UNIVERSITÄT KOBLENZ · LANDAU
Faculty 4: Computer Science

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Approximate Inference for Assumption-based Argumentation in AI

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Web Science

submitted by

Chuyi Sun

First supervisor:	PD Dr. Matthias Thimm Institute for Web Science and Technologies
Second supervisor:	Dr. Tjitze Rienstra Institute for Web Science and Technologies

Koblenz, March 2021

Statement

I hereby certify that this thesis has been composed by me and is based on my own work, that I did not use any further resources than specified – in particular no references unmentioned in the reference section – and that I did not submit this thesis to another examination before. The paper submission is identical to the submitted electronic version.

	Yes	No
I agree to have this thesis published in the library.	☐	☐
I agree to have this thesis published on the Web.	☐	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐
The source code is available under a GNU General Public License (GPLv3).	☐	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐

. . . .

(Place, Date) (Signature)

Note

- If you would like us to contact you for the graduation ceremony,
  please provide your personal E-mail address: . . . .
- If you would like us to send you an invite to join the WeST Alumni
  and Members group on LinkedIn, please provide your LinkedIn ID : . . . .

Abstract

This thesis focuses on approximate inference in assumption-based argumentation frameworks. Argumentation provides a significant idea in the computerization of theoretical and practical reasoning in AI. And it has a close connection with AI, engaging in arguments to perform scientific reasoning. The fundamental approach in this field is abstract argumentation frameworks developed by Dung. Assumption-based argumentation can be regarded as an instance of abstract argumentation with structured arguments. When facing a large scale of data, a challenge of reasoning in assumption-based argumentation is how to construct arguments and resolve attacks over a given claim with minimal cost of computation and acceptable accuracy at the same time. This thesis proposes and investigates approximate methods that randomly select and construct samples of frameworks based on graphical dispute derivations to solve this problem. The presented approach aims to improve reasoning performance and get an acceptable trade-off between computational time and accuracy. The evaluation shows that for reasoning in assumption-based argumentation, in general, the running time is reduced with the cost of slightly low accuracy by randomly sampling and constructing inference rules for potential arguments over a query.

Contents

1. Introduction

Over the last ten years, argumentation has come to be increasingly central as a core study within Artificial Intelligence (AI). Argumentation informally is concerned with how claims are proposed, discussed, and resolved in issues upon diverging opinions. Our aims in this introduction are, firstly, to present the motivation in the context of approximate reasoning in argumentation and, subsequently, to discuss two research questions that we address in this thesis and, finally, to give an overview structure of this thesis.

1.1. Motivation

Formal argumentation aims to reason with conflicting or defeasible information and derive a meaningful conclusion from it. One of the most prominent argumentation formalisms is abstract argumentation (AA) [1], which treats arguments as abstract entities. Assumption-based argumentation (ABA) was developed as a computational framework to solve reasoning problems. It was inspired by AA, and preferred extension semantics for logic programming [2]. The central innovation of ABA is that arguments and attacks are notions derived from primitive ideas of rules in a deductive system based on assumptions and contraries. ABA semantics provides a way to identify sets of arguments "surviving the conflict together" with the interpretation of an argumentation framework. Thus we can build a reasoner to answer a query or decide whether to accept sets of arguments under specific semantics. Although ABA is equipped with a number of computational mechanisms to determine whether a claim/conclusion can be supported by a "winning" set of arguments, many reasoning problems in this formalism usually have high computational complexity [3].

As reasoning in ABA usually has high computational complexity, when it comes to implementation issues, solving reasoning problems for large scale of data in an adequate time needs to be considered. It is becoming an important research topic of formal argumentation systems in AI to overcome this obstacle. For instance, the research [3] has shown that the problem of determining whether a set of assumptions is admissible is NP-complete for logic programming. It means that the process of computing is very time-demanding. From a practical perspective, this problem can be solved by sacrificing accuracy slightly to improve the performance by sample selections to construct approximately relevant arguments. With the price of accuracy, we can reduce the time of computing. This thesis aims to improve the reasoning process's performance in ABA, decreasing the running time with the price of lower accuracy using approximate inference.

Dung[1] proposed the concept of abstract argumentation framework and the acceptability of the sets of arguments for nonmonotonic reasoning as fundamental argumentation formalisms in AI. Assumption-based argumentation [2] inspired by AA is one of the structured argumentation formalisms. In contrast to AA, the internal structure of an argument is made explicit through derivations. Constructing a reasoner to answer acceptability queries over ABA frameworks efficiently is becoming a challenge in this field. Lehtonen et al. [4] have proved that counting the number of arguments satisfying a minimality condition in their support is #P-complete. Dvorak et al. [5] presented computational complexity analysis for reasoning problems under different formal argumentation formalisms, and most of them turn out to be of high complexity. Craven and Toni [6] introduced rule-minimal arguments and arguments graphs to solve conceptual redundancy and inefficiency in ABA. Several algorithms for determining the acceptability of sentences in ABA have been proposed, from dispute derivations presented by Dung et al. [7][8] to a generalized framework presented by Toni [9].

1.2. Research Questions

To guide our research, we design two main questions in this thesis:

Question 1 *How to perform approximate reasoning in assumption-based argumentation frameworks?*

In general, the reasoning process in assumption-based argumentation is to answer some queries from a knowledge base. The existing computational mechanisms are dispute derivations [7]. They have been defined in ABA for computing admissible, grounded, and ideal supports for conclusions/claims [8][9]. Usually, the query is whether the conclusion/claim is supported by a set of arguments accepted under specific semantics by a reasoning agent. In this kind of method, each step generates a dispute containing information about supporting and counter-attacking arguments by the proponent and opponent. The idea of approximate reasoning in this context is adding randomness by randomly selecting and constructing inference rules from an ABA framework in different stages of dispute derivations. There are many different variants of these computational mechanisms. Graphical dispute derivation [10] is a good way to answer the query under admissible/grounded semantics by avoiding *flabbyness* and *circularity*, which might add redundancy in arguments. It uses a graph to guarantee termination and completeness, and it checks whether the graph is cyclic after some operations. We implement the approximate reasoning based on the graphical dispute derivation under the grounded semantics by taking samples of inference rules on the proponent, opponent, and both sides in the process of derivations.

Question 2 *Whether approximate methods can reduce runtime with acceptable accuracy? And to what extent?*

We use experiments to evaluate approximate methods compared with the standard graphical dispute derivation as the baseline. And we design experiments to give statistical results of running time and accuracy by testing many different frameworks. To reach this goal, we use a random generator with different settings of parameters to generate the data of ABA frameworks. In the experiments, we use different scales of datasets to test the performance of these methods. Furthermore, the results show that one random sampling method can reduce the runtime still with an accuracy of about 98% in the best situation.

1.3. Outline

The rest of this thesis is organized as follows. Chapter 2 firstly summarizes relevant definitions and semantics of abstract argumentation. Moreover, this chapter introduces assumptions-based frameworks with their arguments and attacks. Subsequently, this chapter presents argument graphs that are useful for understanding methods in the next chapter. We mention complexity classes at the end of this chapter. Chapter 3 describes a general reasoning process in the assumption-based framework and more concrete methods in dispute derivations. This chapter also provides upper bounds of computational complexity for reasoning problems in assumption-based argumentation. Chapter 4 describes approximate inference methods in two categories: framework-based sampling and dispute-based sampling. Moreover, this chapter represents all of the approximate methods in algorithms and explains them with examples. Chapter 5 presents the evaluation of approximate methods compared with the baseline. The experiments show that some approximate methods can reduce runtime with the price of slightly lower accuracy. Chapter 6 puts forward the aspects of future work and concludes the whole research.

2. Foundations

This chapter introduces some definitions and notions of abstract argumentation and assumption-based argumentation as a fundamental part of this thesis. Moreover, this chapter provides some computational complexity classes, which are helpful to present the computational complexity of reasoning in assumption-based argumentation under different semantics. Section 2.1 introduces abstract argumentation (AA) and semantics in argumentation. Section 2.2 offers many definitions concerning assumption-based argumentation (ABA). Section 2.3 presents argument graphs that are useful to understand graphical dispute derivations. Section 2.4 gives basic classes of complexity as a foundation of complexity analysis in the next chapter.

2.1. Abstract Argumentation

Let us see an example about food. Marry likes pizza, but John likes pasta. Moreover, there are two restaurants: Pizza House and Pasta Home. One day they meet, and they have a conversation as follows.

Mary: "*Hi, John, how are you?*"

John: "*Hello, Mary, where are you going?*"

Mary: "*I am going to eat lunch, would you like to join me?*"

John: "*Of course, where?*"

Mary: "*Pizza House, do you like pizza?*"

John: "*I have an allergic to pizza, I like pasta. Would you like to go to Pasta Home.*"

By summarizing this conversation, we can get some statements about which restaurant they will go to if they eat together.

Statement 1: Mary likes pizza, and she suggests Pizza House.

Statement 2: John likes pasta, and he suggests Pasta Home.

If we abstract these two statements as arguments α and β, we could find a conflict about choosing the restaurant between them. The arguments and their relations are shown in Figure 1 to make them more intuitive. The circles represent arguments α and β. Because Mary wants to go to Pizza House, but John does not. He prefers Pasta Home. The two arguments attack each other. So one directed edge is from α to β, another directed edge is from β to α.

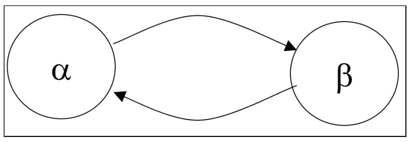

Figure 1: Two abstract arguments α, β and their relations

Abstract argumentation frameworks coined by Dung [1] abstracts from the concrete content of arguments and only considers the relation between them. We regard different semantics in argumentation as selecting the subsets of arguments satisfying specific criteria.

2.1.1. Formal Framework

Argumentation is a multi-faceted word with a variety of informal as well as formal meanings. The abstraction process detaches the word from some of its meanings and properties, keeping only those required by the desired abstraction level. Arguments and their relations are two components for an abstract argumentation framework. An argument is an abstract entity whose role is determined by its relations to other arguments. A general abstract framework centered on conflicts has a wide range of potential applications. We use the following steps to generate argumentation frameworks for different applications. Firstly, we identify an interesting application domain where conflict management plays a key role. Then we define a suitable formalization of problem instances in the selected domain. Finally, we describe the notions of arguments and attacks in formalization to construct an abstract framework. The formal definition of an abstract argumentation framework is from Dung [1] as follows.

Definition 2.1.1 (Abstract Argumentation Framework) *An abstract argumentation framework (F) is a pair (Args,R) where*

- *Args is a set of arguments*
- *$R \subseteq Args \times Args$ is a binary relation*

For two arguments $a, b \in Args$, if $(a, b) \in R$, we say that a attacks b. A set $S \subseteq Args$ defends an argument $a \in Args$, if, for each $b \in Args$ such that $(b, a) \in R$, there exists a $c \in S$, such that $(c, b) \in R$.

To illustrate this definition, we give an example as follows.

Example 2.1 *An abstract argumentation framework is* $F = (Args, R)$ *with* $Args = \{a, b, c, d, e\}$ $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$.

We focus on Example 2.1 and use Figure 2 to represent the framework in this example. In this figure, nodes represent arguments, and edges represent attacks. There are five nodes and six edges in this graph corresponding to $Args$ and R. Argument c is defended by the set of arguments S, where $S = \{a, c\}$. Although c is attacked by d, c can defend itself by attacking d.

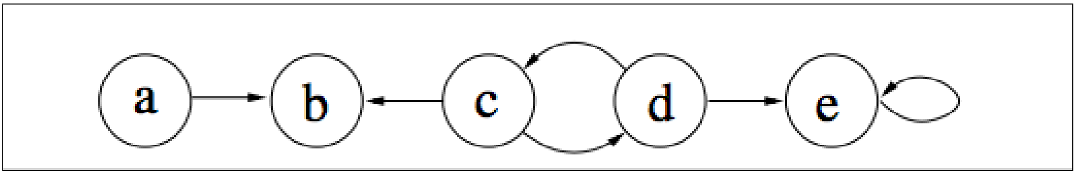

Figure 2: A representation for the framework in Example 2.1

2.1.2. Semantics

Semantics in argumentation is a pre-defined criterion to determine the sets of arguments that can be accepted and plays a vital role in reasoning. Because of attacks between arguments, we are interested in the justification state of arguments. An argument is justified if it has some ways to survive in attacks it receives. The process of determining the state of justification is called argument evaluation. Argumentation semantics is the formal definition of a method that rules the argument evaluation process. There are two main styles of argumentation semantics [11]: extension-based and labelling-based. Both approaches are virtually equivalent, and we can equivalently express any extension-based semantics in a labelling-based formulation. An extension is simply a set of arguments. We define extension-based semantics as a function that assigns a set of extensions to each argumentation framework F.

Definition 2.1.2 (Extension-based semantics) *An extension-based semantics $\mathbb{S}$ associates with each argumentation framework F = (Args, R) a subset of 2^{Args}, denoted as $\varepsilon_{\mathbb{S}}(F)$.*

In labelling-based semantics, let $F = (Args, R)$ be an argument framework and Λ a set of labels. A Λ-labelling is a total function $\mathcal{L}ab$: $Args \to \Lambda$. The set of all labelings of F is denoted as $\varsigma(\Lambda, F)$. A labelling-based semantics prescribes a set of labelings. Generally, if $\Lambda = \{in, out, undec\}$, labelling is a three-valued function that assigns one of the labels in, out, and $undec$ to each argument. An argument is labeled with: in if it is accepted, for example, it is defended by the in labeled arguments; out if it is rejected, for instance, it is attacked by an accepted argument; $undec$ if the argument is neither accepted nor attacked by accepted arguments. We denote labelling functions $\mathcal{L}$ by triples($\mathcal{L}_{in}$, $\mathcal{L}_{out}$, $\mathcal{L}_{undec}$), where $\mathcal{L}_{in}$ is the set of arguments labeled by in, $\mathcal{L}_{out}$ is the set of arguments labeled by out, and $\mathcal{L}_{undec}$ is the set of arguments labeled by $undec$.

Definition 2.1.3 (Labelling-based semantics) *For a set of labels Λ, a labelling-based semantics $\mathbb{S}$ associates with each argumentation framework F a subset of $\varsigma(\Lambda, F)$, denoted as $\mathcal{L}_{\mathbb{S}}(F)$.*

In general, when we compute several extensions under a given semantic $\mathbb{S}$, there are two types of inference from an argumentation system for an agent. Credulous inference: an argument is accepted if it belongs to at least one of the $\mathbb{S}$ extensions. Skeptical inference: an argument is accepted if it belongs to every $\mathbb{S}$ extension. The definition of two types is as follows [12].

Definition 2.1.4 *Given a semantics $\mathbb{S}$ and an argumentation framework F, an argument a is :*

- *skeptically accepted if and only if $\forall E \in \varepsilon_{\mathbb{S}}(F), a \in E$;*
- *credulously accepted if and only if $\exists E \in \varepsilon_{\mathbb{S}}(F), a \in E$.*

We define different semantics in the extension-based style by describing basic conflict-free semantics and five main semantics in the following contents. These definitions are summarized from Baroni et al. [12] and we show the relations between different semantics [13] in Figure 3.

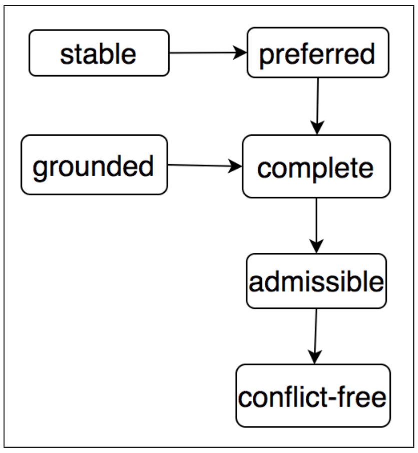

Figure 3: Relations between argumentation semantics: An arrow from a semantics x_1 to another semantics x_2 denotes that each x_1-extension is also a x_2-extension.

The essential requirement for any extension E corresponds to the idea that E is a set of arguments that "can stay together and not attack each other". This requirement is the principle of conflict-freeness. We present the definition of the conflict-free extension as follows.

Definition 2.1.5 (Conflict-free Extension) *Given an argumentation framework F = (Args, R), a set $S \subseteq Args$ is a conflict-free extension of F if for each $a, b \in S$, $(a, b) \notin R$. We denote $\varepsilon_{cf}(F)$ by the set of conflict-free extensions of F.*

If a set of arguments $S \subseteq Args$ defends an argument a, we also say that argument a is acceptable with respect to S. And the definition is as follows.

Definition 2.1.6 *Given an argumentation framework $F = (Args, R)$, an argument $a \in Args$ is acceptable w.r.t. a set $S \subseteq Args$ if for every argument b such that b attacks a, there exits some argument $c \in S$ such that c attacks b.*

A further requirement corresponds to the idea that an extension E is a set of arguments that "can survive and defend itself". It is the property of admissibility. Based on admissible extensions, other extensions can be easily defined.

Definition 2.1.7 (Admissible Extension) *Given an argumentation framework F = (Args, R), a set $S \subseteq Args$ is an admissible extension of F iff S is conflict-free and for all $a \in S$, a is acceptable w.r.t S. We denote by $\varepsilon_{adm}(F)$ the set of admissible extensions.*

We regard complete semantics as a strengthening of the basic requirement of admissible semantics. Complete extensions are based on admissible extensions with the limitation of containing all arguments it can defend. This extension defends arguments in this extension, and every argument that this extension can defend is also in it.

Definition 2.1.8 *Let F = (Args, R) be an argumentation framework. The function $\mathcal{F}$: 2^{Args} $\rightarrow 2^{Args}$ such that $\mathcal{F}$(Ar) = $\{a|$ Ar defends a, $a \in Args\}$ is called the characteristic function of this framework.*

Definition 2.1.9 (Complete extension) *Given an argumentation framework F = ($Args$, R), a set $S \subseteq$ Args is a complete extension of F iff S is admissible and S = $\mathcal{F}$(S). We denote by $\varepsilon_{com}(F)$ the set of complete extensions.*

The idea of grounded semantics is to accept only arguments that one cannot avoid to accept, to reject only the arguments that one cannot avoid to reject. A grounded extension is a minimal complete extension. The grounded extension is always unique.

Definition 2.1.10 (Grounded extension) *Given an argumentation framework F = (Arg-s , R), a set $S \subseteq$ Args is a grounded extension of F iff S is a minimal (w.r.t. $\subseteq$) complete extension of F. We denote by $\varepsilon_{grd}(F)$ the set of the grounded extension.*

When we consider the alternative view oriented at accepting as many arguments as reasonably possible, the idea of preferred semantics is to maximize accepted arguments.

Definition 2.1.11 (Preferred Extension) *Given an argumentation framework F=($Args$, R), a set $S \subseteq Args$ is a preferred extension of F iff S is a maximal (w.r.t. $\subseteq$) admissible extension of F. We denote by $\varepsilon_{prf}(F)$ the set of preferred extensions.*

Before defining a stable extension, we define the notion of Ar^+. In an argumentation framework $F = (Args, R)$, for $Ar \subseteq Args$ then we write Ar^+ for $\{b| \; \exists a \in Ar : (a, b) \in R, b \in Args\}$. Ar^+ represents the set of arguments that an argument a in Ar attacks. Stable semantics gives a view of "not bad is good". A set of arguments is a stable extension if it is conflict-free and attacks any argument outside the set. One argument is either in this extension and it is not attacked by other arguments in this extension or not in this extension and it is attacked by this extension.

Definition 2.1.12 (Stable extension) *Given an argumentation framework F = ($Args$, R), a set $S \subseteq$ Args is a stable extension of F iff S is conflict-free and $S \cup S^+$ = $Args$. We denote by $\varepsilon_{sta}(F)$ the set of stable extensions.*

Not every argumentation framework has a stable extension, so extensions are not guaranteed to exist under stable semantics.

Now we consider Example 2.1. The number of possible extensions is 2^5 because there are five arguments in F. One particular situation is an empty set which is a conflict-free extension because it always satisfies the requirement that no arguments attack each other. For extensions with one argument, except for argument e attacking itself, extensions containing other single arguments are conflict-free. Then we get conflict-free extensions as follows:

- $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$

When we consider extensions with two arguments, argument a and b, c and b, c and d, d and e can't stay together, because (a,b) $\in R$, (c,b) $\in R$, (c,d) $\in R$, (d,c) $\in R$ and (d,e) $\in R$. And extensions with two arguments can't contain argument e. Then we get conflict-free extensions as follows:

- $\{a, c\}$, $\{a, d\}$, $\{b, d\}$

According to conflict-free extensions containing two arguments, we create the set with three arguments. For the set $\{a, c\}$, we can't add any arguments because adding one of the arguments b, d, e will result in conflicts. For the same reason, we can't add any arguments to sets $\{a, d\}$ and $\{b, d\}$. So we don't find any conflict-free extensions with three arguments. Further, there is no need to expand the number of the arguments to four and five. We get the conflict-free extensions of F as follows:

◊ $\varepsilon_{cf}(F) = \{\{a, c\}, \{a, d\}, \{b, d\}, \{a\}, \{b\}, \{c\}, \{d\}, \emptyset\}$

Based on the conflict-free extensions, we can construct admissible extensions by judging every argument in each conflict-free extension whether the corresponding extension can defend them. For the set $\{a, c\}$, argument a is not attacked by any arguments. Although argument c is attacked by argument d, it defends itself by counter-attacking d. The set $\{a, c\}$ defends argument c. It is an admissible extension of the given argumentation framework. We construct each extension with the similar procedure, and finally the admissible extensions of F are:

◊ $\varepsilon_{adm}(F) = \{\{a, c\}, \{a, d\}, \{a\}, \{c\}, \{d\}, \emptyset\}$

Complete extensions are based on admissible extensions. The set $\{a, c\}$ defends argument c. And this extension contains c, so it is a complete extension. The set $\{a, d\}$ defends argument d, and it is also a complete extension. The complete extensions of F are:

◊ $\varepsilon_{com}(F) = \{\{a, c\}, \{a, d\}, \{a\}\}$

The grounded extension is unique. According to the definition, the grounded extension of F is the minimal complete extension as follows:

◊ $\varepsilon_{grd}(F) = \{\{a\}\}$

The definition of a preferred extension is the maximal admissible extension. We construct the preferred extensions of F as follows:

◊ $\varepsilon_{prf}(F) = \{\{a, c\}, \{a, d\}\}$

We obtain stable extensions from conflict-free extensions with the limitation of the arguments that are not in the stable extension are attacked by the arguments that are in the stable extension. $\{a, c\}^+$ is the set $\{b, d\}$ and $\{a, c\}^+ \cup \{a, c\} \neq Args$. If we continue to visit other conflict-free extensions of F, then we have the following stable extension of F:

◊ $\varepsilon_{stb}(F) = \{\{a, d\}\}$

This section began by introducing a scenario where two persons chose a restaurant. They could not reach an agreement because they had a conflict about their favorite food. We abstracted their statements as arguments and identified their conflicts as attacks. Then this section defined abstract arguments and frameworks. Any entity was an abstract argument, and the abstract frameworks were a pair of abstract arguments and their relations. Afterwards, this section presented different semantics in abstract argumentation frameworks, including conflict-free, admissible, complete, preferred, grounded, and stable in the extension-based style rather than in the labelling-based style. In the end, this section gave sets of arguments under different semantics with an example of the abstract argumentation framework.

2.2. Assumption-based Argumentation

Because of abstract argumentation abstracting away from the structure and meaning of arguments and attacks, it can hardly be adapted to model formalisms directly used by different applications. There is a wide gap between the practical problem and its representation as an abstract argument system. ABA was developed as a computational framework with its dialectical interpretation of different semantics to determine whether the given claim can be supported by a set of arguments. We derive arguments and attacks from a deductive system, assumptions, and contraries. Assumption-based argumentation is seen as an instance of AA with lower abstraction, and all the notions of semantics in AA can also be applied in ABA.

2.2.1. Deductive Systems in ABA

We describe a deductive system by a collection of judgments and a collection of steps, which has a list of judgments as hypotheses and a single judgment as a conclusion. The step is usually generated using inference rules, which is a schematic way of describing collections of steps, generally involving metavariables. It is a formal setup of reasoning. For a given language, a deductive system consists of a set of formulas in the language $\mathcal{L}$ and a set of binary relations $\mathcal{R}$ on subsets of the language. Each relation is called a rule of inference. The language is not limited to propositional atoms. It can be represented in different formalisms with propositional language, first-order language, etc. Each $\sigma \in \mathcal{L}$ is called a sentence in the language. The definition of deductive systems is as follows [7] [8].

Definition 2.2.1 (Deductive system) *A deductive system is a pair ($\mathcal{L}$,$\mathcal{R}$) where*

- *$\mathcal{L}$ is a formal language consisting of countably many sentences, and*
- *$\mathcal{R}$ is a countable set of inference rules of the form*

$$\frac{\sigma_1, ..., \sigma_n}{\sigma}$$

$\sigma \in \mathcal{L}$ is called the conclusion of the inference rule, $\sigma_1, ..., \sigma_n \in \mathcal{L}$ are called the premises of the inference rule and $n \geq 0$.

When $n = 0$, then the inference rule represents an axiom. Assumption-based argumentation frameworks are based on deductive systems, for convenience, we assume that the inference rules in $\mathcal{R}$ have the syntax $\sigma \leftarrow \sigma_1, ...\sigma_n$ (for $n \geq 0$) where $\sigma, \sigma_i \in \mathcal{L}$ and refer to σ as the *head* of the rule and $\sigma_1, ...\sigma_n$ as the *body* of the rule instead of using $\frac{\sigma_1,...,\sigma_n}{\sigma}$.

The deduction is a reasoning model. According to some premises, it can identify whether we can reach a logical conclusion. The process of a deduction describes how to obtain a decision from a set of beliefs. If we construct a path to get the desired conclusion from the forward perspective, then the deduction [7] is defined as follows:

Definition 2.2.2 (Forward deduction) *Given a deductive system ($\mathcal{L}$,$\mathcal{R}$), a forward deduction of a conclusion α based on (or supported by) a set of premises P is a sequence of of sets $\beta_1, ..., \beta_m$ of sentences in $\mathcal{L}$, where $m > 0$ and $\beta_m = \{\alpha\}$, and for every $1 \leq i < m$,*

- *$\beta_i \in P$ or*
- *there exists $\beta_i \leftarrow \alpha_1, ..., \alpha_n \in \mathcal{R}$ such that $\alpha_1, ...\alpha_n \in \{\beta_1, ..., \beta_{i-1}\}$.*

If there is a deduction of a conclusion α based on a set of premises P, we write as $P \vdash \alpha$. The forward deduction can contain the application of inference rules that are not relevant to the derivation of the conclusion, and it may contain premises that are not relevant to the rest of the deduction. For instance, if $P \vdash \alpha$, for any $P \subseteq P'$ then $P' \vdash \alpha$. When we change the perspective and view in a backward style, it might avoid some premises not relevant to the conclusion.

When we review the deduction in a backward style. The deduction can be seen as proof trees: the conclusion of the deduction is the root, and the leaves are labelled by the premises supporting the conclusion. Non-leave nodes matching the conclusion of an inference rule as a parent are connected to the children nodes of corresponding premises of the inference rule. Considering the fact that a proof tree can be labelled by the same sentence, multi-sets are used to describe several occurrences of a sentence precisely. To generate this kind of tree, we need to identify which node to expand next. We use a selection function f to formalize the selection strategy. The function f takes a sequence of multi-sets S_i as the input and returns a sentence occurrence in S_i as the output. Each S_i is a step in the deduction. The definition of backward deductions is as follows [7] [8].

Definition 2.2.3 (Backward deduction) *Given a deductive system ($\mathcal{L}$,$\mathcal{R}$) and a selection function f, a backward deduction of a conclusion σ based on (or supported by) a set of premises P is a sequence of multi-sets $S_1, ..., S_m$ of sentences in $\mathcal{L}$, where $m > 0$ and $S_1 = \{\sigma\}$, $S_m = P$, and for every $1 \leq i < m$, where α = $f(S_i)$:*

- *If α is not in P then $S_{i+1} = S_i - \{\alpha\} \cup S$ for some inference rule of the form $\alpha \leftarrow S \in \mathcal{R}$.*
- *If α is in P then $S_{i+1} = S_i$*

Definition 2.2.4 *Given a deductive system ($\mathcal{L}$,$\mathcal{R}$), a proof for $\sigma \in \mathcal{L}$ supported by $S \subseteq \mathcal{L}$ is a (finite) tree with nodes labelled by sentences in $\mathcal{L}$ or by τ (an empty set of premises), such that*

- *the root is labelled by σ*
- *for every node N*
  - *if N is a leaf then N is labelled either by an assumption or by τ;*

  - *if N is not a leaf and l_N is the label of N, then there is an inference rule $l_N \leftarrow b_1, ..., b_m (m \geq 0)$ and*
    either $m = 0$ and the child of N is τ
    or $m > 0$ and N has m children, labelled by $b_1, ..., b_m$ (respectively)

- *$\mathcal{S}$ is the set of all assumptions labelling the leaves.*

The above definitions are forward deductions, backward deductions, and proof. They refer to a similar process in a sense. Forward deductions and backward deductions can generate the same form of a conclusion supported by a set of premises. For example, $P \vdash \alpha$, the forward deduction is from left to right, and the backward deduction is from right to left to construct the internal process. In many literatures, the backward deduction is the common one. In the rest of the thesis, deductions refer to backward deductions. The proof describes the same inference processes as deductions. However, the proof is defined in a tree-based way as the core idea of *arguments* in the following subsection and gives a more intuitive structure for each node. Based on the definition of the proof, we define arguments and make them easier to understand.

2.2.2. Arguments and Attacks

In ABA, *arguments* are deductions of claims supported by sets of assumptions. Assumptions are sentences in the language that are open to challenge. And *attacks* are directed at the assumptions in the support of arguments [2][14]. An argument for a sentence $\sigma \in \mathcal{L}$ supported by a set of assumptions A is a defeasible proof of σ from A, obtained by applying backward the rules in $\mathcal{R}$ until only assumptions are left [9]. Arguments can be represented as trees, which indicate the structural relationship between claims and assumptions justified by rules. By searching the space and applications of inference rules, namely a proof procedure, we generate arguments. The formal definition is as follows.

Definition 2.2.5 (Arguments in ABA) *An argument a for $\sigma \in \mathcal{L}$ supported by s set of assumptions $A \subseteq \mathcal{L}$ is a proof for σ supported by A. We denote the argument a as $A \vdash \sigma$, the claim for the argument is σ =claim(a), the supporting for the argument is A= support(a).*

The notion of $A \vdash \sigma$ ignores the internal structure of the argument. However, it encapsulates the essence of the argument, the set of assumptions supporting the argument, and the conclusion of the argument. Given a deductive system ($\mathcal{L}$,$\mathcal{R}$), where $\mathcal{L} = \{p, q, r, s, a, b\}$, $\mathcal{R} = \{p \leftarrow q, r \; , \; q \leftarrow, \; r \leftarrow a \; , \; s \leftarrow b\}$ and the set of assumptions $\{a, b\}$, we can construct an argument for p such as: $\{a\} \vdash p$. In the tree of this argument, the root is p and it has two children q and r according to the inference rule $p \leftarrow q, r$. Then q is expanded to an empty set because of the rule $q \leftarrow$. The premise for r is a according to the rule $r \leftarrow a$. Then the argument $\{a\} \vdash p$ is generated. Other arguments can be generated by the similar process until leaves are empty or assumptions.

We define the notion of *attacks* between arguments in ABA in terms of the *contrary* of assumptions. *Contrary* is a total mapping from a set of assumptions into a language denoted as $^{-}$. The contrary of an assumption indicates a challenge against the assumption and an open debate about the assumption. For example, if one of the assumptions is "it will be rainy at 1:00 pm tomorrow", the contrary of this assumption might be "it will be sunny at 1:00 pm tomorrow". The contrary of an assumption "X is true" might be "there is evidence against X". Contrary is different from negations. Contrary requires the sentences occur in $\mathcal{L}$, but negation may not. Contrary is only given for assumptions, but negation may apply to any sentence. Contrary can be simplified to negation in propositional logic when a contrary for an assumption is unique. The only way to attack an argument is to attack one of its assumptions supporting the argument's conclusion. We can understand the contrary of an assumption as a sentence in the language representing a challenge against the assumption. $\bar{a}$ refers to the contrary of the assumption a. The *attacks* between arguments depend on attacking assumptions. The definition of *attacks* is as follows [2].

Definition 2.2.6 (Attacks in ABA) *Given a notion of contrary of assumptions,*

- *an argument $A_1 \vdash \sigma_1$ attacks an argument $A_2 \vdash \sigma_2$ if and only if the conclusion σ_1 of the first argument is the contrary of one of the assumptions in the support A_2 of the second argument;*
- *a set of arguments Arg_1 attacks a set of arguments Arg_2 if an argument in Arg_1 attacks an argument in Arg_2.*

Moreover, Toni [14] also introduces the attacks between sets of assumptions to which attacks between corresponding arguments in ABA. It describes an argument supported by a subset of A attacks an argument supported by a subset of A'. From this, in general:

- if an argument arg attacks another argument arg' then the set of assumptions supporting arg attacks the set of assumptions supporting arg'
- if a set of assumptions A attacks another set of assumptions A' then some argument supported by s subset of A attacks some argument supported by a subset of A'

Definition 2.2.7 *A set of assumptions A attacks a set of assumptions A' iff there exists an argument $A_1 \vdash \bar{\sigma}$ such that $A_1 \subseteq A$ and $\sigma \in A'$*

Until now, we have attacks between arguments, sets of arguments, and assumptions, respectively. From a hybrid view, attacks can also happen between arguments and a set of assumptions.

- an argument arg attacks a set of assumptions A iff arg attacks some argument arg' supported by $A' \subseteq A$

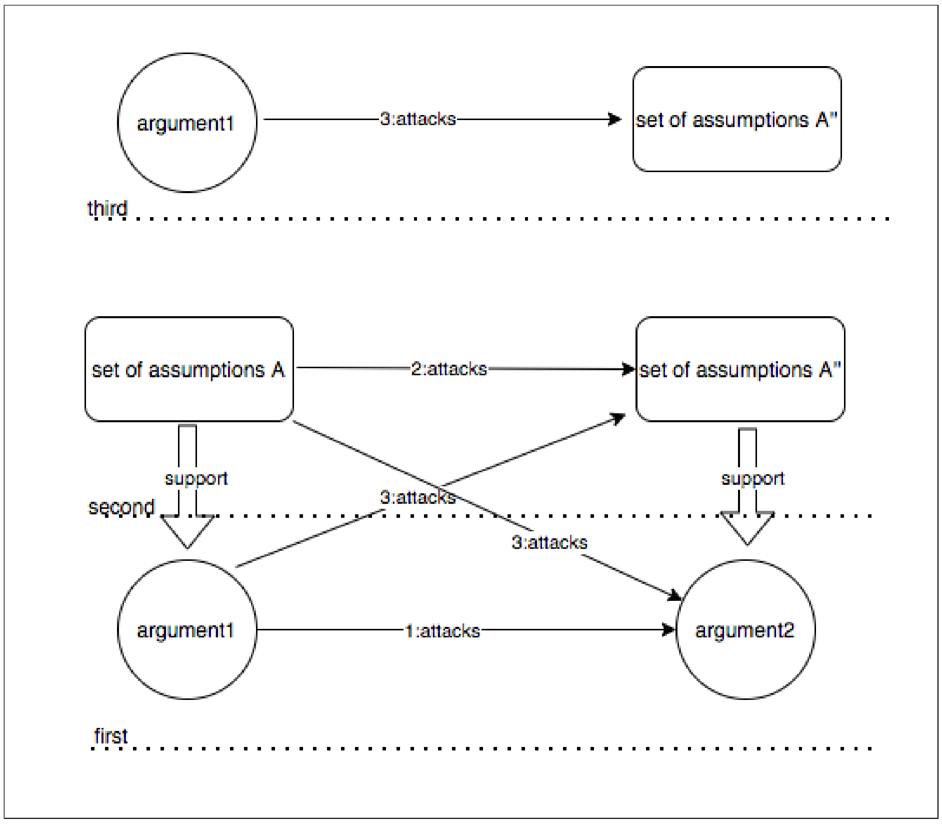

Figure 4: Attacks between arguments and assumptions. (The first level is between arguments. The second level is between sets of assumptions. The third level is between corresponding arguments and assumptions.)

- a set of assumptions A attacks an argument arg iff there is an argument supported by $A' \subseteq A$ that attacks arg

These attacks discussed above are depicted in Figure 4.

2.2.3. ABA Frameworks

To make arguments and attacks complete, we introduce ABA frameworks. An ABA framework is a general-purpose argumentation framework with well-understood theoretical foundations and computational mechanisms compared to abstract argumentation. Arguments are deductions of claims supported by sets of assumptions, and attacks are directed at the assumptions supporting arguments. An ABA framework is based on a deductive system with assumptions and contraries of the assumptions. It consists of four parts: a language, rules, assumptions, and contraries. The rules in the context of ABA frameworks could be domain-independent or domain-dependent based on knowledge bases in the form of $p \leftarrow q, a$. They can also be written in the form of schemata, e.g. $p(X) \leftarrow q(X), a(X)$. It can be instantiated over an implicit vocabulary by using variables. The rules can be chained to derive a conclusion in the process of proof for the conclusion. Assumptions is a subset of the language $\mathcal{L}$ and provides some potential points to debate. Assumptions may be hypothesis to construct arguments. The definition of ABA frameworks is as follows [14].

Definition 2.2.8 (ABA framework) *An ABA framework is a tuple ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$^{-}$) where*

- *($\mathcal{L}$, $\mathcal{R}$) is a deductive system, with a language $\mathcal{L}$ and a set of inference rules $\mathcal{R}$,*
- *$\mathcal{A} \subseteq \mathcal{L}$ is a (non-empty) set, whose elements are referred to as assumptions,*
- *$^{-}$is a total mapping from $\mathcal{A}$ into $\mathcal{L}$, where $\bar{\alpha}$ is the contrary of α .*

We can derive arguments and attacks from ABA frameworks and we show an example of the ABA framework as follows.

Example 2.2 *An ABA framework is ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$^{-}$) where:*
$\mathcal{L}$ = $\{p, q, r, s, a, b\}$

$\mathcal{R}$ = $\{p \leftarrow q, r,\ q \leftarrow,\ r \leftarrow a,\ s \leftarrow b\}$

$\mathcal{A} = \{a, b\}$

$\bar{a} = s, \bar{b} = p$

From Example 2.2, we can obtain several arguments as follows.

- Argument 1: $\{a\} \vdash p$
- Argument 2: $\{\} \vdash q$
- Argument 3: $\{a\} \vdash r$
- Argument 4: $\{b\} \vdash s$

The first argument is for p with the supporting set $\{a\}$. p is expanded to q and r by the rule $q \leftarrow q, r$. The rules $q \leftarrow$ and $r \leftarrow a$ are used to continue the inference. The second argument is for q supported by an empty set. The rule $q \leftarrow$ is used to inference with the head of q, so the end of this path is empty. The third argument is for r supported by a because the rule $r \leftarrow a$. The fourth argument is for s supported by b using the rule $s \leftarrow b$.

Then we identify their relations as follows:

- Attack 1: (Argument 1, Argument 4)
- Attack 2: (Argument 4, Argument 1)

- Attack 3: (Argument 4, Argument 3)

Argument 1 attacks Argument 4 because the conclusion p in Argument 1 is the contrary of the assumption b in Argument 4. Because the contrary of the assumption a is s, Argument 4 attacks Argument 1 and Argument 3 that both use a as the support.

2.2.4. Property and Semantics

Argumentation semantics offer a way to determine the survived set of arguments from a dialectical viewpoint. ABA is also equipped with different semantics for determining the acceptable set of arguments. Along the lines of semantics in AA concerning arguments and attacks between arguments, the set of arguments A is

- admissible iff it does not attack itself and it attacks all arguments that attack it
- preferred iff it is maximally admissible, where there is no set of arguments $A' \supset A$ is admissible.
- complete iff it is admissible and contains all arguments it defends, where A defends α iff A attacks all arguments that attacks α
- grounded iff it is minimally complete, where there is no set of arguments $A' \subset A$ is complete
- stable iff it is conflict-free and it attacks all arguments it does not contain.

Assumption-level view:

Definition 2.2.9 (Admissible assumptions) *A set of assumptions A is admissible, iff*

- *A attacks every set of assumptions that attacks A, and*
- *A does not attack itself.*

A set of assumptions A is

- preferred iff it is maximally (w.r.t. $\subseteq$) admissible;
- complete iff it is admissible and contains all assumption it defends, where A defends a iff A attacks all sets of assumptions that attacks a;
- grounded iff it is minimally (w.r.t. $\subseteq$)complete
- stable iff it is conflict-free and it attacks all assumptions it does not contain.

In ABA, a rational agent can determine whether a given claim would to be accepted or justified. So the agent needs to find an argument for the claim that can be defended against attacks from other arguments. A sentence $s \in \mathcal{L}$ is admissible/-grounded iff

- there is an argument a with the claim(a) = s such that $a \in E$ (for some admissible/grounded extension E).

When we consider Example 2.2, arguments and their relations are depicted in Figure 5 (the directional line indicating an attack from one argument to another). The upper case letter represents the argument for the corresponding lower case letter shown as follows:

- P: $\{a\} \vdash p$
- Q: $\{\} \vdash q$
- R: $\{a\} \vdash r$
- S: $\{b\} \vdash s$
- A: $\{a\} \vdash a$
- B: $\{b\} \vdash b$

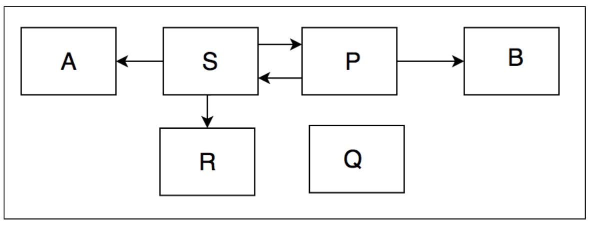

Figure 5: The attacks between arguments in Example 2.2

Index	Complete Extensions
1	$\{Q\}$
2	$\{S, B, Q\}$
3	$\{P, Q, R, A\}$

Table 1: The complete extensions in Example 2.2

If the query is "q is grounded or not," after finding arguments and their relationships in Example 2.2, we aim to find the grounded extension. Because the grounded extension is the minimal complete extension, we construct the complete extensions shown in Table 1. The minimal complete extension is the set $\{Q\}$.

Now backing to the query "whether q is grounded or not," the answer is "true." Because there is an argument Q with claim(Q) =q satisfying Q in the grounded extension shown in Table 2, we can answer that q is grounded.

Index	Grounded Extension
1	$\{Q\}$

Table 2: The grounded extension in Example 2.2

This section focused on notions and definitions of ABA frameworks. Firstly, we started with a deductive system as the foundation of the ABA framework. The system contained the deductions for conclusions chained by inference rules. Then, we gave the definition of arguments and attacks in ABA based on a deductive system. Arguments were the deductions for claims and attacks were between two arguments if the first argument's claim was the contrary of the assumption which was in the support set for the second argument. Subsequently, to make the arguments and attacks complete, we formally presented ABA frameworks, which contained four components language, rules, assumptions and contraries. Finally, we applied the argumentation semantics to ABA in the same way as in abstract argumentation and gave an example to illustrate the application of arguments and semantics.

2.3. Argument Graph

We use argument graphs to show a complete inference process for an argument with rules. By doing so, we get an intuitive impression of the internal structure of arguments. For example, if we are answering a query for making a decision based on conflicting information. The instantiated argument graphs generated from a knowledge base can easily reflect the attacking relationship between arguments according to the contrary of assumptions. Each argument can be seen as a tree-based structure. An argument graph may represent not only one argument. There is a relation between the argument graph and the original notion of tree-based arguments in ABA, and we will explain it in the following definition.

In the definition of an argument graph from Craven and Toni [10], G is a directed graph, and $v(G)$ denotes the vertices of the directed graph, and $e(G)$ denotes the edges of the directed graph. The $sinks(G)$ is the set of vertices without outgoing edges. The vertices in the argument graph is a single sentence in the language. It gives a more concrete internal structure compared with the form of standard arguments.

Definition 2.3.1 (Argument graph) *Given an ABA framework ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$^{-}$), an argument graph G is a directed, acyclic graph where $v(G) \subseteq \mathcal{L}$ and for all $s \in v(G)$:*

- *if $s \in \mathcal{A}$, then $s \in sinks(G)$*
- *if $s \notin \mathcal{A}$, then there is a rule $(s \leftarrow s_1, ..., s_m) \in \mathcal{R}$ such that there is an edge (s, s') in $e(G)$ iff $s' \in \{s_1, ..., s_m\}$*

G is an argument graph, the support of G denoted as $support(G)$ is $v(G) \cap \mathcal{A}$

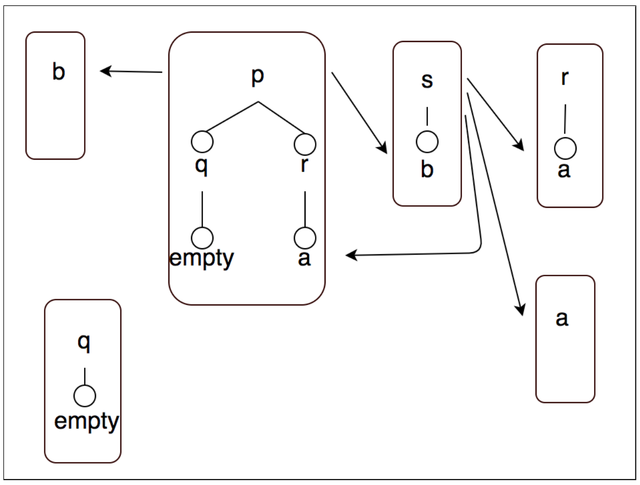

Figure 6: The argument graphs and their relationships in example2.2

If we use argument graphs to represent arguments in Example 2.2, these argument graphs and their attack relationships are depicted in Figure 6. We use rectangles to represent arguments. The argument for a is supported by assumption a and the argument for b is supported by assumption b. The argument graphs show the internal structure of arguments in a more detailed way. In this figure, an empty circle represents the direction from head to body in a rule. With the information of the contrary of assumptions $\bar{a} = s$ and $\bar{b} = p$, we find attacks from the argument for s to the argument whose set of assumptions contains a; from argument for p to the argument whose set of assumptions contains b.

Arguments defined in the previous section (see Definition 2.2.4) are tree-based arguments. When a is an argument, we use $nodes(a)$ representing the set of nodes of a. When $n \in nodes(a)$, $label(n)$ is the sentence or τ (the empty premise in rules) which, labels n. And $children(n)$ is the set of children of n. This kind of argument can be represented in a formal argument graph. To illustrate relations between tree-based arguments and argument graphs, the definition is as follows [10].

Definition 2.3.2 *Let G be an argument graph, and a be a tree-based argument. a is represented in G if there is a function f : (nodes(a) $\setminus$ $\{n \mid n \in nodes(a) \wedge label(n) = \tau\}$) $\rightarrow$ $v(G)$ mapping nodes of a not labelled by τ to nodes of G such that where $n \in (nodes(a)/$ $\{n \mid n \in nodes(a) \wedge label(n) = \tau\})$:*

- *$f(n)$ = label(n);*
- *if $f(n) = s$, then labels $(\{n' \mid n' \in children(n)\})\backslash\{\tau\} = \{s' \mid s'(s, s') \in e(G)\}$*

Based on this relation, one argument graph can include more than one tree-based argument in some situations. An argument graph can represent a combination of several tree-based arguments. If there is a set of rules:

$\mathcal{R} = \{p \leftarrow b, r \leftarrow b\}$

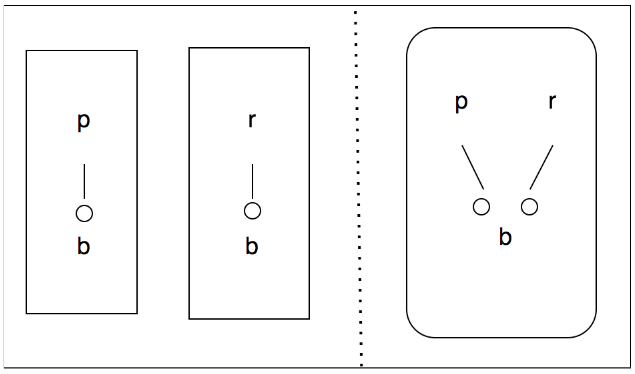

Figure 7: The left: two arguments and the right is an argument graph

In this set, there are two rules with the same body support different heads. If b is in the assumptions, we can construct two arguments with the same leaves node labeled by b. When we use an argument graph to represent, this argument graph can avoid showing the part b twice shown in Figure 7.

An argument in ABA may contain some repeated information, causing redundancy when constructing arguments and their relations. The research [10] captures two types of problem namely *circularity* and *flabbyiness* in arguments defined as follows.

Definition 2.3.3 *An argument a is circular if it contains a directed path from a node n labelled by some $s \in \mathcal{L}$ to another node n' labelled by s.*

Definition 2.3.4 *An argument a is flabby if it is non-circular and there are different nodes n, n' labelled by $s \in \mathcal{L}$ such that the children of n are labelled by different members of $\mathcal{L} \cup \{\tau\}$ from the children of n'.*

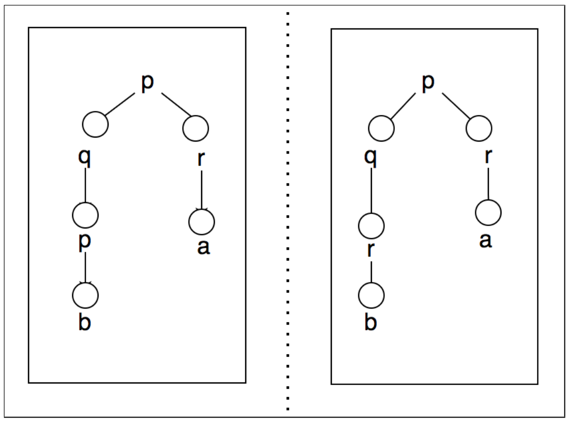

Figure 8: The left argument is circular and the right argument is flabby

To make these definitions more intuitive, if there is such a set of rules:

$\mathcal{R} = \{p \leftarrow q, r, p \leftarrow b, q \leftarrow p, q \leftarrow b, r \leftarrow a, r \leftarrow b\}$

and a set of assumptions $\mathcal{A} = \{a, b\}$. We could construct two different arguments, both for p supported by $\{a, b\}$. These two arguments have different inference processes using different rules shown in Figure 8. The left argument is circular because, in one path, p appears again after the first appearance at root. The right argument is flabby because it is non-circular, and two different nodes labelled by r have different children. One node labelled by r has a child b by the rules $r \leftarrow b$. Another node labelled by r has a child a by the rule $r \leftarrow a$.

Argument graphs can be seen as to represent tree-based arguments without the undesirable properties *circularity* and *flabbyiness* indentified in Definition 2.3.3 and 2.3.4. Let G be an argument graph. Craven and Toni [10] has proved that for each $s \in v(G)$, there is an argument a such that $claim(a) = s$, $support(a) \subseteq support(G)$ and a is represented in G; a is neither circular nor flabby.

After showing the representation of argument graphs, we consider the conversion from tree-based arguments to argument graphs. Argument graphs can represent a set of arguments and have the advantage of retaining the claims of arguments in the set as nodes and pruning some parts of arguments to remove circularity and flabbiness. So we need a mapping from tree-based arguments to argument graphs representing non-flabby and non-circular arguments in argument graphs. A source vertex is a vertex with no incoming edges in an argument graph. And a focused argument graph is an argument graph iff it has a unique source represented as $claim(G)$. The following is the definition of argument graphs graphical conversion to arguments [10].

Definition 2.3.5 *Let a be a tree-based argument. A focused argument graph G is a graphical conversion of a if:*

- *claim(a) = claim(G);*
- *if (s, s') $\in$ G, then there are n, $n' \in$ nodes(a) such that $n' \in$ children(n), label(n) = s, and label(n') = s'*

The different rules with the same head may cause redundancies in arguments. An argument is rule-minimal iff it is neither circular nor flabby [10].

Definition 2.3.6 *An argument a is rule-minimal iff for any two nodes n, n' in a labelled by the same $s \in \mathcal{L}$ the children of n and n' are labelled by the same elements of $\mathcal{L} \cup \{\tau\}$.*

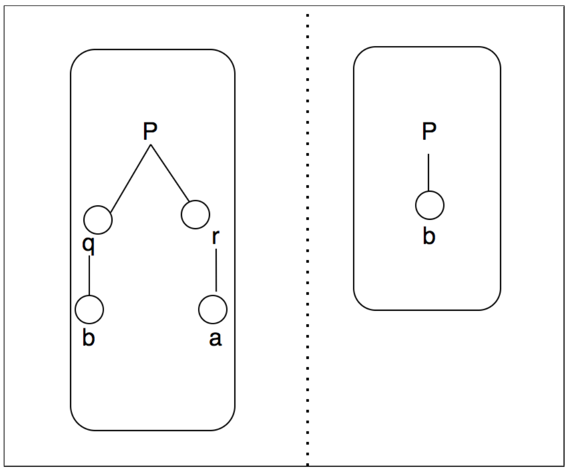

Figure 9: Two argument graphs for p

If we apply the minimality idea to argument graphs, then we have the definition as follows [10].

Definition 2.3.7 *A focused argument graph G is support-minimal iff there is no focused argument graph G' with claim(G) = claim(G') such that support(G') $\subset$ support(G).*

If there is s set of rules as follows:
$\mathcal{R} = \{p \leftarrow q, r, p \leftarrow b, q \leftarrow p, q \leftarrow b, r \leftarrow a\}$

and a set of assumptions $\mathcal{A} = \{a, b\}$, then we get two argument graphs presented in Figure 9. The left argument graph is supported by $\{a, b\}$. The right argument graph is supported by $\{b\}$. The support of the left argument graph is non-minimal because there exists the right argument graph supported by $\{b\}$. In some situations, the support minimality can not guarantee the rule minimality. In other words, the support minimality does not mean non-circular and non-flabby.

This section summarized the argument graphs, which showed the arguments' internal structures in a more intuitive way and the relations with the original notion of tree-based arguments. The argument graph was the basic concept in the graphical dispute method, and we will discuss it in Chapter 3. In the view of the direction from argument graphs to tree-based arguments, an argument represented in an argument graph was neither circular and flabby. We defined the graphical conversion to investigate the conversion from tree-based arguments to argument graphs. A graphical conversion of an argument was a focused argument graph. Moreover, a focused argument graph had the advantages of reducing redundancy by pruning arguments to remove circularity and flabbiness. In the end, we discussed the relations between rule minimal of arguments and support minimal of argument graphs.

2.4. Complexity Class

Assumption-based argumentation was proposed with computational mechanisms for determining "a set of arguments can survive together " under specific semantics. Different methods have been developed but they have high complexity. As a foundation for the analysis of complexity, we will introduce some classes from computational complexity theory. The complexity classes are from the references [15][16]. In this context, we deal with a decision problem that can be posed as a "yes" or "no" question, i.e., problems that admit a boolean answer. Decision problems are significant in computational complexity because they are convenient to reason, and we can transform many other types of problems into them.

For decision problems, the P class is the set of problems that a Turing machine can answer in polynomial time. The NP class is the set of problems that a non-deterministic Turing machine can solve in polynomial time. The class of problems whose answer is always the complement of those in NP is denoted as co-NP. A particular type of computation is called computation with oracles. Oracles are intuitively subroutines without cost. Given a class of decision problems C, the class P^C (NP^C) is the class of decision problems that can be solved in polynomial time by a deterministic (non-deterministic) machine that uses an oracle for the problems in C. For decision problems

- The class P is the set of problems that can be answered by a Turing machine in polynomial time.
- The class NP is the set of problems that can be answered by a non-deterministic Turing machine in polynomial time.
- The class co-NP is the set of problems whose answer is always the complement of those in NP
- The class NP-hard is the set of problems that if an algorithm for solving them can be translated into those for solving any NP problem. This set of problems is at least as hard as any NP problem.
- The class NP-complete is both NP and NP-hard. NP-complete problems can be verified in polynomial time and that any NP problem can be reduced to this problem in polynomial time.

In computational complexity theory, directly solving the NP problem is very difficult. It can be transformed into a verification problem. NP is the class of situation where, given a potential solution, we can check if it is a real solution in polynomial-time. It is evident that P $\subseteq$ NP. However, it is unknown whether P$\subset$ NP. The most well-known problem is deciding whether P = NP. If P $\neq$ NP is true, some issues exist in NP that are more challenging to solve than problems in P. The complement of a decision problem is to reverse the answer "yes" or "no" to the original question.

An oracle machine is a black box capable of solving any instance of a given computational problem. Oracles are intuitively subroutines without cost. When we add oracle to the class of problem, the class P is the class of decision problem that can be solved in polynomial time by a deterministic machine that uses an oracle for the problem. The definition of polynomial hierarchy is as follows.

Definition 2.4.1 (Polynomial Hierarchy) *The classes Σ_k^p , Π_k^p , Δ_k^p are defined by $\Sigma_0^p = \Pi_0^p = \Delta_0^p = P$, and for all $k \geq 0$,*

$$\Sigma_{k+1}^p = NP^{\Sigma_k^p} \quad \Pi_{k+1}^p = co - \Sigma_{k+1}^p \quad \Delta_{k+1}^p = P^{\Sigma_k^p}$$

It is believed that problems belonging to higher levels of the polynomial hierarchy are harder to solve than those in the lower level. Notice that

$$\Sigma_1^p = NP \quad \Pi_1^p = co - NP \quad \Delta_1^p = P$$

This section mainly presented different complexity classes in order to build the foundation of the analysis of computational complexity in reasoning processes. We mentioned several basic complexity classes such as P, NP, and co-NP. At the end of this section, we defined the polynomial hierarchy to represent more complex classes. We will use these classes to describe the computational complexity of different reasoning problems in ABA in Chapter 3.

3. Reasoning and Computational Mechanisms in ABA

Assumption-based argumentation (ABA) is equipped with viable computational mechanisms in dispute derivations to answer queries under a specific semantic. This section provides an overview of argumentation inference and variants of dispute derivation methods to illustrate how to answer a query by a rational agent. Section 3.1 presents the general process related to answering a question. Section 3.2 briefly introduces the dispute tree. Section 3.3 introduces the process of dispute derivations and takes GB-dispute derivations as an example. Section 3.4 discusses graphical dispute derivations under the grounded semantics in detail. This computational mechanism is the baseline for evaluating approximate methods in the next section. Section 3.5 shows the complexity of reasoning problems under different semantics in ABA.

3.1. Reasoning Process

Argumentation provides a formalism to apply scientific reasoning in the study of how to get acceptable conclusions through arguments and their relationships. Semantics in argumentation provides a way to evaluate arguments resulting in the validity of a claim. Whether a claim is accepted under specific semantics depends on the relationships between supporting arguments and possible counterarguments of this claim. Argumentation for inference is divided into three steps [12] shown in Figure 10. The first step is to use a knowledge base to generate an argumentation framework. In the second step, the result from the first step is the input to determine the sets of arguments accepted according to argumentation semantics. The third step is to identify accepted conclusions based on the sets of arguments in the second step. The previous section defined semantics in argumentation and gave examples. However, this section focuses on describing computational mechanisms to realize the third step's target from an argumentation framework.

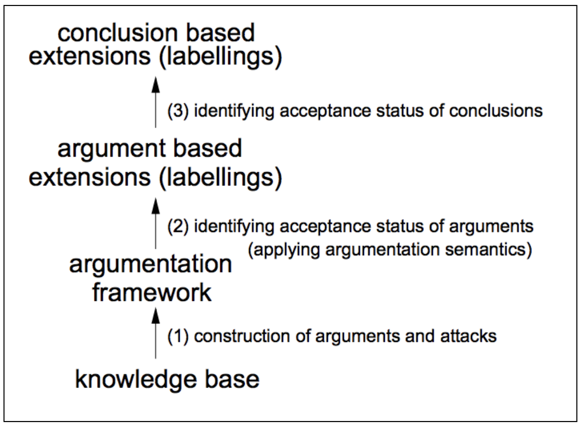
conclusion based extensions (labellings)
(3) identifying acceptance status of conclusions
argument based extensions (labellings)
(2) identifying acceptance status of arguments (applying argumentation semantics)
argumentation framework
(1) construction of arguments and attacks
knowledge base

Figure 10: Argumentation inference

Given an ABA framework, the reasoning process is to answer whether the claim c is accepted under admissible/grounded semantics. To answer this question, we need to construct arguments supporting the claim c, arguments attacking these arguments, and arguments attacking the attackers until supporters or attackers cannot attack each other. Then we identify the relations between them. If an admissible/-grounded extension includes an argument supporting the claim c, then the claim is accepted under admissible/grounded. In this scenario, the query is whether the reasoner accepts a claim representing by a sentence. After the reasoning procedure under specific semantics such as admissible or grounded, the answer is "true" or "false," meaning that the claim is admissible or grounded or not. Figure 11 shows a flowchart to describe this process.

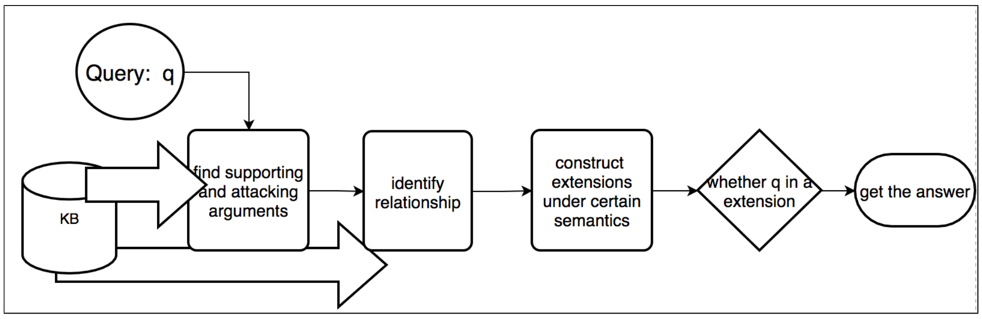
Query: q
KB
find supporting and attacking arguments
identify relationship
construct extensions under certain semantics
whether q in a extension
get the answer

Figure 11: The general reasoning process to answer a query in ABA

The above Figure 11 describes a general process to answer a query. Moreover, if we have a query, generate arguments, and determine the "acceptability" of the generated arguments, these two tasks are called a proof procedure. To construct a proof procedure for a claim, dispute trees and dispute derivations were proposed by Dung [7]. The proof procedure's underlying idea is that there are two sides: proponent and opponent over a claim; they alternatively attack each other until one side wins. We regard it as a game between two fictional players.

This section summarized the general reasoning process to answer a query whether a claim/conclusion can be accepted under specific semantics in ABA. The rest parts of Chapter 3 will explain how to answer queries by some already existing computational mechanisms.

3.2. Dispute Tree

Dispute trees [7] can determine the acceptability of arguments that are already constructed. Literally, there are two types of dispute trees: abstract trees and concrete trees. In abstract trees, every node is labelled by arguments and is assigned the status of the proponent node or the opponent node. An abstract dispute tree shows an abstraction of a winning strategy for a given and desired conclusion. However, it does not show the construction of arguments and counter-arguments. It contains all possible attacks by the opponent but contains only one successful counter-attack by the proponent. The root of the tree is the starting point of the dispute and is put forward by the proponent.

A finite branch represents a winning dispute that ends with an argument by the proponent that the opponent is unable to attack.The abstract dispute trees can be expanded to concrete dispute trees by incorporate the incremental construction of arguments. We focus on the concrete tree in this section out of the practical reason. In concrete dispute trees, individual nodes are labelled by steps of potential arguments. Every branch of a concrete dispute tree represents a sequence of alternating attacks by the opponent and counter-attacks by the proponent. In the process of constructing a concrete dispute tree, we use a selection function. The selection function takes input as a multi-sets and returns a sentence occurrence in the multi-sets as output. The role of the selection function for the proponent and opponent is also different. For both of them, the selection function chooses a non-assumption sentence to expand the potential argument constructed into a complete argument. When the selection function selects an assumption in the proponent, it determines an order in which attacks against the proponent's argument are considered. However, when the selection function selects an opponent's assumption, the assumption becomes a potential culprit for the proponent's counter-attack. Formally, the definition of dispute trees and admissible is directly referencing [7] shown as follows.

Definition 3.2.1 *Given a selection function, a concrete dispute tree for a sentence α is a (possibly infinite) tree $\mathcal{T}$ such that*

1. *Every node of $\mathcal{T}$ is labelled by a multi-set of sentences(representing a potential argument) and is assigned the status of proponent node or opponent node, but not both.*
2. *The root of $\mathcal{T}$ is a proponent node labelled by $\{\alpha\}$.*
3. *Let N be a proponent node labelled by $\mathcal{P}$. If $\mathcal{P}$ is empty, then N is a terminal node. Otherwise, $\mathcal{P}$ is not empty, and there exists some selected occurrence of a sentence σ in $\mathcal{P}$.*
    a) *If σ is an assumption, then there exists one child of N, which is an opponent node labelled by $\{\bar{\sigma}\}$ and a second child of N that is a proponent node labelled by $\mathcal{P} - \{\sigma\}$ (to consider all attacks against $\mathcal{P}$).*

   b) *If σ is not an assumption, then there exists some inference rule $\sigma \leftarrow S \in \mathcal{R}$ and there exists exactly one child of N, which is a proponent node labelled by $\mathcal{P} - \{\sigma\} \cup S$.*

4. *Let N be an opponent node labelled by $\mathcal{O}$. Then $\mathcal{O}$ is not empty, and there exists some selected occurrence of a sentence σ in $\mathcal{O}$.*
   a) *If σ is an assumption, then*
      i. *either σ is ignored and there exists exactly one child of N, which is an opponent node labelled by $\mathcal{O} - \{\sigma\}$,*
      ii. *or σ is a culprit, and there exists exactly one child of N, which is a proponent node labelled by $\{\bar{\sigma}\}$*
   b) *If σ is not an assumption and there exists no inference rule $\sigma \leftarrow S \in \mathcal{R}$, then N is a terminal node (and the potential attack $\mathcal{O}$ fails of its own accord). Otherwise, for every $\sigma \leftarrow S \in \mathcal{R}$, there exists a child of N, which is an opponent node labelled by the multi-set of sentences $\mathcal{O} - \{\sigma\} \cup S$.*
5. *There is no infinite sequence of consecutive nodes all of which are proponent nodes.*
6. *There are no other nodes in $\mathcal{T}$ except those given by $1 - 4$ above.*

The set of all assumptions belonging to the proponent nodes in $\mathcal{T}$ is called the defence set of $\mathcal{T}$

Definition 3.2.2 *A concrete dispute tree $\mathcal{T}$ for a sentence α is admissible if and only if no culprit at an opponent node belongs to the defence set of $\mathcal{T}$.*

Based on the following corollary [7], we can construct a dispute tree to evaluate whether an argument for a conclusion is accepted under admissible semantics.

Corollary 3.1 *The corollary is about concrete dispute trees*

1. *If $\mathcal{T}$ is an admissible concrete dispute tree and A is the defence set of $\mathcal{T}$, then A is an admissible set of assumptions.*
2. *If there is an argument for a conclusion α supported by a set of assumptions A_0 and A is an admissible set of assumptions such that $A_0 \subseteq A$, then for every selection function there exists an admissible concrete dispute tree for α with defence set A' and $A_0 \subseteq A' \subseteq A$ and A' is admissible.*

The following example illustrates these definitions.

Example 3.1 *Given an ABA framework ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$^-$) where*
$\mathcal{L} = \{q, r, s, u, t, w, x, y\}$
$\mathcal{R} = \{t \leftarrow q, w \leftarrow r, s, w \leftarrow u, v, x \leftarrow, y \leftarrow\}$
$\mathcal{A} = \{q, r, s, u\}$
$\bar{s} = t$, $\bar{q} = w$, $\bar{r} = x$, $\bar{u} = y$

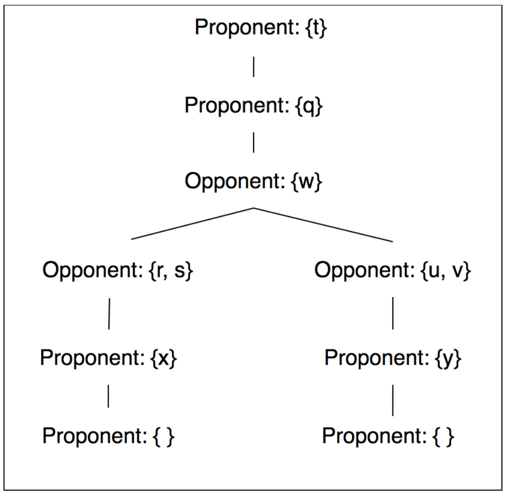

Figure 12: The concrete dispute tree for example 3.1

We give a concrete dispute tree for t in Figure 12. If the task is to judge whether t can be accepted under admissible semantics, then we need to construct an admissible dispute tree. Firstly, we take t as the root of the tree. Then the proponent finds an inference rule $t \leftarrow q$ to support this claim. Next, the opponent puts forward w as the contrary of the assumption q to attack the proponent. The sentence w can be expanded in two different directions with the same head according to $w \leftarrow r, s$ and $w \leftarrow u, v$. In the left branch, we ignore s, and find the contrary of r in the proponent. Finally, we end with an empty set. In the right branch, we find the contrary of assumption u. The leaves of the tree are empty labelled by the proponent. It means the proponent wins in the end, and t is accepted under admissible semantics.

This section introduced the terminology of dispute trees as a part of a proof procedure for ABA frameworks. Dispute trees presented a tree structure for determining the acceptability of a conclusion/claim under specific semantics. This kind of tree showed a winning strategy for the reasoning process over a sentence, which provided a foundation and brought us closer to dispute derivations in the following section.

3.3. Dispute Derivations

Dispute derivations are computational mechanisms to compute top-down dispute trees and figure out how to construct a winning strategy over a conclusion/claim. These kinds of mechanisms allow us to determine whether the claim which is put forward by the proponent and is supported by a set of arguments can be deemed to be acceptable under different semantics. As discussed in dispute trees, dispute derivation can be understood as a game between two players − a *proponent* and an *opponent* with rules as follows [9]:

- the claim (the root of a dispute tree) is a starting point, and the two players alternatively attack each other
- the proponent's task is to support the claim by constructing arguments and defend itself by counter-attacking the opponent
- the opponent's task is to attack the proponent by attacking arguments constructed by the proponent and defend itself by counter-attacking the proponent.

If the proponent cannot defend itself to support the claim, the proponent would fail. Otherwise, the proponent would have a successful outcome and return an acceptable set of assumptions supporting and protecting the given claim under corresponding semantics. Dispute derivations have different variants out of different argumentation semantics.

A generalized framework for dispute derivations in assumption-based argumentation was proposed by Toni[9]. It bases on the several existing computational mechanisms such as GB-, AB- and IB- dispute derivations[7] [8] under the grounded, admissible and ideal semantics, respectively, and give a general notion of existing mechanisms with different parameters. This generalized framework for dispute derivation in ABA has been proved soundness and completeness results w.r.t. the grounded, admissible and ideal semantics. So these computational mechanisms are helpful to compute reasoning problems. We give one kind of dispute derivations in the following, namely GB-dispute derivations. Based on the following theorem[8], GB-dispute derivations determine the acceptability for a sentence under the grounded semantics.

Theorem 3.3.1 *Given a GB-dispute derivation of a defence set A for a sentence α:*

- *A is admissible and it is contained in the grounded set of assumptions;*
- *there exists $A' \subseteq A$ and an argument $A' \vdash \alpha$.*

To make a clear statement of the process of derivations, the definition of GB-dispute derivations is from Toni [9].

GB-dispute derivation The frontier of a dispute tree is a set of proponent and opponent nodes labeled by multi-sets of sentences, which means steps of potential arguments. A dispute derivation represents the current state of this frontier, together with the set of defence assumptions D_i and culprits C_i generated so far as a quadruple: $\langle\, \mathcal{P}_i \,, \mathcal{O}_i \,, \mathcal{D}_i \,, \mathcal{C}_i \,\rangle$. The initial step of a dispute derivation is to represent the root of the dispute tree. There is a selection of a node in the dispute tree's frontier for every step, and we replace the node with its children. We use the set $\mathcal{C}_i$ to filter the potential defense arguments and the set of defense assumptions $\mathcal{D}_i$ to filter potential culprits. So the final defense assumption set constructed does not attack itself.

Definition 3.3.1 *Let $\langle\, \mathcal{L} \,, \mathcal{R} \,, \mathcal{A} \,,^{-} \rangle$ be an ABA framework. Given a selection function, a GB-dispute derivation of a defence set Δ for a sentence α is a finite sequence of quadruples*
$\langle\, \mathcal{P}_0 \,, \mathcal{O}_0 \,, \mathcal{D}_0 \,, \mathcal{C}_0 \,\rangle, \ldots, \langle\, \mathcal{P}_i \,, \mathcal{O}_i \,, \mathcal{D}_i \,, \mathcal{C}_i \,\rangle, \ldots, \langle\, \mathcal{P}_n \,, \mathcal{O}_n \,, \mathcal{D}_n \,, \mathcal{C}_n \,\rangle$
where

$$\begin{array}{lll} \mathcal{P}_0 = \{\alpha\} & D_0 = \mathcal{A} \cap \{\alpha\} & \mathcal{O}_0 = \mathcal{C}_0 = \{\} \\ \mathcal{P}_n = \mathcal{O}_n = \{\} & \Delta = D_n & \end{array}$$

and for every $0 \leq i < n$, only one σ in $\mathcal{P}_i$ or one S in $\mathcal{O}_i$ is selected, and:

1. *If $\sigma \in \mathcal{P}_i$ is selected then*
   a) *if σ is an assumption, then*

$$\mathcal{P}_{i+1} = \mathcal{P}_i - \{\sigma\} \quad D_{i+1} = D_i \quad C_{i+1} = C_i \quad \mathcal{O}_{i+1} = \mathcal{O}_i \cup \{\{\bar{\sigma}\}\}$$

   b) *if σ is not an assumption, then there exists some inference rule $\sigma \leftarrow R \in \mathcal{R}$ such that $C_i \cap R = \{\}$ (filtering of potential defence arguments by culprits) and*

$$\begin{array}{ll} \mathcal{P}_{i+1} = \mathcal{P}_i - \{\sigma\} \cup R & D_{i+1} = D_i \cap (\mathcal{A} \cap R) \\ C_{i+1} = C_i & \mathcal{O}_{i+1} = \mathcal{O}_i \end{array}$$

2. *If S is selected in $\mathcal{O}_i$ and σ is selected in S then*
   a) *if σ is an assumption, then*
      i. *either σ is ignored, i.e.*

$$\begin{array}{ll} \mathcal{P}_{i+1} = \mathcal{P}_i & D_{i+1} = D_i \\ C_{i+1} = C_i & \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup \{S - \{\sigma\}\} \end{array}$$

      ii. *or $\sigma \notin A_i$ (filtering of culprits by defence assumptions) and*

$$\begin{array}{ll} \mathcal{P}_{i+1} = \mathcal{P}_i \cup \{\bar{\sigma}\} & D_{i+1} = D_i \\ C_{i+1} = C_i & \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup \{S - \{\sigma\}\} \end{array}$$

   b) *if σ is not an assumption, then*

$$\begin{array}{ll} \mathcal{P}_{i+1} = \mathcal{P}_i & D_{i+1} = D_i \\ C_{i+1} = C_i & \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup \Big\{ S - \{\sigma\} \cup R | \sigma \leftarrow R \in \mathcal{R} \Big\} \end{array}$$

To illustrate GB- dispute derivations, we consider the following example.

Example 3.2 *Given an assumption-based framework ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$^{-}$) where*

$\mathcal{L} = \{\ q, r, s, t, u, v, w, x, y\ \}$

$\mathcal{R} = \{\ t \leftarrow q,\ w \leftarrow r, s,\ w \leftarrow u, v,\ x \leftarrow,\ y \leftarrow\ \}$

$\mathcal{A} = \{\ q, r, s, u\ \}$

$\bar{q} = w,\ \bar{r} = x,\ \bar{u} = y,\ \bar{s} = t$

According to Definition 3.3.1, when we construct a dispute derivation for the sentence t of the defence set $\{q\}$, we can obtain a sequence of quadruples ($\mathcal{P}_i$, $\mathcal{O}_i$, $\mathcal{D}_i$, $\mathcal{C}_i$) for Example 3.2 as follows:

1. $(\{\underline{t}\}, \{\}, \{\}, \{\})$,
2. $(\{\underline{q}\}, \{\}, \{q\}, \{\}\ )$,
3. $(\ \{\}, \{\{\underline{w}\}\}, \{q\}, \{\})$,
4. $(\{\}, \{\{\underline{r}, s\}, \{u, v\}\}, \{q\}, \{\})$,
5. $(\{\underline{x}\}, \{\{u, v\}\}, \{q\}, \{r\})$,
6. $(\{\}, \{\{\underline{u}, v\}\}, \{q\},\{r\})$,
7. $(\{\underline{y}\}, \{\}, \{q\}, \{r, u\})$,
8. $(\{\}, \{\}, \{q\}, \{r, u\})$.

The underline represents the output of the selection function in each step. The first step is to put the sentence t into $\mathcal{P}_0$. Because t is not an assumption, the other three sets are empty. If t is selected in $\mathcal{P}_0$ in the first step, then we replace t with q and add q into $\mathcal{D}$ according to the condition of 1(ii) in Definition 3.3.1. In the second step, q is selected as the only one sentence. Moreover, q is an assumption, according to the condition of 1(i) in Definition 3.3.1, we remove q and update $\mathcal{O}$ with $\bar{q}$ that is w, resulting in the third step. Then, according to the condition of 2(ii) in Definition 3.3.1, we expand w to set $\{r, s\}$ and $\{u, v\}$ because two rules with the same head of w. In the fourth step, r is selected as an assumption and is not in $\mathcal{D}_i$,according to the condition of 2(i)(b) in Definition 3.3.1, we add x into $\mathcal{P}_i$ and add r into culprits set $\mathcal{C}_i$. In the next step x is selected, we expand x by the rule $x \leftarrow$ with an empty premise. So in the sixth step, we remove x and look at the set in $\mathcal{O}_i$, then we select u. By doing the similar process in Definition 3.3.1, finally, we end with $\mathcal{P}_i$ and $\mathcal{O}_i$ are empty and $\mathcal{D}_i$ is the defence set of sentence t. For every finite dispute tree for a sentence α with defence set D, there exists a dispute derivation for α of a defence set $A' \subseteq A$. This theorem is proved[7], which makes dispute derivations as useful computational mechanisms.

This section introduced the computational mechanisms for ABA, namely dispute derivations. There existed several different variants GB-, AB-, and IB-dispute derivations working for different semantics. We presented the GB-dispute derivation and an example of how to apply it in this section. Dispute derivation was an important method for the reasoning process of answering a query. This section also gave an overview of the process of dispute derivations. The underlying idea was similar to the graphical dispute derivation that we will introduce in the following section.

3.4. Graphical Dispute Derivations

In this section, we will introduce a computational mechanism of graphical dispute derivations. They can be used to determine whether a given claim is accepted under admissible or grounded semantics for ABA. They have the same spirit as the dispute derivations mentioned in the previous section. However, this computational mechanism represents arguments by argument graphs to avoid the problem of flabbiness and circularity proposed by Craven and Toni [10]. Moreover, this computational mechanism consists of five parts $\mathcal{P}_i$, $\mathcal{O}_i$, $\mathcal{G}_i$, $\mathcal{D}_i$, $\mathcal{C}_i$ in each dispute sequence. For simplicity, we only consider the grounded cases. According to the theorems for soundness and completeness as follows[10], we can determine the acceptability of a sentence under the grounded semantics by constructing a grounded graphical dispute derivation.

Theorem 3.4.1 (Soundness) *For any grounded graphical dispute derivation with resulting argument graph G, there is some grounded argument graph G' such that $G' \subseteq G$.*

Theorem 3.4.2 (Completeness) *Let $\mathcal{L}$ be finite. If G is a grounded argument graph such that $s_0 \in v(G)$, then there is a grounded graphical dispute derivation for s_0 with resulting argument graph some G' such that $G' \subseteq G$.*

The definition of the grounded graphical dispute sequence is from Craven and Toni [10] as follows. $\mathcal{P}_i$ is a proponent potential argument graph, and $\mathcal{O}_i$ is an opponent argument graphs set. $\mathcal{G}_i$ represents a graph, $\mathcal{D}_i$ and $\mathcal{C}_i$ are sets of assumptions. A grounded graphical dispute derivation is a grounded graphical dispute sequence with the requirement that there is no unmarked node in $\mathcal{P}_i$ and no unmarked graph in $\mathcal{O}_i$.

Definition 3.4.1 *Let $s_0 \in \mathcal{L}$. Let n be such that $0 \leq n \leq w$. A grounded graphical dispute sequence for s_0 of length n is a sequence $((P_i, O_i, G_i, D_i, C_i))_{i=0}^{n}$, where:*

$$\begin{array}{lll} \mathcal{P}_0 = newgraph(s_0) & \mathcal{D}_0 = \mathcal{A} \cap \{s_0\} & \mathcal{O}_0 = \{\} \\ \mathcal{G}_n = (\{s_0\}, \emptyset) & \mathcal{C}_0 = \{\} & \end{array}$$

and for every I such that $0 \leq i < n$, only one $s \in u(\mathcal{P}_i)$ or one $G \in u(\mathcal{O}_i)$ is selected and

1. *if $s \in u(\mathcal{P}_i)$ is selected, then*
   a) *if $s \in \mathcal{A}$, then*

$$\mathcal{P}_{i+1} = \mathcal{P}_i \cup_m \{s\}$$
$$\mathcal{O}_{i+1} = \begin{cases} \mathcal{O}_i & if \exists G \in \mathcal{O}_i such that \bar{s} = claim(G) \\ \mathcal{O}_i \cup_u \{newgraph(\bar{s})\} & Otherwise \end{cases}$$
$$\mathcal{G}_{i+1} = \mathcal{G}_i \cup_g \{(\bar{s}, s)\}$$

   and $acyclic(\mathcal{G}_{i+1})$

   b) *if $s \notin \mathcal{A}$, then there is some $(s \leftarrow R) \in \mathcal{R}$ such that $R \cap C_i = \emptyset$, and*

$\mathcal{P}_{i+1}$ = *updgraph*($\mathcal{P}_i$, $s \leftarrow R$, $\emptyset$)
$\mathcal{G}_{i+1} = \mathcal{G}_i \cup_g \{(s\prime, s)|s\prime \in R\}$
$\mathcal{D}_{i+1} = \mathcal{D}_i \cup (R \cap \mathcal{A})$

and $acyclic(\mathcal{P}_{i+1})$,$acyclic(\mathcal{G}_{i+1})$

2. *if* $G \in u(\mathcal{O}_i)$ *and* $s \in u(G)$ *are selected, then*
   a) *if* $s \in \mathcal{A}$*, then:*
      i. *either s is ignored, i.e.:*
         $\mathcal{O}_{i+1} = (\mathcal{O}_i \setminus \{G\}) \cup_u \{G \cup_m \{s\}\}$
      ii. *or* $s \notin \mathcal{D}_i$ *and* $s \in \mathcal{C}_i$*, and:*
         $\mathcal{O}_{i+1} = (\mathcal{O}_i \setminus \{G\}) \cup_m \{G \cup_m \{s\}\}$
         $\mathcal{G}_{i+1} = \mathcal{G}_i \cup_g \{(\bar{s}, claim(G))\}$
         and $acyclic(\mathcal{G}_{i+1})$*;*
      iii. *or* $s \notin \mathcal{D}_i$ *and* $s \notin \mathcal{C}_i$*, and:*
         $\mathcal{P}_{i+1} = \begin{cases} \mathcal{P}_i & if \bar{s} \in v(\mathcal{P}_i) \\ \mathcal{P}_i \cup_u \{\bar{s}\} & Otherwise \end{cases}$
         $\mathcal{O}_{i+1} = (\mathcal{O}_i \setminus \{G\}) \cup_m \{G \cup_m \{s\}\}$
         $\mathcal{G}_{i+1} = \mathcal{G}_i \cup_g \{(\bar{s}, claim(G))\}$
         $\mathcal{D}_{i+1} = \mathcal{D}_i \cup (\{\bar{s}\} \cap \mathcal{A})$
         $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{s\}$
         and $acyclic(\mathcal{G}_{i+1})$*;*
3. *if* $s \notin \mathcal{A}$*, let:*
   $R_c = \emptyset$
   $R_{\not c} = \{R|(s \leftarrow R) \in \mathcal{R}, acyclic(updgraph(G, s \leftarrow R, \emptyset)), R \notin R_c\}$*,then:*
   $\mathcal{O}_{i+1} = ((\mathcal{O}_i \setminus \{G\}) \cup_m \{updgraph(G, s \leftarrow R, C_i)|R \in R_c\}) \cup_u \{updgraph(G, s \leftarrow R, \emptyset)|R \in R_{\not c}\}$.

Notion	Operation
$newgraph(s)$	create a G, where $v(G) = u(G) = \{s\}$, $e(G) = \emptyset$
$G \cup_u S$	generate a G', where $v(G') = v(G) \cup S$, $e(G') = e(G)$ and $u(G') = u(G) \cup S$
$G \cup_m S$	generate a G', where $v(G') = v(G) \cup S$, $e(G') = e(G)$ and $u(G') = u(G) \setminus S$
$\mathcal{O} \cup_u X$	get a $\mathcal{O}'$, where contains just the argument graph in $\mathcal{O}$ and X, $u(\mathcal{O}') = u(\mathcal{O}) \cup X$
$\mathcal{O} \cup_m X$	get a $\mathcal{O}'$, where contains just the argument graph in $\mathcal{O}$ and X, $u(\mathcal{O}') = u(\mathcal{O}) \setminus X$
$\mathcal{O} \setminus X$	get a $\mathcal{O}'$, where $\mathcal{O}'$ contains just the argument graph in $\mathcal{O}$ but not in X, $u(\mathcal{O}') = u(\mathcal{O}) \setminus X$
$updgraph(G, s \leftarrow s_1, ..., s_n, S)$	get a G', where $v(G') = v(G) \cup \{s_1, ..., s_n\}$ $e(G') = e(G) \cup \{(s, s')|s' \in \{s_1, ..., s_n\}\}$ $u(G') = (u(G) \cup \{s' \in \{s_1, ...s_n\}|s' \notin m(G)\}) \setminus (\{s\} \cup S)$
$\mathcal{G} \cup_g E$	get a $\mathcal{G}'$, where $e(\mathcal{G}') = e(\mathcal{G}) \cup E$ $v(\mathcal{G}') = \mathcal{G} \cup \{x|(s, s') \in E, (x = s \vee x = s')\}$

Table 3: The operations in the graphical dispute derivation

We perform a lot of operations in the process of the graphical dispute derivation. And these operations are described in detail shown in Table 3. G and G' represent potential argument graphs and $s \in \mathcal{L}$, $S \subset \mathcal{L}$. $\mathcal{O}$ and $\mathcal{O}'$ are sets of argument graphs;

Step	Case	$\mathcal{P}_i$	$u(\mathcal{O}_i)$	$\mathcal{G}_i$	$\mathcal{D}_i$	$\mathcal{C}_i$
0	-	$(\{\boldsymbol{a}\}, \emptyset)$	$\emptyset$	$(\{a\}, \emptyset)$	$\{a\}$	$\emptyset$
1	1(i)	$(\{a\}, \emptyset)$	$\{(\{\boldsymbol{p}\}, \emptyset)\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
2	2(ii)	$(\{a\}, \emptyset)$	$\{(\{p, \boldsymbol{b}\}, \{(p, b)\})\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
3	2(i)(c)	$(\{\boldsymbol{q}, a\}, \emptyset)$	$\emptyset$	$(\{a, p, q\}, \{(p, a), (q, p)\})$	$\{a\}$	$\{b\}$
4	1(ii)	$(\{q, a\}, \emptyset)$	$\emptyset$	$(\{a, p, q\}, \{(p, a), (q, p)\})$	$\{a\}$	$\{b\}$

Table 4: The dispute sequence for a in example 3.3

and X is a set of potential argument graphs. $e(G)$ represents the set of edges in G, and $v(G)$ represents the set of vertices in G; $u(G)$ is the set of unmarked vertices in G; $u(\mathcal{O})$ is the set of unmarked graph in $\mathcal{O}$. A grounded graphical dispute derivation for s_0 is a grounded dispute sequence for s_0: ($\mathcal{P}_i$, $\mathcal{O}_i$, $\mathcal{G}_i$, $\mathcal{D}_i$, $\mathcal{C}_i$) and no unmarked node and unmarked graph in the last $\mathcal{P}$ and $\mathcal{O}$, respectively. The description of the above definition is not easy to understand. We consider the following example that employs this definition as an illustration.

Example 3.3 *Given an ABA framework($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$\bar{\ }$), where*
$\mathcal{L} = \{p, q, a, b\}$
$\mathcal{R} = \{p \leftarrow b, q \leftarrow \}$
$\mathcal{A} = \{a, b\}$
$\bar{a} = p, \bar{b} = q$

The grounded graphical dispute sequence for a of length 4 is shown in Table 4. We represent unmarked nodes in each graph in the bold style. In step 0, we initialize P_0 as the graph with a single node a, which is unmarked, and $\mathcal{D}_0$ as the set with an assumption a. The other parts of the sequence are empty. Then the unmarked node a is selected. a is an assumption, so we add a new graph with a single unmarked node p (the contrary of assumption a) to $\mathcal{O}_i$ in step 1. Then p is expanded by the rule $p \leftarrow b$ and b is an unmarked node. In step 3, q is added into $\mathcal{P}_i$ as an unmarked node, and b is put into $\mathcal{C}_i$ according to case 2(i)(c) in the above definition. Moreover, the graph in $\mathcal{O}_i$ is marked. In the last step, select the only unmarked node q and update the sequence by the rule $q \leftarrow$ according to case 1(ii).

This section introduced the graphical dispute derivation, which was the essential part of the approximate methods described in Chapter 4. This computational mechanism was complicated. Afterwards, we presented an example to illustrate the process of the graphical dispute derivation. We will take this method as the baseline to compare with approximate methods in experiments. And we will present the computational complexity for reasoning in ABA frameworks in the following section.

3.5. Computational Complexity and Upper Bonds

Reasoning problems mentioned in the previous section for ABA are complicated, which results in a high computational complexity. This section will provide complexity results for reasoning under admissible/preferred/stable semantics. And all of these results are from Dimopoulos et al. [3].

Given a theory $T \subseteq \mathcal{L}$ and a formula $\alpha \in \mathcal{L}$, $Th(T) = \{\alpha \in \mathcal{L} | T \vdash \alpha\}$ is the deductive closure of T. "*a set of assumption is sanctioned by the semantics*" means that the set of assumptions is admissible/stable/preferred, respectively. Then the reasoning problem can be divided in two categories :

- the credulous reasoning problem, i.e. the problem of deciding for any given sentence $\psi \in \mathcal{L}$ whether $\psi \in Th(T \cup \Delta)$ for *some* assumption set Δ sanctioned by the semantics
- the sceptical reasoning problem, i.e. the problem of deciding for any given sentence $\psi \in \mathcal{L}$ whether $\psi \in Th(T \cup \Delta)$ for *all* assumption sets Δ sanctioned by the semantics

We often consider the complementary of sceptical reasoning problem for complexity analysis, as follows

- the co-sceptical reasoning problem, i.e. the problem of deciding for any given sentence $\psi \in \mathcal{L}$ whether $\psi \notin Th(T \cup \Delta)$ for *some* assumption set Δ sanctioned by the semantics

Semantics	Verification	Credulous Reasoning.	Sceptical Reasoning
Admissible	co-NP^C	NP^{NP^C}	co-NP^{NP^C}
Preferred	co-NP^{NP^C}	NP^{NP^C}	co-$NP^{NP^{NP^C}}$
Stable	P^C	NP^C	co-NP^C

Table 5: Upper bound complexity results for general ABA frameworks [3]

All these problems have the sub-problem: the assumption set verification problem, which means deciding whether a given set of assumptions Δ is sanctioned by the semantics. And they are located at the lower end of the polynomial hierarchy.

Recall that an ABA framework is based on a deductive system. Thus its complexity depends on the complexity of the derivability problem in the system.

- derivability: Given an ABA framework ($\mathcal{L}$,$\mathcal{R}$,$\mathcal{A}$,$^-$) ,a set of sentences $A \subseteq \mathcal{L}$ and a sentence $\alpha \in \mathcal{L}$, does there exist a deduction of the form $A \vdash^R \alpha$

We identify upper bounds for the credulous and skeptical reasoning problems by exploiting sub-problem. For any DL like propositional monotonic rule system S, the problem of checking whether $S \vdash \varphi$ is NP-complete. Assuming that the derivability problem belongs to a problem class C, for a general ABA framework, upper bounds for these problems are shown in Table 5.

Semantics	Verification	Credulous Reasoning.	Sceptical Reasoning
Admissible	P^C	NP^C	C
Preferred	co-NP^C	NP^C	co-NP^{NP^C}
Stable	P^C	NP^C	co-NP^C

Table 6: Upper bound complexity results for general ABA frameworks [3]

The flat ABA framework has restrictions on the general framework. Thus upper bonds are different, and we show them in Table 6. The complexity results show that skeptical reasoning under admissibility and preferability semantics is trivial and highly complex, respectively. In general, the results indicate that reasoning under different semantics is hard. The exact complexity and concrete analysis of using different underlying logic frameworks can be found from Dimopoulos et al. [3].

This section briefly presented computational complexity results for credulous and skeptical reasoning under the admissible/ preferred/stable semantics in ABA frameworks. These results theoretically indicated that reasoning under most semantics is hard in ABA frameworks. So we need to consider approximate reasoning approaches in Chapter 4.

4. Approximate Reasoning in ABA

From the previous sections, the reasoning procedure under specific semantics in ABA has a high computational complexity. Because the reasoning process is time-consuming, this research aims to reduce the runtime with the cost of slightly lower accuracy. The underlying idea of approximate reasoning in ABA is inspired by the approximate reasoning in ASPIC+ proposed to reason with structured argumentation by sampling arguments in ASPIC+ [17]. Approximate methods can give the correct answer with a certain probability. If the probability is high enough, we view approximate methods as valuable methods to overcome high complexity. In this chapter, we will introduce two categories of approximate approaches: framework-based sampling and dispute-based sampling. The framework-based method samples ABA frameworks to construct new frameworks before the actual reasoning process. The dispute-based method samples inference rules in different stages of derivations during the actual reasoning process. The reasoning procedures in ABA for constructing argument extensions are different under different semantics. The requirement for grounded extensions has more limitations than admissible extensions. For simplicity, the reasoning procedure in this chapter is under the grounded semantics. This section focuses on approximate methods based on graphical dispute derivations. Section 4.1 presents the framework-based sampling method as a way to get the sample from the framework. Section 4.2 offers three algorithms of dispute-based sampling methods. They are different because we apply the random selection function on three distinct stages: proponent, opponent, and both of them. We illustrate all of them with examples in this chapter.

4.1. Framework-based Sampling

In this section, we will introduce the framework-based sampling (FS) method. This method is simple to sample assumptions and inference rules of ABA frameworks before the actual dispute derivations. As we already mentioned in Definition 2.2.8, an ABA framework consists of four components: a language, rules, assumptions, and contraries. When we sample assumptions, we also sample the corresponding contraries to keep the mapping relations. Not all rules and assumptions are used in the reasoning procedure to answer one query. Thus, we implement this approximate method by sampling some rules, assumptions, and contraries with probabilities. And the sample probabilities of assumptions and contraries are the same. The FS method does not employ the information in queries because it directly samples frameworks before starting the reasoning process of dispute derivations. It randomly obtains part of the framework. If we use sampled frameworks, we hypothesize that the running time of reasoning would be shorter than using original frameworks. Then the new framework is used for the reasoning process of the graphical dispute derivation. This method provides an easy way to reduce the computing time for derivations, and we describe the algorithm in Algorithm 1. Moreover, Figure 13 shows an example of an original ABA framework on the left, and a sampled frame-

Original ABA framework :

-
```
A18 <- A10, A5, A16
A19 <- A7, A17
A19 <- A12
A3 <- true
A11 <- true
A13 <- true
A18 <- A4, A16, A19
A17 <- A6, A7, A8
A11 <- A15, A16
A18 <- A19
{A10,A5,A15,A6,A16,A8}
not A15 = A1
not A16 = A11
not A6 = A9
not A5 = A11
not A10 = A2
not A8 = A4
```

Sampled ABA framework :

-
```
A18 <- A10, A5, A16
A19 <- A7, A17
A19 <- A12
A11 <- true
A18 <- A4, A16, A19
A13 <- true
A18 <- A19
{A5,A6,A15,A8}
not A15 = A1
not A6 = A9
not A5 = A11
not A8 = A4
```

Figure 13: Comparison between original framework and sampled framework

work under the two parameters 0.7 and 0.7 on the right. The size of the sampled framework is smaller than the original framework.

Algorithm 1: Framework-based Sampling

```
Data: Abaf:(originalAssum, originalNegation, originalRules),
      pAssums, pRules
Result: nAbaf:(newAssum, newNegation, newRules)
1  initilization nAbaf;
2  int numAssum = pAssums * size(originalAssum);
3  while size(newAssum) < numAssum do
4      newA ← randomSelection(originalAssum);
5      newAssum ← newAssum ∪ newA;
6      newANega ← getNegation(originalNegation, newA);
7      newNegation ← newNegation ∪ newANega;
8  end
9  int numRule = pRules * size(originalRules);
10 while size(newRules) < numRule do
11     int i = rand(size(originalRules));
12     newR ← get(i, originalRules);
13     newRules ← newRules ∪ newR;
14 end
15 nAbaf ← newAssum, newNegation, newRules
```

Algorithm 1 for the framework-based sampling method takes an ABA framework, a probability for sampling the assumptions as well as contraries, and a probability for sampling the inference rules as inputs, represented by `Abaf`, `pAssums`, `pRules`, respectively. The result is a new ABA framework, denoted by `nAbaf`.

Function Name	Input	Output	illustration
$size$	a set	a number	get the number of elements in one set
$randomSelection$	a set	an element in a set	randomly choose an element in a set
$getNegation$	a set negations, an assumption	the negation of an assumption	get the negation for the corresponding assumption
get	index number, a set	the element at the index number	convert the input set into a list and get the element of the input index from the list

Table 7: Functions in Algorithm 1

The ABA framework includes a set of assumptions `originalAssum`, a set of corresponding negations [1] `originalNegation` for assumptions, and a set of rules `orignalRules`. The initialization is to initialize an empty ABA framework consisting of assumptions `newAssum`, corresponding negations `newNegation`, and rules `newRules`. The function $size()$ is to get the number of elements for a set. First, this algorithm calculates the number `numAssum` of assumptions in a new constructed ABA framework by multiplying the probability for sampling assumptions and the size of assumptions in the original ABA framework. Then this algorithm randomly chooses an assumption from the original set of assumptions and adds it into a new set of assumptions. After that, to keep the relationship between assumptions and contraries, this algorithm finds a corresponding negation `newANega` for the chosen assumption and adds it to a new set of negations. The function $getNegation()$ is to get the negation for the given assumption. The function$randomSelection()$ is to select an assumption randomly from the set of assumptions. A similar operation is applied to construct the new set of inference rules. Finally, this algorithm adds all these new sets into the new ABA framework. This algorithm is a straightforward way to construct a sample ABA framework from the original ABA framework. After this process, the new ABA framework may lose some parts of information before starting the derivation. It would help reduce the running time for dispute deriva-

[1]The negations refer to contraries here and in the rest of methods

tions because of simplifying the whole knowledge base. Thus, we would not expect to obtain the answer to a query with a high accuracy. The functions used in this algorithm are summarized in Table 7.

This section proposed a framework-based sampling (FS) method implemented by sampling assumptions, contraries, and rules with a certain probability on ABA frameworks to construct a new ABA framework, which can be seen as a pre-process before actual dispute derivations. This method is not directly related to the query we answer after reasoning, so we do not expect an excellent accuracy. This section showed an example of the result and the algorithm of this approach. We will evaluate this method in experiments by comparing it with other approximate methods, which we will introduce in the next section.

4.2. Dispute-based Sampling

The dispute-based sampling approaches are implemented by randomly sampling inference rules from frameworks based on different stages of the graphical dispute derivation (see Definition 3.4.1 in Section 3). Because the rules mainly decide the expansion of dispute trees for finding a winning strategy. These approximate approaches are different from the framework-based sampling concerning the query and dispute derivations. These methods directly connect to the query and actual reasoning processes by randomly sampling rules during derivations. We consider the process with two players: *proponent* and *opponent* over a claim q as follows:

1. proponent: find a inferences rule $q \leftarrow s$ to support the claim q
2. opponent: find a possible way to attack the premise s in previous, for example there exists $not\ s \leftarrow m$
3. proponent: according to the already existing information, find a possible way to counterattack the opponent
4. opponent: find a possible way to attack the proponent
5. repeat this alternating process until one side can not attack the other side

The proponent and the opponent both attack each other in turn until one side cannot attack the other side. Furthermore, after several turns, if the final turn is on the proponent's side, we say that the proponent wins the game. The reasoner will accept the claim q. In other words, the answer for the query q is yes. However, if the final turn is on the opponent's side, the opponent wins the game. The reasoner will not accept the claim q. The answer for this query q is no. We can not predict the result if we do not compute step by step. We consider a structure over a dispute shown in Figure 14:

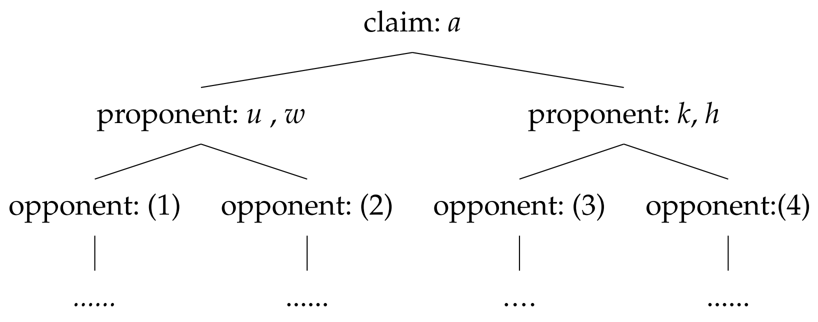

Figure 14: The structure over a dispute in the first situation

In the first situation, the claim is supported by two different rules: $a \leftarrow u, w$ and $a \leftarrow k, h$. The proponent with u, w could be attacked by the opponent (1) (2). The proponent with k, h could be attacked by the opponent (3) (4). If all paths contain a winning strategy in this structure, we can still get a correct answer when we randomly sample the proponent rules. If only the path including proponent with u, w results in a winning result, when we randomly sample the rule with k, h, we can never get the correct answer. This sampling will affect the generation of the latter branches of the opponent and the proponent, respectively. However, this random sampling is still relevant to construct arguments supporting the proponent, and we expect a higher accuracy than the framework-based sampling. When we consider another structure in Figure 15, and we apply a similar idea to the opponent. The

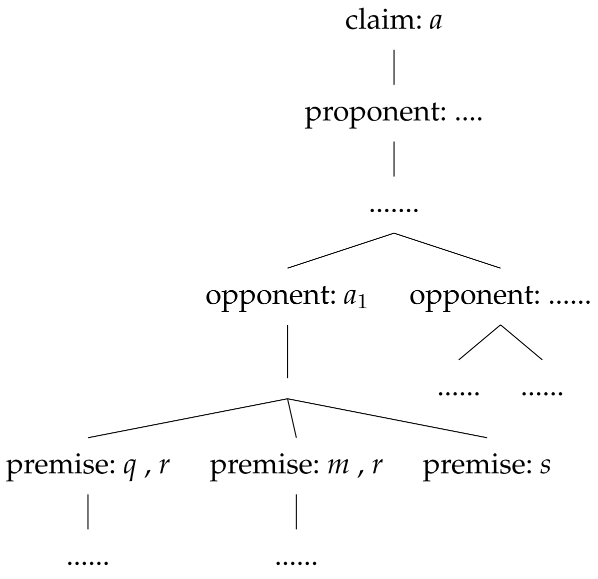

Figure 15: The structure over a dispute in the second situation

structure in the second situation contains a winning strategy for the claim a. After the stages supported by the proponent, the opponent finds a_1 attacks proponent. If a_1 is not an assumption, then the opponent needs to construct a new set of argument graphs to support the sentence a_1 by expanding to different premises with the same head of a_1, and there are three different rules to support it as follows: $\mathcal{R}$ = { $a_1 \leftarrow q, r$, $a_1 \leftarrow m, r$, $a_1 \leftarrow s$ }. If m and s are assumptions, m can be ignored to get a winning strategy, and s is in a set of culprits. When we look at the opponent in detail, we need to consider all these three sub-branches after the opponent. If all of them can lead to a successful state, we can still get the right answer when we randomly cut two sub-branches. The random selection is based on Definition 3.4.1 2(ii) when we expand the potential argument graphs to attack the proponent. If we construct some graphs, we can get a correct answer with a lower probability than constructing all graphs. In this situation, we randomly sample and construct inference rules to support the opponent. The expected accuracy may be similar to the proponent one and still higher than the framework-based sampling. Moreover, if we combine both perspectives of

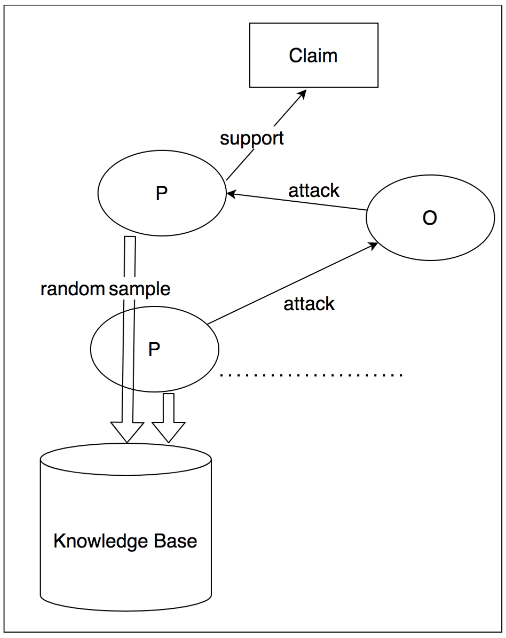

Figure 16: A sample strategy on the proponent side

the proponent and the opponent, this kind of sampling may dramatically reduce the running time for answering a query. However, it may not get an excellent accuracy. Dispute-based sampling approaches randomly select some parts of inference rules which are relevant to a query. We used three strategies to add the random selection: proponent, opponent, both of them. We will present each of the algorithms in the following subsection.

4.2.1. Sampling on the Proponent

The dispute-based sampling on the proponent (DSP) method is performed based on the graphical dispute derivations (see Definition 3.4.1). Figure 16 intuitively describes the process of reasoning for a claim and sampling on the proponent. In this figure, P and O represent the proponent and the opponent, respectively, in the process of derivations. They attack each other to defend themselves. We apply random sampling to get the approximate inference rules on the proponent. The method is described by Algorithm 2 in detail. Moreover, the algorithm presents the main function $Derive()$. In the function $Derive()$, we use a random selection function $findRandomRules()$. The data of the algorithm consists of an ABA framework `Abaf`, a query `Query` and a probability `probability`. The `Query` is in the form of a sentence $s \in \mathcal{L}$. The initialization is to create a dispute sequence with the query and frameworks as $(\mathtt{P}_0, \mathtt{O}_0, \mathtt{G}_0, \mathtt{D}_0, \mathtt{C}_0)$ (see Definition 3.4.1). $\mathtt{P}_0$ is a potential argument graph and $\mathtt{O}_0$ is an empty set. $\mathtt{G}_0$ is a graph with a single node `Query`: s. If s is an assumption, then $\mathtt{D}_0$ is the set of the node s. Otherwise, $\mathtt{D}_0$ is an empty set. And

$\mathtt{C}_0$ is empty. After that, we use the function *Derive*() to get a state which decides the final answer.

Algorithm 2: Dispute-based Sampling on the Proponent

```
Data: Abaf, Query, probability
Result: Answer
Initialization disputeSequence: $P_0, O_0, G_0, D_0, C_0$ by Query and Abaf;
state ← Derive(disputeSequence, Abaf,probability);
if state = successful then
    Answer ← yes;
else
    Answer ← no;
end
Function Derive (disputeSequence, Abaf, probability):
    if getUnNode($P_i$) =∅ and getUnGraph($O_i$) = ∅ then
        state ← succesful;
    end
    while getUnNode($P_i$) ≠∅ or getUnGraph($O_i$) ≠∅ do
        side ← selectPorO($P_i$, $O_i$) ;
        if side="P" then
            node ← selectNode(getUnNode($P_i$));
            if node ∈ Assums then
                state ← Case1i(disputeSequence, Abaf, node);
            else
                Rules← findRandomRules(node,probability);
                state ← Case1ii(disputeSequence, Abaf, node, Rules,
                 probability)
            end
        else
            //opponent part;
            graph ← selectGraph(getUnGraph($O_i$));
            node ← selectNode(getUnNode(graph) ) ;
            if node ∈ Assums then
                state ←Case2i(disputeSequence, Abaf, node, graph,
                 probability);
            else
                Rules←findRules(node);
                Case2ii(disputeSequence, Abaf, node,graph,Rules)
            end
        end
    end
    return state;
```

The function $Derive()$ takes a dispute sequence and an ABA framework as inputs, and it returns a successful or failed state. Firstly, the function tests whether there are unmarked nodes and unmarked graphs in $\texttt{P}_i$ or $\texttt{O}_i$ by judging whether the returns of functions $getUnNode()$ and $getUnGraph()$ are empty or not. If not, the function returns a successful state. Otherwise, the function chooses an unmarked node from the proponent or the opponent. For the opponent part, the function select an unmarked graph to choose unmarked nodes. According to whether the selected node is an assumption or not, we use the different functions of $Case\text{-}X$ ($1i$, $1ii$, $2i$, $2ii$). To sample and construct the approximate rules for a node, we use the function $findRandomRules()$ when we select the node from $\texttt{P}_i$ and the node is not an assumption. The standard algorithm can be recovered by changing the line 19 in Algorithm 2 with the function $findRules()$ so that `Rules` will contain all rules for the given node. The function $findRandomRules()$ is to get approximate rules for the given head of rules by selecting and constructing approximate rules with a certain probability based on the original rules. It is the core function to implement the approximate reasoning and we describe it in Algorithm 3. The function $findRandomRules()$ takes `node` and `probability` as inputs, and returns a set of approximate rules `randRules`.

Algorithm 3: findRandomRules

```
1  Function findRandomRules(node, probability):
2      ruleList ← findRules(node) ;
3      numOfRules ← size(ruleList) * probability ;
4      initial(randRules );
5      while size(randRules) < numOfRules do
6          randNum ← randInteger(0, size(ruleList));
7          rule ← get(ruleList, randNum);
8          premises ← getPremise(rule);
9          initial(randPremises );
10         for i = 0; i < size(premises); i = i + 1 do
11             randNum2 ← randDouble (0, 1);
12             if randNum2 > probability then
13                 continue;
14             else
15                 randPremises ← addPremise(randPremises
                    ,get(premises, i));
16             end
17         end
18         nRule ← construct(randPremises,node);
19         randRules ← addRule(randRules, nRule );
20     end
21     return randRules;
```

Firstly, Algorithm 3 uses the function $findRules()$ to find all rules whose head is labelled by the given `node`. Then, it randomly samples on rules with the number of `numOfRules` from the set of rules. The `numOfRules` is calculated by the size of all rules multiplying with the `probability`. The function $size()$ is to get the size of the input set. Next, it initializes an empty set of rules by the function $initial()$. The functions $randInteger()$ and $randDouble()$ are to generate a random number of the int and double type respectively. For each sampled rule `rule`, Algorithm 3 constructs corresponding approximate rule by adding every sentence in `premise` to `randPremises` if the random number `randNum2` is less than the variable `probability`. The function $construct()$ is used to construct a rule according to the given premises `randPremise` as the body, and the label of `node` as the head. Finally, this algorithm constructs a set of approximate rules for the given node and probability.

Function Name	Input	Output	illustration
$getUnNode$	a graph	a set of unmarked node	get the set of unmarked nodes
$getUnGraph$	a set of graphs	a set of unmarked graphs	get the set of unmarked graphs
$selectPorO$	a graph, a set of graphs	"P" / "O"	select p or o according to the input.
$selectNode$	a set of nodes	node	randomly get a node from the input set
$selectGraph$	a set of graphs	graph	randomly get a graph from the input set
$findRules$	node	set of rules	get all the rules whose head is the label of the input node
get	an ordered collection, an index	an element	get the element at the input index position from the input collection
$addPremise$	a set of premises, a premise	a set of premises	add a premise to the set of premises
$addRule$	a set of rules, a rule	a set of rules	add a rule to the set of rules

Table 8: Functions used in Algorithm 2 and 3

In Algorithm 2, we use four complicated functions $Case1i()$, $Case1ii$, $Case2i()$,

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initilization	$(\{\boldsymbol{p}\}, \emptyset)$	$\emptyset$	$(\{p\}, \emptyset)$	$\emptyset$	$\emptyset$
1	Case1ii	$(\{p, \boldsymbol{b}\}, \{(p, b)\})$	$\emptyset$	$(\{p, b\}, \{(p, b)\})$	$\{b\}$	$\emptyset$
2	Case1i	$(\{p, b\}, \{(p, b)\})$	$\{(\{\boldsymbol{y}\}, \emptyset)\}$	$(\{p, b, y\}, \{(p, b), (y, b)\})$	$\{b\}$	$\emptyset$
3	Case2ii	$(\{p, b\}, \{(p, b)\})$	$\emptyset$	$(\{p, b, y\}, \{(p, b), (y, b)\})$	$\{b\}$	$\emptyset$

Table 9: A dispute sequence in example 4.1 using DSP

and $Case2ii()$, which are described in the appendix of algorithm functions. And we explain other functions that we use in Algorithm 2 and Algorithm 3 in Table 8. To illustrate these algorithms, we give an example 4.1 as follows.

Example 4.1 *Given an ABA framework ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$,$^{-}$) where*
$\mathcal{L}$ = { a, b, c, p, s, t, x, y, z }
$\mathcal{A}$ = { a, b, c }
$\mathcal{R}$ = { $p \leftarrow b$, $p \leftarrow s, t$, $s \leftarrow a$, $t \leftarrow c$ }
$\bar{a} = x$, $\bar{b} = y$, $\bar{c} = z$

If we use the DSP method to answer the query "p". In the step 0, we initialize the dispute sequence as P_0 = ({p}, $\emptyset$), G_0 = ({p}, $\emptyset$). Furhermore, O_0, D_0, and C_0 are empty. In each step, the dispute sequence is changed shown in Table 9. The unmarked node is shown in the bold style. And O_i only shows the unmarked graph in the table. Then we select the unmarked node p in the proponent, and expand it to construct the potential arguments supporting the claim p. By randomly selecting and constructing the inference rules $p \leftarrow b$, we can finally get a successful dispute sequence. However, if we randomly select the rule $p \leftarrow s, t$ and construct it as the rule $p \leftarrow s$, the dispute sequence would be different shown in Table 10. In this situation, the DSP method gets the correct answer for both choices.

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initilization	$(\{\boldsymbol{p}\}, \emptyset)$	$\emptyset$	$(\{p\}, \emptyset)$	$\emptyset$	$\emptyset$
1*	Case1ii	$(\{p, \boldsymbol{s}\}, \{(p, s)\})$	$\emptyset$	$(\{p, s\}, \{(p, s)\})$	$\emptyset$	$\emptyset$
2	Case1ii	$(\{p, s\}, \{(p, s)\})$	$\emptyset$	$(\{p, s, a\}, \{(s, p), (a, s)\})$	$\{a\}$	$\emptyset$

Table 10: An alternative dispute sequence in example 4.1 using DSP

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initilization	$(\{\boldsymbol{p}\}, \emptyset)$	$\emptyset$	$(\{p\}, \emptyset)$	$\emptyset$	$\emptyset$
1*	Case1ii	$(\{p, \boldsymbol{x}\}, \{(p, x)\})$	$\emptyset$	$(\{p, x\}, \{(x, p)\})$	$\emptyset$	$\emptyset$
2	Failure	-	-	-	-	-

Table 11: A failed dispute sequence in example 4.1 using DSP

When we consider Example 4.1, if $\mathcal{R}$ = { $p \leftarrow b$, $p \leftarrow s, t$, $s \leftarrow a$, $t \leftarrow c$, $p \leftarrow x$ }, the query is still p. The initialization is the same as in the Table 10. However, if the rule $p \leftarrow x$ is randomly sampled in the step 1, p is expanded to x by this rule. The dispute sequence is shown in Table 11. Because x is not an assumption and there is no rules starting with the head of x, the sequence can never reach a successful state and we can not get the correct answer.

4.2.2. Sampling on the Opponent

Along the lines of the method mentioned in the previous subsection, the dispute-based sampling on the opponent (DSO) method is also based on graphical dispute derivations, but it randomly samples and constructs inference rules on the opponent side. From the procedure of graphical dispute derivations, the opponent needs to consider all possible inference rules when constructing argument graphs to attack the proponent. Moreover, there are many rules to expand an unmarked node in the set of argument graphs. We apply the random sampling on the opponent to construct an approximate set of argument graphs by randomly selecting and constructing inference rules. Figure 17 gives a more intuitive presentation for this method over a claim. In this figure, P and O represent the proponent and the opponent, respectively. They attack each other in turn until one side cannot attack the other side. This kind of random sampling may lose some inference information during the reasoning. It can be seen as a random pruning way to reduce the searching space. However, it can not guarantee the answer is always right.

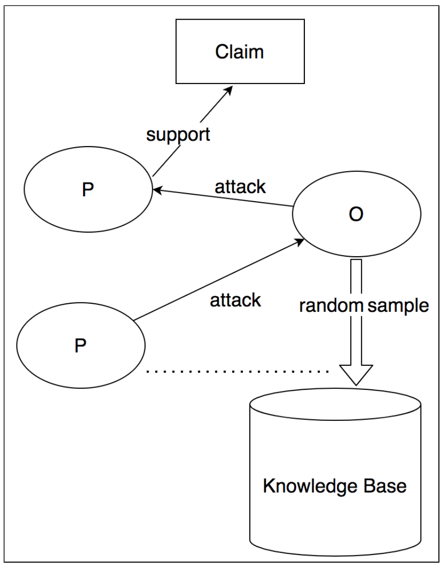

Figure 17: A sample strategy on the opponent side

Algorithm 4 describes the method DSO in detail. Most of the variables used in this algorithm are the same as those appearing in the previous method. This algorithm

Algorithm 4: Dispute-based Sampling on the Opponent

```
Data: Abaf, Query, probability
Result: Answer
1  initialization disputeSequence: $P_0, O_0, G_0, D_0, C_0$ with Abaf and Query ;
2  state ← Derive(disputeSequence, Abaf);
3  if state = successful then
4      Answer ← yes;
5  else
6      Answer ← no;
7  end
8  Function Derive (disputeSequence, Abaf, probability):
9      if getUnNode($P_i$) =∅ and getUnGraph($O_i$) = ∅ then
10         state ← succesful;
11     end
12     while getUnNode($P_i$) ≠∅ or getUnGraph($O_i$) ≠∅ do
13         side ← selectPorO($P_i$, $O_i$) ;
14         if side=P then
15             node = selectNode(getUnNode($P_i$) ) ;
16             if node ∈ Assums then
17                 state ← Case1i(disputeSequence, Abaf, node);
18             else
19                 Rules← findRules(node);
20                 state ← Case1ii(disputeSequence, Abaf, node, Rules,
                    probability)
21             end
22         else
23             //opponent part;
24             graph ← selectGraph(getUnGraph($O_i$));
25             node ← selectNode(getUnNode(graph) ) ;
26             if node ∈ Assums then
27                 state ← Case2i(disputeSequence, Abaf, node, graph,
                    probability);
28             else
29                 Rules← findRandomRules(node,probability);
30                 Case2ii(disputeSequence, Abaf, node, graph, Rules)
31             end
32         end
33     end
34     return state;
```

mainly uses the function $Derive()$, and takes a data structure of dispute sequence consisting of five components $\mathtt{P}_i$, $\mathtt{O}_i$, $\mathtt{G}_i$, $\mathtt{D}_i$, $\mathtt{C}_i$, an ABA framework `Abaf` and a probability `probability` as inputs. Moreover, it returns a `state` with successful or not. This function ends with the condition if there is no unmarked node in the graph $\mathtt{P}_i$ and no unmarked graph in the set of graphs $\mathtt{O}_i$. If not, we select an unmarked node from $\mathtt{P}_i$ or an unmarked graph in $\mathtt{O}_i$. If the former one is chosen, the operations are based on definition 3.4.1- condition 1. If the label of the node is an assumption, we apply function $Case1i()$, otherwise, apply function $findRules()$ and $Case1ii()$ so as to get the `state`. If the latter one is chosen, the operations are based on definition 3.4.1- condition 2 and we still need to judge whether the node's label is in the set of assumptions `Assums`. Before judging, the program selects an unmarked graph `graph`, then selects an unmarked node from the `graph` by function $selectGraph()$ and $selectNode()$. If the label of the selected node is an assumption, we apply function $Case2i()$, otherwise perform functions $findRandomRules()$ and $Case2ii()$. The standard method can also be recovered by replacing line 29 with $findRules()$ so that `Rules` contains all rules for the given node. These functions are the same as those in the previous method.

Let us see an example for using the DSO method.

Example 4.2 *Given an ABA framework ($\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$, $\bar{\ }$) where*
$\mathcal{L}$ = { a, b, p, q, s, t }
$\mathcal{R} = \{p \leftarrow b, t,\ p \leftarrow b, s\ ,\ q \leftarrow\ \}$
$\mathcal{A}$ = { a, b}
$\bar{a} = p$, $\bar{b} = q$

If the query is "a", the dispute sequence generated by the DSO method is shown in Table 12. In the step 2, the function $findRandomRules()$ returns $p \leftarrow b, t$ rather than all rules with the head of p in standard graphical dispute derivations. Finally, the function $Derive()$ returns a successful state, and we get the correct answer by this sampling on rules compared to the standard method.

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initilization	$(\{\boldsymbol{a}\}, \emptyset)$	$\emptyset$	$(\{a\}, \emptyset)$	$\{a\}$	$\emptyset$
1	Case1i	$(\{a\}, \emptyset)$	$\{(\{\boldsymbol{p}\}, \emptyset)\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
2	Case2ii	$(\{a\}, \emptyset)$	$\{(\{p, \boldsymbol{b}, \boldsymbol{t}\}, \{(p, b), (p, t)\})\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
3	Case2i	$(\{a, \boldsymbol{q}\}, \emptyset)$	$\emptyset$	$(\{a, p, q\}, \{(p, a), (q, p)\})$	$\{a\}$	$\{b\}$
4	Case1ii	$(\{a, q\}, \emptyset)$	$\emptyset$	$(\{a, p, q\}, \{(p, a), (q, p)\})$	$\{a\}$	$\{b\}$

Table 12: A dispute sequence in example 4.2 using DSO

In the step 2, if the rule $p \leftarrow b, s$ is randomly selected and constructed, the corresponding dispute sequence is shown in Table 13. In this situation, we can get a successful dispute sequence and then still get the correct answer for this example.

We continue to consider Example 4.2, if $\mathcal{R} = \{p \leftarrow b, t,\ p \leftarrow b, s\ ,\ q \leftarrow,\ p \leftarrow$ c $\}$, and c is an assumption with its contrary m. In the step 2, we randomly select and

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initilization	$(\{\boldsymbol{a}\}, \emptyset)$	$\emptyset$	$(\{a\}, \emptyset)$	$\{a\}$	$\emptyset$
1	Case1i	$(\{a\}, \emptyset)$	$\{(\{\boldsymbol{p}\}, \emptyset)\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
2*	Case2ii	$(\{a\}, \emptyset)$	$\{(\{p, \boldsymbol{b}, \boldsymbol{s}\}, \{(p, b), (p, s)\})\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
3	Case2i	$(\{a, \boldsymbol{q}\}, \emptyset)$	$\emptyset$	$(\{a, p, q\}, \{(p, a), (q, p)\})$	$\{a\}$	$\{b\}$
4	Case1ii	$(\{a, q\}, \emptyset)$	$\emptyset$	$(\{a, p, q\}, \{(p, a), (q, p)\})$	$\{a\}$	$\{b\}$

Table 13: An alternative dispute sequence in example 4.2 using DSO

construct the rule $p \leftarrow c$. Table 14 represents the sequence for this situation. The standard dispute derivation doesn't finish with a successful dispute sequence, but if we use the DSO method and get a dispute sequence shown in Table 14, we can not get a correct answer in this situation.

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initilization	$(\{\boldsymbol{a}\}, \emptyset)$	$\emptyset$	$(\{a\}, \emptyset)$	$\{a\}$	$\emptyset$
1	Case1i	$(\{a\}, \emptyset)$	$\{(\{\boldsymbol{p}\}, \emptyset)\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
2*	Case2ii	$(\{a\}, \emptyset)$	$\{(\{p, \boldsymbol{c}\}, \{(p, c)\})\}$	$(\{a, p\}, \{(p, a)\})$	$\{a\}$	$\emptyset$
3	Case2i	$(\{a, \boldsymbol{m}\}, \emptyset)$	$\emptyset$	$(\{a, p, m\}, \{(p, a), (m, p)\})$	$\{a\}$	$\{c\}$
4	Failure	-	-	-	-	-

Table 14: A failed dispute sequence in example 4.2 using DSO

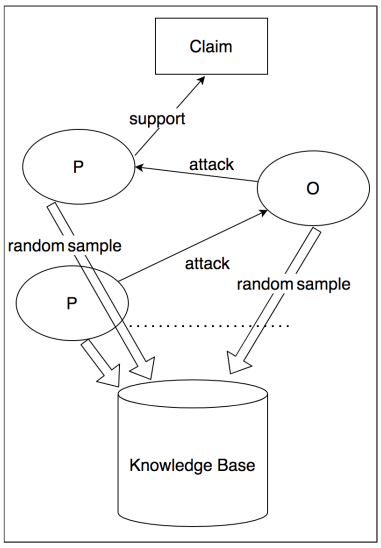

Figure 18: A sample strategy on the proponent and opponent sides

4.2.3. Sampling on Both Sides

We implement the dispute-based sampling on both sides (DSB) method by sampling inference rules on the proponent and the opponent based on graphical dispute derivations (see Definition 3.4.1). It can be seen as a combination of two methods DSP and DSO mentioned in the previous subsections. Moreover, the process for this method is depicted in Figure 18. The claim represents a query of whether the claim is accepted under the grounded semantics. "P" and "O" represent the proponent and opponent, respectively. Random sampling is applied on both sides to construct approximate reasoning. This process combines the sample operation on both sides so that the running time might be shorter than the DSP or the DSO.

The algorithm for the DSB method is similar to the previous methods DSP and DSO. Besides, it uses function $findRandomRules()$ on both of the proponent and opponent sides. The random sampling on the proponent would influence the generation of unmarked nodes on the opponent. And the random sampling on the opponent would affect the generation of unmarked nodes on the proponent. The DSB method may lead to a lower accuracy than the DSP and the DSO. Algorithm 5 describes this method as follows. The initialization is to create a dispute sequence by initializing P_0, G_0, O_0, D, C_0 with the `Query` and `Abaf` based on Definition 3.4.1. P_0 is a graph with the single node labelled by the `Query` and the single node is unmarked. O_0 is an empty set. G_0 is a graph with single node labelled by the `Query`. If the query is an assumption in `Abaf`, then D_0 is the set of the single node labelled by the `Query`, otherwise D_0 is an empty set. C_0 is an empty set. Then the algorithm

Algorithm 5: Dispute-based Sampling on Both

```
  Data: Abaf, Query, probability
  Result: Answer
1 Initialization disputeSequence: $P_0, O_0, G_0, D_0, C_0$ with Abaf and Query ;
2 state ← $Derive$(disputeSequence, Abaf, probability);
3 if state = $successful$ then
4 |   Answer ← yes;
5 else
6 |   Answer ← no;
7 end
```

performs the function $Derive()$. According to the return of the function $Derive()$ whether the `state` is successful or not, we can obtain the answer to the query.

Note that the function $Derive()$ is slightly different from the function described in previous, although it has the same name and same parameters. In this $Derive()$ function, the $findRandomRules()$ is used in both the "P" and "O" sides so that the `Rules` contains the set of approximate rules for the given head compared to the standard method. The set of approximate rules is obtained by randomly sampling inference rules for a given `node`. For each sampled rule, we reconstruct the body of the rule by randomly choosing premises with a `probability` from the original set of premises. Other functions used in Algorithm 5 are the same as those in the DSP and the DSO.

```
8  Function Derive(disputeSequence, Abaf,probability):
9      if getUnNode($P_i$) =∅ and getUnGraph($O_i$) = ∅ then
10         state ← succesful;
11     end
12     while getUnNode($P_i$) ≠∅ or getUnGraph($O_i$) ≠∅ do
13         side ← $selectPorO$($P_i$, $O_i$) ;
14         if side="P" then
15             node ← $selectNode$($getUnNode$($P_i$));
16             if node ∈ Assums then
17                 state ← $Case1i$(disputeSequence, Abaf, node);
18             else
19                 Rules← findRandomRules(node,probability);
20                 state ← $Case1ii$(disputeSequence, Abaf, node, Rules,
                    probability)
21             end
22         else
23             //opponent part;
24             graph ← $selectGraph$($getUnGraph$($O_i$));
25             node ← $selectNode$($getUnNode$(graph) ) ;
26             if node ∈ Assums then
27                 state ←$Case2i$(disputeSequence, Abaf, node, graph,
                    probability);
28             else
29                 Rules←findRandomRules(node, probability);
30                 $Case2ii$(disputeSequence, Abaf, node,graph,Rules)
31             end
32         end
33     end
34     return state;
```

To illustrate this method, we consider the following example.

Example 4.3 *Given an ABA framework ($\mathcal{L}$, $\mathcal{R}$,$\mathcal{A}$,$\bar{}$) where*
$\mathcal{L}$ = {p, q, s, t, a, b, c }
$\mathcal{R}$ = { $p \leftarrow a$, $p \leftarrow b$, $t \leftarrow c$, $t \leftarrow$, $q \leftarrow$, $s \leftarrow$ }
$\mathcal{A} = \{a, b, c\}$
$\bar{a} = t$, $\bar{b} = q$, $\bar{c} = s$

For the above example, if we use the DSB method to determine whether the claim p is accepted under the grounded semantics, the query is "p." Then we get the dispute sequence shown in Table 15. In the step 1, this method randomly samples and constructs the rule $p \leftarrow a$ on the proponent side. And in the step 3, it randomly samples and constructs the rule $t \leftarrow c$ on the opponent side. From the Table 15, we can get a successful dispute sequence in the step 5 and the answer for this query is "yes". The actual answer should be "no" in the standard method, but the DSO method can not get the correct answer in this situation.

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initialization	$(\{\boldsymbol{p}\}, \emptyset)$	$\emptyset$	$(\{p\}, \emptyset)$	$\emptyset$	$\emptyset$
1	Case1ii	$(\{p, \boldsymbol{a}\}, \{(p, a)\})$	$\emptyset$	$(\{p, a\}, \{(a, p)\})$	$\{a\}$	$\emptyset$
2	Case1i	$(\{p, a\}, \{(p, a)\})$	$\{(\{\boldsymbol{t}\}, \emptyset)\}$	$(\{p, a, t\}, \{(a, p), (t, a)\})$	$\{a\}$	$\emptyset$
3	Case2ii	$(\{p, a\}, \{(p, a)\})$	$\{(\{t, \boldsymbol{c}\}, \{(t, c)\})\}$	$(\{p, a, t\}, \{(a, p), (t, a)\})$	$\{a\}$	$\emptyset$
4.	Case2i	$(\{p, a, \boldsymbol{q}\}, \{(p, a)\})$	$\emptyset$	$(\{p, a, t, q\}, \{(a, p), (t, a), (q, t)\})$	$\{a\}$	$\emptyset$
5.	Case1ii	$(\{p, a, q\}, \{(p, a)\})$	$\emptyset$	$(\{p, a, t, s\}, \{(a, p), (t, a), (s, t)\})$	$\{a\}$	$\emptyset$

Table 15: A dispute sequence in example 4.3 using DSB

We continue to Example 4.3, for the same query, if in the step 1, this method randomly samples and constructs the rule $p \leftarrow b$ to update the graph P_i. According to $\bar{b}$ = q, a new unmarked graph with a single unmarked node q is added into O_i. Finally, we can get the dispute sequence which indicates a successful state shown in Table 16. Thus this method does not give the correct answer.

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initialization	$(\{\boldsymbol{p}\}, \emptyset)$	$\emptyset$	$(\{p\}, \emptyset)$	$\emptyset$	$\emptyset$
1*	Case1ii	$(\{p, \boldsymbol{b}\}, \{(p, b)\})$	$\emptyset$	$(\{p, b\}, \{(b, p)\})$	$\{b\}$	$\emptyset$
2	Case1i	$(\{p, b\}, \{(p, b)\})$	$\{(\{\boldsymbol{q}\}, \emptyset)\}$	$(\{p, b, q\}, \{(a, p), (q, b)\})$	$\{b\}$	$\emptyset$
3	Case2ii	$(\{p, b\}, \{(p, b)\})$	$\{(\{q\}, \emptyset)\}$	$(\{p, a, t\}, \{(a, p), (t, a)\})$	$\{b\}$	$\emptyset$

Table 16: An alternative dispute sequence in example 4.3 using DSB

However, if in the step 3 of Table 15, the DSO method gets the rule $t \leftarrow$ rather than the rule $t \leftarrow c$ by the random selection function, the following process is different. We shoe the new dispute sequence for this situation in Table 17. And this leads to a failed state so that the DSO method provides a correct answer in this situation.

Step	Case	P_i	O_i	G_i	D_i	C_i
0	Initialization	$(\{\boldsymbol{p}\}, \emptyset)$	$\emptyset$	$(\{p\}, \emptyset)$	$\emptyset$	$\emptyset$
1	Case1ii	$(\{p, \boldsymbol{a}\}, \{(p, a)\})$	$\emptyset$	$(\{p, a\}, \{(a, p)\})$	$\{a\}$	$\emptyset$
2	Case1i	$(\{p, a\}, \{(p, a)\})$	$\{(\{\boldsymbol{t}\}, \emptyset)\}$	$(\{p, a, t\}, \{(a, p), (t, a)\})$	$\{a\}$	$\emptyset$
3*	Case2ii	$(\{p, a\}, \{(p, a)\})$	$\{(\{t\}, \emptyset)\}$	$(\{p, a, t\}, \{(a, p), (t, a)\})$	$\{a\}$	$\emptyset$

Table 17: A failed dispute sequence in example 4.3 using DSB

This section proposed three different dispute-based sampling methods. The three methods DSP, DSO, and DSB, randomly sampled and constructed inference rules in the process of graphical dispute derivations. Based on the different operations in different stages of derivations, we used the random selection function $findRandom$-$Rules()$ on the proponent, the opponent, and both of them for the DSO, the DSP, and the DSB method, respectively. We presented these methods in different algorithms with some same functions. Algorithm 2, 4, and 5 were similar, except for the position where we used the random selection function. The expected results were that the DSP, DSO, and DSO would have a better accuracy than the FS method mentioned in the previous section, and the FS method would use less time than other methods. The proponent was more active than the opponent because it put forward arguments to support a claim, which drove the opponent to attack. Moreover, each of the rules supporting the proponent would lead to a possible way to get a successful branch. Nevertheless, all the rules together supporting the opponent decided to continue a branch. Random sampling on both sides might lead to a lower accuracy and less time than on only one side because the proponent and the opponent affected each other in every step, which resulted in a smaller searching space. We will evaluate all of these methods compared with the baseline in the next chapter.

5. Evaluation

This section aims to evaluate the approximate methods described in Section 4 by experiments. All the methods are implemented in Java. These experiments are designed to test the performance of different methods. We take the standard graphical dispute derivation method as the baseline. The experiments are performed by running different methods to answer the same query and comparing each method's performance according to the statistic result. Section 5.1 describes the data generated for the experiment and evaluation metrics for different methods. Section 5.2 shows the statistical results by comparing the baseline and approximate methods. Moreover, it summarizes the experiment and discusses the results. The results indicate that, in general, the approximate methods can reduce running time and get a slightly lower accuracy.

5.1. Experiment

5.1.1. Data

We used an ABA theory generator with different parameters to generate the data that would be used in the following experiments. The generator is also implemented in Java. This program can generate different ABA frameworks with various settings of parameters. Furthermore, each parameter has its influence on the generated frameworks. One of the components of an ABA framework is the language. nS is a parameter to control the number of elements in the language. nA is the number of assumptions in the language, and it can not be greater than the number of sentences. The number of inference rules are affected by nDH, $minR$, and $maxR$. And the number of sentences in the body of inference rules is affected by $minB$ and $maxB$. The parameter $minA$ and $maxA$ decide the number of assumptions in the body of rules. They are all described in Table 18.

Name of parameter	Meaning
nS	the number of sentences
nA	the number of assumptions
nDH	the number of distinct head
$minR$	minimum number of rules per head
$maxR$	maximum number of rules per head
$minB$	minimum number of sentences in the body for one rule
$maxB$	maximum number of sentences in the body for one rule
$minA$	minimum number of assumptions in the body for one rule
$maxA$	maximum number of assumptions in the body for one rule

Table 18: parameters in random generate ABA frameworks

The following Algorithm 6 illustrates the process of the generation of an ABA framework. In this algorithm, firstly, the program generates a set of sentences to represent languages according to the parameter nS, and each sentence is in the form of Ai, $i \in \{0...n\}$. Then it randomly generates a subset with the size nA of the set of sentences as assumptions. Moreover, the program randomly generates corresponding negation for each assumption from the subtraction between the set of sentences and assumptions. Finally, it constructs the set of inference rules. In this process, firstly, it creates heads and bodies for rules from sentences. The head is only in the set of subtraction between the set of all sentences and assumptions. Moreover, the premise in each body is an assumption or a non-assumption. It depends on the number of assumptions in the body.

Algorithm 6: Random ABA Theory Generator

```
   Data: $nS$, $nA$, $nDH$, $minR$,$maxR$,$minB$,$maxB$,$minA$,$maxA$
   Result: Abaf
 1 for int i=0; i<nS; i++ do
 2     generate a sentence s;
 3     $L \leftarrow L \cup s$;
 4 end
 5 for int i=0;i<nA;i++ do
 6     random choose a sentence $a \in L$;
 7     $A \leftarrow A \cup a$;
 8     random choose a sentence $n \in L\text{-}A$;
 9     $N \leftarrow N \cup \{not\ a{=}n\}$;
10 end
11 for int i=0;i<nDH;i++ do
12     random choose a sentence $h \in L\text{-}A$;
13     generate numOfRulesin [minR,maxR];
14     for int j=0;j<numOfRules;j++ do
15         generate numOfBodys in [minB,maxB];
16         generate numOfAssums in [minA,maxA];
17         for int t=0;t<numOfAssums;t++ do
18             random choose a sentence $a_1 \in A$;
19             $r_{body} \leftarrow r_{body} \cup a_1$
20         end
21         for int k=0; k<numOfBodys-numOfAssums; k++ do
22             random choose a sentence $l_1$;
23             $r_{body} \leftarrow r_{body} \cup l_1$;
24         end
25         construct the $r : h \leftarrow r_{body}$;
26         $rules \leftarrow rules \cup r$
27     end
28 end
29 return Abaf( L, A, N, rules);
```

•

```
A11 <- A14
A17 <- A1, A2
A18 <- A10, A5, A16
A19 <- A7, A17
A19 <- A12
A3 <- true
A11 <- true
A13 <- true
A18 <- A4, A16, A19
A17 <- A6, A7, A8
A11 <- A15, A16
A18 <- A19
{A10,A5,A15,A6,A16,A8}
not A15 = A1
not A16 = A11
not A6 = A9
not A5 = A11
not A10 = A2
not A8 = A4
```

Figure 19: An ABA framework generated from the generator

By applying this generator several times, we generate different ABA frameworks. One example of the results is shown in Figure 19. This framework consists of three parts. First part is the set of inference rules in the form of $A_i \leftarrow A_j, A_k.....A_z$. A_i is the head of the rule. And $A_j, A_k.....A_z$ is the body of the rule. The body consists of literals or "true", which means this rule is strict enough not to be attacked by others. Then, the collection in a curly bracket is the set of assumptions. The last part is the set of negations and their corresponding assumptions. The negation is represented by adding a "not" before an assumption. For instance, the "not $A15 = A1$", we say the negation of the assumption $A15$ is $A1$. In order to test the performance of these methods, we design three datasets with a small, medium, and large number of instances, respectively. Moreover, each instance in different datasets is set with different parameters. The parameters for generating ABA frameworks are not independent. The sentences are all the elements in the language for ABA frameworks. The number of assumptions and the number of distinct heads can not be greater than sentences. If the number of distinct heads is higher, the number of rules would be higher. These different datasets are used to test the performance for the baseline and approximate methods.

DataSet1 The first dataset that we used has 80 ABA frameworks. Furthermore, the number of sentences is in the range [20,90]. We have done some informal experiments to test the generator. The informal experiments show that if parameters

Parameters	Values
nS	$\{20, 30, 40, 50, 60, 70, 80, 90\}$
nA	10
nDH	20
$minR$	1
$maxR$	3
$minB$	0
$maxB$	3
$minA$	0
$maxA$	3

Table 19: Parameters used to generate DataSet1

are smaller and the framework is simple. Parameters in this dataset are set to be small. The smallest number of sentences is 20. The number of assumptions and distinct heads can not be greater than this number. Parameters are shown in Table 19 in detail.

DataSet2 The second dataset we used has 900 ABA frameworks, and parameters are shown in Table 20. This dataset's parameters are set to be larger than the first dataset to make the instance in this dataset more complex than the first dataset. The number of sentences is one thousand for each instance. The number of assumptions is 150. We have done some informal experiments to test the running time for different methods. The informal experiments show that if the maximum number of inference rules is in the range [1400, 2200], the running time is significantly large. According to the $minR$ and $maxR$ shown in Table 20, the number of distinct head per rule is in the range [350,550].

Parameters	Values
nS	1000
nA	150
nDH	[350, 550]
$minR$	1
$maxR$	4
$minB$	0
$maxB$	4
$minA$	0
$maxA$	4

Table 20: Parameters used to generate DataSet1

DataSet3 The third dataset that we used has 4000 ABA frameworks, and in each framework, the number of sentences is randomly generated within the maximum number of 100. This dataset is larger than the first dataset in the number of frameworks. If the parameters are small, the result of the running time would be short. It is not enough to illustrate that the approximate methods reduce the running time dramatically. The number of assumptions and rules are also randomly generated by a probability of sentences. The parameters are shown in Table 21. These parameters are more flexible, and the range of some values is wider than the first and second datasets.

Parameters	Values
nS	$[1, 100]$
nA	$\{0.1, 0.2, 0.3, 0.4\}$
nDH	$\{0.1, 0.2, 0.3, 0.4\}$
$minR$	1
$maxR$	10
$minB$	0
$maxB$	10
$minA$	0
$maxA$	10

Table 21: Parameters used to generate DataSet2

5.1.2. Experiment Design

To evaluate the performance of the FS, DSP, DSO, and DSB methods, we design experiments based on three different scales of datasets. We use the DataSet1, DatSet2, and DataSet3. The experiments are performed by answering a randomly chosen query in different datasets using different methods. The first dataset represents a small and simple dataset. The second dataset describes the situations that thousands of inference rules in each instance. The third data set has 4000 frameworks randomly generated with a maximum number of 100 sentences. The maximum number of literals in the body is 10. The generator with these parameters could generate different frameworks, including simple and complex frameworks. We randomly select a query from the set of sentences for each framework. We design to run experiments using the baseline, FS, DSP, DSO, and DSB methods for answering the same query. The two parameters in the FS method are both 0.8, and the parameter of the probability in the DSP, DSO, DSB are all 0.8. Simultaneously, we record the computing time and answers. If the running time is more than five minutes, this query would be seen as an unsolved one. We compare the answers in the approximate methods with the baseline. In the end, we compute the statistical results and record them into tables.

Environment All the experiments mentioned above, including the data generation, were performed on a Mac OS equipped with the 2.9GHz Intel Core i7 processor and 16GB memory. The codes were implemented in Java and run on Eclipse with jdk-15.0.1.

Metrics We used two principal metrics to evaluate the performance of the reasoning process. The time slot computes the running time from starting the reasoning process until it ends. The accuracy is calculated by the number of answers that are the same as the baseline divided by the number of all tests. After that, we compared the average running time and accuracy for different methods. The statistical data provides a basis for analyzing approximate methods' performance and solving the second research question.

This section presented the design of experiments and the data for investigations. We used a random generator to generate the datasets with different parameters. The experiments were designed to compare the performance of the baseline and different approximate methods (FS, DSP, DSO, DSB) by two principal metrics. We will show the results of these experiments in the next section.

5.2. Results

Experiments designed in the previous section were performed, and results for each dataset are presented in this section. The results are shown in the form of tables. By doing so, we can get an intuitive impression of different methods' performance from different datasets. In the end, we summarize these experiments and results.

Table 22 shows the running time and accuracy for different methods in DataSet1. The FS, DSP, DSO, and DSB methods show lower values of time than the baseline. In this dataset, the running time is too short, so this does not actively demonstrates that approximate methods can reduce the running time. The FS shows the lowest accuracy of around 60% than other approximate methods. The DSP offers the highest accuracy of around 98%.

	Time	Accuracy	Solved
Baseline	3.2750	1.0000	1.0
FS	0.3000	0.6625	1.0
DSP	1.1000	0.9875	1.0
DSO	0.7875	0.9500	1.0
DSB	0.3250	0.8000	1.0

Table 22: The result for DataSet1 (The time is the average running time in milliseconds for answering different queries. The accuracy is calculated by the percentage of the correct answer. The solved is the percentage of the answer obtained in 5 minutes.)

	Time	Accuracy	Solved
Baseline	1150.564651	1.000000	0.905556
FS	315.083037	0.838285	0.908889
DSP	894.775458	0.983865	0.896667
DSO	1471.525697	0.905720	0.966667
DSB	352.970258	0.870617	0.930000

Table 23: The result for Dataset2 (The time is the average running time in milliseconds for answering different queries. The accuracy is calculated by the percentage of the correct answer. The solved is the percentage of the answer obtained in 5 minutes.)

The result for DataSet2 is shown in Table 23. This dataset has 800 ABA frameworks, and each of the frameworks has thousands of inference rules. It is more complicated than the first dataset according to the solved rate, which is consistent with the datasets' setting. The DSO method shows the longest running time than other methods. The particular reason for this circumstance is that the sample rules on the opponent side may lead to more sentences to expand and paths to search in dispute trees. The FS shows the shortest time from this result than other approximate methods, but the accuracy is the lowest. The reason for the circumstances is that the FS method without employing the query information may lose some rules that are crucial for constructing supporting arguments and attacking arguments over a query. The DSP shows a higher accuracy around 98% than other approximate methods. The FS and DSB show an accuracy of around 80%, and the DSO shows an accuracy of around 90%.

The result for DataSet3 is shown in Table 24. This dataset has 4000 frameworks, and each framework has hundreds of rules. For the third dataset, the parameter is more flexible than other datasets. It shows that the FS, DSO, DSB, and DSP approximate methods have a shorter running time than the baseline. It is consistent with our expectations, but it is not consistent with the result in DataSet2. The particular reason for this situation might be that the number of rules in DataSet2 is larger than in DataSet3 and the more complex paths are selected by the random selection function in the DSO method for DataSet2. The FS and DSB methods show a significant decrease in the running time. The DSP method has a higher accuracy of about 98% than other approximate methods. The FS has the lowest accuracy of about 80%, which is consistent with DataSet2 due to the same reason.

	Time	Accuracy	Solved
Baseline	221.145475	1.000000	0.988515
FS	23.745933	0.808872	0.989274
DSP	120.759371	0.984274	0.988931
DSO	199.813362	0.957797	0.993495
DSB	22.988376	0.917809	0.994338

Table 24: The result for DataSet3. (The time is the average running time in milliseconds for answering different queries. The accuracy is calculated by the percentage of the correct answer. The solved is the percentage of the answer obtained in 5 minutes.)

To sum up every result that has been stated so far, the DSP method had better accuracy than other approximate methods to reduce the running time. The DSO method did not perform better than the DSP, although they used the same function in different positions. In some situations, the DSO showed a longer runtime time than the baseline. It might be thought an explanation for this fact that the random function that was used on the opponent in the graphical dispute derivation resulted in the more complex searching paths. Moreover, for the IS method, it significantly reduced the running time but had a lower accuracy than most directed random sample methods. The FS used the random sample function before starting the actual reasoning process. The running time in the DSB was similar to it in the FS. However, the DSB showed a higher accuracy than the FS. Randomly sampling on both sides might cut more inference rules than the DSP or the DSO, but it still kept relevant to the query. The DSB increased the likelihood to reserve the crucial information for answering a query than the FS.

6. Future Work and Conclusion

6.1. Future Work

The evaluation indicates that the approximate methods based on graphical dispute derivations can reduce the running time with a lower accuracy. Although there are many different variants for the dispute derivations method, they all use the same underlying idea of simulating two players over a dispute. Therefore, one main aspect of future work is to investigate how a claim/conclusion can be determined to be accepted under other different semantics than the grounded semantics in ABA frameworks. One possible way is to directly construct the specific extensions according to the definition of different semantics. From the foundation in Chapter 2, the semantics in argumentation can also be represented in the labelling-based style. Therefore, another possible way is to construct labels for arguments under different semantics. Once different reasoning methods have been proposed, another aspect of future work would be to construct approximate methods based on these reasoning methods. Apart from randomly sampling a knowledge base to implement the approximate reasoning, we could do pre-computing for each inference rule and give each of them a weight value according to specific criteria. And then, we could use the rules that have a higher value with priority in reasoning.

As for the dataset, one interesting aspect of future work would be to investigate some practical applications for assumption-based argumentation. We can get knowledge bases from these applications as the data source to test whether approximate methods can work well on the real datasets.

6.2. Conclusion

In this thesis, we focused on reasoning problems in the formalism of assumption-based argumentation (ABA). Along the lines of semantics in abstract argumentation, ABA has many different semantics, including admissible, complete, grounded, etc. These different semantics provide a basis for reasoning problems. ABA is equipped with different computational mechanisms under different semantics for reasoning problems. Therefore, we mainly solved the research questions of how to perform approximate inference under the grounded semantics and whether approximate methods can improve the performance in ABA.

We proposed approximate methods based on the graphical dispute derivation under the grounded semantics in assumption-based argumentation to answer these research questions. In this context, the reasoning problem is to determine a claim/conclusion whether it can be accepted under specific semantics. Concretely, the query is a sentence in the language as a component of an ABA framework. For the exact purpose of improving the performance for answering a query, we presented the framework-based sampling (FS) method by randomly sampling frameworks to construct new frameworks and the dispute-based sampling methods (DSP, DSO, DSB) by randomly sampling in the process of the dispute derivation. We implemented the

DSP, DSO, and DSB methods (see Chapter 4) using the random selection function in different stages of the derivation. The FS method is different from others because it resizes ABA frameworks without employing the query's information before the actual reasoning process. The DSP, DSO, and DSB use a random selection function on the proponent, the opponent, and both, respectively, to get a set of approximate inference rules with the given head.

The evaluation for approximate methods has revealed that adding randomness in the process of constructing arguments to answer a query can reduce the computing time with the cost of a lower accuracy. However, when we used the random selection function in different stages, the result was different. The DSP method had a better accuracy than other approximate methods. The DSO method reduced potential argument graphs that we used to counterattack the proponent in the dispute sequence, resulting in a lower accuracy than the DSP method. In the standard graphical dispute derivations, the opponent needs to find all possible counterattacks. However, randomly sampling and constructing rules that support opponents reduces the chance to get the correct answer. From the experiment's results, the DSP method showed a shorter running time than the baseline and an accuracy of around 98% in the best situation. It indicated that the approximate method could reduce the running time with an acceptable accuracy. The FS method can reduce the running time but with a lower accuracy than other approximate methods. We can sample parts of rules in ABA frameworks in the process of the graphical dispute derivations to perform the approximate inference in ABA. If the reasoning problems are time emergency and do not require a high accuracy, we would consider the DSO method. For the reasoning problems that require high accuracy, we recommend the DSP method.

Acknowledgement

Throughout the writing of this thesis and my studies at the University of Koblenz-Landau, I have received a great deal of support and assistance.

I would like to thank my first supervisor, PD Dr. Matthias Thimm, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank my second supervisor, Dr. Tjitze Rienstra, for his valuable guidance throughout my thesis. I want to thank you for your patient support whenever I ran into a trouble spot or had a question with my research or writing and for all of the opportunities I was given to further my research. You provided me with the advice that I needed to choose the right direction and successfully complete my thesis.

I would also like to thank Ms. Ines Tobis, from International Office, for her assistance on many different procedures during my study program. I would also like to thank Ms. Anja Feege and Mr. Winfried Riemer, for their practical training on how to start studying and living in Germany.

In addition, I would like to thank my fellow students and friends Juli, Muhan, and Haoze, who provided useful suggestions for starting my thesis as well as happy distractions to rest my mind outside of my research. I would also like to say special thank you to my friend Mr. Li, for your support and patient when I was in bad and negative emotions.

Finally, I would like to thank my parents for providing me with financial support and encouragement throughout my years of study and throughout the process of researching and writing this thesis. You are always there for me. I could not have completed this thesis without the support of my parents.

A. Algorithm Functions

Algorithm 7: Case1i

```
Function Case1i(disputeSequence, Abaf, node):
    v($P_{i+1}$) ← v($P_i$) ∪ node;
    add node in unmarked set in $P_{i+1}$;
    negationNode ← findNegation(node);
    if negationNode = claim(graph), graph ∈ $O_i$ then
        $O_{i+1}$ ← $O_i$
    else
        $O_{i+1}$ ← $O_i$ ∪ newgraph(negationNode)
    end
    v($G_{i+1}$) ← v($G_i$) ∪ negationNode ∪ node ;
    e($G_{i+1}$) ← e($G_i$) ∪ (negationNode,node) ;
    if $G_{i+1}$ is cyclic then
        state ← failed;
    else
        state ← successful;
    end
    return state;
```

Algorithm 8: Case1i i

```
Function Case1ii(disputeSequence, Abaf, node, Rules, probability):
    for R in Rules do
        if $R \cap C_i = \emptyset$ then
            $v(P_{i+1}) \leftarrow v(P_i) \cup R$;
            $e(P_{i+1}) \leftarrow e(P_i) \cup (node, r), r \in R$;
            remove node from the unmarked set;
            $v(G_{i+1}) \leftarrow v(G_{i+1}) \cup node \cup r, r \in R$;
            $D_{i+1} \leftarrow D_{i+1} \cup (R \cap Assums)$;
            if $P_{i+1}$ or $G_{i+1}$ is cyclic then
                state $\leftarrow$ failed ;
                return state
            end
            state $\leftarrow$ Derive(disputeSequence, Abaf, probability);
        end
    end
    return state;
```

Algorithm 9: Case2i

```
Function Case2i(disputeSequence, Abaf, node, graph, probability):
    for int i=0;i<2;i++ do
        if i=0 then
            Operations(disputeSequence, Abaf, node, graph);
            set graph unmarked;
            state ← Derive(disputeSequence, Abaf, probability);
            if state is successful then
                return state;
            end
        end
        if i=1 then
            if node $\notin D_i$ and node $\in C_i$ then
                Operation2(disputeSequence, Abaf, node, graph);
                if $G_{i+1}$ is cyclic then
                    state ← failed;
                    return state;
                end
            else if node $\notin D_i$ and node $\notin C_i$ then
                Operation2(disputeSequence, Abaf, node, graph);
                if negationNode $\notin P_i$ then
                    $v(P_{i+1}) \leftarrow v(P_i) \cup$ negationNode;
                end
                $D_{i+1} \leftarrow D_i \cup$ (negationNode $\cap$ Assums);
                $C_{i+1} \leftarrow C_i \cup$ node;
                if $G_{i+1}$ is cyclic then
                    state ← failed;
                    return state;
                end
            else
                Operation1(disputeSequence, Abaf, node, graph);
                set graph unmarked;
            end
            state ← Derive(disputeSequence, Abaf, probability);
        end
    end
    return state;
Function Operation1(disputeSequence, Abaf, node, graph):
    remove graph from $O_i$;
    v(graph) ← v(graph) ∪ node;
    add node in marked set in graph;
    $O_{i+1} \leftarrow O_i \cup$ graph;
Function Operation2(disputeSequence, Abaf, node, graph):
    Operation1(disputeSequence, Abaf, node, graph);
    set graph marked;
    negationNode← findNegation(node);
    $v(G_{i+1}) \leftarrow v(G_i) \cup$ negationNode $\cup$ claim(node) ;
    $e(G_{i+1}) \leftarrow e(G_i) \cup$ (negationNode,claim(node)) ;
```

Algorithm 10: Case2ii

```
Function Case2ii(disputeSequence, Abaf, node):
    remove graph from $O_i$;
    for R in Rules do
        if R is true then
            add node in marked set in graph;
            add $C_i$ in marked set in graph;
            $O_{i+1} \leftarrow O_i \cup$ graph;
            set graph marked;
        end
        v(graph) $\leftarrow$ v(graph) $\cup$ R;
        e(graph) $\leftarrow$ e(graph) $\cup$ (node, r), $r \in R$;
        add node in marked set in graph;
        if graph is not cyclic then
            $O_{i+1} \leftarrow O_i \cup$ graph;
            set graph unmarked;
        end
    end
```

References

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

[2] Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. Assumption-based argumentation. In Guillermo Ricardo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 199–218. Springer, 2009.

[3] Yannis Dimopoulos, Bernhard Nebel, and Francesca Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artif. Intell.*, 141(1/2):57–78, 2002.

[4] Tuomo Lehtonen, Johannes Peter Wallner, and Matti Järvisalo. From structured to abstract argumentation: Assumption-based acceptance via AF reasoning. In Alessandro Antonucci, Laurence Cholvy, and Odile Papini, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 14th European Conference, ECSQARU 2017, Lugano, Switzerland, July 10-14, 2017, Proceedings*, volume 10369 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2017.

[5] Wolfgang Dvorák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. *FLAP*, 4(8), 2017.

[6] Robert Craven, Francesca Toni, and Matthew Williams. Graph-based dispute derivations in assumption-based argumentation. In Elizabeth Black, Sanjay Modgil, and Nir Oren, editors, *Theory and Applications of Formal Argumentation - Second International Workshop, TAFA 2013, Beijing, China, August 3-5, 2013, Revised Selected papers*, volume 8306 of *Lecture Notes in Computer Science*, pages 46–62. Springer, 2013.

[7] Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. Dialectic proof procedures for assumption-based, admissible argumentation. *Artif. Intell.*, 170(2):114–159, 2006.

[8] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artif. Intell.*, 171(10-15):642–674, 2007.

[9] Francesca Toni. A generalised framework for dispute derivations in assumption-based argumentation. *Artif. Intell.*, 195:1–43, 2013.

[10] Robert Craven and Francesca Toni. Argument graphs and assumption-based argumentation. *Artif. Intell.*, 233:1–59, 2016.

[11] Pietro Baroni and Massimiliano Giacomin. Semantics of abstract argument systems. In Guillermo Ricardo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

[12] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowl. Eng. Rev.*, 26(4):365–410, 2011.

[13] Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.

[14] Francesca Toni. A tutorial on assumption-based argumentation. *Argument Comput.*, 5(1):89–117, 2014.

[15] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[16] David S. Johnson. A catalog of complexity classes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 67–161. Elsevier and MIT Press, 1990.

[17] Matthias Thimm and Tjitze Rienstra. Approximate reasoning with ASPIC+ by argument sampling. In Sarah Alice Gaggl, Matthias Thimm, and Mauro Vallati, editors, *Proceedings of the Third International Workshop on Systems and Algorithms for Formal Argumentation co-located with the 8th International Conference on Computational Models of Argument (COMMA 2020), September 8, 2020*, volume 2672 of *CEUR Workshop Proceedings*, pages 22–33. CEUR-WS.org, 2020.