0) 0	(-2) 0	(-6) 7	(-1) 12	(+30) 66	(+29) 68
	total time [h]	(-14.81) 23.58	(-15.53) 20.17	(-0.36) 3.37	(-0.37) 3.68	(+0.25) 7.62	(-0.45) 7.39	(+7.75) 19.59	(+5.93) 18.49
S_{1a}	#timeout.	(-393) 475	(-156) 213	(-9) 0	(-9) 2	(-9) 8	(-6) 16	(+40) 122	(+63) 139
	total time [h]	(-60.89) 93.08	(-22.11) 46.39	(-1.67) 4.70	(-1.61) 4.71	(-0.77) 8.52	(+0.06) 9.31	(+8.55) 29.79	(+11.54) 32.06
S_{1b}	#timeout.	(-114) 79	(-120) 59	(-117) 78	(-110) 70	(-94) 178	(-90) 146	(-6) 294	(-14) 243
	total time [h]	(-17.30) 20.94	(-18.38) 17.41	(-16.85) 21.70	(-16.40) 19.29	(-11.64) 40.59	(-10.20) 35.13	(+2.46) 58.74	(+1.02) 49.86
S_2	#timeout.	(-94) 39	(-83) 32	(+0) 0	(+0) 0	(+136) 316	(+92) 352	(+112) 279	(+99) 339
	total time [h]	(-16.40) 12.24	(-16.06) 10.02	(+0.90) 3.96	(+0.57) 3.69	(+22.67) 63.49	(+13.64) 70.05	(+18.22) 57.34	(+16.07) 69.11
S_{2a}	#timeout.	(-371) 389	(-165) 172	(-1) 0	(-4) 0	(+118) 299	(+92) 354	(+117) 290	(+84) 336
	total time [h]	(-60.50) 75.93	(-22.48) 39.50	(-0.71) 3.75	(-1.17) 3.71	(+19.11) 60.21	(+16.49) 72.97	(+18.30) 58.10	(+13.53) 67.99
S_{2b}	#timeout.	(-86) 47	(-85) 30	(-101) 32	(-87) 28	(+55) 471	(-134) 704	(+79) 495	(-106) 694
	total time [h]	(-15.99) 12.56	(-14.61) 11.55	(-17.73) 10.88	(-15.41) 10.43	(+12.32) 93.39	(-21.18) 145.60	(+14.44) 94.92	(-17.45) 142.37
S_{AG}	#timeout.	(-259) 307	(-138) 193	(-1) 0	(-2) 0	(+0) 11	(-1) 14	(+62) 116	(+41) 100
	total time [h]	(-38.89) 63.16	(-19.08) 43.18	(-0.07) 4.44	(-0.51) 4.14	(+0.53) 8.07	(+0.74) 8.23	(+11.46) 26.62	(+8.17) 23.50

(a) $p_{prop} = 0.6, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+9) 202	(-8) 171	(+2) 2	(-1) 1	(-2) 11	(-6) 7	(-7) 29	(+1) 40
	total time [h]	(+2.86) 41.25	(-1.87) 33.83	(+1.90) 5.63	(+0.26) 4.31	(-0.24) 7.13	(-1.44) 6.40	(-0.41) 11.43	(+1.35) 13.91
S_{1a}	#timeout.	(-56) 812	(-71) 298	(-2) 7	(-5) 6	(+2) 19	(-5) 17	(-18) 64	(-16) 60
	total time [h]	(-8.49) 145.48	(-12.16) 56.34	(+0.14) 6.51	(-0.61) 5.71	(-0.17) 9.12	(+1.57) 10.82	(-3.75) 17.49	(-4.23) 16.29
S_{1b}	#timeout.	(-9) 184	(-4) 175	(+9) 204	(+3) 183	(+7) 279	(-20) 216	(-26) 274	(-9) 248
	total time [h]	(-0.98) 37.26	(-0.74) 35.05	(+2.17) 40.72	(-0.29) 35.40	(+0.93) 53.16	(-3.72) 41.61	(-3.40) 52.88	(-1.94) 46.90
S_2	#timeout.	(-7) 126	(+12) 127	(+0) 0	(+0) 0	(-32) 148	(-20) 240	(-28) 139	(-13) 227
	total time [h]	(-1.99) 26.65	(+1.42) 27.50	(+0.16) 3.22	(+0.33) 3.45	(-5.23) 35.59	(-3.74) 52.67	(-5.74) 33.38	(-2.64) 50.40
S_{2a}	#timeout.	(-65) 695	(-56) 281	(+0) 1	(+0) 4	(-30) 151	(-15) 247	(-28) 145	(-21) 231
	total time [h]	(-10.30) 126.13	(-9.95) 52.03	(-0.04) 4.42	(-0.20) 4.68	(-5.01) 36.09	(-1.83) 54.65	(-5.63) 34.17	(-3.66) 50.80
S_{2b}	#timeout.	(+0) 133	(+8) 123	(-2) 131	(+20) 135	(+11) 427	(+58) 896	(+2) 418	(+53) 853
	total time [h]	(-0.48) 28.07	(+0.39) 26.55	(-0.51) 28.10	(+2.75) 28.59	(-0.02) 81.05	(+9.51) 176.29	(-0.40) 80.08	(+7.98) 167.80
S_{AG}	#timeout.	(-79) 487	(-57) 274	(-1) 0	(-2) 0	(-1) 10	(-3) 12	(-23) 31	(-26) 33
	total time [h]	(-14.76) 87.29	(-10.46) 51.80	(+0.15) 4.66	(-0.14) 4.51	(-0.02) 7.52	(-0.60) 6.89	(-2.78) 12.38	(-4.34) 10.99

(b) $p_{prop} = 1, p_{opp} = 0.6$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-97) 96	(-102) 77	(+0) 0	(-1) 1	(-1) 12	(+2) 15	(+9) 45	(+17) 56
	total time [h]	(-14.60) 23.79	(-14.96) 20.74	(-0.09) 3.64	(-0.24) 3.81	(+0.34) 7.71	(+0.44) 8.28	(+2.35) 14.19	(+4.01) 16.57
S_{1a}	#timeout.	(-425) 443	(-160) 209	(-8) 1	(-11) 0	(+0) 17	(-2) 20	(+16) 98	(+19) 95
	total time [h]	(-66.50) 87.47	(-24.50) 44.0	(-1.89) 4.48	(-2.18) 4.14	(+0.59) 9.88	(+1.02) 10.27	(+3.03) 24.27	(+2.65) 23.17
S_{1b}	#timeout.	(-85) 108	(-97) 82	(-84) 111	(-83) 97	(-51) 221	(-68) 168	(-1) 299	(-24) 233
	total time [h]	(-12.91) 25.33	(-13.59) 22.20	(-12.14) 26.41	(-11.42) 24.27	(-5.18) 47.05	(-6.31) 39.02	(+2.27) 58.55	(-1.86) 46.98
S_2	#timeout.	(-73) 60	(-66) 49	(+0) 0	(+0) 0	(+57) 237	(+82) 342	(+95) 262	(+97) 337
	total time [h]	(-13.08) 15.56	(-12.20) 13.88	(+0.42) 3.48	(+0.76) 3.88	(+9.42) 50.24	(+12.73) 69.14	(+14.30) 53.42	(+16.02) 69.06
S_{2a}	#timeout.	(-387) 373	(-173) 164	(-1) 0	(-4) 0	(+78) 259	(+94) 356	(+79) 252	(+73) 325
	total time [h]	(-64.45) 71.98	(-26.45) 35.53	(-0.96) 3.50	(-1.50) 3.38	(+13.17) 54.27	(+13.71) 70.19	(+11.69) 51.49	(+11.52) 65.98
S_{2b}	#timeout.	(-63) 70	(-59) 56	(-67) 66	(-68) 47	(+89) 505	(-66) 772	(+64) 480	(-49) 751
	total time [h]	(-10.74) 17.81	(-10.66) 15.50	(-11.77) 16.84	(-11.81) 14.03	(+16.54) 97.61	(-12.59) 154.19	(+12.56) 93.04	(-10.69) 149.13
S_{AG}	#timeout.	(-298) 268	(-183) 148	(-1) 0	(-2) 0	(+6) 17	(-3) 12	(+12) 66	(+12) 71
	total time [h]	(-46.50) 55.55	(-28.67) 33.59	(-0.99) 3.52	(-0.86) 3.79	(+1.46) 9.0	(+0.63) 8.12	(+1.64) 16.80	(+2.21) 17.54

(c) $p_{prop} = 0.6, p_{opp} = 0.6$

Table 10: "Static" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.6.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-50) 143	(-39) 140	(+1) 1	(-2) 0	(+2) 15	(-2) 11	(+9) 45	(+22) 61
	total time [h]	(-6.92) 31.47	(-5.13) 30.57	(+0.02) 3.75	(+0.26) 4.31	(+4.11) 11.48	(+0.02) 7.86	(+2.98) 14.82	(+4.87) 17.43
S_{1a}	#timeout.	(-153) 715	(-21) 348	(-5) 4	(-5) 6	(-1) 16	(-2) 20	(+34) 116	(+22) 98
	total time [h]	(-23.32) 130.65	(-2.47) 66.03	(-0.77) 5.60	(-0.03) 6.29	(+0.93) 10.22	(+0.06) 9.31	(+7.17) 28.41	(+4.36) 24.88
S_{1b}	#timeout.	(-34) 159	(-32) 147	(-42) 153	(-29) 151	(-49) 223	(-28) 208	(+4) 304	(-5) 252
	total time [h]	(-4.57) 33.67	(-3.41) 32.38	(-5.25) 33.30	(-3.98) 31.71	(-4.88) 47.35	(-2.46) 42.87	(+3.88) 60.16	(+0.85) 49.69
S_2	#timeout.	(-44) 89	(-39) 76	(+0) 0	(+0) 0	(+57) 237	(+52) 312	(+70) 237	(+55) 295
	total time [h]	(-8.34) 20.30	(-7.05) 19.03	(+0.33) 3.39	(+0.49) 3.61	(+9.68) 50.50	(+8.65) 65.06	(+12.80) 51.92	(+9.94) 62.98
S_{2a}	#timeout.	(-174) 586	(-59) 278	(+0) 1	(-2) 2	(+65) 246	(+58) 320	(+70) 243	(+66) 318
	total time [h]	(-26.32) 110.11	(-5.09) 56.89	(-0.18) 4.28	(-0.68) 4.20	(+10.96) 52.06	(+10.37) 66.85	(+12.03) 51.83	(+9.62) 64.08
S_{2b}	#timeout.	(-43) 90	(-35) 80	(-35) 98	(-37) 78	(+42) 458	(-51) 787	(+14) 430	(-45) 755
	total time [h]	(-6.54) 22.01	(-7.05) 19.11	(-5.72) 22.89	(-6.48) 19.36	(+8.81) 89.88	(-8.10) 158.68	(+4.06) 84.54	(-6.42) 153.40
S_{AG}	#timeout.	(-99) 467	(-39) 292	(-1) 0	(-2) 0	(+4) 15	(-6) 9	(+26) 80	(+23) 82
	total time [h]	(-15.53) 86.52	(-5.22) 57.04	(-0.49) 4.02	(+0.14) 4.79	(+1.48) 9.02	(-0.08) 7.41	(+4.74) 19.90	(+6.28) 21.61

(a) $p_{prop} = 0.8, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+15) 208	(+3) 182	(+0) 0	(-1) 1	(-3) 10	(+3) 16	(-2) 34	(+1) 40
	total time [h]	(+3.40) 41.79	(+0.91) 36.61	(+0.27) 4.0	(-0.24) 3.81	(+0.76) 8.13	(+0.40) 8.24	(+0.45) 12.29	(+0.30) 12.86
S_{1a}	#timeout.	(-11) 857	(-33) 336	(-2) 7	(-5) 6	(+5) 22	(-2) 20	(-3) 79	(-6) 70
	total time [h]	(-1.62) 152.35	(-5.49) 63.01	(-0.34) 6.03	(-0.45) 5.87	(+1.16) 10.45	(+1.37) 10.62	(-0.84) 20.40	(-1.10) 19.42
S_{1b}	#timeout.	(+10) 203	(+6) 185	(-3) 192	(-1) 179	(+0) 272	(+0) 236	(+8) 308	(+0) 257
	total time [h]	(+2.25) 40.49	(+1.07) 36.86	(+0.51) 39.06	(+0.57) 36.26	(+0.39) 52.62	(+0.05) 45.38	(+0.99) 57.27	(-0.25) 48.59
S_2	#timeout.	(+13) 146	(+3) 118	(+0) 0	(+0) 0	(-10) 170	(-4) 256	(-2) 165	(-5) 235
	total time [h]	(+1.96) 30.60	(+0.78) 26.86	(+0.75) 3.81	(+0.16) 3.28	(-1.88) 38.94	(-0.40) 56.01	(+0.40) 39.52	(+0.13) 53.17
S_{2a}	#timeout.	(-19) 741	(-18) 319	(+0) 1	(-1) 3	(-16) 165	(-4) 258	(-13) 160	(-7) 245
	total time [h]	(-3.47) 132.96	(-2.61) 59.37	(+0.36) 4.82	(-0.43) 4.45	(-3.32) 37.78	(-0.21) 56.27	(-1.81) 37.99	(-1.22) 53.24
S_{2b}	#timeout.	(+6) 139	(+6) 121	(+4) 137	(+9) 124	(+26) 442	(+68) 906	(+23) 439	(+46) 846
	total time [h]	(+1.02) 29.57	(+0.78) 26.94	(+0.90) 29.51	(+1.08) 26.92	(+4.20) 85.27	(+11.92) 178.70	(+3.81) 84.29	(+7.06) 166.88
S_{AG}	#timeout.	(-36) 530	(-26) 305	(+0) 1	(-1) 1	(+0) 11	(-3) 12	(+0) 54	(-12) 47
	total time [h]	(-5.91) 96.14	(-4.46) 57.80	(-0.30) 4.21	(-0.18) 4.47	(-0.64) 6.90	(+0.96) 8.45	(-0.67) 14.49	(-1.73) 13.60

(b) $p_{prop} = 1, p_{opp} = 0.8$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-24) 169	(-35) 144	(+2) 2	(-1) 1	(-1) 12	(+5) 18	(+15) 51	(-1) 38
	total time [h]	(-1.24) 37.15	(-2.81) 32.89	(+1.35) 5.08	(-0.08) 3.97	(+1.27) 8.64	(+1.20) 9.04	(+3.91) 15.75	(+0.91) 13.47
S_{1a}	#timeout.	(-176) 692	(-53) 316	(-4) 5	(-6) 5	(+2) 19	(-2) 20	(+22) 104	(+11) 87
	total time [h]	(-27.56) 126.41	(-8.39) 60.11	(+0.0) 6.37	(-1.10) 5.22	(+1.16) 10.45	(+0.84) 10.09	(+5.49) 26.73	(+2.74) 23.26
S_{1b}	#timeout.	(-26) 167	(-34) 145	(-7) 188	(-26) 154	(-30) 242	(-19) 217	(-5) 295	(-2) 255
	total time [h]	(-2.70) 35.54	(-4.48) 31.31	(+0.18) 38.73	(-3.45) 32.24	(-3.40) 48.83	(-0.74) 44.59	(+1.44) 57.72	(+1.64) 50.48
S_2	#timeout.	(-19) 114	(-34) 81	(+0) 0	(+0) 0	(+52) 232	(+67) 327	(+41) 208	(+64) 304
	total time [h]	(-3.02) 25.62	(-5.96) 20.12	(+0.31) 3.37	(+1.07) 4.19	(+8.13) 48.95	(+12.17) 68.58	(+6.94) 46.06	(+10.18) 63.22
S_{2a}	#timeout.	(-182) 578	(-79) 258	(+2) 3	(-1) 3	(+32) 213	(+48) 310	(+56) 229	(+53) 305
	total time [h]	(-29.68) 106.75	(-10.52) 51.46	(-0.23) 4.23	(-0.35) 4.53	(+5.80) 46.90	(+7.72) 64.20	(+8.51) 48.31	(+9.20) 63.66
S_{2b}	#timeout.	(-37) 96	(-37) 78	(-44) 89	(-26) 89	(+58) 474	(-28) 810	(+46) 462	(+6) 806
	total time [h]	(-6.20) 22.35	(-7.81) 18.35	(-6.92) 21.69	(-4.45) 21.39	(+9.20) 90.27	(-2.0) 164.78	(+8.08) 88.56	(+0.31) 160.13
S_{AG}	#timeout.	(-130) 436	(-65) 266	(+0) 1	(+0) 2	(+3) 14	(+1) 16	(+13) 67	(+8) 67
	total time [h]	(-20.49) 81.56	(-9.46) 52.80	(+0.73) 5.24	(-0.19) 4.46	(+0.75) 8.29	(+0.82) 8.31	(+2.41) 17.57	(+1.84) 17.17

(c) $p_{prop} = 0.8, p_{opp} = 0.8$

Table 11: "Static" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.8.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-23) 170	(+0) 179	(+0) 0	(-1) 1	(+2) 15	(+3) 16	(+8) 44	(+13) 52
	total time [h]	(-2.78) 35.61	(+0.76) 36.46	(+0.12) 3.85	(+0.77) 4.82	(+1.69) 9.06	(+1.30) 9.14	(+2.11) 13.95	(+2.90) 15.46
S_{1a}	#timeout.	(-71) 797	(+5) 374	(-6) 3	(-3) 8	(+4) 21	(-4) 18	(+14) 96	(+23) 99
	total time [h]	(-9.75) 144.22	(+0.78) 69.28	(+1.86) 8.23	(-0.13) 6.19	(+1.25) 10.54	(+1.18) 10.43	(+5.57) 26.81	(+5.13) 25.65
S_{1b}	#timeout.	(-10) 183	(-12) 167	(-11) 184	(-11) 169	(-21) 251	(+4) 240	(+13) 313	(-2) 255
	total time [h]	(-0.73) 37.51	(-1.63) 34.16	(-1.04) 37.51	(-1.72) 33.97	(-1.95) 50.28	(+2.0) 47.33	(+2.80) 59.08	(+0.84) 49.68
S_2	#timeout.	(-16) 117	(-24) 91	(+0) 0	(+0) 0	(+29) 209	(+25) 285	(+38) 205	(+45) 285
	total time [h]	(-3.18) 25.46	(-3.94) 22.14	(+0.37) 3.43	(+0.69) 3.81	(+5.84) 46.66	(+4.19) 60.60	(+6.51) 45.63	(+7.36) 60.40
S_{2a}	#timeout.	(-61) 699	(-10) 327	(+1) 2	(+1) 5	(+38) 219	(+34) 296	(+41) 214	(+33) 285
	total time [h]	(-9.59) 126.84	(-0.62) 61.36	(+0.13) 4.59	(+0.70) 5.58	(+7.98) 49.08	(+6.41) 62.89	(+6.48) 46.28	(+5.44) 59.90
S_{2b}	#timeout.	(-24) 109	(-24) 91	(-20) 113	(-14) 101	(+18) 434	(-28) 810	(+26) 442	(-9) 791
	total time [h]	(-2.23) 26.32	(-4.32) 21.84	(-3.38) 25.23	(-2.68) 23.16	(+3.17) 84.24	(-2.20) 164.58	(+4.65) 85.13	(-2.23) 157.59
S_{AG}	#timeout.	(-50) 516	(-20) 311	(+0) 1	(+0) 2	(+1) 12	(-1) 14	(+15) 69	(+8) 67
	total time [h]	(-6.67) 95.38	(-2.83) 59.43	(+0.18) 4.69	(+1.18) 5.83	(+1.15) 8.69	(+0.73) 8.22	(+4.25) 19.41	(+1.77) 17.10

(a) $p_{prop} = 0.9, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+3) 196	(+9) 188	(+0) 0	(+0) 2	(+0) 13	(-1) 12	(-1) 35	(+0) 39
	total time [h]	(+1.16) 39.55	(+2.05) 37.75	(+0.55) 4.28	(+0.28) 4.33	(+0.10) 7.47	(+0.29) 8.13	(+0.47) 12.31	(+0.51) 13.07
S_{1a}	#timeout.	(-15) 853	(-13) 356	(+0) 9	(+0) 11	(+4) 21	(-2) 20	(-2) 80	(-2) 74
	total time [h]	(-1.49) 152.48	(-2.29) 66.21	(+0.20) 6.57	(+1.50) 7.82	(+1.97) 11.26	(+0.31) 9.56	(-0.08) 21.16	(-1.19) 19.33
S_{1b}	#timeout.	(+11) 204	(+1) 180	(-1) 194	(+7) 187	(+2) 274	(-8) 228	(-2) 298	(+0) 257
	total time [h]	(+2.75) 40.99	(+0.22) 36.01	(+0.64) 39.19	(+1.72) 37.41	(+0.21) 52.44	(-0.56) 44.77	(-0.59) 55.69	(+0.18) 49.02
S_2	#timeout.	(+6) 139	(+9) 124	(+0) 0	(+0) 0	(-2) 178	(-1) 259	(-1) 166	(+3) 243
	total time [h]	(+1.11) 29.75	(+1.27) 27.35	(+0.30) 3.36	(+0.18) 3.30	(-0.20) 40.62	(+0.02) 56.43	(-0.58) 38.54	(+0.90) 53.94
S_{2a}	#timeout.	(-12) 748	(-16) 321	(+1) 2	(+1) 5	(-6) 175	(-6) 256	(-4) 169	(-4) 248
	total time [h]	(-1.96) 134.47	(-1.92) 60.06	(+0.32) 4.78	(+0.83) 5.71	(-0.49) 40.61	(+0.10) 56.58	(-0.30) 39.50	(+0.06) 54.52
S_{2b}	#timeout.	(-2) 131	(+6) 121	(+0) 133	(+14) 129	(+10) 426	(+30) 868	(+18) 434	(+31) 831
	total time [h]	(+0.13) 28.68	(+1.05) 27.21	(+0.37) 28.98	(+2.91) 28.75	(+1.41) 82.48	(+5.97) 172.75	(+3.06) 83.54	(+5.10) 164.92
S_{AG}	#timeout.	(-15) 551	(-13) 318	(+1) 2	(+0) 2	(+0) 11	(+1) 16	(-4) 50	(-11) 48
	total time [h]	(-2.08) 99.97	(-1.86) 60.40	(+0.61) 5.12	(+0.21) 4.86	(-0.20) 7.34	(+1.43) 8.92	(-1.0) 14.16	(-0.96) 14.37

(b) $p_{prop} = 1, p_{opp} = 0.9$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-7) 186	(+6) 185	(+0) 0	(-2) 0	(+2) 15	(+0) 13	(+9) 45	(+15) 54
	total time [h]	(-0.46) 37.93	(+1.84) 37.54	(+0.38) 4.11	(-0.02) 4.03	(+1.21) 8.58	(+0.67) 8.51	(+2.23) 14.07	(+2.90) 15.46
S_{1a}	#timeout.	(-82) 786	(-12) 357	(-5) 4	(-5) 6	(+5) 22	(-2) 20	(+11) 93	(+7) 83
	total time [h]	(-10.87) 143.10	(-2.04) 66.46	(-0.37) 6.0	(-0.24) 6.08	(+1.84) 11.13	(+0.99) 10.24	(+2.85) 24.09	(+1.29) 21.81
S_{1b}	#timeout.	(-14) 179	(-9) 170	(-7) 188	(-12) 168	(-19) 253	(+6) 242	(+2) 302	(-2) 255
	total time [h]	(-1.63) 36.61	(-1.27) 34.52	(-0.14) 38.41	(-0.38) 35.31	(-1.97) 50.26	(+0.96) 46.29	(+1.55) 57.83	(+0.96) 49.80
S_2	#timeout.	(-19) 114	(-11) 104	(+0) 0	(+0) 0	(+27) 207	(+30) 290	(+26) 193	(+27) 267
	total time [h]	(-2.82) 25.82	(-2.85) 23.23	(+0.97) 4.03	(+0.60) 3.72	(+5.35) 46.17	(+4.25) 60.66	(+4.86) 43.98	(+3.67) 56.71
S_{2a}	#timeout.	(-89) 671	(-40) 297	(+5) 6	(+0) 4	(+27) 208	(+31) 293	(+33) 206	(+20) 272
	total time [h]	(-12.82) 123.61	(-4.86) 57.12	(+0.81) 5.27	(+0.21) 5.09	(+3.57) 44.67	(+5.02) 61.50	(+5.24) 45.04	(+4.15) 58.61
S_{2b}	#timeout.	(-9) 124	(-19) 96	(-6) 127	(-10) 105	(+35) 451	(+19) 857	(+28) 444	(+9) 809
	total time [h]	(-1.93) 26.62	(-3.38) 22.78	(-0.88) 27.73	(-2.40) 23.44	(+5.78) 86.85	(+3.90) 170.68	(+4.85) 85.33	(+2.88) 162.70
S_{AG}	#timeout.	(-61) 505	(-25) 306	(+0) 1	(-1) 1	(-3) 8	(-5) 10	(+4) 58	(+5) 64
	total time [h]	(-9.83) 92.22	(-3.91) 58.35	(-0.22) 4.29	(+0.28) 4.93	(+0.14) 7.68	(+1.01) 8.50	(+0.36) 15.52	(+1.56) 16.89

(c) $p_{prop} = 0.9, p_{opp} = 0.9$

Table 12: "Static" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.9.

"Dynamic" sampling

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-143) 50	(-142) 37	(+0) 0	(-2) 0	(-12) 1	(-10) 3	(-22) 14	(-24) 15
	total time [h]	(-23.89) 14.50	(-23.96) 11.74	(-0.48) 3.25	(-0.76) 3.29	(-2.77) 4.60	(-3.02) 4.82	(-5.09) 6.75	(-5.55) 7.01
S_{1a}	#timeout.	(-292) 576	(-50) 319	(-9) 0	(-11) 0	(-13) 4	(-17) 5	(-40) 42	(-31) 45
	total time [h]	(-45.81) 108.16	(-5.0) 63.50	(-3.18) 3.19	(-2.58) 3.74	(-4.05) 5.24	(-4.22) 5.03	(-7.94) 13.30	(-7.40) 13.12
S_{1b}	#timeout.	(-142) 51	(-148) 31	(-139) 56	(-138) 42	(-184) 88	(-173) 63	(-68) 232	(-72) 185
	total time [h]	(-24.33) 13.91	(-25.14) 10.65	(-23.70) 14.85	(-23.41) 12.28	(-28.81) 23.42	(-26.11) 19.22	(-8.48) 47.80	(-9.79) 39.05
S_2	#timeout.	(-103) 30	(-98) 17	(+0) 0	(+0) 0	(-67) 113	(-86) 174	(-65) 102	(-85) 155
	total time [h]	(-19.23) 9.41	(-19.14) 6.94	(+0.31) 3.37	(+0.91) 4.03	(-14.23) 26.59	(-17.04) 39.37	(-14.44) 24.68	(-17.39) 35.65
S_{2a}	#timeout.	(-285) 475	(-78) 259	(-1) 0	(-4) 0	(-69) 112	(-88) 174	(-65) 108	(-79) 173
	total time [h]	(-46.81) 89.62	(-9.70) 52.28	(-0.99) 3.47	(-1.88) 3.0	(-14.26) 26.84	(-17.09) 39.39	(-13.20) 26.60	(-14.76) 39.70
S_{2b}	#timeout.	(-94) 39	(-91) 24	(-101) 32	(-93) 22	(-87) 329	(-338) 500	(-97) 319	(-342) 458
	total time [h]	(-17.76) 10.79	(-18.07) 8.09	(-19.23) 9.38	(-16.08) 9.76	(-10.47) 70.60	(-56.45) 110.33	(-12.41) 68.07	(-58.08) 101.74
S_{AG}	#timeout.	(-274) 292	(-150) 181	(-1) 0	(-2) 0	(-8) 3	(-10) 5	(-13) 41	(-20) 39
	total time [h]	(-41.28) 60.77	(-21.52) 40.74	(-0.97) 3.54	(-1.62) 3.03	(-2.72) 4.82	(-2.39) 5.10	(-2.25) 12.91	(-3.82) 11.51

(a) $p_{prop} = 0.5, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+53) 246	(+58) 237	(+0) 0	(-1) 1	(-6) 7	(-5) 8	(-1) 35	(-1) 38
	total time [h]	(+9.07) 47.46	(+10.55) 46.25	(+0.10) 3.83	(+0.61) 4.66	(-1.96) 5.41	(-1.76) 6.08	(+0.52) 12.36	(-0.11) 12.45
S_{1a}	#timeout.	(-14) 854	(-17) 352	(-1) 8	(-5) 6	(-5) 12	(-7) 15	(-24) 58	(-20) 56
	total time [h]	(-2.08) 151.89	(-3.32) 65.18	(-0.43) 5.94	(-0.71) 5.61	(-1.36) 7.93	(+0.70) 9.95	(-3.40) 17.84	(-3.51) 17.01
S_{1b}	#timeout.	(+54) 247	(+66) 245	(+67) 262	(+54) 234	(+8) 280	(+26) 262	(+50) 350	(+60) 317
	total time [h]	(+9.58) 47.82	(+11.57) 47.36	(+11.53) 50.08	(+10.16) 45.85	(+1.31) 53.54	(+4.67) 50.0	(+8.30) 64.58	(+10.55) 59.39
S_2	#timeout.	(+17) 150	(+8) 123	(+0) 0	(+0) 0	(-72) 108	(-117) 143	(-66) 101	(-108) 132
	total time [h]	(+3.14) 31.78	(+1.30) 27.38	(+0.41) 3.47	(+0.26) 3.38	(-13.67) 27.15	(-23.16) 33.25	(-12.97) 26.15	(-21.20) 31.84
S_{2a}	#timeout.	(-45) 715	(-93) 244	(+0) 1	(-1) 3	(-72) 109	(-111) 151	(-72) 101	(-118) 134
	total time [h]	(-7.50) 128.93	(-14.38) 47.60	(+0.63) 5.09	(+0.19) 5.07	(-13.73) 27.37	(-20.04) 36.44	(-13.84) 25.96	(-22.40) 32.06
S_{2b}	#timeout.	(+11) 144	(+10) 125	(+13) 146	(+13) 128	(-44) 372	(-56) 782	(-26) 390	(-52) 748
	total time [h]	(+2.47) 31.02	(+0.72) 26.88	(+1.99) 30.60	(+2.60) 28.44	(-10.15) 70.92	(-12.31) 154.47	(-5.45) 75.03	(-12.34) 147.48
S_{AG}	#timeout.	(-16) 550	(-14) 317	(-1) 0	(-1) 1	(-6) 5	(-5) 10	(-23) 31	(-31) 28
	total time [h]	(-3.19) 98.86	(-2.72) 59.54	(-0.25) 4.26	(-0.54) 4.11	(-2.15) 5.39	(-1.09) 6.40	(-3.15) 12.01	(-5.30) 10.03

(b) $p_{prop} = 1, p_{opp} = 0.5$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-123) 70	(-134) 45	(+0) 0	(-2) 0	(-12) 1	(-13) 0	(-23) 13	(-21) 18
	total time [h]	(-20.87) 17.52	(-21.83) 13.87	(-0.09) 3.64	(-0.67) 3.38	(-2.89) 4.48	(-3.86) 3.98	(-5.44) 6.40	(-5.62) 6.94
S_{1a}	#timeout.	(-303) 565	(-60) 309	(-9) 0	(-11) 0	(-16) 1	(-21) 1	(-56) 26	(-54) 22
	total time [h]	(-49.22) 104.75	(-7.17) 61.33	(-2.68) 3.69	(-2.90) 3.42	(-4.82) 4.47	(-5.56) 3.69	(-12.36) 8.88	(-11.74) 8.78
S_{1b}	#timeout.	(-108) 85	(-110) 69	(-103) 92	(-117) 63	(-161) 111	(-161) 75	(-23) 277	(-26) 231
	total time [h]	(-17.46) 20.78	(-18.12) 17.67	(-17.83) 20.72	(-19.08) 16.61	(-27.01) 25.22	(-26.36) 18.97	(-0.53) 55.75	(-1.69) 47.15
S_2	#timeout.	(-90) 43	(-93) 22	(+0) 0	(+0) 0	(-145) 35	(-198) 62	(-135) 32	(-191) 49
	total time [h]	(-16.76) 11.88	(-17.56) 8.52	(-0.03) 3.03	(+0.69) 3.81	(-28.48) 12.34	(-40.14) 16.27	(-28.02) 11.10	(-38.43) 14.61
S_{2a}	#timeout.	(-322) 438	(-126) 211	(-1) 0	(-4) 0	(-144) 37	(-203) 59	(-140) 33	(-202) 50
	total time [h]	(-53.82) 82.61	(-18.95) 43.03	(-0.71) 3.75	(-1.13) 3.75	(-29.06) 12.04	(-40.14) 16.34	(-28.47) 11.33	(-39.19) 15.27
S_{2b}	#timeout.	(-98) 35	(-90) 25	(-82) 51	(-89) 26	(-156) 260	(-401) 437	(-163) 253	(-408) 392
	total time [h]	(-17.89) 10.66	(-17.77) 8.39	(-15.11) 13.50	(-17.08) 8.76	(-23.26) 57.81	(-69.01) 97.77	(-23.47) 57.01	(-70.86) 88.96
S_{AG}	#timeout.	(-296) 270	(-147) 184	(-1) 0	(-2) 0	(-10) 1	(-12) 3	(-34) 20	(-33) 26
	total time [h]	(-47.36) 54.69	(-21.08) 41.18	(-0.91) 3.60	(-1.46) 3.19	(-3.77) 3.77	(-3.04) 4.45	(-7.62) 7.54	(-7.17) 8.16

(c) $p_{prop} = 0.5, p_{opp} = 0.5$

Table 13: "Dynamic" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.5.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-151) 42	(-141) 38	(+0) 0	(-2) 0	(-11) 2	(-11) 2	(-12) 24	(-17) 22
	total time [h]	(-24.87) 13.52	(-22.12) 13.58	(+0.02) 3.75	(-0.64) 3.41	(-1.33) 6.04	(-2.56) 5.28	(-3.25) 8.59	(-4.60) 7.96
S_{1a}	#timeout.	(-189) 679	(-24) 345	(-9) 0	(-11) 0	(-11) 6	(-14) 8	(-35) 47	(-31) 45
	total time [h]	(-29.63) 124.34	(-3.23) 65.27	(-2.36) 4.01	(-2.74) 3.58	(-2.71) 6.58	(-3.68) 5.57	(-6.74) 14.50	(-6.16) 14.36
S_{1b}	#timeout.	(-145) 48	(-143) 36	(-149) 46	(-151) 29	(-177) 95	(-174) 62	(-64) 236	(-53) 204
	total time [h]	(-22.84) 15.40	(-23.53) 12.26	(-23.29) 15.26	(-23.78) 11.91	(-27.27) 24.96	(-25.93) 19.40	(-7.79) 48.49	(-5.30) 43.54
S_2	#timeout.	(-107) 26	(-102) 13	(+0) 0	(+0) 0	(-64) 116	(-77) 183	(-58) 109	(-78) 162
	total time [h]	(-20.19) 8.45	(-19.89) 6.19	(-0.04) 3.02	(+0.93) 4.05	(-12.98) 27.84	(-13.58) 42.83	(-11.98) 27.14	(-14.35) 38.69
S_{2a}	#timeout.	(-202) 558	(-39) 298	(-1) 0	(-4) 0	(-66) 115	(-80) 182	(-62) 111	(-77) 175
	total time [h]	(-33.43) 103.0	(-3.66) 58.32	(-1.08) 3.38	(-1.58) 3.30	(-12.89) 28.21	(-14.58) 41.90	(-12.52) 27.28	(-15.51) 38.95
S_{2b}	#timeout.	(-102) 31	(-106) 9	(-103) 30	(-104) 11	(-55) 361	(-248) 590	(-56) 360	(-255) 545
	total time [h]	(-17.89) 10.66	(-20.36) 5.80	(-19.23) 9.38	(-19.57) 6.27	(-6.81) 74.26	(-40.38) 126.40	(-5.90) 74.58	(-42.52) 117.30
S_{AG}	#timeout.	(-193) 373	(-101) 230	(-1) 0	(-2) 0	(-7) 4	(-7) 8	(-5) 49	(-11) 48
	total time [h]	(-27.94) 74.11	(-13.69) 48.57	(-0.55) 3.96	(-0.84) 3.81	(-1.85) 5.69	(-2.06) 5.43	(+0.96) 16.12	(-2.24) 13.09

(a) $p_{prop} = 0.6, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+43) 236	(+55) 234	(+0) 0	(+0) 2	(-4) 9	(-3) 10	(-7) 29	(-3) 36
	total time [h]	(+7.75) 46.14	(+9.90) 45.60	(+0.67) 4.40	(+0.73) 4.78	(-1.23) 6.14	(-0.96) 6.88	(-0.30) 11.54	(+0.27) 12.83
S_{1a}	#timeout.	(-6) 862	(-7) 362	(+1) 10	(+0) 11	(-2) 15	(-5) 17	(-19) 63	(-9) 67
	total time [h]	(-0.72) 153.25	(-1.01) 67.49	(-0.07) 6.30	(+1.08) 7.40	(-0.73) 8.56	(-1.28) 7.97	(-3.67) 17.57	(-2.08) 18.44
S_{1b}	#timeout.	(+53) 246	(+55) 234	(+45) 240	(+42) 222	(+21) 293	(+23) 259	(+46) 346	(+29) 286
	total time [h]	(+8.81) 47.05	(+9.91) 45.70	(+7.80) 46.35	(+7.61) 43.30	(+4.27) 56.50	(+4.62) 49.95	(+8.43) 64.71	(+5.03) 53.87
S_2	#timeout.	(+32) 165	(+1) 116	(+0) 0	(+0) 0	(-66) 114	(-104) 156	(-63) 104	(-100) 140
	total time [h]	(+7.40) 36.04	(+0.60) 26.68	(+0.14) 3.20	(-0.16) 2.96	(-11.86) 28.96	(-20.16) 36.25	(-11.62) 27.50	(-18.91) 34.13
S_{2a}	#timeout.	(-36) 724	(-73) 264	(+0) 1	(+1) 5	(-67) 114	(-110) 152	(-55) 118	(-111) 141
	total time [h]	(-4.58) 131.85	(-11.81) 50.17	(+0.39) 4.85	(+0.69) 5.57	(-12.26) 28.84	(-20.88) 35.60	(-9.15) 30.65	(-20.68) 33.78
S_{2b}	#timeout.	(+16) 149	(+10) 125	(+18) 151	(+18) 133	(-19) 397	(-14) 824	(+4) 420	(-19) 781
	total time [h]	(+3.02) 31.57	(+1.95) 28.11	(+4.38) 32.99	(+2.82) 28.66	(-5.39) 75.68	(-5.18) 161.60	(-1.22) 79.26	(-6.29) 153.53
S_{AG}	#timeout.	(-14) 552	(-10) 321	(+0) 1	(+1) 3	(-3) 8	(-5) 10	(-14) 40	(-21) 38
	total time [h]	(-2.47) 99.58	(-1.71) 60.55	(-0.05) 4.46	(+0.03) 4.68	(-0.68) 6.86	(-0.66) 6.83	(-2.73) 12.43	(-3.43) 11.90

(b) $p_{prop} = 1, p_{opp} = 0.6$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-121) 72	(-129) 50	(+0) 0	(-2) 0	(-13) 0	(-10) 3	(-16) 20	(-18) 21
	total time [h]	(-19.15) 19.24	(-20.29) 15.41	(-0.27) 3.46	(-0.71) 3.34	(-3.70) 3.67	(-3.28) 4.56	(-3.64) 8.20	(-2.95) 9.61
S_{1a}	#timeout.	(-207) 661	(-15) 354	(-9) 0	(-11) 0	(-14) 3	(-17) 5	(-50) 32	(-41) 35
	total time [h]	(-33.72) 120.25	(-1.61) 66.89	(-2.75) 3.62	(-2.61) 3.71	(-4.75) 4.54	(-4.78) 4.47	(-10.13) 11.11	(-8.23) 12.29
S_{1b}	#timeout.	(-127) 66	(-129) 50	(-123) 72	(-126) 54	(-153) 119	(-147) 89	(-11) 289	(-14) 243
	total time [h]	(-19.72) 18.52	(-19.89) 15.90	(-18.30) 20.25	(-18.85) 16.84	(-23.16) 29.07	(-23.35) 21.98	(+2.24) 58.52	(+0.56) 49.40
S_2	#timeout.	(-99) 34	(-96) 19	(+0) 0	(+0) 0	(-131) 49	(-183) 77	(-123) 44	(-171) 69
	total time [h]	(-17.88) 10.76	(-17.69) 8.39	(+0.17) 3.23	(+0.17) 3.29	(-25.48) 15.34	(-36.68) 19.73	(-24.25) 14.87	(-33.77) 19.27
S_{2a}	#timeout.	(-242) 518	(-113) 224	(-1) 0	(-4) 0	(-132) 49	(-189) 73	(-130) 43	(-180) 72
	total time [h]	(-39.60) 96.83	(-16.13) 45.85	(-1.37) 3.09	(-1.38) 3.50	(-25.14) 15.96	(-36.16) 20.32	(-25.38) 14.42	(-34.95) 19.51
S_{2b}	#timeout.	(-95) 38	(-95) 20	(-88) 45	(-101) 14	(-113) 303	(-274) 564	(-76) 340	(-274) 526
	total time [h]	(-17.70) 10.85	(-19.04) 7.12	(-16.41) 12.20	(-19.44) 6.40	(-16.14) 64.93	(-49.13) 117.65	(-10.08) 70.40	(-48.41) 111.41
S_{AG}	#timeout.	(-215) 351	(-110) 221	(-1) 0	(-2) 0	(-9) 2	(-11) 4	(-28) 26	(-22) 37
	total time [h]	(-32.59) 69.46	(-15.14) 47.12	(-0.88) 3.63	(-1.06) 3.59	(-3.20) 4.34	(-2.18) 5.31	(-5.74) 9.42	(-3.71) 11.62

(c) $p_{prop} = 0.6, p_{opp} = 0.6$

Table 14: "Dynamic" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.6.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-88) 105	(-77) 102	(+0) 0	(-2) 0	(-8) 5	(-6) 7	(-7) 29	(-12) 27
	total time [h]	(-10.57) 27.82	(-10.52) 25.18	(+0.25) 3.98	(-0.58) 3.47	(-1.90) 5.47	(-2.08) 5.76	(-1.42) 10.42	(-1.54) 11.02
S_{1a}	#timeout.	(-74) 794	(+12) 381	(-9) 0	(-11) 0	(-8) 9	(-12) 10	(-29) 53	(-14) 62
	total time [h]	(-10.63) 143.34	(+2.49) 70.99	(-1.98) 4.39	(-1.19) 5.13	(-2.73) 6.56	(-2.58) 6.67	(-4.45) 16.79	(-3.09) 17.43
S_{1b}	#timeout.	(-84) 109	(-79) 100	(-91) 104	(-86) 94	(-115) 157	(-90) 146	(-23) 277	(-3) 254
	total time [h]	(-10.28) 27.96	(-10.76) 25.03	(-12.17) 26.38	(-10.29) 25.40	(-17.49) 34.74	(-13.10) 32.23	(-2.13) 54.15	(+1.52) 50.36
S_2	#timeout.	(-86) 47	(-88) 27	(+0) 0	(+0) 0	(-36) 144	(-47) 213	(-34) 133	(-45) 195
	total time [h]	(-14.32) 14.32	(-15.49) 10.59	(-0.05) 3.01	(+1.09) 4.21	(-7.77) 33.05	(-8.94) 47.47	(-7.93) 31.19	(-8.35) 44.69
S_{2a}	#timeout.	(-86) 674	(-11) 326	(-1) 0	(-4) 0	(-37) 144	(-51) 211	(-36) 137	(-52) 200
	total time [h]	(-13.12) 123.31	(+0.04) 62.02	(-0.47) 3.99	(-0.89) 3.99	(-6.54) 34.56	(-9.73) 46.75	(-7.41) 32.39	(-9.46) 45.0
S_{2b}	#timeout.	(-87) 46	(-73) 42	(-94) 39	(-78) 37	(-19) 397	(-116) 722	(-25) 391	(-127) 673
	total time [h]	(-13.83) 14.72	(-13.65) 12.51	(-16.06) 12.55	(-13.72) 12.12	(-1.29) 79.78	(-15.97) 150.81	(-2.19) 78.29	(-20.25) 139.57
S_{AG}	#timeout.	(-76) 490	(-19) 312	(-1) 0	(-2) 0	(-6) 5	(-5) 10	(-5) 49	(-4) 55
	total time [h]	(-11.40) 90.65	(-2.53) 59.73	(-0.93) 3.58	(-0.60) 4.05	(-1.01) 6.53	(-1.09) 6.40	(-1.34) 13.82	(+0.98) 16.31

(a) $p_{prop} = 0.8, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+24) 217	(+22) 201	(+0) 0	(+1) 3	(-1) 12	(-2) 11	(+0) 36	(+7) 46
	total time [h]	(+4.15) 42.54	(+3.94) 39.64	(+1.02) 4.75	(+2.75) 6.80	(-0.30) 7.07	(-0.28) 7.56	(+0.43) 12.27	(+1.55) 14.11
S_{1a}	#timeout.	(-2) 866	(-5) 364	(+0) 9	(-2) 9	(+0) 17	(-1) 21	(-4) 78	(-1) 75
	total time [h]	(+0.13) 154.10	(-0.88) 67.62	(-0.05) 6.32	(+0.43) 6.75	(+0.67) 9.96	(+0.24) 9.49	(-0.26) 20.98	(-0.25) 20.27
S_{1b}	#timeout.	(+24) 217	(+26) 205	(+20) 215	(+18) 198	(+14) 286	(+18) 254	(+25) 325	(+13) 270
	total time [h]	(+4.43) 42.67	(+4.40) 40.19	(+3.55) 42.10	(+3.64) 39.33	(+2.75) 54.98	(+3.52) 48.85	(+6.30) 62.58	(+2.15) 50.99
S_2	#timeout.	(+8) 141	(+4) 119	(+0) 0	(+0) 0	(-37) 143	(-70) 190	(-33) 134	(-61) 179
	total time [h]	(+1.53) 30.17	(+1.25) 27.33	(+0.36) 3.42	(+0.25) 3.37	(-5.84) 34.98	(-12.28) 44.13	(-5.72) 33.40	(-10.81) 42.23
S_{2a}	#timeout.	(-4) 756	(-30) 307	(+1) 2	(+0) 4	(-42) 139	(-65) 197	(-29) 144	(-57) 195
	total time [h]	(-0.82) 135.61	(-3.85) 58.13	(+0.46) 4.92	(-0.03) 4.85	(-6.83) 34.27	(-11.78) 44.70	(-4.96) 34.84	(-10.78) 43.68
S_{2b}	#timeout.	(+11) 144	(+8) 123	(+7) 140	(+21) 136	(+38) 454	(+53) 891	(+53) 469	(+44) 844
	total time [h]	(+3.82) 32.37	(+1.50) 27.66	(+1.95) 30.56	(+3.12) 28.96	(+4.82) 85.89	(+6.69) 173.47	(+7.98) 88.46	(+6.38) 166.20
S_{AG}	#timeout.	(-2) 564	(+3) 334	(+0) 1	(+1) 3	(+0) 11	(-4) 11	(-7) 47	(-11) 48
	total time [h]	(-0.05) 102.0	(+0.55) 62.81	(-0.25) 4.26	(+0.27) 4.92	(-0.13) 7.41	(-0.06) 7.43	(-0.78) 14.38	(-0.99) 14.34

(b) $p_{prop} = 1, p_{opp} = 0.8$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-53) 140	(-63) 116	(+0) 0	(-2) 0	(-7) 6	(-8) 5	(-12) 24	(-18) 21
	total time [h]	(-5.65) 32.74	(-7.72) 27.98	(+0.71) 4.44	(-0.77) 3.28	(-1.97) 5.40	(-2.92) 4.92	(-2.36) 9.48	(-3.69) 8.87
S_{1a}	#timeout.	(-72) 796	(+8) 377	(-7) 2	(-9) 2	(-10) 7	(-12) 10	(-27) 55	(-19) 57
	total time [h]	(-10.77) 143.20	(+1.05) 69.55	(-1.78) 4.59	(-0.94) 5.38	(-2.55) 6.74	(-2.75) 6.50	(-4.15) 17.09	(-3.88) 16.64
S_{1b}	#timeout.	(-69) 124	(-63) 116	(-68) 127	(-67) 113	(-97) 175	(-74) 162	(+3) 303	(+8) 265
	total time [h]	(-6.83) 31.41	(-6.92) 28.87	(-7.92) 30.63	(-7.87) 27.82	(-14.03) 38.20	(-11.45) 33.88	(+3.13) 59.41	(+3.02) 51.86
S_2	#timeout.	(-84) 49	(-79) 36	(+0) 0	(+0) 0	(-84) 96	(-108) 152	(-63) 104	(-100) 140
	total time [h]	(-14.36) 14.28	(-13.99) 12.09	(+0.17) 3.23	(+0.51) 3.63	(-14.08) 26.74	(-20.15) 36.26	(-11.67) 27.45	(-19.29) 33.75
S_{2a}	#timeout.	(-111) 649	(-39) 298	(-1) 0	(-4) 0	(-81) 100	(-116) 146	(-71) 102	(-114) 138
	total time [h]	(-17.10) 119.33	(-4.53) 57.45	(-0.84) 3.62	(-1.18) 3.70	(-14.63) 26.47	(-21.73) 34.75	(-13.19) 26.61	(-21.69) 32.77
S_{2b}	#timeout.	(-84) 49	(-77) 38	(-85) 48	(-72) 43	(-4) 412	(-94) 744	(+8) 424	(-91) 709
	total time [h]	(-13.27) 15.28	(-13.41) 12.75	(-13.62) 14.99	(-12.40) 13.44	(+0.62) 81.69	(-14.21) 152.57	(+2.11) 82.59	(-15.67) 144.15
S_{AG}	#timeout.	(-80) 486	(-24) 307	(-1) 0	(-2) 0	(-4) 7	(-5) 10	(-12) 42	(-11) 48
	total time [h]	(-12.53) 89.52	(-3.29) 58.97	(-0.70) 3.81	(-1.03) 3.62	(-1.72) 5.82	(-0.98) 6.51	(-0.89) 14.27	(-1.74) 13.59

(c) $p_{prop} = 0.8, p_{opp} = 0.8$

Table 15: "Dynamic" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.8.

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-32) 161	(-26) 153	(+0) 0	(-2) 0	(-1) 12	(-3) 10	(-9) 27	(-9) 30
	total time [h]	(-4.59) 33.80	(-3.31) 32.39	(-0.01) 3.72	(-0.39) 3.66	(+0.57) 7.94	(-1.20) 6.64	(-1.19) 10.65	(-1.64) 10.92
S_{1a}	#timeout.	(-30) 838	(+13) 382	(-6) 3	(-4) 7	(-3) 14	(-5) 17	(-10) 72	(+1) 77
	total time [h]	(-4.44) 149.53	(+2.51) 71.01	(-0.98) 5.39	(-0.69) 5.63	(-1.06) 8.23	(-0.30) 8.95	(-1.37) 19.87	(+1.44) 21.96
S_{1b}	#timeout.	(-38) 155	(-30) 149	(-39) 156	(-29) 151	(-64) 208	(-52) 184	(-11) 289	(-4) 253
	total time [h]	(-4.81) 33.43	(-4.24) 31.55	(-4.58) 33.97	(-3.61) 32.08	(-10.65) 41.58	(-8.40) 36.93	(+0.11) 56.39	(+0.41) 49.25
S_2	#timeout.	(-53) 80	(-47) 68	(+0) 0	(+0) 0	(-18) 162	(-25) 235	(-22) 145	(-28) 212
	total time [h]	(-7.59) 21.05	(-7.78) 18.30	(+0.09) 3.15	(+0.30) 3.42	(-3.76) 37.06	(-5.62) 50.79	(-4.74) 34.38	(-5.19) 47.85
S_{2a}	#timeout.	(-34) 726	(+3) 340	(-1) 0	(-4) 0	(-22) 159	(-27) 235	(-21) 152	(-31) 221
	total time [h]	(-5.29) 131.14	(+1.65) 63.63	(+0.33) 4.79	(-0.42) 4.46	(-4.71) 36.39	(-3.60) 52.88	(-4.14) 35.66	(-5.72) 48.74
S_{2b}	#timeout.	(-48) 85	(-46) 69	(-52) 81	(-37) 78	(+3) 419	(-60) 778	(-1) 415	(-55) 745
	total time [h]	(-7.85) 20.70	(-7.62) 18.54	(-7.60) 21.01	(-6.62) 19.22	(+0.94) 82.01	(-7.38) 159.40	(+1.70) 82.18	(-7.62) 152.20
S_{AG}	#timeout.	(-32) 534	(-14) 317	(-1) 0	(-2) 0	(-3) 8	(-5) 10	(-3) 51	(-4) 55
	total time [h]	(-4.87) 97.18	(-0.95) 61.31	(-0.55) 3.96	(-0.74) 3.91	(-0.71) 6.83	(-0.67) 6.82	(-0.13) 15.03	(-0.33) 15.0

(a) $p_{prop} = 0.9, p_{opp} = 1$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(+13) 206	(+11) 190	(+0) 0	(-1) 1	(-1) 12	(-1) 12	(+1) 37	(+0) 39
	total time [h]	(+2.52) 40.91	(+2.29) 37.99	(+0.51) 4.24	(-0.14) 3.91	(+0.50) 7.87	(-0.28) 7.56	(+0.90) 12.74	(+0.37) 12.93
S_{1a}	#timeout.	(+0) 868	(+8) 377	(+0) 9	(+0) 11	(+0) 17	(-1) 21	(-1) 81	(-2) 74
	total time [h]	(+0.42) 154.39	(+2.33) 70.83	(+0.17) 6.54	(+0.34) 6.66	(+0.41) 9.70	(+0.15) 9.40	(+0.51) 21.75	(-0.01) 20.51
S_{1b}	#timeout.	(+12) 205	(+10) 189	(+16) 211	(+10) 190	(+5) 277	(+5) 241	(+6) 306	(+7) 264
	total time [h]	(+2.98) 41.22	(+1.70) 37.49	(+3.05) 41.60	(+2.30) 37.99	(+1.67) 53.90	(+1.54) 46.87	(+1.25) 57.53	(+1.27) 50.11
S_2	#timeout.	(+8) 141	(+6) 121	(+0) 0	(+0) 0	(-15) 165	(-30) 230	(-14) 153	(-30) 210
	total time [h]	(+1.41) 30.05	(+0.88) 26.96	(+0.71) 3.77	(+0.55) 3.67	(-2.41) 38.41	(-5.52) 50.89	(-2.22) 36.90	(-5.56) 47.48
S_{2a}	#timeout.	(+0) 760	(-18) 319	(+0) 1	(+0) 4	(-13) 168	(-35) 227	(-17) 156	(-38) 214
	total time [h]	(+0.08) 136.51	(-2.38) 59.60	(+0.28) 4.74	(-0.05) 4.83	(-0.47) 40.63	(-6.14) 50.34	(-2.75) 37.05	(-6.15) 48.31
S_{2b}	#timeout.	(+9) 142	(+2) 117	(+6) 139	(+4) 119	(+64) 480	(+71) 909	(+61) 477	(+66) 866
	total time [h]	(+1.28) 29.83	(+0.58) 26.74	(+1.14) 29.75	(+1.02) 26.86	(+9.38) 90.45	(+10.83) 177.61	(+9.90) 90.38	(+10.11) 169.93
S_{AG}	#timeout.	(+1) 567	(+1) 332	(+1) 2	(+0) 2	(+1) 12	(+0) 15	(-5) 49	(-5) 54
	total time [h]	(-0.03) 102.02	(+0.96) 63.22	(+0.51) 5.02	(+0.49) 5.14	(-0.05) 7.49	(+0.76) 8.25	(-0.46) 14.70	(-0.12) 15.21

(b) $p_{prop} = 1, p_{opp} = 0.9$

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	(-32) 161	(-13) 166	(+0) 0	(-2) 0	(-3) 10	(-2) 11	(-5) 31	(-11) 28
	total time [h]	(-3.09) 35.30	(-1.62) 34.08	(-0.02) 3.71	(-0.56) 3.49	(+0.65) 8.02	(-0.66) 7.18	(-1.25) 10.59	(-1.73) 10.83
S_{1a}	#timeout.	(-30) 838	(+7) 376	(-8) 1	(-7) 4	(-5) 12	(-8) 14	(-10) 72	(-13) 63
	total time [h]	(-4.89) 149.08	(+0.71) 69.21	(-1.08) 5.29	(-0.86) 5.46	(-1.73) 7.56	(-1.40) 7.85	(-1.95) 19.29	(-2.02) 18.50
S_{1b}	#timeout.	(-26) 167	(-13) 166	(-27) 168	(-14) 166	(-64) 208	(-41) 195	(+3) 303	(+1) 258
	total time [h]	(-3.40) 34.84	(-1.63) 34.16	(-3.21) 35.34	(-1.11) 34.58	(-9.48) 42.75	(-6.0) 39.33	(+1.46) 57.74	(+1.33) 50.17
S_2	#timeout.	(-44) 89	(-50) 65	(+0) 0	(+0) 0	(-38) 142	(-63) 197	(-40) 127	(-66) 174
	total time [h]	(-7.02) 21.62	(-7.72) 18.36	(+0.37) 3.43	(+0.43) 3.55	(-5.94) 34.88	(-12.47) 43.94	(-7.05) 32.07	(-11.98) 41.06
S_{2a}	#timeout.	(-28) 732	(-10) 327	(-1) 0	(-3) 1	(-47) 134	(-70) 192	(-43) 130	(-62) 190
	total time [h]	(-4.07) 132.36	(-0.61) 61.37	(-0.15) 4.31	(-0.91) 3.97	(-8.51) 32.59	(-12.94) 43.54	(-7.33) 32.47	(-10.56) 43.90
S_{2b}	#timeout.	(-49) 84	(-35) 80	(-47) 86	(-38) 77	(+60) 476	(+7) 845	(+55) 471	(-9) 791
	total time [h]	(-8.04) 20.51	(-6.48) 19.68	(-7.25) 21.36	(-6.30) 19.54	(+9.25) 90.32	(+1.80) 168.58	(+9.82) 90.30	(-1.57) 158.25
S_{AG}	#timeout.	(-30) 536	(-20) 311	(-1) 0	(-2) 0	(-3) 8	(-5) 10	(-4) 50	(-5) 54
	total time [h]	(-4.91) 97.14	(-3.54) 58.72	(-0.72) 3.79	(-0.74) 3.91	(-0.90) 6.64	(-0.48) 7.01	(-0.97) 14.19	(-0.86) 14.47

(c) $p_{prop} = 0.9, p_{opp} = 0.9$

Table 16: "Dynamic" rule sampling results for combinations of (p_{prop}, p_{opp}) with value 0.9.

Grounded and Preferred Semantics

Table 17 and Table 18 present the obtained results for grounded semantics using `abagraph` and `flexABle` respectively. Additionally, the latter table includes results for preferred semantics using `flexABle`.

Algorithms for grounded and preferred semantics involve high exponential complexity, as the correct subset of assumptions has to be guessed, thus posing a problem for the algorithm used in `flexABle`. Since computation exceeded our time limits (and anyway the results obtained in the

meantime were not optimistic), we decided to present the results in a slightly different manner. For each setup we show the results obtained within the given time-frame, i.e. 307h for grounded and 300h for the preferred semantics. Within that time-frame each `flexABle` setup was fed the same sequence of benchmarks. Note that in Table 18 we additionally introduce a "#solved" column denoting the number of instances for which the system returned an answer without running out of time, without which it would be difficult to interpret the results.

Results for grounded semantics show the overwhelming performance-wise dominance of `abagraph` over `flexABle`. For instance, as presented in Table 17, Strategy 3 calculated answers for 6903 instances, timing out only for 97 of them (1.4% of all examined instances) while the entire calculation took little over 73 hours. On the other hand, one of the "fastest" `flexABle` setups (S_1, BFS, Start w/`DABF + TA`) examined only 4035 instances, timing out for 635 of them (15%), requiring over 4 times more time. This clearly indicates that the algorithm for grounded (and most probably for preferred) semantics does not perform well and is not the optimal approach.

	ABAGRAPH strategy				
	1	2	3	4	5
#timeout.	111	210	69	410	111
total time [h]	45.66	81.66	30.96	145.80	45.40
95% time [h]	26.33	62.33	11.63	126.47	26.06
min [s]	0.64	0.66	0.66	0.68	0.64
median [s]	9.48	44.58	11.13	41	10.74
mean [s]	30.04	44.71	26.54	44.51	29.12
max [s]	1057.04	1000.78	1114.45	727.40	1159.01

Table 17: Results for the grounded semantics using `abagraph`. As before, green and red cell backgrounds indicate best and worst results obtained globally. Greyed-out cells represent setups which exceeded the total available running time

Str.		**grounded** semantics				**preferred** semantics			
		Start w/ DABF+TA		Start w/ DC+ TC		Start w/ DABF+TA		Start w/ DC+ TC	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#solved	3400	3400	1568	1524	2892	2819	1241	1176
	#timeout.	608	635	620	610	484	484	598	598
	total time [h]	306.09	306.60	306.35	303.90	289.28	287.43	299.08	289.04
	95% time [h]	11.41	9.92	125.33	140.89	13.86	15.01	120.07	131.70
	min [s]	1.38	1.38	1.42	1.87	1.80	2.17	2.14	2.13
	median [s]	71.64	61.78	190.96	200.13	98.98	106.71	252.85	242.05
	mean [s]	109.49	100.50	228.81	237.52	159.25	161.01	289.32	274.60
	max [s]	1130.66	1128.63	945.40	931.30	1061.00	1058.36	1062.28	1066.04
S_{1a}	#solved	3374	3380	1540	1540	2451	2641	1134	1141
	#timeout.	600	600	616	614	497	494	598	598
	total time [h]	303.79	306.26	306.28	305.74	287.28	292.89	287.82	288.92
	95% time [h]	11.44	11.65	135.93	136.05	17.92	16.53	144.48	143.25
	min [s]	1.29	1.35	1.51	1.45	2.13	2.38	2.07	2.14
	median [s]	71.16	73.32	189.36	191.91	111.58	111.23	245.96	249.32
	mean [s]	110.72	113.15	235.94	236.22	178.60	174.76	280.88	282.64
	max [s]	1001.16	1024.32	962.46	957.06	1165.33	1062.17	1012.63	1012.72
S_{1b}	#solved	1480	1480	840	825	1498	1485	499	499
	#timeout.	780	778	711	711	677	718	829	825
	total time [h]	305.19	305.41	306.44	306.00	283.47	289.70	299.74	298.52
	95% time [h]	100.17	101.03	306.44	306.00	24.08	22.95	291.08	291.19
	min [s]	1.59	1.48	1.51	1.48	1.66	1.82	3.20	2.84
	median [s]	47.45	47.85	65.36	62.66	62.17	59.94	79.29	78.97
	mean [s]	109.85	111.96	297.51	300.91	138.85	122.00	168.83	169.63
	max [s]	1073.25	1094.39	1195.69	1197.11	1168.80	1192.47	1131.09	1123.19
S_2	#solved	3400	3400	1510	1462	2916	2916	1017	1026
	#timeout.	612	626	602	611	501	492	625	625
	total time [h]	303.94	307.14	305.64	306.21	295.69	294.22	287.49	288.97
	95% time [h]	11.30	11.03	149.96	163.53	14.21	14.86	174.15	172.64
	min [s]	1.34	1.42	1.40	1.73	1.90	2.07	2.67	2.81
	median [s]	72.62	72.63	198.04	204.91	102.84	107.00	229.84	231.68
	mean [s]	105.80	104.24	250.22	252.45	158.85	160.74	280.17	282.92
	max [s]	939.37	930.81	1186.44	1197.39	941.85	950.88	1190.84	1179.55
S_{2a}	#solved	3400	3400	1416	1418	2772	2888	1009	1099
	#timeout.	609	609	620	620	484	484	625	625
	total time [h]	303.43	304.10	303.54	304.99	288.32	296.31	289.41	298.89
	95% time [h]	11.03	11.06	173.19	173.97	14.88	14.88	178.74	158.22
	min [s]	1.25	1.28	1.82	1.66	1.95	2.15	2.76	2.08
	median [s]	70.56	70.96	191.75	198.00	100.58	107.19	233.89	251.48
	mean [s]	106.31	107.02	246.22	249.55	164.90	168.22	289.26	296.63
	max [s]	981.16	985.17	1069.56	1098.82	1119.79	1133.39	1193.37	1185.43
S_{2b}	#solved	1570	1570	619	618	1617	1595	462	460
	#timeout.	805	822	815	811	750	729	811	799
	total time [h]	303.33	306.96	305.39	304.61	294.95	288.18	292.58	288.13
	95% time [h]	59.97	57.90	305.39	304.61	15.95	16.30	292.58	288.13
	min [s]	1.63	1.39	1.55	1.73	2.15	1.60	2.87	2.31
	median [s]	33.89	29.17	35.30	35.62	56.12	54.06	66.31	60.52
	mean [s]	80.18	75.44	195.99	199.46	100.01	101.89	173.26	170.52
	max [s]	1128.23	1141.28	1193.91	1194.95	1193.32	1189.46	1171.28	1181.22
S_{AG}	#solved	2620	2600	1096	1083	1810	1823	695	691
	#timeout.	550	548	640	649	543	547	678	680
	total time [h]	306.16	305.72	306.01	305.37	289.14	296.17	288.38	287.79
	95% time [h]	25.65	25.98	275.68	276.37	27.30	30.18	264.71	264.79
	min [s]	1.43	1.44	1.48	1.51	1.84	1.90	2.66	2.63
	median [s]	100.63	102.34	109.00	111.06	112.84	122.17	133.18	132.14
	mean [s]	168.74	170.35	304.35	295.90	215.05	224.76	323.07	318.40
	max [s]	1014.93	1035.74	1195.07	1199.30	1185.41	1191.99	1199.58	1151.79

Table 18: Results for the grounded and preferred semantics using `flexABle`.