Algorithms for computational argumentation in artificial intelligence

Vasiliki Efstathiou

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

UCL.

Department of Computer Science

University College London

2010

I, Vasiliki Efstathiou, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Argumentation is a vital aspect of intelligent behaviour by humans. It provides the means for comparing information by analysing pros and cons when trying to make a decision. Formalising argumentation in computational environment has become a topic of increasing interest in artificial intelligence research over the last decade.

Computational argumentation involves reasoning with uncertainty by making use of logic in order to formalize the presentation of arguments and counterarguments and deal with conflicting information. A common assumption for logic-based argumentation is that an argument is a pair $\langle \Phi, \alpha \rangle$ where Φ is a consistent set which is minimal for entailing a claim α. Different logics provide different definitions for consistency and entailment and hence give different options for formalising arguments and counterarguments. The expressivity of classical propositional logic allows for complicated knowledge to be represented but its computational cost is an issue. This thesis is based on monological argumentation using classical propositional logic [12] and aims in developing algorithms that are viable despite the computational cost. The proposed solution adapts well established techniques for automated theorem proving, based on resolution and connection graphs. A connection graph is a graph where each node is a clause and each arc denotes there exist complementary disjuncts between nodes. A connection graph allows for a substantially reduced search space to be used when seeking all the arguments for a claim from a given knowledgebase. In addition, its structure provides information on how its nodes can be linked with each other by resolution, providing this way the basis for applying algorithms which search for arguments by traversing the graph. The correctness of this approach is supported by theoretical results, while experimental evaluation demonstrates the viability of the algorithms developed. In addition, an extension of the theoretical work for propositional logic to first-order logic is introduced.

Acknowledgements

I would like to thank my supervisor Tony Hunter for his guidance on my work and his continuous encouragement and advice during my PhD. I am grateful to Tony for introducing me to the area of argumentation and giving me the chance to work with him on the Argumentation Factory project. I also wish to thank Robin Hirsch and Emmanuel Letier for assessing my progress during my PhD. Thanks to Nikos Gorogiannis for being always willing to help with technical issues and Matt Williams for providing actual medical data for experimentation. Thanks to my examiners Paul Krause and Odinaldo Rodrigues for reading this thesis and taking part in my exam. I thank my family and friends for their moral support, special thanks to Anna without whose help at earlier stages of my studies I might had not managed to start this PhD. Finally, I would like to express my gratitude to EPSRC for funding this research.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This chapter provides an overview of the work conducted for this thesis. It starts with a brief description of the area where this work is involved. It continues with a description of the problem on which this thesis is focused, and motivation on why this is an interesting problem to address. It proceeds with a brief presentation of the proposed solution to this problem and a summary of the contribution of this work and how this is presented in the chapters that follow.

1.1 Logical argumentation

Argumentation is a vital aspect of intelligent behaviour by humans. Consider diverse professionals such as politicians, journalists, clinicians, scientists, and administrators, who all need to collate and analyse information looking for pros and cons for consequences of importance when attempting to understand problems and make decisions. It involves reasoning with uncertainty by making use of logic to formalize the presentation of arguments and counterarguments and deal with conflicting information. There are a number of proposals for logic-based formalisations of argumentation (for reviews see [15, 11, 23, 68]). These proposals allow for the representation of arguments for and against some claim, and for counterargument relationships between arguments. In a number of key examples of argumentation systems, an argument is a pair where the first item in the pair is a minimal consistent set of formulae that proves the second item which is a formula (see for example [4, 10, 12, 37, 45]). Proof procedures and algorithms have been developed for finding preferred arguments from a knowledgebase using defeasible logic and following for example Dung's preferred semantics (see for example [21, 22, 25, 27, 54, 67, 78]).

Argumentation may involve a group of agents where the exchange of arguments among the agents can be used as the tool for resolving conflicts or sharing information, or only one agent where given a knowledgebase, a monological process of listing arguments and counterarguments for a case can be used to evaluate a situation. In either case, in many formalisations argumentation requires obtaining subsets of a knowledgebase that minimally and consistently entail a claim. This process is computationally expensive, especially when using classical logic.

The aim of this work is to develop algorithms that are viable despite the computational cost. This work is based on monological argumentation using classical logic where given a background knowledge and a claim, all the possible arguments supporting this claim are generated from the knowledge.

Entailment of the claim is identified with deduction in classical logic.

1.2 Problem statement

This thesis is concerned with argumentation based on classical logic, mainly classical propositional logic [12]. In this context, an argument is a tuple $\langle \Phi, \alpha \rangle$ where Φ is a consistent set of propositional formulae that entails α and there is no $\Phi' \subset \Phi$ that entails α. In this framework, α is the claim of the argument, Φ is the support of the argument and $\langle \Phi, \alpha \rangle$ is an argument for α. An argument $\langle \Phi, \alpha \rangle$ may be challenged by another argument, a counterargument. A counterargument for an argument $\langle \Phi, \alpha \rangle$ is an argument $\langle \Psi, \beta \rangle$ where the claim β contradicts the support Φ. Like in [12], this work focuses on a particular kind of counterargument called a canonical undercut. A canonical undercut for an argument $\langle \Phi, \alpha \rangle$ is an argument $\langle \Psi, \beta \rangle$ where β is equivalent to $\neg(\phi_1 \wedge .. \wedge \phi_n)$ and $\{\phi_1, ..., \phi_n\}$ is the support of the argument being undercut. Since a canonical undercut $\langle \Psi, \beta \rangle$ is itself an argument, a canonical undercut for $\langle \Psi, \diamond \rangle$ can be produced and so on, generating this way series of counterarguments.

Given a set Δ of propositional formulae and a formula α, searching through Δ for supports for arguments for α constitutes a complex task. For each argument, we need a minimal and consistent set of formulae that proves the claim. An automated theorem prover (an ATP) may use a "goal-directed" approach, bringing in extra premises when required, but they are not guaranteed to be minimal and consistent. For example, supposing we have a knowledgebase $\{\neg a \vee b, b\}$, for proving b, the ATP may start with the premise $\neg a \vee b$, then to prove b, a second premise is required, which would be b, and so the net result is $\{\neg a \vee b, b\}$, which does not involve a minimal set of premises. In addition, an ATP is not guaranteed to use a consistent set of premises since by classical logic it is valid to prove anything from an inconsistency.

So if we seek arguments for a claim δ, we need to post queries to an ATP to ensure that a particular set of premises entails δ, that the set of premises is minimal for this, and that it is consistent. So finding arguments for a claim α involves considering subsets Φ of Δ and testing them with the ATP to ascertain whether $\Phi \vdash \alpha$ and $\Phi \nvdash \bot$ hold. For $\Phi \subseteq \Delta$, and a formula α, let $\Phi ? \alpha$ denote a call (a query) to an ATP. If Φ classically entails α, then we get the answer $\Phi \vdash \alpha$, otherwise we get the answer $\Phi \nvdash \alpha$. In this way, we do not give the whole of Δ to the ATP. Rather we call it with particular subsets of Δ. So for example, if we want to know if $\langle \Phi, \alpha \rangle$ is an argument, then we have a series of calls $\Phi ? \alpha$, $\Phi ? \bot$, $\Phi'_1 ? \alpha$,...,$\Phi'_k ? \alpha$, where for all $i = 1, \ldots, k$, $\Phi'_i \subset \Phi$. So the first call is to ensure that $\Phi \vdash \alpha$, the second call is to ensure that $\Phi \nvdash \bot$, the remaining calls are to ensure that there is no subset Φ' of Φ such that $\Phi' \vdash \alpha$. This then raises the question of which subsets Φ of Δ to investigate to determine whether $\langle \Phi, \alpha \rangle$ holds when we are seeking for an argument for α. A further problem we need to consider is that if we want to generate all arguments for a particular claim in the worst case we may have to send each subset Φ of Δ to the ATP to determine whether $\Phi \vdash \alpha$ and $\Phi \nvdash \bot$. So in the worst case, if $|\Delta| = n$, then we may need to make 2^{n+1} calls to the ATP. Even for a small knowledgebase of say 20 or 30 formulae, this can become prohibitively expensive.

The work of this thesis is focused on exploring alternative ways of finding all the arguments from a knowledgebase Δ for a claim α. A summary of the solution proposed follows in the next section.

1.3 Proposed solution

My proposal to address the problem of providing viable algorithms for computational argumentation in classical logic is based on an existing proposal for automated theorem proving where connection graphs of clauses are introduced [55, 56]. A connection graph is a graph where each node represents a clause and an arc (ϕ, ψ) denotes that there is a disjunct in ϕ with its complement being a disjunct in ψ. Given propositional knowledge Δ in the form of clauses and a claim α that is a disjunctive clause, by using connection graphs for Δ and the resolution proof rule I have developed algorithms that:

1. Reduce the search space for arguments for α by isolating a connected component of the connection graph that contains all the arguments for α.

2. Retrieve arguments by walking over the subgraph that corresponds to the reduced search space isolated in 1.

3. Generate counterarguments for a given argument $\langle \Phi, \alpha \rangle$ efficiently by extending the theory in 1 and 2.

In addition, the work in 1 and 2 has has been extended and adapted to deal with a subset of first-order logic.

More details on the solution proposed in the thesis follow in section 1.5.

1.4 Scope of this thesis

The work conducted for this thesis focuses on developing a proposal that addresses the problem of producing arguments in classical logic. It is based on an existing argumentation framework [12], so no new proposal for formalising argumentation in logic is developed in this work. The problem addressed is the difficulty in producing arguments in classical propositional logic, and the technical difficulties that arise in practical application of classical logic argumentation due to the high computational cost of the specific problem. So, the algorithms developed to deal with this problem are assessed through software implementation and experimentation that evaluate the viability of the approach and its suitability in practical applications rather than a theoretical complexity analysis related to the problem itself or the given solution.

1.5 Contribution of this work

The work presented in this thesis has contributed in the area of computational argumentation by providing a theoretical background with algorithms that addresses the computational problem of producing arguments in classical propositional logic. Moreover, an implementation of this theory provides the first software system that models argumentation in classical propositional logic. In addition, an extension of the work concerning propositional logic to classical first-order logic provides a first approach to develop algorithms for argumentation in a first-order language.

This work has been submitted in the form of papers and reviewed in conferences and journals where it has been accepted for publication [31, 32, 33, 34]. In addition, the software produced as part of this

work has been accepted for demonstration at COMMA 2010 conference [30]. Sections 1.5.1 to 1.5.5 that follow give a summary of the contribution made by this work and provide an overview of the way this is presented in chapters 3 to 7.

1.5.1 Algorithms for reducing the search space for arguments

The starting point in addressing the problem of retrieving arguments is harnessing the notion of a connection graph to reduce the search space when seeking all the arguments for a claim from a propositional knowledgebase [32]. For a set of propositional clauses, a connection graph is a graph where each node is a clause and each arc denotes that there exist complementary disjuncts in the pair of nodes. For a set of formulae in conjunctive normal form, the notion of the connection graph is used for the set of clauses obtained from the conjuncts in the formulae. When seeking arguments for a claim α from a knowledgebase Δ, we can focus the search on a particular subgraph of the connection graph of $\Delta \cup \{\neg\alpha\}$ where all the elements of $\Delta \cup \{\neg\alpha\}$ are in conjunctive normal form. This subgraph, defined as the query graph, consists of the connected components of the connection graph that are linked to the clauses that appear as conjuncts in $\neg\alpha$. Locating this subgraph is relatively inexpensive in terms of computational cost. In addition, using (as the search space) the formulae of the initial knowledgebase whose conjuncts relate to this subgraph, can substantially reduce the cost of looking for arguments. A theoretical framework and algorithms for this proposal are presented in chapter 3, along with experimental results on software implementation of the algorithms.

1.5.2 Algorithms for producing arguments

The proposal described in section 1.5.1 provides a reduced search space for arguments. It focuses the search for arguments on the subset of a propositional knowledgebase Δ that corresponds to the query graph. Apart from providing a reduced search space, the query graph gives information on how the elements of a knowledgebase relate to each other. The arcs of the graphs connect clauses that contain complementary literals and hence indicate pairs of clauses on which the resolution proof rule can be applied. This motivates for developing search algorithms which by walking over the graph in a structured way give a proof for the claim. By applying certain restrictions on the way this walk on the query graph takes place, this proof can be minimal and consistent and hence provide a support for an argument for the claim [31, 34]. In Chapter 4, I describe how it is possible to build proof trees that represent the steps of a structured walk on the query graph when looking for arguments for a claim α from a knowledgebase Δ where α is a clause and Δ is a set of clauses. In addition, I provide experimental results on software implementation of algorithms that are based on proof trees.

1.5.3 Algorithms for generating counterarguments

The proposal decribed in the previous paragraph that generates arguments by walking over the query graph can handle claims that are disjunctive clauses. In this thesis, as mentioned in section 1.2, a canonical undercut is used as a counteragument. A canonical undercut for an argument $\langle \Phi, \alpha \rangle$, where $\Phi = \{\phi_1, \ldots, \phi_n\}$ is an argument for $\neg(\phi_1 \wedge \ldots \wedge \phi_n)$. The claim $\neg(\phi_1 \wedge \ldots \wedge \phi_n)$ is not necessarily a disjunctive clause, and hence looking for arguments for this claim is not supported by the mechanism

desribed in section 1.5.2. In chapter 5, I describe how with some additional theory we can use the proof trees introduced in chapter 1.5.2 as the means for generating canonical undercuts for an argument $\langle \Phi, \alpha \rangle$. Moreover, I describe how we can use the fact that the clauses in Φ are linked with each other through resolution, and make the search for canonical undercuts efficient.

1.5.4 Implementation

Apart from developing theory and algorithms that implement argumentation in classical propositional logic efficiently, this work has contributed in the area of computational argumentation by providing the first software argumentation system based on classical logic. Chapter 6 presents system JArgue, a software system implemented in Java, based on the algorithms described in sections 1.5.1-1.5.3. This chapter also provides an evaluation of the algorithms developed by experimentation on the system.

1.5.5 Extension to first-order logic

Part of the work of this thesis deals with classical first-order logic. The work for propositional logic introduced in sections 1.5.1-1.5.2 is extended to first order logic where resolution with unification is used for retrieving arguments [33]. This proposal deals with a restricted function-free first-order language of quantified clauses in prenex normal form. Each such clause is composed of both existential and universal quantifiers, and n-ary ($n \geq 1$) predicates. Given a knowledgebase Δ of such first-order clauses and a first-order unit clause α as a claim for an argument, a connection graph is produced where the nodes are the clauses from Δ together with the complement of α, and the arcs are defined in a way similar to the propositional case. As in section 1.5.1, the query graph, is a subgraph of the connection graph that essentially contains all the arguments for α. This is isolated and used as the search space for arguments for α where a search algorithm walks over the graph and at the same time unifies complementary literals. Again this walk on the graph is done in a structured way that provides a minimal and consistent proof for α. The steps of this walk are represented by tree structures defined as assignment trees.

1.6 List of publications

The papers produced out of the work of this thesis are listed below, along with the chapters of the thesis where the related work is presented.

1. V. Efstathiou and A. Hunter. Focused search for arguments from propositional knowledge. In *Proceedings of the Second International Conference on Computational Models of Argument (COMMA'08)*. IOS Press, 2008. (Chapter 3)

2. V. Efstathiou and A. Hunter. Algorithms for effective argumentation in classical propositional logic: A connection graph approach. In *FoIKS*, pages 272–290. Springer, 2008.(Chapter 4)

3. V. Efstathiou and A. Hunter. An algorithm for generating arguments in classical predicate logic. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'09)*, pages 119–130. Springer, 2009. (Chapter 7)

4. V. Efstathiou and A. Hunter. Algorithms for generating arguments and counterarguments in

propositional logic. In *International Journal of Approximate Reasoning* (accepted for publication) (Chapters 4 and 5)

5. V. Efstathiou and A.Hunter. JArgue: An implemented argumentation system for classical propositional logic (software demo). `http://www.ing.unibs.it/comma2010/demos/Efstathiou_etal.pdf`, 2010. (Chapter 6)

6. V. Efstathiou and A. Hunter. JArgue: A java engine for argumentation in classical propositional logic. (in preparation). (Chapter 6)

1.7 Structure of the thesis

This thesis starts with a brief review on argumentation literature in chapter 2. Then follow chapters 3-7 that deal with the issues discussed in each of paragraphs 1.5.1-1.5.5. Finally, chapter 8 provides a discussion on the work presented in the thesis and related issues that could be addressed by extending this work.

Chapter 2

Background

This chapter provides an outline of proposals for modeling argumentation in a computational environment. Further literature that was reviewed for this work involves automated theorem proving and will not be mentioned in this chapter, but in later chapters where it is closely associated with the corresponding theory.

The chapter starts with a brief discussion on various existing approaches to computational argumentation and it continues with a more descriptive presentation of argumentation based on classical logic. It focuses on a specific proposal which is the basis of the work of this thesis [12] and provides a detailed review of this proposal. The chapter closes with a discussion on the conclusions of the review on argumentation systems.

2.1 Existing argumentation systems

There are a number of formalisations for argumentation. (see for reviews [8, 15, 20, 23, 68]). These vary from simple high level representation of arguments where arguments are entities for which no detailed information on the knowledge represented is provided, to more involved representations where the details of the knowledge represented and the inference mechanisms are provided together with the formalisation of an argument. Some widely used argumentation systems are listed below.

2.1.1 Abstract argumentation

Abstract argumentation is one the fundamental proposals for modeling argumentation in computational environment. Dung's proposal [26] suggests a simple, yet illustrative way for formalising the mechanism of argumentation. In this proposal, arguments are depicted as nodes in a directed graph where arcs linking pairs of nodes denote the attack relation between the nodes of the pair. Apart from the binary attack relation between arguments, no other information is given on the internal inference relations of each indivual argument and how an attack is associated to them. This provides a structural way for representing conflicting arguments. Further analysis on the structure of the interrelated attacks depicted in the graph can provide an evaluation whether some arguments are defeated.

Abstract argumentation provides the means for presenting the attack relation in a set of arguments and additional theory for justifying the acceptablility of an argument according to the interrelated attacks between the arguments in the set. It has the advantage that it can instantiate other argumentation systems

since the internal structure of the arguments is not taken into account. However, it assumes a set of arguments and an attack relation as given. Therefore, it does not provide a method for constructing individual arguments. This does not ensure that all the arguments relevant to an issue have been identified and presented. Moreover, since it does not provide the internal strucure of each argument, it does not determine why this may attack or be attacked by another argument. These issues limit the strength of the system in representing and reasoning with knowledge.

2.1.2 Argumentation based on defeasible logic

A formalisation that can provide representation of more detailed information and is widely used is that of defeasible logic programming (i.e. DeLP) [45]. In DeLP two kinds of rules are considered: *defeasible rules* that represent weak information and *strict rules* that represent sound information. The underlying meaning of the notion of a defeasible rule is that it represents tentative information that can be used until nothing is posed against it. According to Nute [62], an inference is defeasible if it can be blocked or defeated in some way. In DeLP an argument is a tuple $\langle \mathscr{A}, h \rangle$ where $\mathscr{A}$ is a set that consists of defeasible rules which together with the set of strict rules available is consistent, h is derived by $\mathscr{A}$, and $\mathscr{A}$ is minimal for this derivation. Then, conflict between two arguments arises when these together contain information which is contradictory with the set of all strict rules.

Defeasible reasoning captures the nature of human reasoning where people tend to make up defeasible rules out of observations and use them to reach conclusions which may be then withdrawn in the presence of additional information that contradicts their defeasible rules. In this sense, defeasible logic programming provides a good tool for formalising human reasoning. Moreover, since it provides the details of the information represented, the deductive mechanism and the exact information that makes two arguments contradictory, defeasible logic programming provides a system for modeling argumentation that offers much greater expressivity than abstract argumentation systems. However, it lacks the advantages of classical logic for representing and reasoning with knowledge including syntax, proof theory and semantics for the intuitive language incorporating negation, conjunction, disjunction and implication.

Apart from DeLP discussed in this paragraph, work on argumentation that is based on defeasible logics includes [53, 67].

2.1.3 Assumption based argumentation

Another approach to formalising argumentation is that of *assumption-based argumentation* [18, 27]. Assumption-based argumentation provides an instantiation of abstract argumentation by providing a way to generate arguments from assumptions and rules. An assumption-based argumentation (ABA) system is a tuple $\langle L, R, A, \overline{\bullet} \rangle$ such that R is a set of rules of the form $s_1 \leftarrow s_2, ..., s_n$ where each s_i is a sentence, $A \subseteq L$ is a set of candidate assumptions, and each assumption cannot be the head of any rule and $\overline{a}$ is the contrary of assumption a. In this system, an argument is a backward chaining deduction supported by a set of assumptions. An argument A attacks another argument A' if the conclusion of A is the contrary of one of the assumptions supporting A'. Assumption-based argumentation adapts notions from abstract argumentation in order to evaluate arguments with respect to the attacks that appear in a set of arguments. In addition, assumption-based argumentation supports constructing dispute trees where

each node is an argument that attacks its parent in the tree, in order to determine if the root argument is acceptable. This provides a way for evaluating arguments without constructing a complete argument graph.

Assumption-based argumentation provides information on the internal structure of arguments, giving a more detailed presentation of the knowledge represented than that of abstract argumentation. The power of the underlying logic though is similar to that of defeasible logic programming and hence has the weaknesses discussed in section 2.1.2. The language it can handle and the proof theory it involves are much simpler and restricted than that of classical logic.

2.2 Argumentation based on classical logic

This section provides a brief review of formalisations of argumentation based on classical logic. It starts with Pollock's work and then presents some formalisations inspired by this approach. One of these formalisations [12] is the basis of this thesis and is reviewed in detail later in this section.

2.2.1 Pollock's proposal

In [64, 65] Pollock emphasizes the significance of inductive reasoning and how it should be regarded as equally important to deductive reasoning in philosophy and AI. Human reasoning is usually inductive, since people tend to come to conclusions by judging situations by their perception and experience rather than listing a set of reasons from which their conclusions strictly follow by deduction. If additional information that may change their opinion on a matter is presented, then they may revise their knowledge and change their mind regarding a previously justified conclusion. This reasoning has a non-monotonic character and it is important to take this into account when trying to formalise reasoning.

Defeasible reasoning is a form of non-monotonic reasoning where some reasons may justify accepting a conclusion but when additional information is added, that conclusion may no longer be justified. Pollock characterizes defeasible reasoning using arguments where the underlying logic is classical logic and arguments are chains of reasons that may lead to a conclusion where defeaters may exists at each step. This approach depicts defeasible reasoning, since the presence of additional information may destroy the reason connection.

Pollock's work provides the starting point for many logic-based argumentation proposals where an argument is regarded as a tuple $\langle \Phi, \alpha \rangle$ where Φ is a set of premises that represents the reasons and α is a formula that represents a conclusion. This definition is very general and does not necessarily require that Φ logically entails α or that Φ is a consistent set. In the system presented in [64] two kinds of reasons are introduced, **defeasible** and **nondefeasible**. Nondefeasible reasons are the ones that logically entail their conclusions. Defeasible reasons are reasons that justify their conlusion but together with another reason may destroy the reason connection. Information that can mandate the retraction of the conclusion of a defeasible argument constitutes a defeater for the argument. There are two kinds of defeaters. The **rebutting defeaters**, which attack an argument by attacking its conclusion, and the **undercutting defeaters**, which attack the defeasible inference itself, without doing so by giving us a reason for thinking it has a false conclusion. A simplified version of the formal definitions adapted from

[15] follows.

Definition 2.2.1 *An argument* $\langle \Psi, \beta \rangle$ **rebuts** *an argument* $\langle \Phi, \alpha \rangle$ *iff* $\vdash \beta \leftrightarrow \neg\alpha$.

An argument $\langle \Psi, \beta \rangle$ **undercuts** *an argument* $\langle \Phi, \alpha \rangle$ *iff* $\vdash \beta \leftrightarrow \neg \wedge \Phi$.

An argument A_1 **attacks** *an argument* A_2 *iff* A_1 *rebuts* A_2 *or* A_1 *undercuts* A_2.

The definition of attack indicates that if an argument A_2 is attacked by another argument A_1, then there are reasons to believe that A_1 is not justified and A_1 can be regarded as defeated. Using the definition of attack on its own in order to evaluate the credibility of an argument is not sufficient though. If A_2 is an argument for which exists an argument A_1 such that A_1 attacks A_2, then this does not necessarily mean that A_2 is defeated, since another argument A_3 may exist such that A_3 attacks A_2, reinstating in this way A_2 and so on. In order to decide the warranted propositions (i.e. arguments conclusions) that are justified by a set of arguments we need a mechanism which by recursion analyses the presentation of sets of arguments where each argument attacks another in the set. In [64] such a mechanism is introduced which proceeds in 'levels', where given a set of arguments $\mathscr{A}$,

- at level 0 no argument is considered as provisionally defeated
- at level 1 those arguments that are attacked by some other argument in the set are regarded as provisionally defeated
- at level 2 those arguments that are attacked by arguments that have been provisionally defeated at level 1 are 'reinstated'

and so on. Then, for an argument $\langle \Phi, \alpha \rangle \in \mathscr{A}$, the proposition α is **warranted** iff $\langle \Phi, \alpha \rangle$ is undefeated in $\mathscr{A}$.

Pollock's approach is important in logic-based argumentation because it introduced the notion of an argument in the format $\langle \Phi, \alpha \rangle$ where Φ is a set of premises and α is a proposition. This format has been adapted by other proposals based on classical or defeasible logics, usually with the additional constraint that Φ entails α, and/or Φ is a consistent set and is minimal for entailing α. Moreover, the ideas for rebut and undercut have been adapted by other proposals as well as the recursive characterisation of warrant.

A formalism of logic-based argumentation [4] that has adapted ideas from Pollock's approach is briefly reviewed below.

2.2.2 Amgoud and Cayrol's proposal

An argumentation framework that has adapted the ideas of argument, rebut and undercut as presented in the previous section is [4]. This framework adapts the form of an argument $\langle \Phi, \alpha \rangle$ where Φ is a set of reasons and α is a conclusion, with the restrictions that $\Phi \vdash \alpha$ i.e. $\langle \Phi, \alpha \rangle$ is a *deductive argument*, Φ is consistent and Φ is minimal for entailing α. In this system, the definition for a rebut remains the same with the one given in definition 2.2.1 while the definition for an undercut is as follows.

Definition 2.2.2 $\langle \Psi, \beta \rangle$ **undercuts** $\langle \Phi, \alpha \rangle$ *iff there is some* $\gamma \in \Phi$ *such that* $\vdash \gamma \leftrightarrow \neg\beta$

So, in this system, an undercut $\langle \Psi, \beta \rangle$ for an argument $\langle \Phi, \alpha \rangle$ negates a particular element of Φ. For instance, for the argument $A_1 = \langle \{\neg a, a \vee b\}, b \rangle$, according to definition 2.2.1, $\langle \Psi_1, \beta_1 \rangle = \langle \{a \vee$

$\neg b\}, a \vee \neg b\rangle$ is an undercut while $\langle \Psi_2, \beta_2\rangle = \langle\{a\}, a\rangle$ is not. With definition 2.2.2, $\langle \Psi_2, \beta_2\rangle = \langle\{a\}, a\rangle$ is an undercut for A_1 while $\langle \Psi_1, \beta_1\rangle = \langle\{a \vee \neg b\}, a \vee \neg b\rangle$ is not. With definition 2.2.2, an undercut for an argument $\langle \Phi, \alpha\rangle$ focuses on a particular premise from Φ, restricting this way the number of arguments that satisfy the definition for an undercut for $\langle \Phi, \alpha\rangle$. This may lead to omitting arguments that could intuitively represent a counterargument for $\langle \Phi, \alpha\rangle$. With definition 2.2.1 on the other hand, the conclusion of an undercut is less specific, yet sufficient for formalising how the reason connection in $\langle \Phi, \alpha\rangle$ is attacked by the undercut. In addition, Pollock's proposal provides a more compact definition for an undercut where information already stated in Φ may avoid being repeated in the reasons of an undercut for $\langle \Phi, \alpha\rangle$. Consider argument $A_2 = \langle\{a, b, c\}, a \wedge b \wedge c\rangle$ for instance. With definition 2.2.2, $\langle\{a, b, \neg a \vee \neg b \vee \neg c\}, \neg c\rangle$ is an undercut for A_2 and the set of reasons $\{a, b\}$ which is included in the premises of the original argument is also included in the premises of the undercut. With definition 2.2.1 though this does not happen. $\{\neg a \vee \neg b \vee \neg c\}$ is sufficient to be the set of premises for an undercut for A_2 and so $\langle\{\neg a \vee \neg b \vee \neg c\}, \neg(a \wedge b \wedge c)\rangle$ is an undercut for A_2.

Apart from the definition for undercut, the warrant criteria differ in this framework from that of Pollock's framework on that they are based on acceptability. The attack relation between arguments depends on an ordering over the set of arguments where these arguments belong, and this ordering is based on preference relations. Given a set of arguments $\mathscr{A}$, where $Pref$ is a preordering on $\mathscr{A} \times \mathscr{A}$, let $>^{Pref}$ denote the strict ordering associated with $Pref$. Then, if $\mathscr{R}$ is a binary relation between arguments, then for $A, B \in \mathscr{A}$, B attacks A iff $B\mathscr{R}A$ and $A \not>^{Pref} B$. For an argument $A \in \mathscr{A}$, it holds that A *defends itself* iff for all B, if $B\mathscr{R}A$, then $A >^{Pref} B$. Moreover, a set of arguments $S \subseteq \mathscr{A}$ *defends* A iff for all $B \in \mathscr{A}$, if $B\mathscr{R}A$ and $A \not>^{Pref} B$, then there exists $C \in S$ such that $C\mathscr{R}B$ and $B \not>^{Pref} C$. The acceptable arguments are the ones that defend themselves against their defeaters and also the arguments that are defended by the arguments that defend themeselves. Given a set of arguments $\mathscr{A}$ where for pairs of arguments A, B the relation $B\mathscr{R}A$ may hold and a preference ordering over this set, it is possible to decide whether an argument $A \in \mathscr{A}$ is acceptable by drawing a *dialogue tree*. This is a tree where where A is the root and each node is an argument. The arcs are defined by the attack relation, each argument node attacks its parent node in the tree. By marking in this tree as *undefeated* all the leaf nodes and then by recursion as *defeated* all the nodes that have at least one child which is marked as undeafeated, the root node is acceptable iff by this process it is marked as undefeated. This mechanism for deciding whether an argument is acceptable is analogous to Pollock's warrant citerion for deciding whether the conclusion of an argument is warranted.

2.2.3 Besnard and Hunter's proposal

The formalisation for argumentation reviewed in this section [12], which is also based on classical logic, is the basis for this thesis. For this reason, a detailed review of this proposal is provided rather than a brief discussion. This proposal adapts the ideas introduced in Pollock's proposal with the restriction of employing only deductive arguments with consistent sets of reasons. An important feature of this framework is that it introduces a special kind of undercut, the *canonical undercut* which subsumes many other kinds of undercut, providing an efficient way for presenting counterarguments. Most of the examples in

this section are adapted from [11] and consist of propositional formulae. Towards the end of this section, examples based on classical first-order logic are also introduced.

2.2.3.1 Arguments

In the definition for an argument given in this section, the knowledgebases and claims are formulae in classical logic and the method of inference by which the claim follows from a set of formulae is deductive inference and is denoted $\vdash$. Then, an argument in classical logic is defined as follows.

Definition 2.2.3 *An* **argument** *is a pair* $\langle \Phi, \alpha \rangle$ *such that:*

(1) $\Phi \nvdash \bot$*;*

(2) $\Phi \vdash \alpha$*;*

(3) there is no $\Phi' \subset \Phi$ *such that* $\Phi' \vdash \alpha$*.*

$\langle \Phi, \alpha \rangle$ *is said to be an argument for* α*, where* α *is called the* **claim** *of the argument and* Φ *the* **support** *of the argument (it is also said that* Φ *is a support for* α*).*

Example 2.2.1 *For a knowledgebase* $\Delta = \{a, b, c, a \rightarrow \neg b \vee c, a \rightarrow \neg b, d, \neg d, \neg c, d \rightarrow a, d \rightarrow e\}$ *some arguments include:*

$$\langle \{a \rightarrow \neg b\}, \neg(a \wedge b) \rangle \qquad \langle \{a, a \rightarrow \neg b\}, \neg b \vee \neg c \rangle \qquad \langle \{a, a \rightarrow \neg b\}, \neg b \rangle$$
$$\langle \{\neg c\}, \neg c \vee b \rangle \qquad \langle \{\neg c\}, \neg c \rangle \qquad \langle \{a, b, a \rightarrow \neg b \vee c\}, c \rangle$$

Because in classical logic from a fallacy anything can be inferred, condition (1) of the definition for an argument ensures that the reasoning of the argument is coherent and the claim does not follow from the support because of an inconsistency. This is illustrated in the following example adapted from [11].

Example 2.2.2 *Consider the following atoms.*

a *The office phone number is 020 4545 8721*

b *I am a billionaire*

Now let $\{a, \neg a\} \subseteq \Delta$*, and so by classical logic, we have* $\{a, \neg a\} \vdash b$*. However, we do not want to have* $\{a, \neg a\}$ *as the support for an argument with claim* b*. If we were to allow that as an argument, then we would have an argument with this support and with any claim in the language. Hence, if we were to allow inconsistent supports, then we would have an overwhelming number of useless arguments.*

Real arguments, that is arguments presented by humans, are usually enthymemes [79]. Enthymemes do not explicitly state all the premises that are necessary to reach a claim. This often happens because some premises are considered to be obvious and so are not explicitly assumed. More details about the use of enthymemes can be found in [51]. The use of enthymemes is illustrated in the following example from [11].

Example 2.2.3 *Consider the informal argument which is acceptable.*

'It is an even number, and therefore we can infer it is not an odd number'

Now consider the following atoms

e	*It is an even number*
o	*It is an odd number*

So we can represent the premise of the informal argument by the set $\{e\}$*. However, by classical logic we have that* $\{e\} \nvdash \neg o$*, and hence the following is not an argument* $\langle\{e\}, \neg o\rangle$*. If we want to turn the informal argument (which is an enthymeme) into an argument, we need to make explicit all the premises. So we can represent the above informal argument by the following formal argument* $\langle\{e, e \rightarrow \neg o\}, \neg o\rangle$*.*

So, when formalising argumentation, all the premises of the reasoning have to be included explicitly in the support so that it is sufficient for the consequent to hold. Condition (2) of definition 2.2.3 ensures this requirement is met.

Condition (3) of definition 2.2.3 ensures the minimality of the support for proving the claim. The underlying idea for condition (3) is that an argument makes explicit the connection between reasons for a claim and the claim itself. The following example taken from [11] shows the effect of this condition.

Example 2.2.4 *Consider the following formulae.*

p	*I like paprika*
r	*It is raining*
$r \rightarrow q$	*If it is raining, then I should use my umbrella*

It is possible to argue that "I should use my umbrella, because I should use my umbrella, if it is raining, and indeed it is", to be captured formally by the argument

$$\langle\{r, r \rightarrow q\}, q\rangle$$

In contrast, it is counter-intuitive to argue that "I should use my umbrella, because I like paprika and I should use my umbrella, if it is raining, and indeed it is", to be captured formally by

$$\langle\{p, r, r \rightarrow q\}, q\rangle$$

which fails to be an argument because condition 3 is not satisfied.

Examples 2.2.3 and 2.2.4 demonstrate how conditions (2) and (3) of the definition for an argument ensure that the support for an argument contains exactly the formulae that are necessary to entail the claim of the argument.

2.2.3.2 Comparing arguments

Given two arguments, it is possible to compare them in terms of how more general is one from the other. The following definition captures this relation between arguments.

Definition 2.2.4 *An argument* $\langle\Phi, \alpha\rangle$ *is* **more conservative** *than an argument* $\langle\Psi, \beta\rangle$ *iff* $\Phi \subseteq \Psi$ *and* $\beta \vdash \alpha$*.*

Example 2.2.5 *Continuing example 2.2.1,* $\langle\{a \rightarrow \neg b\}, \neg(a \wedge b)\rangle$*, is a more conservative argument than* $\langle\{a, a \rightarrow \neg b\}, \neg b\rangle$ *and* $\langle\{a, a \rightarrow \neg b\}, \neg b \vee \neg c\rangle$ *is a more conservative argument than* $\langle\{a, a \rightarrow \neg b\}, \neg b\rangle$*. Also,* $\langle\{\neg c\}, \neg c \vee b\rangle$ *is a more conservative argument than* $\langle\{\neg c\}, \neg c\rangle$*.*

A more conservative argument is more general: it is less demanding on the support and less specific about the consequent. This is demonstrated in the following example.

Example 2.2.6 *Consider the following knowledgebase* $\Delta = \{p, p \rightarrow q, q \rightarrow r\}$*. Then, the following is an argument with the claim q.*

$$\langle\{p, p \rightarrow q\}, q\rangle$$

Similarly, the following is argument with claim $r \wedge q$*.*

$$\langle\{p, p \rightarrow q, q \rightarrow r\}, r \wedge q\rangle$$

However, the first argument $\langle\{p, p \rightarrow q\}, q\rangle$ *is more conservative than the second argument* $\langle\{p, p \rightarrow q, q \rightarrow r\}, r \wedge q\rangle$ *which can be retrieved from it:*

$$\left.\begin{array}{l} \langle\{p, p \rightarrow q\}, q\rangle \\ \{q, q \rightarrow r\} \vdash r \wedge q \end{array}\right\} \quad \Rightarrow \quad \langle\{p, p \rightarrow q, q \rightarrow r\}, r \wedge q\rangle$$

The notion of "more conservative" argument is useful in identifying the most useful counterarguments amongst the potentially large number of counterarguments.

2.2.3.3 Counterarguments

In argumentation literature a counterargument for an argument A is described as an argument A' that disagrees with A. Specifically in logic-based approaches to argumentation, the notion of a counterargument is usually associated to that of defeater, an argument whose claim refutes the support of another argument [62, 72, 53, 77, 68]. This gives a general way for an argument to challenge another.

Definition 2.2.5 *A* **defeater** *for an argument* $\langle\Phi, \alpha\rangle$ *is an argument* $\langle\Psi, \beta\rangle$ *such that* $\beta \vdash \neg(\phi_1 \wedge \ldots \wedge \phi_n)$ *for some* $\{\phi_1, \ldots, \phi_n\} \subseteq \Phi$*.*

Example 2.2.7 *Let* $\Delta = \{\neg a, a \vee b, a \leftrightarrow b, c \rightarrow a\}$*. Then,* $\langle\{a \vee b, a \leftrightarrow b\}, a \wedge b\rangle$ *is a defeater for* $\langle\{\neg a, c \rightarrow a\}, \neg c\rangle$*. A more conservative defeater for* $\langle\{\neg a, c \rightarrow a\}, \neg c\rangle$ *is* $\langle\{a \vee b, a \leftrightarrow b\}, a \vee c\rangle$*.*

A less general case of a defeater is that of an undercut defined below.

Definition 2.2.6 *Let* $A = \langle\Phi, \alpha\rangle$ *be an argument. An* **undercut** *for A is an argument* $\langle\Psi, \neg(\phi_1 \wedge \ldots \phi_n)\rangle$ *where* $\{\phi_1, \ldots, \phi_n\} \subseteq \Phi$*.*

Example 2.2.8 *For the argument* $\langle\{a, b, a \rightarrow \neg b \vee c\}, c\rangle$ *of example 2.2.1 it holds that it has* $\langle\{a \rightarrow \neg b\}, \neg(a \wedge b)\rangle$*, and* $\langle\{a, a \rightarrow \neg b\}, \neg b\rangle$ *as undercuts.*

Another case of conflict between arguments which has already been mentioned is when two arguments have opposite claims.

Definition 2.2.7 *An argument* $\langle\Psi, \beta\rangle$ *is a* **rebuttal** *for an argument* $\langle\Phi, \alpha\rangle$ *iff* $\beta \leftrightarrow \neg\alpha$ *is a tautology.*

Example 2.2.9 *$\langle\{\neg b\}, \neg b\rangle$ is a rebuttal for $\langle\{a, a \rightarrow b\}, b\rangle$.*

By the definition of an undercut it follows that undercuts are defeaters but it can also be shown that rebuttals are defeaters. Although both undercuts and rebuttals are defeaters, an undercut for an argument need not be a rebuttal for that argument, and a rebuttal for an argument need not be an undercut for that argument. It can be the case where an undercut may even agree with the claim of the argument it objects to as the following examples illustrates.

Example 2.2.10 *Let $A_1 = \langle\{a, a \rightarrow b\}, b\rangle$ and $A_2 = \langle\{b \wedge \neg a\}, \neg a\rangle$. Then, A_2 is an undercut for A_1. A_2 not only is not a rebuttal for A_1, but the support of A_2 is a support for an argument for the claim of A_1: $\langle\{b \wedge \neg a\}, b\rangle$ is an argument.*

So, an undercut need not question the claim of an argument but only the reasons given by that argument to support its claim. Moreover, a rebuttal for an argument may not be an undercut for that argument.

Example 2.2.11 *$\langle\{\neg b\}, \neg b\rangle$ is a rebuttal for $\langle\{a, a \rightarrow b\}, b\rangle$ but is not an undercut for it because b is not in $\{a, a \rightarrow b\}$.*

Rebuttals and undercuts provide different ways to challenge an argument. However, there is a special kind of undercut that can effectively capture all the information needed in order to challenge an argument. This is the canonical undercut defined in the next section.

2.2.3.4 Canonical undercuts

Undercuts for a specific argument can be compared on which is more conservative than another. So for instance, for the argument $\langle\{a, b, a \rightarrow \neg b \vee c\}, c\rangle$ of example 2.2.8 and its undercuts we can say that $\langle\{a \rightarrow \neg b\}, \neg(a \wedge b)\rangle$, is a more conservative undercut than $\langle\{a, a \rightarrow \neg b\}, \neg b\rangle$. Given a set of undercuts for an argument, the notion of the most conservative undercut is captured in the definition of a maximally conservative undercut defined below.

Definition 2.2.8 *$\langle\Psi, \beta\rangle$ is a* **maximally conservative undercut** *for $\langle\Phi, \alpha\rangle$ iff for all undercuts $\langle\Psi', \beta'\rangle$ of $\langle\Phi, \alpha\rangle$, if $\Psi' \subseteq \Psi$ and $\beta \vdash \beta'$ then $\Psi \subseteq \Psi'$ and $\beta' \vdash \beta$.*

Example 2.2.12 *Continuing example 2.2.8, $\langle\{a \rightarrow \neg b\}, \neg(a \wedge b)\rangle$ is a maximally conservative undercut for $\langle\{a, b, a \rightarrow \neg b \vee c\}, c\rangle$.*

The next example from [11] shows that a collection of counterarguments for the same argument can sometimes be summarized in the form of a single maximally conservative undercut of the argument, thereby avoiding some amount of redundancy among counterarguments.

Example 2.2.13 *Consider the following formulae concerning who is going to a party.*

$r \rightarrow \neg p \wedge \neg q$	*If Rachel goes, neither Paul nor Quincy go*
p	*Paul goes*
q	*Quincy goes*

Hence both Paul and Quincy go (initial argument)

$\langle\{p, q\}, p \wedge q\rangle$

Now assume the following additional piece of information

r *Rachel goes Hence Paul does not go (a first counterargument)*

$\langle\{r, r \rightarrow \neg p \wedge \neg q\}, \neg p\rangle$

Hence Quincy does not go (a second counterargument)

$\langle r, r \rightarrow \neg p \wedge \neg q\}, \neg q\rangle$

A maximally conservative undercut (for the initial argument) that subsumes both counterarguments above is

$\langle\{r, r \rightarrow \neg p \wedge \neg q\}, \neg(p \wedge q)\rangle$

The next example highlights the advantage of maximally consevative undercuts when considering counterarguments for an argument.

Example 2.2.14 *Consider the following knowledgebase* $\{a, b, c, \neg a \vee \neg b \vee \neg c\}$. *Suppose we start with the following argument* $\langle\{a, b, c\}, a \wedge b \wedge c\rangle$. *There are numerous undercuts to this argument including the following.*

$$\langle\{b, c, \neg a \vee \neg b \vee \neg c\}, \neg a\rangle$$
$$\langle\{a, c, \neg a \vee \neg b \vee \neg c\}, \neg b\rangle$$
$$\langle\{a, b, \neg a \vee \neg b \vee \neg c\}, \neg c\rangle$$
$$\langle\{a, \neg a \vee \neg b \vee \neg c\}, \neg b \vee \neg c\rangle$$
$$\langle\{b, \neg a \vee \neg b \vee \neg c\}, \neg a \vee \neg c\rangle$$
$$\langle\{c, \neg a \vee \neg b \vee \neg c\}, \neg a \vee \neg b\rangle$$
$$\langle\{\neg a \vee \neg b \vee \neg c\}, \neg a \vee \neg b \vee \neg c\rangle$$

All these undercuts say the same thing which is that the set $\{a, b, c\}$ *is inconsistent together with the formula* $\neg a \vee \neg b \vee \neg c$. *As a result, this can be captured by the last undercut listed above which is the maximally conservative undercut amongst the undercuts listed.*

The claim of a maximally conservative undercut for an argument is exactly the negation of the full support of the argument. In other words, if $\langle\Psi, \neg(\phi_1 \wedge \ldots \wedge \phi_n)\rangle$ is a maximally conservative undercut for an argument $\langle\Phi, \alpha\rangle$, then $\Phi = \{\phi_1, \ldots, \phi_n\}$. Then, if $\langle\Psi, \neg(\phi_1 \wedge \ldots \wedge \phi_n)\rangle$ is a maximally conservative undercut for an argument $\langle\Phi, \alpha\rangle$, so are $\langle\Psi, \neg(\phi_2 \wedge \ldots \wedge \phi_n \wedge \phi_1)\rangle$ and $\langle\Psi, \neg(\phi_3 \wedge \ldots \wedge \phi_n \wedge \phi_1 \wedge \phi_2)\rangle$ and so on. However, they are all equivalent. We can ignore the unnecessary variants by just considering the canonical undercuts defined as follows.

Definition 2.2.9 $\langle\Psi, \neg(\phi_1 \wedge \ldots \phi_n)\rangle$ *is a* **canonical undercut** *for* $\langle\Phi, \alpha\rangle$ *iff it is a maximally conservative undercut for* $\langle\Phi, \alpha\rangle$ *and* $\{\phi_1, \ldots, \phi_n\}$ *is the canonical enumeration of* Φ.

Example 2.2.15 *Returning to Example 2.2.13, suppose the canonical enumeration of the support of initial argument* $\langle\{p, q\}, p \wedge q\rangle$ *is:* p, q. *Then both the following are maximally conservative undercuts,*

but only the first is a canonical undercut.

$$\langle\{r, r \rightarrow \neg p \wedge \neg q\}, \neg(p \wedge q)\rangle$$
$$\langle\{r, r \rightarrow \neg p \wedge \neg q\}, \neg(q \wedge p)\rangle$$

Clearly, an argument may have more than one canonical undercut. Any two different canonical undercuts for the same argument have the same claim, but distinct supports, and none is more conservative than the other.

Example 2.2.16 *Let* $\Delta = \{a, b, \neg a, \neg b\}$. *Both the following are canonical undercuts for* $\langle\{a, b\}, a \leftrightarrow b\rangle$, *but neither is more conservative than the other.*

$$\langle\{\neg a\}, \neg(a \wedge b)\rangle$$
$$\langle\{\neg b\}, \neg(a \wedge b)\rangle$$

For simplicity, since all the canonical undercuts for an argument have the same claim, the notation $\langle\Psi, \diamond\rangle$ will be used to denote a canonical undercut for $\langle\Phi, \alpha\rangle$.

The next section describes how the exchange of arguments and counterarguments (where a counterargument for an argument is considered to be a canonical undercut for that argument) can be structured in trees.

2.2.3.5 Argument trees

Usually the argumentation process takes place by putting an argument forward in order to support a claim of interest. If there is conflicting information that can be used against this argument, this is posed in the form of a counteragument and in the same way there can be a counter-counterargument that can be put against this counterargument and so on. For each argument there can be a number of counterarguments and for each of these counterarguments there can be a number of counter-counterarguments and so on, and depending on which of these is presented against an argument, the argumentation process can unfold in different ways. So, an issue to address when evaluating information about a case, is that we need to take into account all the different possibilities for the course of argumentation from the given background.

Example 2.2.17 *Let* $\Delta = \{a \wedge b, \neg a \wedge b, \neg b\}$. *Then,*

$A_1 = \langle\{a \wedge b\}, a\rangle$ *is an argument.*

Moreover

$A_2 = \langle\{\neg a \wedge b\}, \diamond\rangle$ *is a counterargument for* A_1
$A_3 = \langle\{\neg b\}, \diamond\rangle$ *is a counterargument for* A_2.

In addition,

$A'_2 = \langle\{\neg b\}, \diamond\rangle$ *is a counteragument for* A_1 *and*

$A'_3 = \langle\{\neg a \wedge b\}, \diamond\rangle$ *is a counterargument for* A'_2.

For the above arguments it holds that A_2 *has equal support with* A'_3 *and* A_3 *has equal support with* A'_2. *We can see though that in the first case* A_2 *can be put against* A_1 *and* A_3 *can be put against* A_2 *so in a sense* A_3 *is used in favour of the initial argument while in the second case* A'_2 *is used against the initial argument and* A'_3 *which is used against* A'_2 *is acting in favour of the initial argument.*

The last example highlights the need to express all the possible courses argumentation can take. Another point to address in argumentation, is that an argument can continue forever by recycling already stated information unless a stopping condition is introduced.

Example 2.2.18 *Continuing example 2.2.17, the following infinite exchange of arguments and counterarguments may be produced.*

$$\begin{array}{c}
\langle\{a \wedge b\}, a\rangle \\
\uparrow \\
\langle\{\neg a \wedge b\}, \diamond\rangle \\
\uparrow \\
\langle\{\neg b\}, \diamond\rangle \\
\uparrow \\
\langle\{\neg a \wedge b\}, \diamond\rangle \\
\uparrow \\
\langle\{\neg b\}, \diamond\rangle \\
\uparrow \\
\langle\{\neg a \wedge b\}, \diamond\rangle \\
\vdots
\end{array}$$

A structure that takes the issues discussed above into account and provides an efficient way for representing the exchange of arguments and counterarguments is the argument tree defined next.

Definition 2.2.10 *An* **argument tree** *for* α *is a tree where the nodes are arguments such that*

(1) *The root is an argument for* α.

(2) *For no node* $\langle\Phi, \beta\rangle$ *with ancestor nodes* $\langle\Phi_1, \beta_1\rangle \ldots \langle\Phi_n, \beta_n\rangle$ *is* Φ *a subset of* $\Phi_1 \cup \ldots \cup \Phi_n$.

(3) *The children nodes of a node* N *consist of all canonical undercuts for* N *that obey* (2).

Example 2.2.19 *Let* Δ *be the knowledgebase from example 2.2.1,* $\Delta = \{a, b, c, a \rightarrow \neg b \vee c, a \rightarrow \neg b, d, \neg d, \neg c, d \rightarrow a, d \rightarrow e\}$. *Then, the following is an argument tree for* $\alpha = c$.

$$\begin{array}{ccc}
 & \langle\{a, b, a \to \neg b \vee c\}, c\rangle & \\
 & / \qquad\qquad \backslash & \\
\langle\{a \to \neg b\}, \diamond\rangle & & \langle\{\neg c\}, \diamond\rangle \\
| & & | \\
\langle\{d, d \to a, b\}, \diamond\rangle & & \langle\{c\}, \diamond\rangle \\
/ \quad \backslash & & \\
\langle\{a \to \neg b \vee c, \neg c\}, \diamond\rangle \quad \langle\{\neg d\}, \diamond\rangle & &
\end{array}$$

The argument tree is a compact way of presenting all the possible sequences of conflicting arguments from a knowledgebase initialized by the root node. Condition (2) of definition 2.2.10 ensures that repeating the same piece of information in different nodes in a branch will be avoided and so the argument tree will not contain duplication of information in a sequence of arguments and counterarguments. Moreover, each such sequence will always be finite. Infinite repetition of a set of formulae in the support sets of the nodes of a branch cannot exist and considering finite knowledgebases this condition can only allow for finite branches to exist.

Example 2.2.20 *Let* $\Delta = \{a, a \to b, c \to \neg a, c\}$.

$$\begin{array}{c}
\langle\{a, a \to b\}, b\rangle \\
| \\
\langle\{c, c \to \neg a\}, \diamond\rangle \\
| \\
\langle\{a, c \to \neg a\}, \diamond\rangle
\end{array}$$

This is not an argument tree because Condition (2) *of the definition of the argument tree is not met. The undercut to the undercut is actually making exactly the same point (that* a *and* c *are incompatible) as the undercut itself does, just by using modus tollens instead of modus ponens.*

Condition (3) ensures that each node in the tree is a canonical undercut for its parent node avoiding this way situations like the one in the next example.

Example 2.2.21 *Given* $\Delta = \{a, b, a \to c, b \to d, \neg a \vee \neg b\}$*, consider the following tree.*

$$\begin{array}{ccc}
 & \langle\{a, b, a \to c, b \to d\}, c \wedge d\rangle & \\
 & / \qquad\qquad \backslash & \\
\langle\{a, \neg a \vee \neg b\}, \neg b\rangle & & \langle\{b, \neg a \vee \neg b\}, \neg a\rangle
\end{array}$$

This is not an argument tree because the two children nodes of the root are not maximally conservative undercuts. The first undercut is essentially the same argument as the second undercut in a rearranged form (relying on a *and* b *being incompatible, assume one and then conclude that the other doesn't hold). If we replace these by the maximally conservative undercut* $\langle\{\neg a \vee \neg b\}, \diamond\rangle$*, we obtain the following argument tree.*

$$\langle\{a, b, a \rightarrow c, b \rightarrow d\}, c \wedge d\rangle$$
$$|$$
$$\langle\{\neg a \vee \neg b\}, \diamond\rangle$$

So, an argument tree that has $\langle\Phi, \alpha\rangle$ as the root provides an exhaustive presentation of arguments and counterarguments that can be produced with $\langle\Phi, \alpha\rangle$ as the initial argument. Then, the argument tree can be evaluated and the results can be analysed to judge whether the initial argument is warranted. The mechanism for determining whether the argument at the root of the argument tree is warranted is adapted from [45]. For this, each node of the argument tree is marked as either U for **undefeated** or D for **defeated**. All the leaf nodes are marked undefeated and then all the nodes that have at least one undefeated child are marked as defeated. By using the value assigned to the root node of the argument tree by this process, the Judge function assigns either Warranted or Unwarranted to the tree.

Definition 2.2.11 *The* **judge function**, *denoted* Judge, *assigns either* Warranted *or* Unwarranted *to each argument tree T such that* Judge(T) = Warranted *iff* Mark(A_r) = U *where A_r is the root node of T. For all nodes A_i in T, if there is child A_j of A_i such that* Mark$(A_j) = U$, *then* Mark$(A_i) = D$, *otherwise* Mark$(A_i) = U$.

As a direct consequence of the above definition, the root is undefeated iff all its children are defeated.

Example 2.2.22 *For the argument tree of example 2.2.19, the nodes are marked as defeated or undefeated as follows:*

$$\langle\{a, b, a \rightarrow \neg b \vee c\}, c\rangle\ \mathbf{D}$$
$$/ \qquad \backslash$$
$$\langle\{a \rightarrow \neg b\}, \diamond\rangle\ \mathbf{U} \qquad \langle\{\neg c\}, \diamond\rangle \mathbf{D}$$
$$| \qquad\qquad |$$
$$\langle\{d, d \rightarrow a, b\}, \diamond\rangle\ \mathbf{D} \qquad \langle\{c\}, \diamond\rangle\ \mathbf{U}$$
$$/ \quad \backslash$$
$$\langle\{a \rightarrow \neg b \vee c, \neg c\}, \diamond\rangle\ \mathbf{U} \quad \langle\{\neg d\}, \diamond\rangle\ \mathbf{U}$$

Then, by the definition of the judge function, since for the root node A_r of the above tree T it holds that Mark$(A_r) \neq U$, *then* Judge(T) = Unwarranted.

An argument tree provides an overview of all the possible ways argumentation can develop starting from an initial argument that is the root of the argument tree. Then, depending on the structure of the argument tree and by using the judge function, the tree can be classified as warranted or not, providing a way for evaluating the initial argument.

2.2.3.6 Examples in classical predicate logic

All the examples given so far in section 2.2.3 are based on classical propositional logic. The definitions presented so far in this section though cover classical logic, including both propositional and first-order

logic. More details on argumentation focused on classical predicate logic can be found in [13]. This paragraph presents some examples in classical first-order logic adapted from [13].

Example 2.2.23 *Let* $\Delta = \{\forall x(P(x) \rightarrow Q(x) \vee R(x)), P(a), \neg\forall x(S(x)), \neg\exists x(R(x)), \neg\exists x(P(x) \rightarrow Q(x) \vee R(x))\}$. *Some arguments from* Δ *are:*

$$\langle\{P(a), \forall x(P(x) \rightarrow Q(x) \vee R(x))\}, Q(a) \vee R(a)\rangle$$
$$\langle\{\neg\forall x(S(x))\}, \neg\forall x(S(x))\rangle$$
$$\langle\{\neg\exists x(R(x))\}, \neg\exists x(R(x))\rangle$$
$$\langle\{\neg\exists x(R(x))\}, \forall x(\neg R(x))\rangle$$

The following example demonstarates how arguments in first-order logic can be compared on which is more conservative than the other by using definition 2.2.4.

Example 2.2.24 $\langle\{P(a), \forall x(P(x) \rightarrow Q(x) \vee R(x)), Q(a) \vee R(a)\}\rangle$ *is a more conservative argument than* $\langle\{P(a), \forall x(P(x) \rightarrow Q(x) \vee R(x)), \neg\exists x(R(x))\}, Q(a)\rangle$.

The next example involves undercuts in first-order logic.

Example 2.2.25 *Let* $A_1 = \langle\{P(a), \forall x(P(x) \rightarrow Q(x) \vee R(x)), Q(a) \vee R(a)\}\rangle$. *Then* A_1 *is an argument and* $A_2 = \langle\{\neg\exists x(P(x) \rightarrow Q(x) \vee R(x))\}, \neg\forall x(P(x) \rightarrow Q(x) \vee R(x))\rangle$ *is an undercut for* A_1. *A more conservative undercut for* A_1 *is* $A_3 = \langle\{\neg\exists x(P(x) \rightarrow Q(x) \vee R(x))\}, \neg(P(a) \wedge \forall x(P(x) \rightarrow Q(x) \vee R(x)))\rangle$. *Moreover,* A_3 *is a maximally conservative undercut and canonical undercut for* A_1.

Finally, the argument trees are also defined for arguments in first-order logic. Then, according to definition 2.2.10, the following is an argument tree for $\phi = Q(a) \vee R(a)$.

Example 2.2.26 *Continuing example 2.2.24, the following is an argument tree for* $Q(a) \vee R(a)$.

$$\langle\{P(a), \forall x(P(x) \rightarrow Q(x) \vee R(x))\}, Q(a) \vee R(a)\rangle$$
$$|$$
$$\langle\{\neg\exists x(P(x) \rightarrow Q(x) \vee R(x))\}, \diamond\rangle$$

Using the judge function this tree is classified as Unwarranted.

2.2.4 Computational issues in argumentation

Various approaches to argumentation yield computational issues that sometimes seem to make argumentation unsuitable for practical applications. Classical logic has many advantages for representing and reasoning with knowledge however, for argumentation, it is computationally challenging to generate arguments from a knowledgebase using classical logic. If we consider the problem as an abduction problem, where we seek the existence of a minimal subset of a set of formulae that implies the consequent, then the problem is in the second level of the polynomial hierarchy [36].

The difficult nature of argumentation has been underlined by studies concerning the complexity of finding individual arguments [63] and the complexity of finding argument trees [49]. Furthermore,

encoding of these tasks as quantified Boolean formulae also indicate that development of algorithms is a difficult challenge [16]. Further work regarding the complexity of argumentation related problems can be found in [80, 25, 28].

2.3 Existing implementations of argumentation systems

There are a number of argumentation engines based on some of the proposals discussed in this chapter. Some of these are mentioned below.

1. DeLP interpreter. Implemented argumentation system based on defeasible logic programming [45, 44].

2. Argue tuProlog. A Java-based argumentation engine built using a Prolog engine as its foundation. Accepts formulae in a first-order language where each formula comes with a numerical value that quantifies the degree of belief for this formula [21, 19].

3. CaSAPI. Prolog implementation that combines abstract and assumption-based argumentation. Various different versions released [40, 39].

4. ASPARTIX Answer-set-programming based argumentation tool that adapts abstract argumentation [35, 1].

5. Dungine A Java reasoner that implements abstract argumentation. [74, 73].

6. The ASPIC argumentation engine. It uses a prolog-like syntax and is based on defeasible logic [2].

Other implemented systems exist that focus on specific applications of argumentation like e-democracy [6] and sharing of ideas [71, 3], as well as argument mapping applications [47] and argument graph drawing applications [83, 76].

All known implemented argumentation systems are based on defeasible logics, assumption based argumentation and abstract argumentation, while there is no implemented system based on classical logic that is known.

2.4 Discussion

The formalisations discussed in this chapter provide different approaches to argumentation. With different formalisations arguments can be considered abstract objects whose internal structure is not known, or they can be considered structures built on top of a deductive system where the claim of the argument has to be entailed by its premises. Each formalisation gives a different option on the level of detail of the knowledge represented as well as on the criteria by which an argument is evaluated in terms of information that is put against it.

Classical logic has many advantages for representing and reasoning with knowledge including syntax and proof theory. It provides a powerful language for expressing in detail the premises of an argument, and entailment is based on classical deduction which is well established. It has a simple and

intuitive syntax and semantics, and it is supported by a proof theory and extensive foundational results that make it it an appealing option for argumentation. In addition to these advantages, the framework on which the work of this thesis is based [12] has the advantage of employing canonical undercuts for counterarguments which capture exactly the information that needs to be challenged in order to attack an argument.

Besides its advantages, argumentation in classical logic has the disadvantage of the computational viability of generating arguments. Little work has been done so far to address this issue [50, 14]. Moreover, to our knowledge, prior to this thesis no working software applications have been developed based on classical logic.

Chapter 3

Reducing the search space for arguments

In this chapter I present a proposal for addressing the problem of the computational cost of generating arguments by reducing the search space. This proposal is based on an existing proposal for automated theorem proving [55, 56] where connection graphs are used for sets of clauses. For a set of clauses, a connection graph is a graph where each node is a clause from the set and each arc denotes that there exist complementary literals in the pair of nodes. The idea in using a connection graph for automated theorem proving is that it indicates that the resolution proof rule can be applied for a pair of clauses that are linked by an arc, motivating the application algorithms that form proofs by walking over a connection graph.

The work presented in this chapter does not involve using connection graphs for constructing proofs. It focuses on how using connection graphs can help reduce the search space when looking for arguments. I present how given a formula ψ in CNF (i.e. conjunctive normal form) as the claim for an argument and a knowledgebase Φ that consists of formulae in CNF, we can form connection graphs where each node is a clause that appears as a conjunct in some fomula from Φ. I explain how using a connectivity criterion introduced, we can isolate a subgraph of this connection graph which is associated to a subset of formulae from Φ that contains all the arguments for ψ, and thus potentially provides a reduced search space. This subgraph, defined as the query graph, is relatively inexpensive to locate in terms of computational cost, and using as the search space the formulae of Φ that relate to the nodes of the query graph, can substantially reduce the cost of looking for arguments. A theoretical framework and algorithms for this proposal are presented in this chapter along with some preliminary experimental results to indicate the potential of the approach.

The chapter starts with the definition of the language of clauses $\mathcal{C}$, which is used throughout this thesis, together with some properties of the language and an overview of how the resolution rule applies to the elements of $\mathcal{C}$ and can be used for satisfiability checks. It proceeds by presenting definitions for different types of connection graphs for the elements of $\mathcal{C}$. It continues by introducing knowledgebases that consist of formulae in CNF. It presents how by defining graphs for the clauses that appear in the conjuncts of the formulae of the knowledgebase, we can determine a subset of the knowledgebase that contains all the formulae that may be premises in arguments for a given claim, which is also in CNF. Throughout the chapter, theoretical results that apply to different types of graphs are introduced together with the corresponding definitions of the graphs in order to demonstrate how these definitions are useful

for addressing the problem of generating arguments. The chapter closes with some experimental data on a prototype implementation of the theory presented, that indicate the advantages of the approach.

3.1 The language of clauses $\mathcal{C}$

In this section I present the language of clauses $\mathcal{C}$ on which connection graphs are defined. First I define the language and then I introduce some functions that are defined on the elements of $\mathcal{C}$. I discuss the resolution proof procedure that can be applied on $\mathcal{C}$ and how this can be used for satisfiability checks.

3.1.1 Definition of $\mathcal{C}$

A language of clauses $\mathcal{C}$ is composed by a set of atoms $\mathcal{A}$ as follows: If a is an atom, then a is a **positive literal**, and $\neg a$ is a **negative literal**. If b is a positive literal, or b is a negative literal, then b is a **literal**. The symbol $\mathcal{L}$ denotes the set of all literals. Literals b_1, b_2 are complementary iff they consist of the same atom and one of them is negative and the other one is positive. Then, we say that b_1 is the complement of b_2 and b_2 is the complement of b_1 and we write $b_1 = \overline{b}_2$. If $b_1, .., b_n$ are literals, then $b_1 \vee ... \vee b_n$ is a **clause**. A **clause knowledgebase** is a set of clauses.

Lowercase roman characters denote atoms, lowercase Greek characters denote clauses and uppercase Greek characters denote sets of clauses. The symbol $\perp$ is used to denote the empty clause, that is the clause $b_1 \vee ... \vee b_n$ with $\{b_1, ..., b_n\} = \emptyset$. Then, any set of clauses that contains $\perp$ is unsatisfiable.

3.1.2 Relations in $\mathcal{C}$

The definitions of this section mainly concern disjunctive clauses. I start though by introducing functions Conjuncts and Disjuncts that are defined for formulae in conjunctive normal form and disjunctive normal form respectively, and so can be used for formulae in canonical form that are not necessarily in $\mathcal{C}$. Function Disjuncts is used further in this section to define relations for CNFs of 1 conjunct, i.e. for elements in $\mathcal{C}$.

Definition 3.1.1 *Let ϕ be a formula in conjunctive normal form: $\phi = \gamma_1 \wedge \ldots \wedge \gamma_n$, where each of $\gamma_1, \ldots, \gamma_n$ is a disjunction of literals and let ψ be a formula in disjunctive normal form, $\psi = \delta_1 \vee \ldots \vee \delta_n$, where each of $\delta_1, \ldots, \delta_n$ is a conjunction of literals. Then,* $\mathsf{Conjuncts}(\phi)$ *returns the set of disjunctive clauses in ϕ i.e.* $\mathsf{Conjuncts}(\phi) = \{\gamma_1, \ldots, \gamma_n\}$ *and* $\mathsf{Disjuncts}(\psi)$ *returns the set of conjunctive clauses in ψ i.e.* $\mathsf{Disjuncts}(\psi) = \{\delta_1, \ldots, \delta_n\}$.

Example 3.1.1 *Let $\phi = (a \vee b) \wedge (a \vee d) \wedge q$ and $\psi = (a \wedge c) \vee (d \wedge p)$. Then,* $\mathsf{Conjuncts}(\phi) = \{a \vee b, a \vee d, q\}$ *and* $\mathsf{Disjuncts}(\psi) = \{a \wedge c, d \wedge p\}$. $\mathsf{Disjuncts}(\phi)$ *and* $\mathsf{Conjuncts}(\psi)$ *are not defined.*

Hence, functions Conjuncts and Disjuncts simply return the set of conjuncts or disjuncts respectively for formula in CNF or DNF (i.e. disjunctive normal form). Using the Disjuncts function on disjunctive clauses the following binary relations are defined in $\mathcal{C}$.

Definition 3.1.2 *Let $\phi, \psi \in \mathcal{C}$ and $b \in \mathcal{C} \cap \mathcal{L}$. Then, functions* Preattacks *and* Attacks *are defined as*

follows:

$i)$ $\mathsf{Preattacks}(\phi, \psi) = \{b \mid b \in \mathsf{Disjuncts}(\phi)$ *and* $\bar{b} \in \mathsf{Disjuncts}(\psi)\}$

$ii)$ *If* $\mathsf{Preattacks}(\phi, \psi) = \{b\}$ *for some b then*

$\mathsf{Attacks}(\phi, \psi) = b$ *otherwise* $\mathsf{Attacks}(\phi, \psi) = null$

Hence, the Preattacks relation is defined for any pair of clauses ϕ, ψ and returns the set of complementary literals between these clauses while the Attacks relation is defined for a pair of clauses ϕ, ψ for which $|\mathsf{Preattacks}(\phi, \psi)| = 1$ and returns the unique literal that is contained in $\mathsf{Preattacks}(\phi, \psi)$.

Example 3.1.2 *According to definition 3.1.2, the following hold.*

$\mathsf{Preattacks}(a \vee \neg b \vee \neg c \vee d, a \vee b \vee \neg d \vee e) = \{\neg b, d\}$,

$\mathsf{Preattacks}(a \vee b \vee \neg d \vee e, a \vee \neg b \vee \neg c \vee d) = \{b, \neg d\}$,

$\mathsf{Preattacks}(a \vee b \vee \neg d, a \vee b \vee c) = \emptyset$,

$\mathsf{Preattacks}(a \vee b \vee \neg d, a \vee b \vee d) = \{\neg d\}$,

$\mathsf{Preattacks}(a \vee b \vee \neg d, e \vee c \vee d) = \{\neg d\}$.

Example 3.1.3 *According to definition 3.1.2, the following hold.*

$\mathsf{Attacks}(a \vee \neg b \vee \neg c \vee d, a \vee b \vee \neg d \vee e) = null$,

$\mathsf{Preattacks}(a \vee b \vee \neg d \vee e, a \vee \neg b \vee \neg c \vee d) = null$,

$\mathsf{Attacks}(a \vee b \vee \neg d, a \vee b \vee c) = null$,

$\mathsf{Attacks}(a \vee b \vee \neg d, a \vee b \vee d) = \neg d$,

$\mathsf{Attacks}(a \vee b \vee \neg d, e \vee c \vee d) = \neg d$.

For a set of clauses Δ, $\mathsf{Literals}(\Delta)$ returns the set of literals that appear as disjuncts in the elements of Δ.

Definition 3.1.3 *Let* Δ *be a set of clauses. Then* $\mathsf{Literals}(\Delta) = \bigcup_{\delta \in \Delta} \{d \mid d \in \mathsf{Disjuncts}(\delta)\}$

The resolution function is defined for a pair of clauses that have a pair of complementary literals between them. The following definition for resolution applies to a pair of clauses that have exactly one pair of complementary disjuncts between them and is based on function Attacks.

Definition 3.1.4 *If* ϕ *and* ψ *are clauses and* $\mathsf{Attacks}(\phi, \psi) = b$ *for some* $b \in \mathcal{L}$. *Then,*

$$\phi \bullet \psi = \bigvee (\mathsf{Disjuncts}(\phi) \cup \mathsf{Disjuncts}(\psi)) \setminus \{b, \bar{b}\})$$

Hence, $\bullet$ denotes the function of resolution i.e. for a pair of clauses ϕ and ψ, $\phi \bullet \psi$ is the clause that is obtained by resolution from ϕ and ψ. For simplicity, from now on when the function appears in a sequence of more than two clauses, no brackets will be used and the order in which the resolution function applies to the clauses is going to be the order in which the clauses appear in the sequence. For example, $\phi \bullet \chi \bullet \psi \bullet \alpha = ((\phi \bullet \chi) \bullet \psi) \bullet \alpha$

Example 3.1.4 *Continuing examples 3.1.2 and 3.1.3,*

$a \vee \neg b \vee \neg c \vee d \bullet a \vee b \vee \neg d \vee e$ *is not defined,*

$a \vee b \vee \neg d \vee e \bullet a \vee \neg b \vee \neg c \vee d$ *is not defined,*

$a \vee b \vee \neg d \bullet a \vee b \vee c$ *is not defined,*

$a \vee b \vee \neg d \bullet a \vee b \vee d = a \vee b$,

$a \vee b \vee \neg d \bullet e \vee c \vee d = a \vee b \vee e \vee c$.

3.2 Resolution for satisfiability checks

Proof procedures based on the resolution principle have been widely used for automated theorem proving since Robinson's proposal for a resolution precedure [69] (good introductions and reviews on automated theorem proving based on resolution techniques can be found in [43, 75, 81, 60]). Various refinements on Robinson's resolution procedure have been studied, and strategies for applying resolution efficiently have been developed [82, 5, 17, 7, 57, 55, 61, 58]. A resolution procedure is a refutation procedure where unsatisfiability is established as the means to establish validity. Because of the duality of unsatisfiability and validity (i.e. a formula is valid iff its negation is unsatisfiable) the ground resolution procedure provides a sound and complete method for theorem proving.

In this section I present some definitions and results from [69] and then I introduce a refinement of the resolution procedure, the linear resolution [59], which is used in later chapters for searching for arguments.

3.2.1 The resolution proof procedure

Given a set of clauses Φ, we may wish to generate all the possible resolvents that can be produced by applying resolution recursively between pairs of clauses from Φ and their resolvent clauses. The definition that follows, introduces function $\mathsf{Resolve}(\Phi)$ which provides a way for formalising application of resolution between all the possible pairs of clauses from Φ and function $\mathsf{Resolve}^n(\Phi)$, $n \in \mathbb{N}$ which can be used to recursively apply resolution on set of clauses Φ and the set Φ' of resolvents of Φ and so on.

Definition 3.2.1 *Let Φ be a set of clauses. Then,* $\mathsf{Resolve}(\Phi)$ *returns the set of clauses that consists of the members of Φ together with all the resolvents of all pairs of clauses from Φ.* $\mathsf{Resolve}^n(\Phi)$ *is defined for each $n \geq 0$ as follows:* $\mathsf{Resolve}^0(\Phi) = \Phi$ *and* $\mathsf{Resolve}^{n+1}(\Phi) = \mathsf{Resolve}(\mathsf{Resolve}^n(\Phi))$

For a finite set of clauses Φ there is a finite number of sets $\mathsf{Resolve}^1(\Phi)$, . . . , $\mathsf{Resolve}^n(\Phi)$ such that $\mathsf{Resolve}^i(\Phi) \neq \mathsf{Resolve}^{i+1}(\Phi)$.

Definition 3.2.2 *For a set of clauses Φ,* $\mathsf{Resolvents}(\Phi) = \mathsf{Resolve}^n(\Phi)$ *iff for some $n \geq 0$,* $\mathsf{Resolve}^{n+1}(\Phi) = \mathsf{Resolve}^n(\Phi)$.

According to the ground resolution theorem that follows, applying resolution recursively on a set of clauses Φ can be used to test Φ for satisfiability.

Theorem 3.2.1 *(Robinson 1965) Let Φ be a finite set of clauses. Then, Φ is unsatisfiable iff* $\mathsf{Resolve}^n(\Phi)$ *contains $\perp$ for some $n \geq 0$.*

The two examples that follow demonstrate how theorem 3.2.1 can be applied to test a set of clauses for satisfiability. For simplicity equivalent clauses are not included in a set as this does not affect the conditions of the theorem.

Example 3.2.1 *Let* $\Phi = \{a \vee b, \neg b, \neg a \vee c, \neg c\}$*. Then, according to definition 3.2.1,*

$$\begin{aligned}
&\mathsf{Resolve}^0(\Phi) = \Phi,\\
&\mathsf{Resolve}^1(\Phi) = \{a \vee b, \neg b, a, \neg a \vee c, \neg c, \neg a, b \vee c\},\\
&\mathsf{Resolve}^2(\Phi) = \{a \vee b, \neg b, a, \neg a \vee c, \neg c, \neg a, b \vee c, b, c, \bot\}
\end{aligned}$$

where $\bot$ *in* $\mathsf{Resolve}^2(\Phi)$ *is the resolvent of* a *and* $\neg a$ *from* $\mathsf{Resolve}^1(\Phi)$*. Hence,* Φ *is unsatisfiable because* $\mathsf{Resolve}^2(\Phi)$ *contains* $\bot$*.*

Example 3.2.2 *Let* $\Phi = \{a \vee b \vee d, \neg b, \neg a \vee d, \neg d \vee c\}$*. Then, according to definition 3.2.1,*

$$\begin{aligned}
&\mathsf{Resolve}^0(\Phi) = \Phi,\\
&\mathsf{Resolve}^1(\Phi) = \{a \vee b \vee d, \neg b, a \vee d, \neg a \vee d, b \vee d, \neg d \vee c, a \vee b \vee c, \neg a \vee c\},\\
&\mathsf{Resolve}^2(\Phi) = \{a \vee b \vee d, \neg b, a \vee d, \neg a \vee d, b \vee d, d, \neg d \vee c, a \vee c, b \vee c, a \vee b \vee c, b \vee c \vee d, \neg a \vee c, d \vee c\}\\
&\mathsf{Resolve}^3(\Phi) = \{a \vee b \vee d, \neg b, a \vee d, \neg a \vee d, b \vee d, d, \neg d \vee c, c, a \vee c, b \vee c, a \vee b \vee c, b \vee c \vee d, \neg a \vee c, d \vee c\}
\end{aligned}$$

For $i = 4$ *it holds that* $\mathsf{Resolve}^3(\Phi) = \mathsf{Resolve}^4(\Phi)$*, so* $\mathsf{Resolvents}(\Phi) = \mathsf{Resolve}^3(\Phi)$ *and there is no* $m > 3$ *such that* $\mathsf{Resolve}^m(\Phi) \neq \mathsf{Resolve}^3(\Phi)$*. Then, since* $\bot \notin \mathsf{Resolve}^i(\Phi)$*,* $i = 0 \ldots 3$ *there is no* m *such that* $\bot \in \mathsf{Resolve}^m(\Phi)$ *and so* Φ *is satisfiable.*

Exhaustive generation of the resolvents from a set of clauses Φ can be highly repetitive and the number of clauses produced in every iteration can be large. The following section introduces a method that can reduce the number of resolvents produced with the resolution procedure.

3.2.2 Linear resolution

There are proposals introducing restrictions to the way the resolvents of a set of clauses are generated. They introduce strategies that help reduce the number of resolvents produced until the empty clause is reached when testing the set for satisfiability. Some of these proposals do not maintain completeness or maintain completeness for a restricted language (e.g. Horn clauses). Linear resolution is a refinement of the resolution procedure that provides a refutation complete and sound method for a satisfiability check when dealing with arbitrary propositional clauses [59]. To check whether a set of clauses Φ is unsatisfiable, we can check whether the empty clause is deducible by a linear resolution deduction as defined below.

Definition 3.2.3 *Given a set of clauses* Ψ*, the set of* **linear resolution deductions** *from* Ψ *is the set* $\mathsf{Deductions}(\Psi) = \{\Gamma_1, \ldots, \Gamma_m\}$ *where each* $\Gamma_l = \{\gamma_1, \ldots, \gamma_n\}$*,* $1 \leq l \leq m$ *is defined as follows:*

(1) *for each* $\gamma_k \in \Gamma_l$ *such that* $1 < k \leq n$*, either* γ_k *is obtained by resolution from* γ_i *and* γ_j *where* $i, j < k$ *or* $\gamma_k \in \Psi$ *and*

(2) *for each* γ_i *such that* $1 \leq i < n$*, there are* γ_k *and* γ_j *(*$i < k$ *and* $j < k$*) s.t.* γ_k *is obtained by resolution from* γ_i *and* γ_j

(3) *No* $\gamma_k \in \Gamma_l$ *is a tautology*

Each such Γ_l is called a **linear deduction** of δ_n from Ψ. In the above definition, condition (1) ensures that the clauses of a linear deduction are generated by using elements from Ψ and applying resolution recursively. Condition (2) ensures that every clause in the deduction is used to resolve with some other clause and condition (3) ensures that no tautologies will appear in the deduction.

We say that there is a **linear refutation** of Ψ, if there is a linear deduction of $\perp$ from Ψ. Linear resolution is known to be a refutation complete and sound strategy for automated theorem proving. This is restated in the theorem that follows.

Theorem 3.2.2 *(Loveland 1970) A set of clauses* Φ *is unsatisfiable iff there is a linear refutation from* Φ.

Corollary 3.2.1 *Let* Ψ *be a set of clauses and* $\alpha = a_1 \vee \ldots \vee a_n$ *be a clause. Then,* $\Psi \vdash \alpha$ *iff there is a linear deduction* $\Gamma \in \mathsf{Deductions}(\Psi \cup \{\overline{a_1}, \ldots, \overline{a_n}\})$, $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ *such that* γ_n *is the empty clause.*

Proof: If $\Psi \vdash \alpha$ and $\alpha = a_1 \vee \ldots \vee a_n$ then $\Psi \cup \{\overline{a_1}, \ldots, \overline{a_n}\}$ is unsatisfiable and by theorem 3.2.2 there is linear refutation from $(\Psi \cup \{\overline{a_1}, \ldots, \overline{a_n}\})$ so there is a linear deduction $\Gamma \in \mathsf{Deductions}(\Psi \cup \{\overline{a_1}, \ldots, \overline{a_n}\})$, $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ such that $\gamma_n = \perp$. □

Example 3.2.3 *Let* $\Psi = \{a \vee b \vee c, \neg c \vee d, \neg d \vee \neg c\}$ *and* $\alpha = a \vee b \vee e$*. Then* $\Psi \cup \{\overline{a}, \overline{b}, \overline{e}\}) = \{a \vee b \vee c, \neg c \vee d, \neg d \vee \neg c, \neg a, \neg b, \neg e\}$*. Then there is* $\Gamma \in \mathsf{Deductions}(\Psi \cup \{\overline{a}, \overline{b}, \overline{e}\})$ *with* $\Gamma = \{\neg d \vee \neg c, \neg c \vee d, \neg c, a \vee b \vee c, a \vee b, \neg a, b, \neg b, \perp\}$ *and it holds that* $\Psi \vdash \alpha$.

Moreover, for a minimal inconsistent set of clauses (i.e. a set which is unsatisfiable but all its proper subsets are satisfiable) the following proposition holds.

Proposition 3.2.1 *Let* Ψ *be a minimal inconsistent set of clauses. Then, there is a* $\Gamma \in \mathsf{Deductions}(\Psi)$, $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ *such that* $\gamma_n = \perp$ *and* $\Psi \subseteq \Gamma$.

Proof: Let Ψ be a minimal inconsistent set of clauses. Then, by corollary 3.2.1 there is a linear deduction $\Gamma \in \mathsf{Deductions}(\Psi \cup \emptyset)$, $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ such that γ_n is the empty clause. For a proof by contradiction assume that $\Psi \not\subseteq \Gamma$, so for some $\gamma_i \in \Psi$ it holds that $\gamma_i \notin \Gamma$. Then $\Gamma \cap \Psi \vdash \perp$ and $\Gamma \cap \Psi \subseteq \Psi \setminus \{\gamma\} \subset \Psi$, which contradicts the assumption that Ψ is a minimal inconsistent set □

3.3 Connection Graphs for propositional clauses

In this section I introduce definitions for connection graphs for sets of clauses. Generally, a connection graph is a graph that has clauses for nodes and arcs link clauses that contain complementary literals.

I give definitions for different types of connection graphs according to the number of complementary literals that indicate arcs, or according to the connectivity of the graph. Then, I present how these can be used when we look for arguments for a claim in CNF from a knowledgebase of formulae in CNF. I explain how by using connection graphs for the clauses that appear as conjuncts in the formulae of the knowledgebase, we can narrow the search down to a subgraph that is associated to a subset of the knowledgebase that contains all the formulae that can be premises in arguments for the given claim.

3.3.1 Connection graph definitions

Connection graphs were introduced by Kowalski [55, 56] as a method for automated theorem proving where graphs have disjunctive clauses for nodes and arcs link nodes that contain complementary literals. Kowalski's technique is based on a refutation procedure where a connection graph is traversed until a refutation occurs by applying the resolution proof rule. The connection graph resolution operation resolves on a link in the connection graph, forms the resolvent, and adds it to the connection graph giving the search a dynamic nature. In this chapter the definition of the connection graph is adapted as in [55, 56] and further definitions of special kinds of connection graph are introduced that have some properties with respect to the search for arguments. In Chapter 4, I introduce a search technique for traversing the graphs that does not adapt the dynamic nature of Kowalski's technique. With this technique the structure of the graph does not change by adding nodes that occur by resolution during the search. The clauses visited during the search are instead stored in tree structures that offer the grounds for taking into account properties like minimality and consistency of a proof that are essential for the search for arguments.

A **connection graph** for Δ is an undirected graph (N, A) where N is a set of nodes each of which corresponds to a clause from Δ and A is the set of arcs that connect pairs of clauses with complementary literals. The connection graph as defined here consists of exactly the elements of Δ as its nodes and is therefore unique. The **attack graph** for Δ is a graph where N is a set of nodes each of which corresponds to a clause from Δ and A is the set of arcs that connect pairs of clauses with exactly one complementary literal between them. The **closed graph** introduces a kind of connected subgraph of the attack graph where for each clause ϕ in the subgraph and for each disjunct b in ϕ there is another clause ψ in the subgraph such that $\mathsf{Attacks}(\phi, \psi) = b$. The formal definitions are given below along with examples taken from [32].

Definition 3.3.1 *Let Δ be a clause knowledgebase. The* **connection graph** *for Δ, denoted $\mathsf{Connect}(\Delta)$, is a graph (N, A) where $N = \Delta$ and $A = \{(\phi, \psi) \mid \mathsf{Preattacks}(\phi, \psi) \neq \emptyset\}$.*

Example 3.3.1 *The following is the connection graph for $\Delta = \{\neg b, \neg c \vee \neg g, \neg c, f \vee p, \neg l \vee \neg k, a \vee b, \neg b \vee d, c \vee g, \neg h \vee l, l \vee k, \neg a \vee d, \neg d, \neg g, h \vee \neg l, \neg k, n \vee m \vee \neg q, \neg m, \neg n, m, q\}$.*

$$
\begin{array}{ccccccccccc}
\neg b & & \neg c \vee \neg g & & \neg c & & \neg h \vee l & - & \neg l \vee \neg k & & n \vee m \vee \neg q \\
| & & & \diagdown & | & & | & & | & & | \quad | \quad | \\
a \vee b & - & \neg b \vee d & & c \vee g & & h \vee \neg l & - & l \vee k & & \neg n \quad \neg m \quad q \\
| & & | & & | & & & & | & & | \\
\neg a \vee d & - & \neg d & & \neg g & & f \vee p & & \neg k & & m
\end{array}
$$

The attack graph defined below is a subgraph of the connection graph identified using the Attacks function.

Definition 3.3.2 *Let Δ be a clause knowledgebase. The* **attack graph** *for Δ, denoted* $\mathsf{AttackGraph}(\Delta)$*, is a graph (N, A) where $N = \Delta$ and $A = \{(\phi, \psi) \mid \mathsf{Attacks}(\phi, \psi) \neq null\}$.*

Example 3.3.2 *Continuing Example 3.3.1, the following is the attack graph for Δ.*

$$
\begin{array}{ccccccccccc}
\neg b & & \neg c \vee \neg g & & \neg c & & \neg h \vee l & - & \neg l \vee \neg k & & n \vee m \vee \neg q \\
| & & & & | & & & & & & | \quad | \quad | \\
a \vee b & - & \neg b \vee d & & c \vee g & & h \vee \neg l & - & l \vee k & & \neg n \quad \neg m \quad q \\
| & & | & & | & & & & | & & | \\
\neg a \vee d & - & \neg d & & \neg g & & f \vee p & & \neg k & & m
\end{array}
$$

The **closed graph** for Δ is the subgraph of the attack graph where for each clause ϕ in the subgraph and for each disjunct b in ϕ there is another clause ψ in the subgraph such that $\mathsf{Attacks}(\phi, \psi) = b$ holds.

Definition 3.3.3 *Let Δ be a clause knowledgebase. The* **closed graph** *for Δ, denoted* $\mathsf{Closed}(\Delta)$*, is the largest subgraph (N, A) of* $\mathsf{AttackGraph}(\Delta)$*, such that for each $\phi \in N$, for each $b \in \mathsf{Disjuncts}(\phi)$ there is a $\psi \in N$ with* $\mathsf{Attacks}(\phi, \psi) = b$*.*

Example 3.3.3 *Continuing Example 3.3.2, the following is the closed graph for Δ.*

$$
\begin{array}{ccccccccccc}
\neg b & & & & \neg c & & & & & & n \vee m \vee \neg q \\
| & & & & | & & & & & & | \quad | \quad | \\
a \vee b & - & \neg b \vee d & & c \vee g & & & & & & \neg n \quad \neg m \quad q \\
| & & | & & | & & & & & & | \\
\neg a \vee d & - & \neg d & & \neg g & & & & & & m
\end{array}
$$

The above definition assumes that there is a unique largest subgraph of the attack graph that meets the conditions presented. This is justified because having a node from the attack graph in the closed graph does not exclude any other node from the attack graph also being in the closed graph. Any subset of nodes is included when each of the disjuncts is negated by disjuncts in the other nodes. Moreover, we can consider the closed graph being composed of components where for each component Y, and for each node ϕ in Y, and for each disjunct b in ϕ, there is another node ψ in Y such that there is a disjunct $\neg b$ in ψ. So the nodes in each component work together to ensure each disjunct is negated by a disjunct

in another node in the component, and the largest subgraph of the attack graph is obtained by just taking the union of these components.

For a minimal inconsistent set of clauses Ψ, the nodes of the closed graph for Ψ are exactly the elements of Ψ. This is formalised in the proposition that follows.

Proposition 3.3.1 *Let Γ be a minimal inconsistent set of clauses and let $(N, A) = \mathsf{Closed}(\Gamma)$. Then $N = \Gamma$.*

Proof: Γ is an inconsistent set and so there is a Δ in $\mathsf{Deductions}(\Gamma)$, $\Delta = \{\delta_1, \ldots, \delta_n\}$, where $\delta = \delta_n$ is the empty clause. Also because Γ is a minimal inconsistent set, by proposition 3.2.1 it follows that $\Gamma \subseteq \Delta$. Then:

(1) For all $\delta_i \in \Delta$, for all $a \in \mathsf{Disjuncts}(\delta_i)$ there is $\delta'_i \in \Delta \cap \Gamma$ (possibly $\delta'_i = \delta_i$) such that $a \in \mathsf{Disjuncts}(\delta'_i)$.

(2) For all $\delta_i \in \Delta$, for all $a \in \mathsf{Disjuncts}(\delta_i)$ there is a $\delta_j \in \Delta$ with $\overline{a} \in \mathsf{Disjuncts}(\delta_j)$.

(3) For all $\delta_i \in \Delta$ and for all $a \in \mathsf{Disjuncts}(\delta_i)$ there are $\delta'_i \in \Delta \cap \Gamma$ and $\delta'_j \in \Delta \cap \Gamma$ such that $a \in \mathsf{Disjuncts}(\delta'_i)$ and $\overline{a} \in \mathsf{Disjuncts}(\delta'_j)$: Follows from (1) and (2).

(4) For all $\delta'_i \in \Delta \cap \Gamma$ and for all $a \in \mathsf{Disjuncts}(\delta'_i)$ there is a $\delta'_j \in \Delta \cap \Gamma$ such that $\mathsf{Attacks}(\delta'_i, \delta'_j) = a$: From (3), for all $\delta'_i \in \Delta \cap \Gamma$ and for all $a \in \mathsf{Disjuncts}(\delta'_i)$ there is a $\delta'_j \in \Delta \cap \Gamma$ such that $a \in \mathsf{Preattacks}(\delta'_i, \delta'_j)$. If for all such δ'_i, δ'_j, $\mathsf{Attacks}(\delta'_i, \delta'_j) = null$, then for all δ'_j with $\overline{a} \in \mathsf{Disjuncts}(\delta'_j)$, $|\mathsf{Preattacks}(\delta'_i, \delta'_j)| > 1$, and so for all δ'_j with $\overline{a} \in \mathsf{Disjuncts}(\delta'_j)$ there is a $b \in \mathsf{Disjuncts}(\delta'_i)$ such that $\overline{b} \in \mathsf{Disjuncts}(\delta'_j)$. Since $\Delta \cap \Gamma$ is a minimal inconsistent set, then $\Delta' = \Delta \cap \Gamma \setminus \{\delta'_i\} \vdash \neg\delta'_i$ and Δ' is consistent. So, $\Delta' \vdash \overline{a} \in \mathsf{Conjuncts}(\neg\delta_i)$ and there are $\delta'_j \in \Delta'$ with $\overline{a} \in \mathsf{Disjuncts}(\delta'_j)$. For all the disjuncts b' of $\mathsf{Disjuncts}(\delta'_j) \setminus \{\overline{a}\}$, $\Delta' \vdash \overline{b'}$. But since for all δ'_j with $\overline{a} \in \mathsf{Disjuncts}(\delta'_j)$ there is a $b \in \mathsf{Disjuncts}(\delta'_i)$ such that $\overline{b} \in \mathsf{Disjuncts}(\delta'_j)$ then for $b' = \overline{b}$ holds that $\Delta' \vdash \overline{\overline{b}} = b$. Also, since $b \in \mathsf{Disjuncts}(\delta'_i)$, and $\Delta' \vdash \neg\delta_i$ then $\Delta' \vdash \overline{b} \in \mathsf{Conjuncts}(\neg\delta_i)$ which contradicts the assumption that Δ' is consistent. Hence, the assumption that for some $\delta'_i \in \Delta \cap \Gamma$ there is an $a \in \mathsf{Disjuncts}(\delta'_i)$ for which there is no $\delta'_j \in \Delta \cap \Gamma$ such that $\mathsf{Attacks}(\delta'_i, \delta'_j) = a$, cannot hold and so for all $\delta'_i \in \Delta \cap \Gamma$, for all $a \in \mathsf{Disjuncts}(\delta'_i)$, there is $\delta'_j \in \Delta \cap \Gamma$ such that $\mathsf{Attacks}(\delta'_i, \delta'_j) = a$ from which by the definition of the closed graph follows that $N = \Gamma$. □

For a minimal inconsistent set of clauses Ψ the following proposition also holds, according to which, the closed graph for Ψ consists of a unique component.

Proposition 3.3.2 *Let Ψ be a minimal inconsistent set of clauses. Then, $\mathsf{Closed}(\Psi)$ consists of a unique component.*

Proof: Let Ψ be a minimal inconsistent set of clauses. For a proof by contradiction assume that $\mathsf{Closed}(\Gamma) = (N, A)$ consists of more that one component. Then, there are (N_1, A_1), (N_2, A_2) such that $N_1 \subset N$, $N_2 \subset N$, $A_1 \subset A$, $A_2 \subset A$ and $N_1 \cap N_2 = \emptyset$, $A_1 \cap A_2 = \emptyset$. By proposition 3.2.1, there is a $\Gamma \in \mathsf{Deductions}(\Psi)$, $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ such that $\gamma_n = \bot$ and $\Psi \subseteq \Gamma$. Also, by proposition 3.3.1 it holds that $N = \Psi$. Then, $N \subseteq \Gamma$. Since $N_1 \cap N_2 = \emptyset$, for all $\gamma_i \in N_1$ and for all $\gamma_j \in N_2$,

$\mathsf{Attacks}(\gamma_i, \gamma_j) = null$ so it cannot hold that $N_1 \cup N_2 \subseteq \Gamma$ by the conditions of the definition for a linear deduction and since $N_1 \cup N_2 = N$ and $N = \Psi$, this contradicts the fact that $\Psi \subseteq \Gamma$. □

The **focal graph** (defined next) is a subgraph of the closed graph for Δ which is specified by a clause ϕ from Δ and corresponds to the part of the closed graph that contains ϕ. In the following, a component of a graph means that each node in the component is connected to any other node in the component by a path.

Definition 3.3.4 *Let Δ be a clause knowledgebase and $\phi \in \Delta$. The* **focal graph** *of ϕ in Δ denoted $\mathsf{Focal}(\Delta, \phi)$ is defined as follows: If there is a component X in $\mathsf{Closed}(\Delta)$ containing the node ϕ, then $\mathsf{Focal}(\Delta, \phi) = X$, otherwise $\mathsf{Focal}(\Delta, \phi)$ is the empty graph. Clause ϕ is called the* **epicentre** *of the focal graph.*

Example 3.3.4 *Continuing Example 3.3.3, the following is the focal graph of $\neg b$ in Δ*

$$\begin{array}{ccc} \neg b & & \\ | & & \\ a \vee b & — & \neg b \vee d \\ | & & | \\ \neg a \vee d & — & \neg d \end{array}$$

The last example illustrates how the notion of the focal graph of an epicentre ϕ in Δ can be used in order to focus on the part of the knowledgebase that is relevant to ϕ. From the way the focal graph is defined we obtain the following proposition.

Proposition 3.3.3 *Let Δ be a clause knowledgebase. For all $\gamma_i, \gamma_j \in \Delta$ either $\mathsf{Focal}(\Delta, \gamma_i)$ and $\mathsf{Focal}(\Delta, \gamma_j)$ are the same component of $\mathsf{Closed}(\Delta)$ or they are disjoint components.*

Proof: Follows from definition 3.3.4. If there is a component X_i in $\mathsf{Closed}(\Delta)$ containing the node γ_i, then $\mathsf{Focal}(\Delta, \gamma_i) = X_i$. If X_i contains γ_j then by definition 3.3.4 for γ_j as the epicentre follows that $\mathsf{Focal}(\Delta, \gamma_j) = X_i$ and $\mathsf{Focal}(\Delta, \gamma_i)$ and $\mathsf{Focal}(\Delta, \gamma_j)$ are the same component. If X_i does not contain γ_j then then either there is some component X_j of $\mathsf{Closed}(\Delta)$ that contains γ_j or $\mathsf{Focal}(\Delta, \gamma_j)$ is the empty graph. In the first case X_j and X_i are disjoint otherwise it would hold that X_i contains γ_j. In the second case where $\mathsf{Focal}(\Delta, \gamma_j)$ is the empty graph, $\mathsf{Focal}(\Delta, \gamma_j)$ and $\mathsf{Focal}(\Delta, \gamma_i)$ are trivially disjoint. □

3.3.2 Connection graphs for arguments in CNF

The focal graph can be used to reduce the search space when looking for arguments for a formula ψ in CNF from a knowledgebase Δ that consists of formulae in CNF. This requires using each of the conjuncts of the conjunctive normal form $\overline{\psi}$ of $\neg\psi$ as the epicentre for a focal graph in the set of clauses that appear as conjuncts in the elements of Δ. In order to extend the definitions for connection graphs

to support a language of propositonal formulae in conjunctive normal form, I first introduce the function SetConjuncts which, given a set of formulae where each formula is in CNF, returns the set of clauses that are conjuncts of the formulae in the set.

Definition 3.3.5 *Let $\Phi = \{\phi_1, \ldots, \phi_k\}$ be a set of formulae where each of the $\phi_1, \ldots, \phi_k$ is in conjunctive normal form. Then, function* $\mathsf{SetConjuncts}(\Phi)$ *returns the set clauses that appear as conjuncts in the formulae of the set:*

$$\mathsf{SetConjuncts}(\Phi) = \bigcup_{\phi_i \in \Phi} \mathsf{Conjuncts}(\phi_i)$$

Example 3.3.5 *For $\Phi = \{\neg a, (a \vee b) \wedge \neg d, (c \vee d) \wedge (e \vee f \vee \neg g), \neg d\}$,* $\mathsf{SetConjuncts}(\Phi) = \{\neg a, a \vee b, \neg d, c \vee d, e \vee f \vee \neg g\}$.

Given a set Φ that contains formulae in CNF and a $\gamma_i \in \mathsf{Conjuncts}(\psi)$ for some $\psi \in \Phi$, function $\mathsf{SubFocus}(\Phi, \gamma_i)$ that follows returns the focal graph of γ_i in $\mathsf{SetConjuncts}(\Phi)$.

Definition 3.3.6 *Let Φ be a knowledgebase and $\psi \in \Phi$. Then for each $\gamma_i \in \mathsf{Conjuncts}(\psi)$,*

$$\mathsf{SubFocus}(\Phi, \gamma_i) = \mathsf{Focal}(\mathsf{SetConjuncts}(\Phi), \gamma_i)$$

.

Example 3.3.6 *Let $\Phi = \{(a \vee b) \wedge (f \vee p) \wedge \neg c, (\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), h \vee \neg l, \neg m \wedge \neg n, m \wedge q\}$. Then,* $\mathsf{SetConjuncts}(\Phi)$ *is equal to Δ from example 3.3.1. Let $\phi = (a \vee b) \wedge (f \vee p) \wedge \neg c$, and let γ_1 denote $a \vee b$, γ_2 denote $f \vee p$ and γ_3 denote $\neg c$. So, if* $\mathsf{SubFocus}(\Phi, \gamma_1) = (N_1, A_1)$, $\mathsf{SubFocus}(\Phi, \gamma_2) = (N_2, A_2)$ *and* $\mathsf{SubFocus}(\Phi, \gamma_3) = (N_3, A_3)$ *then*

$$N_1 = \{a \vee b, \neg a \vee d, \neg d, \neg b, \neg b \vee d\},$$
$$N_2 = \emptyset,$$
$$N_3 = \{\neg c, c \vee g, \neg g\}.$$

The next definition introduces the notion of the query graph of a formula ψ in a knowledgebase Φ, where each of the elements of Φ is a formula in CNF. Let $\psi = \delta_1 \wedge \ldots \wedge \delta_n$ be a formula and let $\overline{\psi}$ denote the conjunctive normal form of the negation of ψ, and so $\overline{\psi} = \gamma_1 \wedge \ldots \wedge \gamma_m \equiv \neg\psi$. The query graph is a graph consisting of all the focal graphs of each of the $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ in $\mathsf{SetConjuncts}(\Phi \cup \{\overline{\psi}\})$, i.e. all the $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i)$ for all $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$.

For a graph $C = (N, A)$ let function $\mathsf{Nodes}(C)$ return the set of clauses that correspond to the nodes of the graph (i.e. $\mathsf{Nodes}(C) = N$) and for a formula ψ, let $\overline{\psi}$ denote the conjunctive normal form of $\neg\psi$. Using this notation the query graph of ψ in Φ is defined as follows.

Definition 3.3.7 *Let Φ be a knowledgebase and ψ be a formula. The* **query graph** *of ψ in Φ denoted* $\mathsf{Query}(\Phi, \psi)$ *is the closed graph for the nodes*

$$\bigcup_{\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})} \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i))$$

Example 3.3.7 *Let* $\Phi' = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), h \vee \neg l, \neg m \wedge \neg n, m \wedge q\}$ *and let* $\psi = (\neg a \vee \neg f \vee c) \wedge (\neg a \vee \neg p \vee c) \wedge (\neg b \vee \neg f \vee c) \wedge (\neg b \vee \neg p \vee c)$. *For* Φ' *and* ψ *we have* $\overline{\psi} = (a \vee b) \wedge (f \vee p) \wedge \neg c$, *which is equal to* ϕ *from example 3.3.6 and* $\Phi' \cup \{\overline{\psi}\}$ *is equal to* Φ *from example 3.3.6. Then, continuing example 3.3.6, the query graph of* ψ *in* Φ *is presented below and consists of the subgraphs* $(N_1, A_1), (N_2, A_2)$ *and* (N_3, A_3).

$$\begin{array}{ccccc}
\neg b & & & & \neg c \\
| & & & & | \\
a \vee b & — & \neg b \vee d & & c \vee g \\
| & & | & & | \\
\neg a \vee d & — & \neg d & & \neg g
\end{array}$$

So, using each of the conjuncts γ_i of $\overline{\psi}$ as the epicentre for the focal graph in $\mathsf{SetConjuncts}(\Phi \cup \{\overline{\psi}\})$ we obtain a component of the query graph of ψ in Φ.

The idea behind the notion of the query graph is that if we seek supports for arguments for ψ from a knowledgebase $\Phi = \{\phi_1, \ldots, \phi_k\}$ (where each of the $\phi_1, \ldots, \phi_k$ is in CNF), instead of searching among the elements of Φ, we can search among the elements of Φ that consist of formulae whose conjuncts are contained in each of the components of the query graph i.e. in each $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i)$ for all $\gamma_i \in \mathsf{Conjuncts}(\psi)$. This can provide a reduced knowledgebase to consider, without losing any arguments for ψ. These results can be formalised by using some additional definitions that correlate the clauses of the query graph with the formulae of the knowledgebase. The following definition introduces the notion of a zone, which associates each clause from each subfocus to one or more formulae from knowledgebase Φ.

Definition 3.3.8 *Let* Φ *be a knowledgebase and* ψ *be a formula. Then, for each* $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$,

$$\mathsf{Zone}(\Phi, \gamma_i) = \{\phi \in \Phi \mid \mathsf{Conjuncts}(\phi) \cap \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i)) \neq \emptyset\}$$

Example 3.3.8 *Continuing examples 3.3.6 and 3.3.7, where* $\overline{\psi} = (a \vee b) \wedge (f \vee p) \wedge \neg c$, $\gamma_1 = a \vee b$, $\gamma_2 = f \vee p$ *and* $\gamma_3 = \neg c$, *for each* $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi}), i = 1 \ldots 3$ *we have*

$$\begin{aligned}
&\mathsf{Zone}(\Phi', \gamma_1) = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg b, \neg b \vee d, \neg d \wedge (\neg h \vee l)\} \\
&\mathsf{Zone}(\Phi', \gamma_2) = \emptyset \\
&\mathsf{Zone}(\Phi', \gamma_3) = \{c \vee g, \neg g\}
\end{aligned}$$

From the way SubFocus and Zone are defined, it holds that conjuncts of $\overline{\psi}$ that are in the same SubFocus correspond to the same set Zone of formulae from Φ.

Proposition 3.3.4 *Let* Φ *be a set of formulae and let* ψ *be a formula. If* $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$ *for some* $\gamma_i, \gamma_j \in \mathsf{Conjuncts}(\overline{\psi})$, *then* $\mathsf{Zone}(\Phi, \gamma_i) = \mathsf{Zone}(\Phi, \gamma_j)$.

Proof: Let Φ be a set of formulae and ψ be a formula and let for some $\gamma_i, \gamma_j \in \mathsf{Conjuncts}(\overline{\psi})$ $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$. Then, it holds that $\mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$ so for all $\phi \in \Phi$, $\mathsf{Conjuncts}(\phi) \cap \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) \neq \emptyset$ iff $\mathsf{Conjuncts}(\phi) \cap \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j) \neq \emptyset$ from which follows that $\phi \in \mathsf{Zone}(\Phi, \gamma_i)$ iff $\phi \in \mathsf{Zone}(\Phi, \gamma_j)$ and so $\mathsf{Zone}(\Phi, \gamma_i) = \mathsf{Zone}(\Phi, \gamma_j)$. □

The converse of the last proposition does not hold as the following example illustrates.

Example 3.3.9 *For* $\Phi = \{(c \vee g) \wedge d, d \vee f, \neg q, (d \vee p) \wedge f, \neg n, k \vee \neg m\}$ *and* $\psi = c \vee g \vee d$ *we have* $\overline{\psi} = \neg c \wedge \neg g \wedge \neg d$ *and* $N_1 \equiv \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \neg c) = \{\neg c, \neg g, c \vee g\}$, $N_2 \equiv \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \neg g) = \{\neg c, \neg g, c \vee g\} = N_1$ *and* $N_3 \equiv \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \neg d) = \{\neg d, d\}$. *Furthermore,* $\mathsf{Zone}(\Phi, \neg c) = \mathsf{Zone}(\Phi, \neg g) = \mathsf{Zone}(\Phi, \neg d) = \{(c \vee g) \wedge d\}$ *although* $N_3 \neq N_2$ *and* $N_3 \neq N_1$.

The following proposition demonstrates how the query graph of ψ in Φ, and in particular the set $\mathsf{Zone}(\Phi, \gamma_i)$ that is associated to a component $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i)$ of the query graph is useful in the search for arguments for ψ from Φ.

Proposition 3.3.5 *Let* Φ *be a knowledgebase where each element is in CNF and* ψ *be a clause. If* $\langle \Psi, \psi \rangle$ *is an argument from* Φ*, then there is a* $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ *s.t* $\Psi \subseteq \mathsf{Zone}(\Phi, \gamma_i)$.

Proof: Let $\langle \Psi, \psi \rangle$ be an argument where $\Psi \neq \emptyset$. Then $\Psi \cup \{\overline{\psi}\}$ is a minimal inconsistent set, so there is a $\Gamma \subseteq \mathsf{SetConjuncts}(\Psi \cup \{\overline{\psi}\})$ such that Γ is a minimal inconsistent set. Then for all $\phi \in \Psi \cup \{\overline{\psi}\}$ there is a $\gamma \in \Gamma$ such that $\gamma \in \mathsf{Conjuncts}(\phi)$, otherwise if this did not hold, since $\Gamma \vdash \bot$, and $\mathsf{Conjuncts}(\phi) \cap \Gamma = \emptyset$ then $(\Psi \cup \{\overline{\psi}\}) \setminus \{\phi\} \vdash \bot$ and this contradicts the assumption that $\Psi \cup \{\overline{\psi}\}$ is a minimal inconsistent set. For Γ it holds that $\Gamma \cap \mathsf{SetConjuncts}(\Psi) \neq \emptyset$ otherwise it would hold that $\Gamma \subseteq \mathsf{SetConjuncts}(\{\overline{\psi}\})$, and because $\Gamma \vdash \bot$ it would hold that $\{\overline{\psi}\} \vdash \bot$ which contradicts the assumption that $\Psi \cup \{\overline{\psi}\}$ is a minimal inconsistent set. Similarly, $\Gamma \cap \mathsf{SetConjuncts}(\{\overline{\psi}\}) \neq \emptyset$ otherwise it would hold that $\Gamma \subseteq \mathsf{SetConjuncts}(\Psi)$, and because $\Gamma \vdash \bot$ it would hold that $\Psi \vdash \bot$ which contradicts the assumption that $\Psi \cup \{\overline{\psi}\}$ is a minimal inconsistent set. Then there is a $\Gamma_1 \subseteq \mathsf{SetConjuncts}(\Psi)$, $\Gamma_1 \neq \emptyset$ and a $\Gamma_2 \subseteq \mathsf{SetConjuncts}(\{\overline{\psi}\})$, $\Gamma_2 \neq \emptyset$ such that $\Gamma = \Gamma_1 \cup \Gamma_2$. For all $\gamma_j \in \Gamma_2$ it holds that $\gamma_j \in \mathsf{Nodes}(\mathsf{SubFocus}(\Psi \cup \{\overline{\psi}\}, \gamma_j))$ by the definition of SubFocus. By proposition 3.3.2, $\mathsf{Closed}(\Gamma)$ consists of a unique component. By proposition 3.3.1, if $\mathsf{Closed}(\Gamma) = (N, A)$, then $\Gamma = N$. Since $\Gamma = N$, and (N, A) consists of a unique component, for all $\gamma_j \in \Gamma \cap \Gamma_2$ follows that they are contained in the same componenent of $\mathsf{Closed}(\Gamma)$ and hence the same component of $\mathsf{Closed}(\mathsf{SetConjuncts}(\Psi \cup \{\overline{\psi}\}))$. So there is a component (N', A') of $\mathsf{Closed}(\mathsf{SetConjuncts}(\Psi \cup \{\overline{\psi}\}))$ such that for all $\gamma_i, \gamma_j \in \Gamma \cap \Gamma_2$, $\mathsf{Focal}(\mathsf{SetConjuncts}(\Psi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{Focal}(\mathsf{SetConjuncts}(\Psi \cup \{\overline{\psi}\}, \gamma_j) = (N', A')$ and so by the definition for SubFocus, for all $\gamma_i, \gamma_j \in \Gamma \cap \Gamma_2$, $\mathsf{SubFocus}(\Psi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{SubFocus}(\Psi \cup \{\overline{\psi}\}, \gamma_j) = (N', A')$. By the way the focal graph is defined it holds that (N', A') consists of a unique component. Since for all $\gamma_j \in \Gamma_2$ it holds that $\gamma_j \in N'$, then $\Gamma_2 \subseteq N'$. Because (N, A) and (N', A') are connected subgraphs of $\mathsf{Closed}(\mathsf{SetConjuncts}(\Psi \cup \{\overline{\psi}\})$ each of which consists of a unique component and since $\Gamma_2 \cap N \subset N'$ and $N = \Gamma_1 \cup \Gamma_2$ then it also holds that $\Gamma_1 \cap N \subseteq N'$. Hence, it holds that $(\Gamma_1 \cup \Gamma_2) \cap N \subseteq$

N' and because $N = \Gamma = \Gamma_1 \cup \Gamma_2$ the above is equivalent to $N \subseteq N'$. From the way Γ is defined, the following holds

(1) $\forall \phi \in \Psi \cup \{\overline{\psi}\}$, $\exists \gamma \in \Gamma$ such that $\gamma \in \mathsf{Conjuncts}(\phi)$

from (1) follows that

(2) $\forall \phi \in \Psi$, $\exists \gamma \in \Gamma$ such that $\gamma \in \mathsf{Conjuncts}(\phi)$

since $\Gamma = N$, then from (2) follows that

(3) $\forall \phi \in \Psi$, $\exists \gamma \in N$ such that $\gamma \in \mathsf{Conjuncts}(\phi)$

because $N \subseteq N'$, from (3) follows that

(4) $\forall \phi \in \Psi$, $\exists \gamma \in N'$ such that $\gamma \in \mathsf{Conjuncts}(\phi)$

because for all $\gamma_i \in \Gamma_2$, $(N', A') = \mathsf{SubFocus}(\Psi \cup \{\overline{\psi}\}, \gamma_i)$, from (4) follows that

(5) $\forall \phi \in \Psi$ and for some $\gamma_i \in \Gamma_2$, $\exists \gamma \in \mathsf{Nodes}(\mathsf{SubFocus}(\Psi \cup \{\overline{\psi}\}, \gamma_i))$ s.t. $\gamma \in \mathsf{Conjuncts}(\phi)$

from the definition of Zone and (5) it follows that $\forall \phi \in \Psi$ and for some $\gamma_i \in \Gamma_2$, $\phi \in \mathsf{Zone}(\Phi, \gamma_i)$ from which follows that for some $\gamma_i \in \Gamma_2$, $\Psi \subseteq \mathsf{Zone}(\Phi, \gamma_i)$ and so there is a $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ such that $\Psi \subseteq \mathsf{Zone}(\Phi, \gamma_i)$. □

Example 3.3.10 *Continuing example 3.3.7, let* $\Phi' = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), h \vee \neg l, \neg m \wedge \neg n, m \wedge q\}$ *and* $\psi = (\neg a \vee \neg f \vee c) \wedge (\neg a \vee \neg p \vee c) \wedge (\neg b \vee \neg f \vee c) \wedge (\neg b \vee \neg p \vee c)$. *Then, all the arguments for* ψ *from* Φ *are the following*

$$
\begin{aligned}
A_1 &= \langle \Psi_1, \psi \rangle = \langle \{c \vee g, \neg g\}, \psi \rangle \\
A_2 &= \langle \Psi_2, \psi \rangle = \langle \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg b, \neg d\}, \psi \rangle \\
A_3 &= \langle \Psi_3, \psi \rangle = \langle \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg b \vee d, \neg d \wedge (\neg h \vee l)\}, \psi \rangle \\
A_4 &= \langle \Psi_4, \psi \rangle = \langle \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg b \vee d, \neg d\}, \psi \rangle \\
A_5 &= \langle \Psi_5, \psi \rangle = \langle \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg b, \neg d \wedge \neg h \vee l\}, \psi \rangle
\end{aligned}
$$

From example 3.3.8 we have $\overline{\psi} = (a \vee b) \wedge (f \vee p) \wedge \neg c$, *and for each* $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi}), i = 1 \ldots 3$ *where* $\gamma_1 = a \vee b$, $\gamma_2 = f \vee p$ *and* $\gamma_3 = \neg c$, *we have* $\mathsf{Zone}(\Phi', \gamma_1) = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg b, \neg b \vee d, \neg d \wedge (\neg h \vee l)\}$, $\mathsf{Zone}(\Phi', \gamma_2) = \emptyset$ *and* $\mathsf{Zone}(\Phi', \gamma_3) = \{c \vee g, \neg g\}$. *For the supports of the arguments given above it holds that* $\Psi_1 \subseteq \mathsf{Zone}(\Phi', \gamma_3)$, *and for* $j = 2 \ldots 5$, *it holds that* $\Psi_j \subseteq \mathsf{Zone}(\Phi', \gamma_1)$

So if $\langle \Psi, \psi \rangle$ is an argument, then there is some $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ that delineates the subset $\mathsf{Zone}(\Phi, \gamma_i)$ of knowledgebase Φ which contains the support Ψ. Therefore, in order to find all the arguments for ψ we need to consider the set of formulae that corresponds to the zones of all the $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$. This set is the *supportbase* defined next.

Definition 3.3.9 *For a knowledgebase* Φ *and a formula* ψ *the supportbase is given as follows:*

$$\mathsf{SupportBase}(\Phi, \psi) = \bigcup_{\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})} \mathsf{Zone}(\Phi, \gamma_i)$$

Example 3.3.11 *Continuing example 3.3.10,* $\mathsf{SupportBase}(\Phi, \psi) = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg b, \neg b \vee d, \neg d \wedge (\neg h \vee l), c \vee g, \neg g\}$.

As a consequence of proposition 3.3.5 we obtain the following corollary, according to which, $\mathsf{SupportBase}(\Phi, \psi)$ is the knowledgebase that contains all the arguments for ψ from Φ.

Corollary 3.3.1 *Let Φ be a knowledgebase where each formula from Φ is in conjunctive normal form and let ψ be a formula. If $\langle \Psi, \psi \rangle$ is an argument for ψ from Φ, then $\Psi \subseteq \mathsf{SupportBase}(\Phi, \psi)$.*

Proof: Let $\langle \Psi, \psi \rangle$ be an argument where $\Psi \subseteq \Phi$. From proposition 3.3.5, it follows that there is some $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$ such that $\Psi \subseteq \mathsf{Zone}(\Phi, \gamma_i)$. Then, $\Psi \subseteq \bigcup_{\gamma_j \in \mathsf{Conjuncts}(\overline{\psi})} \mathsf{Zone}(\Phi, \gamma_j)$ which is equivalent to $\Psi \subseteq \mathsf{SupportBase}(\Phi, \psi)$. □

Hence, by corollary 3.3.1, it follows that all the supports for arguments for ψ are contained in $\mathsf{SupportBase}(\Phi, \psi)$. So, instead of using the initial knowledgebase Φ as the background knowledge in order to look for arguments for ψ, we can use $\mathsf{SupportBase}(\Phi, \psi)$.

The next section presents the algorithms that implement the theory introduced in this section.

3.4 Algorithms for producing connection graphs

In this section three algorithms are presented, each of which is associated with a notion defined in the previous section. The first algorithm $\mathsf{GetFocal}(\Delta, \phi)$ returns the focal graph of a clause ϕ in a knowledgebase of clauses Δ. Using algorithm GetFocal, the second algorithm $\mathsf{GetQueryGraph}(\Phi, \psi)$ returns the query graph of a formula ψ in a set Φ that consists of formulae in CNF. Using the output of $\mathsf{GetQueryGraph}(\Phi, \psi)$, the last algorithm $\mathsf{RetrieveZones}(\Phi, \psi)$ returns the sets $\mathsf{Zone}(\Phi, \gamma_i)$ for all $\gamma_i \in \overline{\psi}$.

3.4.1 Algorithm for the focal graph

Algorithm 3.1 returns the set of nodes of the focal graph of a clause ϕ in a clause knowledgebase Δ. Algorithm $\mathsf{GetFocal}(\Delta, \phi)$ finds the focal graph of ϕ in Δ by a depth first search following the links of the component of $\mathsf{AttackGraph}(\Delta)$ that is linked to ϕ. For this, a data structure $Node_\psi$ (for each $\psi \in \Delta$) is used that represents the node for ψ in $\mathsf{AttackGraph}(\Delta)$. The attack graph can be represented by an adjacency matrix. Initially all the nodes are allowed as candidate nodes for the focal graph of ϕ in Δ and stored in the set AllowedNodes. During the search they can be rejected if they do not satisfy the conditions of the definition for the focal graph and removed from the set AllowedNodes. The algorithm chooses the appropriate nodes by using the boolean method $\mathsf{isConnected}(C, Node_\psi)$ which tests whether a node $Node_\psi$ of the attack graph $C = (N, A)$ is such that each literal $b \in \mathsf{Disjuncts}(\psi)$ corresponds to at least one arc to an allowed node (i.e. $\forall b \in \mathsf{Disjuncts}(\psi), \exists Node_{\psi'} \in \mathsf{AllowedNodes}$ s.t. $\mathsf{Attacks}(\psi, \psi') = b$), and so it returns false when there is a $b \in \mathsf{Disjuncts}(\psi)$ for which there is no $Node_{\psi'} \in \mathsf{AllowedNodes}$ s.t. $\mathsf{Attacks}(\psi, \psi') = b$. If this method does return false for a $Node_\psi$, then $Node_\psi$ is rejected (it is removed from set AllowedNodes) and the algorithm backtracks to retest whether its adjacent allowed

nodes are still connected. If some of them are no longer connected, they are rejected and in the same way their adjacent allowed nodes are tested recursively.

Algorithm 3.1 GetFocal(Δ, ϕ)

```
Let C = (N, A) be the attack graph for Δ
Let AllowedNodes = {Node_ψ | ψ ∈ N}
Let VisitedNodes be the empty set.
if φ ∉ Δ or ¬isConnected(C, Node_φ) then
  return ∅
else
  Let S be an empty Stack
  push Node_φ onto S
end if
while S is not empty do
  Let Node_ψ be the top of the stack S
  if Node_ψ ∈ AllowedNodes then
    if isConnected(C, Node_ψ) then
      if Node_ψ ∈ VisitedNodes then
        pop Node_ψ from S
      else
        VisitedNodes = VisitedNodes ∪ {Node_ψ}
        pop Node_ψ from S
        for all Node_ψ' ∈ AllowedNodes with Attacks(ψ, ψ') ≠ null do
          push Node_ψ' onto S
        end for
      end if
    else
      AllowedNodes = AllowedNodes \ {Node_ψ}
      VisitedNodes = VisitedNodes ∪ {Node_ψ}
      pop Node_ψ from S
      for all Node_ψ' ∈ (AllowedNodes \ VisitedNodes) with Attacks(ψ, ψ') ≠ null do
        push Node_ψ' onto S
      end for
    end if
  else
    pop Node_ψ from S
  end if
end while
return AllowedNodes ∩ VisitedNodes
```

3.4.2 Algorithm for the query graph

Using algorithm GetFocal(Δ, ϕ) for Δ = SetConjuncts($\Phi \cup \{\overline{\psi}\}$) and $\phi = \gamma_1, \ldots, \gamma_n$ where $\{\gamma_1, \ldots, \gamma_n\}$ = Conjuncts($\overline{\psi}$), algorithm 3.2 returns a set of sets each of which corresponds to the set of clauses of a component of the query graph of ψ in Φ. Each component of the query graph is the focal graph of a $\gamma_i \in$ Conjuncts($\overline{\psi}$) in SetConjuncts($\Phi \cup \{\overline{\psi}\}$) i.e. SubFocus($\Phi \cup \{\overline{\psi}\}, \gamma_i$). The algorithm stores the set of clauses S_i of each such component that is retrieved by GetFocal for some epicentre γ_i and proceeds to the next epicentre.

In Algorithm 3.2 we can see that it is not always necessary to use Algorithm 3.1 for each of the $\gamma_i \in$ Conjuncts($\overline{\psi}$) when trying to isolate the appropriate subsets of Φ. Testing for containment of a clause $\gamma_j \in$ Conjuncts($\overline{\psi}$) in an already retrieved set S_i = Nodes(SubFocus($\Phi \cup \{\overline{\psi}\}, \gamma_i$)), $i < j$ (where the ordering of the indices describes the order in which the algorithm is applied for each of the

conjuncts) is sufficient to give the $S_j = \mathsf{Nodes}(\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j))$ according to proposition 3.3.3.

Algorithm 3.2 $\mathsf{GetQueryGraph}(\Phi, \psi)$

Let $\overline{\psi}$ be $\neg\psi$ in CNF : $\overline{\psi} \equiv \gamma_1 \wedge \ldots \wedge \gamma_m$
Let S be a set to store sets of clauses, initially empty
Let $\mathsf{Clauses} = \mathsf{SetConjuncts}(\Phi \cup \{\overline{\psi}\})$
for $i = 1 \ldots m$ **do**
  if $\exists S_j \in S$ s.t. $\gamma_i \in S_j$ **then**
    $i = i + 1$
  else
    $S_i = \mathsf{GetFocal}(\mathsf{Clauses}, \gamma_i)$
  end if
  $S = S \cup \{S_i\}$
end for
return S

3.4.3 Algorithm for zones

The last algorithm, $\mathsf{RetrieveZones}(\Phi, \psi)$, returns the set that contains all the sets $\mathsf{Zone}(\Phi, \gamma_i)$ for each $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$. Using the results of algorithm 3.2, it associates each set of clauses S_i from the output $\{S_1 \ldots, S_k\}$ of $\mathsf{GetQueryGraph}(\Phi, \psi)$ to a set of formulae $Z_i \subseteq \Phi$ where for each Z_i, $\phi \in Z_i$ iff $\phi \in \Phi$ and there is a $\gamma \in S_i$ such that $\gamma \in \mathsf{Conjuncts}(\phi)$. Then, by definition 3.3.8, each such Z_i corresponds to a $\mathsf{Zone}(\Phi, \gamma_i)$ for some $\gamma_i \in \mathsf{Conjuncts}(\overline{\psi})$. When this has been applied to all the the sets S_j from $\{S_1 \ldots, S_k\}$ and for all S_j there is a Z_j constructed as described above, all the sets $\mathsf{Zone}(\Phi, \gamma_j)$ are retrieved. From the way algorithm 3.2 works, if for some γ_i, γ_j $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$, then only one occurrence of the set of nodes of this subfocus is stored in the output set $\{S_1 \ldots, S_k\}$. This does not affect the results of algorithm 3.3 in terms of omitting results. By proposition 3.3.4, if $\mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_i) = \mathsf{SubFocus}(\Phi \cup \{\overline{\psi}\}, \gamma_j)$ then $\mathsf{Zone}(\Phi, \gamma_i) = \mathsf{Zone}(\Phi, \gamma_j)$ and so by storing the set of nodes of identical components only once, we can save repeating operations that give the same result without missing useful outputs.

Algorithm 3.3 $\mathsf{RetrieveZones}(\Phi, \psi)$

Let Z be a set to store sets of formulae, initially empty
Let $S = \mathsf{GetQueryGraph}(\Phi, \psi) \equiv \{S_1, \ldots, S_k\}$
for $i = 1 \ldots k$ **do**
  Let Z_i be the emptyset.
  for $j = 1 \ldots |S_i|$ **do**
    Let γ_j be the j-th element of S_i
    Let $C_j = \{\phi \in \Phi \mid \gamma_j \in \mathsf{Conjuncts}(\phi)\}$
    $Z_i = Z_i \cup C_j$
  end for
  $Z = Z \cup \{Z_i\}$
end for
return Z

3.5 Experimental results

This section covers a preliminary experimental evaluation of algorithm 3.1 using a prototype implementation programmed in java running on a modest PC (Core2 Duo 1.8GHz).

The experimental data were obtained using randomly generated clause knowledgebases of a fixed number of 600 clauses according to the fixed clause length model K-SAT [70, 46] where the chosen length (i.e. K) for each clause was either a disjunction of 3 literals or a disjunction of 1 literal. The clauses of length 3 can be regarded as **rules** and clauses of length 1 as **facts**. Each disjunct of each clause was randomly chosen out of a set of N distinct variables (i.e. atoms) and negated with probability 0.5.

In the experiment, two dimensions were considered. The first dimension was the clauses-to-variables ratio. This ratio is considered to be the integer part of the division of the number of clauses in Δ by the number of variables N (i.e. $\lfloor |\Delta|/|N| \rfloor$). The second dimension was the proportion of facts-to-rules in the knowledgebases tested. The preliminary results are presented in Figure 3.1 where each curve in the graph corresponds to one of these variations on the proportion of facts-to-rules. More precisely, each curve relates to one of the following (n, n') tuples where n represents the number of facts and n' represents the number of rules in the set: (150,450), (200,400), (300,300), (400,200), (450,150). Since each clause knowledgebase contains 600 elements, each of these tuples sums to 600. Each point on each curve is the average focal graph size from 1000 repetitions of running Algorithm 3.1 for randomly generated epicentres and randomly generated knowledgebases of a fixed clauses-to-variables ratio represented by coordinate x. For the results presented, since the values on axis x ranges from 1 to 15, the smallest number of variables used throughout the experiment was 40 which corresponds to a clauses-to-variables ratio of 15, while the largest number of variables used was 600 which corresponds to a clauses to variables ratio of 1.

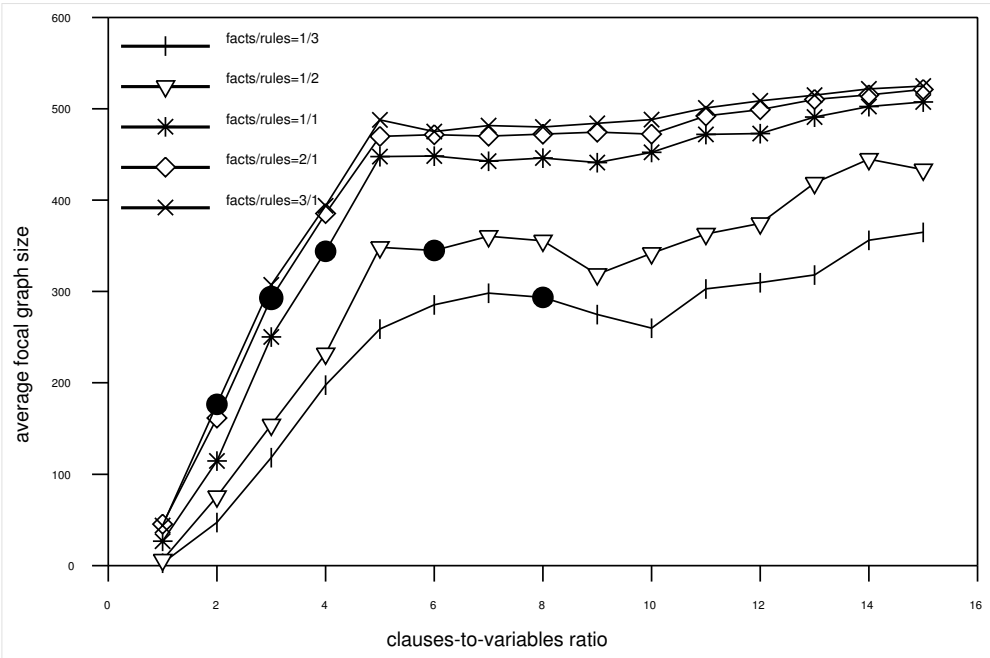


Figure 3.1: Focal graph size variation with the clauses-to-variables ratio

The evaluation of this experiment was based on the size of the focal graph of an epicentre in a clause knowledgebase compared to the cardinality of the knowledgebase. For a fixed number of clauses,

the number of distinct variables that occur in the disjuncts of all these clauses determines the size of the focal graph. In figure 3.1 we see that as the clauses-to-variables ratio increases, the average focal graph size also increases because an increasing clauses-to-variables ratio for a fixed number of clauses implies a decreasing number of variables and this allows for a distribution of the variables amongst the clauses such that it is more likely for a literal to occur in a clause with its opposite occuring in another clause. In previous experiments [31] it was observed that for a clause knowledgebase consisting of 3-place clauses only, an increasing clauses-to-variables ratio in the range $[5, 10]$ ameliorates the performance of the system as it increases the probability of a pair of clauses ϕ, ψ from Δ being such that $|\mathsf{Preattacks}(\phi, \psi)| > 1$. This is because a ratio in this range makes the occurrence of a variable and its negation in the clauses of the set so frequent that it allows the $\mathsf{Attacks}$ relation to be defined only on a small set of clauses from the randomly generated clause knowledgebase. In the graph presented here though, this is less likely to happen as the clause knowledgebases tested involve both facts and rules.

Including literals (facts) in the knowledgebases used during the experiment makes the repeated occurrence of the same fact and its complement in the randomly generated clause knowledgebases, and hence in the subsequent focal graph, more frequent. It is for this reason that the curves with lower facts-to-rules proportion have a lower average focal graph (for each clauses-to-variables ratio). The symbol $\bullet$ on each of the curves indicates the highest possible clauses-to-variables ratio that would allow for a randomly generated clause knowledgebase consisting of the corresponding proportion of facts and rules to contain only distinct elements. Hence, for the data presented in this graph, the largest average focal graph of a randomly generated clause in a randomly generated clause knowledgebase of 600 distinct elements has the value 343.99 which corresponds to 57% of the initial knowledgebase. The values of the parameters with which this maximum is obtained correspond to a clauses-to-variables ratio equal to 4 on knowledgebases with a 1 to 1 proportion of facts-to-rules.

The average time for each repetition of the algorithm ranged from 6.3 seconds (for a facts-to-rules proportion of 1-3) to 13.8 seconds (for a facts-to-rules proportion of 3-1). So, the results show that for an inexpensive process we can substantially reduce the search space for arguments.

3.6 Discussion

In this chapter, I proposed the use of connection graphs as a way of ameliorating the computation cost by reducing the search space when searching for arguments for a formula from knowledgebases that contain formulae in conjunctive normal form. I provided theoretical results to ensure the correctness of the proposal, and provisional empirical results to indicate the potential advantages of the approach.

In the next chapter I explain how apart from reducing the search space for arguments, using the structures defined in this chapter can be useful for developing algorithms that produce arguments.

Chapter 4

Searching for arguments

In this chapter I present how we can use the structure of the query graph in order to search for arguments. By the results of chapter 3 we know that given a claim ψ and a knowledgebase Φ that contains formulae in CNF, we can reduce the cost of searching for arguments for ψ using $\mathsf{SupportBase}(\Phi, \psi)$ rather than Φ as the background knowledge.

So far the query graph has given the advantage of focusing the search on a part of the knowledgebase and thus reducing the search space for arguments but it has not provided a way for generating arguments. In order to decide whether there are any arguments that can be obtained from the set of formulae delineated by the query graph and to determine the support for these arguments, we need to search the query graph. The query graph contains information about how the formulae relate to each other in the sense of how they can potentially form proofs for the claim. In this chapter I present how we can exploit the structure of the query graph and apply a search strategy which, by traversing the graph, generates arguments. The language is restricted to the language of clauses $\mathcal{C}$ where, given a knowledgebase Δ of disjunctive clauses and a claim α that is a disjunctive clause, a depth first search takes place on the query graph of α in Δ. The steps of the search are represented by tree structures that are defined as 'proof trees'.

This chapter starts with a brief presentation of arguments in $\mathcal{C}$. It continues with definitions for proof trees and theoretical results concerning proof trees that motivate the use of this method for generating arguments. It closes with the algorithms that generate the proof trees defined in the chapter and experimental evaluation of these algorithms.

4.1 Arguments in $\mathcal{C}$

The proposal for generating arguments presented in this chapter is based on $\mathcal{C}$. The definition for an argument in $\mathcal{C}$ remains the same with the definition used so far and so the conditions of definition 2.2.3 do not change. The only restriction to that is that the claim for an argument is a disjunctive clause and the knowledgebase consists of disjunctive clauses. Similarly, the definitions for a counterargument, a canonical undercut and an argument tree also remain the same, with the restriction that the knowledgebases and the claims considered are from $\mathcal{C}$.

Although $\mathcal{C}$ is a restricted language in comparison to full propositional logic, it still has the strength

to represent complex information. It is well established that formulae with full syntax of propositional logic that involve implication and conjunction besides disjunction and negation can be rewritten in equivalence to formulae in a CNF. In addition, a set of disjunctive clauses that appear as conjuncts in a CNF can be regarded as equivalent to the formula in CNF. Therefore, although $\mathcal{C}$ lacks the symbols of implication and conjunction compared to full propositional logic, it is strong enough to represent equivalent information. The following examples illustrate how by rewriting a set of formulae into clausal form we can obtain equivalent arguments in $\mathcal{C}$.

Example 4.1.1 *For a knowledgebase* $\Delta = \{a, b, c, a \rightarrow \neg b \vee c, a \rightarrow \neg b, d, \neg d, \neg c, d \rightarrow a, d \rightarrow e\}$ *some arguments include:*

$$\langle\{a \rightarrow \neg b\}, \neg(a \wedge b)\rangle \qquad \langle\{a, a \rightarrow \neg b\}, \neg b \vee \neg c\rangle \qquad \langle\{a, a \rightarrow \neg b\}, \neg b\rangle$$
$$\langle\{\neg c\}, \neg c \vee b\rangle \qquad \langle\{\neg c\}, \neg c\rangle \qquad \langle\{a, b, a \rightarrow \neg b \vee c\}, c\rangle$$

Neither knowledgebase Δ *nor the claims of the arguments above are necessarily in* $\mathcal{C}$*. Let* $\Delta' = \{a, b, c, \neg a \vee \neg b \vee c, \neg a \vee \neg b, d, \neg d, \neg c, \neg d \vee a, \neg d \vee e\}$*. Then,* $\Delta' \subset \mathcal{C}$*. Observe that there is a one-to-one correspodence between the elements of* Δ' *and those of* Δ*, and there is a one-to-one correspodence between the arguments given above and their equivalent from* Δ' *given below.*

$$\langle\{\neg a \vee \neg b\}, \neg a \vee \neg b\rangle \qquad \langle\{a, \neg a \vee \neg b\}, \neg b \vee \neg c\rangle \qquad \langle\{a, \neg a \vee \neg b\}, \neg b\rangle$$
$$\langle\{\neg c\}, \neg c \vee b\rangle \qquad \langle\{\neg c\}, \neg c\rangle \qquad \langle\{a, b, \neg a \vee \neg b \vee c\}, c\rangle$$

Example 4.1.2 *Let* $\Delta = \{a, b, c, a \rightarrow \neg b \vee c, a \rightarrow \neg b, d, \neg d, \neg c, d \rightarrow a, d \rightarrow e\}$*. Then, the following is an argument tree for* $\alpha = c$*.*

$$\begin{array}{ccc}
 & \langle\{a, b, a \rightarrow \neg b \vee c\}, c\rangle & \\
 & / \qquad\qquad \backslash & \\
\langle\{a \rightarrow \neg b\}, \diamond\rangle & & \langle\{\neg c\}, \diamond\rangle \\
| & & | \\
\langle\{d, d \rightarrow a, b\}, \diamond\rangle & & \langle\{c\}, \diamond\rangle \\
/ \qquad \backslash & & \\
\langle\{a \rightarrow \neg b \vee c, \neg c\}, \diamond\rangle & \langle\{\neg d\}, \diamond\rangle &
\end{array}$$

Example 4.1.3 *Using the equivalent knowledgebase* Δ' *of* Δ *as described in example 4.1.1 we obtain the following argument tree for* $\alpha = c$ *where each node is logically equivalent to a node from the argument tree of example 4.1.2.*

$$
\begin{array}{ccc}
 & \langle\{a, b, \neg a \vee \neg b \vee c\}, d\rangle & \\
 & / \qquad\qquad \backslash & \\
\langle\{\neg a \vee \neg b\}, \diamond\rangle & & \langle\{\neg c\}, \diamond\rangle \\
| & & | \\
\langle\{d, \neg d \vee a, b\}, \diamond\rangle & & \langle\{c\}, \diamond\rangle \\
/ \quad \backslash & & \\
\langle\{\neg a \vee \neg b \vee c, \neg c\}, \diamond\rangle & \langle\{\neg d\}, \diamond\rangle &
\end{array}
$$

4.1.1 Properties of deductions and arguments in $\mathcal{C}$

Linear resolution is the basis for the proposal for searching for arguments presented in this chapter. There are some properties that follow from the definition of the linear resolution deduction. These properties are used in proofs throughout this thesis, so they are given in this section in the form of lemmas.

Lemma 4.1.1 *Let Ψ be a set of clauses and let $D \in \mathsf{Deductions}(\Psi)$. Then, for all $a \in \mathsf{Literals}(D)$ there is an $\alpha \in \Psi \cap D$ such that $a \in \mathsf{Disjuncts}(\alpha)$.*

Proof: Follows from condition (1) of the definition of a linear deduction. □

Lemma 4.1.2 *Let Ψ be a set of clauses and let $D \in \mathsf{Deductions}(\Psi)$ be such that $D = \{\delta_1, \ldots, \delta_n\}$. Then, for all $a \in \mathsf{Literals}(D) \setminus \mathsf{Disjuncts}(\delta_n)$ there is an $\alpha \in D \cap \Psi$ and an $\alpha' \in D \cap \Psi$ such that $\mathsf{Attacks}(\alpha, \alpha') = a$.*

Proof: Follows from condition (2) of definition 3.2.3. □

Lemma 4.1.3 *Let $\langle\Phi, \alpha\rangle$ be an argument where Φ is a set of clauses and $\alpha = a_1 \vee \ldots \vee a_n$ is a clause. Then, there is at least one $a_j \in \mathsf{Disjuncts}(\alpha)$ such that there is a $\beta \in \Phi$ with $a_j \in \mathsf{Disjuncts}(\beta)$.*

Proof: $\Phi \vdash \alpha$ and so there is a linear deduction D for α from Φ. Then by lemma 4.1.1, there is a $a_j \in \mathsf{Disjuncts}(\alpha)$ such that there is a $\beta \in \Phi$ with $a_j \in \mathsf{Disjuncts}(\beta)$. □

Lemma 4.1.4 *Let Φ be a minimal inconsistent set of clauses. Then for all $\psi \in \Phi$, for all $a \in \mathsf{Disjuncts}(\psi)$ there is a $\Phi' \subset \Phi$ such that $\langle\Phi', \overline{a}\rangle$ is an argument. For this Φ' it holds that $\Phi' \subseteq \Phi \setminus \{\psi\}$.*

Proof: Let Φ be a minimal inconsistent set of clauses. Let for a clause ψ, $\overline{\psi}$ be the CNF of $\neg\psi$. Then, for all $\langle\Phi \setminus \{\psi\}, \overline{\psi}\rangle$ is an argument hence $\Phi \setminus \{\psi\}$ is consistent and $\Phi \setminus \{\psi\} \vdash \overline{\psi}$. Let for $\psi \in \Phi$, $a \in \mathsf{Disjuncts}(\psi)$. Then $\overline{a} \in \mathsf{Conjuncts}(\overline{\psi})$ so either $\langle\Phi \setminus \{\psi\}, \overline{a}\rangle$ is an argument or there is a subset Φ' of $\Phi \setminus \{\psi\}$ that is minimal for entailing $\overline{a}$ and is also consistent hence $\langle\Phi', \overline{a}\rangle$ is an argument. □

Lemma 4.1.5 *Let $\langle\Phi, \alpha\rangle$ be an argument. Then, there is no $\beta \in \Phi$ such that β is a tautology.*

Proof: Let $\langle\Phi, \alpha\rangle$ be an argument and let $\beta \in \Phi$ be a tautology. Then $\Phi^+ \equiv \Phi \cup \{\neg\alpha\}$ is a minimal inconsistent set and also $\beta \in \Phi^+$ and $\beta \neq \neg\alpha$ because β is a clause consisting of a disjunction of more than one literal while $\neg\alpha$ is a conjuction of one or more literals. So, $\Phi^+ \setminus \{\beta\} \subset \Phi^+$ and $\Phi^+ \setminus \{\beta\} \vdash \neg\beta \vdash \bot$, hence Φ^+ is not a minimal inconsistent set which contradicts the assumption that $\langle\Phi, \alpha\rangle$ is an argument. □

Lemma 4.1.6 *If Φ is a minimal inconsistent set of clauses then for all $\phi \in \Phi$, for all $a \in \mathsf{Disjuncts}(\phi)$, there is a $\phi' \in \Phi$ with $\overline{a} \in \mathsf{Disjuncts}(\phi)$.*

Proof: Let Φ be a minimal inconsistent set of clauses and let $\phi \in \Phi$ be such that $a \in \mathsf{Disjuncts}(\phi)$. To give a proof by contradiction suppose there is no $\phi' \in \Phi$ with $\overline{a} \in \mathsf{Disjuncts}(\phi')$. By lemma 4.1.5, $\overline{a} \notin \mathsf{Disjuncts}(\phi)$. Let $\overline{\phi}$ be the CNF of $\neg\phi$. Then, $\langle \Phi \setminus \{\phi\}, \overline{\phi} \rangle$ is an argument and $\overline{a} \in \mathsf{Conjuncts}(\overline{\phi})$, so $\Phi \setminus \{\phi\} \vdash \overline{a}$. If $\Phi' \subseteq \Phi \setminus \{\phi\}$ is a minimal set that entails $\overline{a}$, then $\langle \Phi', \overline{a} \rangle$ is an argument, but there is no $\phi' \in \Phi'$ such that $\overline{a} \in \mathsf{Disjuncts}(\phi')$ which contradicts lemma 4.1.3. □

Corollary 4.1.1 *If Φ is a minimal inconsistent set of clauses then for all $a \in \mathsf{Literals}(\Phi)$ there is a $\Phi' \subset \Phi$ such that $\langle \Phi', a \rangle$ is an argument and a $\Phi'' \subset \Phi$ such that $\langle \Phi'', \overline{a} \rangle$ is an argument.*

Proof: Follows from lemmas 4.1.4 and 4.1.6. □

4.1.2 The Supportbase in $\mathcal{C}$

Before I turn to introducing mechanisms for searching for arguments I present some simplifications in the theory of chapter 3 when this is applied on $\mathcal{C}$. These simplifications have to do with the fact that for a set of clauses Δ it holds that $\mathsf{SetConjuncts}(\Delta) = \Delta$. Therefore, if we seek arguments for a clause α from Δ, the query graph of α in Δ contains directly elements from Δ as its nodes. This simplifies the definitions of SubFocus, Zone and SupportBase.

According to the following lemma, for a set of clauses Δ and a clause δ_i, $\mathsf{SubFocus}(\Delta, \delta_i)$ and $\mathsf{Focal}(\Delta, \delta_i)$ are identical.

Lemma 4.1.7 *Let Δ be a set of clauses and δ_i be a clause. Then,* $\mathsf{SubFocus}(\Delta, \delta_i) = \mathsf{Focal}(\Delta, \delta_i)$.

Proof: Let Δ be a set of clauses and δ_i be a clause. Clause δ_i can be regarded as a formula in conjunctive normal form that consists of a unique conjunct. Then, by the definition for function Conjuncts follows that $\mathsf{Conjuncts}(\delta_i) = \{\delta_i\}$. Moreover, for $\Delta = \{\delta_1, \ldots, \delta_n\}$ it holds that $\mathsf{SetConjuncts}(\Delta) = \{\delta_1, \ldots, \delta_n\} = \Delta$. Then, for some $\delta_i \in \Delta$, and for $\delta_i \in \mathsf{Conjuncts}(\delta_i)$, $\mathsf{SubFocus}(\Delta, \delta_i) = \mathsf{Focal}(\mathsf{SetConjuncts}(\Delta), \delta_i) = \mathsf{Focal}(\Delta, \delta_i)$. □

Moreover, according to the following lemma, for a claim $\alpha \in \mathcal{C}$ and a knowledgebase $\Delta \subset \mathcal{C}$, the subset of Δ that is associated to $\mathsf{Zone}(\Delta, a_i)$ for some $\overline{a}_i \in \mathsf{Conjuncts}(\overline{\alpha})$ contains elements directly from $\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i)$.

Lemma 4.1.8 *Let Δ be a set of clauses and α be a clause. Then, for all $a_i \in \mathsf{Conjuncts}(\overline{\alpha})$,* $\mathsf{Zone}(\Delta, a_i) = \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i)) \cap \Delta$.

Proof: Let Δ be a set of clauses, α be a clause and $a_i \in \mathsf{Conjuncts}(\overline{\alpha})$. Then, by the definition for zone, $\mathsf{Zone}(\Delta, a_i) = \{\delta \in \Delta \mid \mathsf{Conjuncts}(\delta) \cap \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i)) \neq \emptyset\}$. Because for all $\delta \in \Delta$, $\mathsf{Conjuncts}(\delta) = \{\delta\}$, then $\mathsf{Conjuncts}(\delta) \cap \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i)) \neq \emptyset$ iff $\delta \in \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i))$ i.e. for all $\delta \in \Delta$, $\delta \in \mathsf{Zone}(\Delta, a_i)$ iff $\delta \in \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup$

$\{\overline{\alpha}\}, a_i))$ from which follows that $\mathsf{Zone}(\Delta, a_i) = \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i)) \cap \Delta$. □

Essentially, the last lemma suggests that for all $a_i \in \mathsf{Conjuncts}(\overline{\alpha})$, the nodes of $\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i))$ that are from Δ are the ones $\mathsf{Zone}(\Delta, a_i)$ consists of. The only nodes of $\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i))$ that may not be from Δ are the conjuncts a_j of $\overline{\alpha}$ for which it holds that $a_j \in \mathsf{Conjuncts}(\overline{\alpha})$ and $a_j \notin \Delta$. From the way SubFocus is defined, for all the other nodes δ_k that are such that $\delta_k \notin \mathsf{Conjuncts}(\overline{\alpha})$, it holds that $\delta_k \in \Delta$. As a consequence of lemma 4.1.8, the next lemma follows.

Lemma 4.1.9 *Let Δ be a set of clauses and α be a clause. Then,* $\mathsf{SupportBase}(\Delta, \alpha) = \mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha)) \cap \Delta$.

Proof: Let Δ be a set of clauses and α be a clause. By the definition of the query graph follows

(1) $\mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha)) = \bigcup_{a_i \in \mathsf{Conjuncts}(\overline{\alpha})} \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i))$.

By lemma 4.1.8, follows

(2) for all $a_i \in \mathsf{Conjuncts}(\overline{\alpha})$, $\mathsf{Zone}(\Delta, a_i) = \mathsf{Nodes}(\mathsf{SubFocus}(\Delta \cup \{\overline{\alpha}\}, a_i) \cap \Delta$

From (1) and (2) follows that $\mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha)) \cap \Delta = \bigcup_{a_i \in \mathsf{Conjuncts}(\overline{\alpha})} (\mathsf{Zone}(\Delta, a_i))$ and since by definition $\mathsf{SupportBase}(\Delta, \alpha) = \bigcup_{a_i \in \mathsf{Conjuncts}(\overline{\alpha})} (\mathsf{Zone}(\Delta, a_i))$, it follows that $\mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha)) \cap \Delta = \mathsf{SupportBase}(\Delta, \alpha)$. □

From the last proposition and taking into account previous theoretical results concerning the query graph we obtain the following corollary.

Corollary 4.1.2 *Let Δ be a set of clauses and α be a clause. If $\langle \Phi, \alpha \rangle$ is an argument for α from Δ, then $\Phi \subseteq \mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha))$.*

Proof: By corollary 3.3.1, if $\langle \Phi, \alpha \rangle$ is an argument for α from Δ then $\Phi \subseteq \mathsf{SupportBase}(\Delta, \alpha)$. By lemma 4.1.9, $\mathsf{SupportBase}(\Delta, \alpha) = \mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha)) \cap \Delta$ so from these two relations follows that $\Phi \subseteq \mathsf{Nodes}(\mathsf{Query}(\Delta, \alpha))$. □

According to the last corollary, for a clause knowledgebase Δ and a claim α that is a clause, the nodes of the query graph of α in Δ is the set of clauses that contains all the arguments for α. The topic of the next section is how we can use the query graph of α in Δ in order to retrieve arguments for α.

Tautologies constitute a particular case which is unconventional since as the following proposition suggests the only support for an argument for a tautology is the empty set.

Proposition 4.1.1 *Let α be a tautology. Then, $\langle \Phi, \alpha \rangle$ is an argument iff $\Phi = \emptyset$.*

Proof: Let $\langle \Phi, \alpha \rangle$ be an argument where α is a tautology. Then $\Phi^+ \equiv \Phi \cup \{\neg\alpha\}$ is a minimal inconsistent set, but $\neg\alpha \vdash \bot$, so $\{\neg\alpha\}$ is a minimal inconsistent set and hence $\Phi^+ = \Phi \cup \{\neg\alpha\} = \{\neg\alpha\}$. Also, $\neg\alpha \notin \Phi$ because Φ is consistent since it is a support for an argument and $\neg\alpha \vdash \bot$, so $\Phi \cap \{\neg\alpha\} = \emptyset$

and $\Phi \cup \{\neg\alpha\} = \{\neg\alpha\}$ holds iff $\Phi = \emptyset$. □

Using methods based on refutation when looking for a proof Φ for a clause α requires refuting α and checking whether $\Phi \cup \{\neg\alpha\}$ is unsatisfiable. If we seek a proof for a clause α that is a tautology, the fact that $\neg\alpha$ itself is a contradiction raises complications on deciding whether a set Φ is unsatisfiable together with $\neg\alpha$ because it is the case where Φ actually entails α. In addition, for resolution-refutation based theorem proving methods where the starting point of the search is $\neg\alpha$, if an algorithm does not return a solution then it is hard to distinguish whether this is because no solution exists or because the solution is the empty set. In order to avoid this situation, from now on tautologies will not be considered as claims for arguments.

4.2 Definitions for proof trees for arguments

As stated in the last section, the set of nodes of the query graph of a clause α in a clause knowledgebase Δ contains all the clauses that may be premises in an argument for α. The query graph provides a subset of the original knowledgebase that contains all the arguments for α. In order to generate these arguments we can apply search algorithms based on the structure of the query graph. Apart from providing a reduced search space, the query graph contains information on how the clauses that represent its nodes relate to each other motivating the implementation of algorithms that search for arguments by following the paths of the graph. This is the subject of this section that introduces tree structures that represent the steps of the search of the query graph.

4.2.1 The presupport tree

The search strategies presented in this chapter are based on the idea of a proof by refutation using resolution. From now on, in this chapter only knowledgebases that contain elements from $\mathcal{C}$ and claims that are also from $\mathcal{C}$ are considered. Also, for simplicity no tautologies are considered for claims.

Assume we want to find arguments for α from Δ. Each of the components of the query graph of α in Δ is identified by the complement $\overline{a}$ of one of the disjuncts of α, and contains $\overline{a}$ as a node. Walking over the query graph according to certain conditions with $\overline{a}$ as the starting point, can lead to a proof for α. We can represent the steps of this walk by tree structures. The notion of a presupport tree defined in this section represents the steps of a tentative proof for α constructed in this way. It has the complement of one of the disjuncts of α representing its root node, which is also one of the epicentres of the focal graphs that compose the query graph of α in Δ. The non-root nodes have clauses from the query graph of α in Δ assigned. Essentially a presupport tree is constructed from a subgraph of $\mathsf{Query}(\Delta, \alpha)$. A presupport tree does not necessarily relate to a proof for a claim. It forms the basis for more refined definitions that can be more meaningful in the search for arguments for α from Δ.

Definition 4.2.1 *Let Δ be a clause knowledgebase and let $\alpha = a_1 \vee \ldots \vee a_n$ be a clause that is not a tautology. A* **presupport tree** *for Δ, α and $a_k \in \mathsf{Disjuncts}(\alpha)$ is a tuple (N, A, f) where (N, A, f) is a*

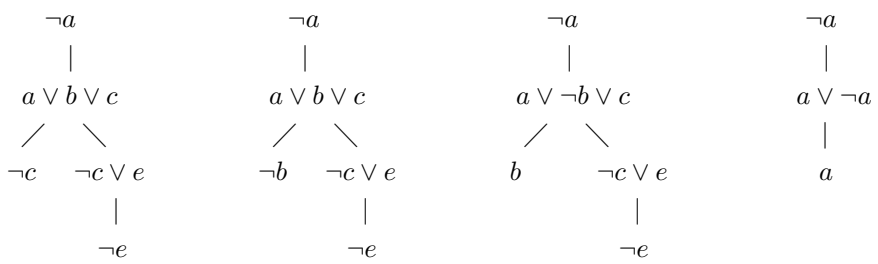
tree, and f is a mapping from N to $\Delta \cup \{\overline{a_k}\}$ such that

(1) *if x is the root of the tree, then $f(x) = \overline{a_k}$ and*
there is exactly one child y of x s.t. $\mathsf{Attacks}(f(y), f(x)) = a_k$,

(2) *for any nodes x, y in the same branch, if $x \neq y$, then $f(x) \neq f(y)$*

(3) *for any nodes x, y in the same branch, if x is the parent of y,*
then $\mathsf{Attacks}(f(x), f(y)) \neq null$

Example 4.2.1 *Some presupport trees for $\Delta = \{a \vee b \vee c, a \vee \neg b \vee c, \neg c, b, \neg c \vee e, \neg e, \neg a \vee c, \neg b, a \vee \neg a, a, a \vee c \vee \neg c, c\}$, $\alpha = a \vee b \vee d$ and a are:*



In the definition for a presupport tree, function f assigns to the set of nodes N a clause from $\Delta \cup \{\overline{a}\}$. The first condition of definition 4.2.1 sets the complement $\overline{a}$ of one of the disjuncts of α as the starting point for the search by placing it at the root of the tree. Condition (2) of the definition prevents repetitions of the same clause on a branch, ensuring this way that infinite branches are avoided in the search. Finally, condition (3) indicates that the search takes place by following the links in the query graph and an arc (x, x') in the presupport tree can exist if there is an arc (ϕ, ϕ') in $\mathsf{Query}(\Delta, \alpha)$ such that $\phi = f(x)$ and $\phi' = f(x')$.

The next definition introduces some functions that are used to refer to the elements of a presupport tree (N, A, f).

Definition 4.2.2 *Let (N, A, f) be a presupport tree for Δ, α and some $a \in \mathsf{Disjuncts}(\alpha)$. Then for a node $x \in N$,* $\mathsf{Ancestors}(x)$ *is the set of ancestors of x in (N, A, f).* $\mathsf{AncestorLabels}(x)$ *is the set of literals that define the arcs between the ancestors of x through the* Attacks *function where the sign of each literal is defined from the child to the parent:* $\mathsf{AncestorLabels}(x) = \{\mathsf{Attacks}(f(w), f(w')) \mid w' \in \mathsf{Ancestors}(x)$ *and* $w \in \mathsf{Ancestors}(x) \cup \{x\}\}$. $\mathsf{Subtree}(x)$ *is the set of successors of x in (N, A, f), together with x, and* $\mathsf{Children}(x)$ *is the set of children nodes of x:* $\mathsf{Children}(x) = \{y \mid (x, y) \in A$ *and* $y \in \mathsf{Subtree}(x)\}$. *A branch is a set of nodes X connected through a sequence of arcs starting from the root node and ending on a leaf node.*

Example 4.2.2 *Let (N, A, f) be the first tree of example 4.2.1. Let $x \in N$ be the node for which $f(x) = \neg c \vee e$. Then,* $\mathsf{Ancestors}(x) = \{z, w\}$ *with $f(z) = a \vee b \vee c$ and $f(w) = \neg a$,* $\mathsf{AncestorLabels}(x) = \{\mathsf{Attacks}(\neg c \vee e, a \vee b \vee c), \mathsf{Attacks}(a \vee b \vee c, \neg a)\} = \{\neg c, a\}$, *and if y is the node such that $f(y) = \neg e$,*

$\mathsf{Subtree}(x) = \{x, y\}$ *and* $\mathsf{Children}(x) = \{y\}$.

The presupport tree does not necessarily provide a proof for α but it provides the basis for a search tree structure which, with additional constraints does provide a proof for α. The next section introduces a special category of presupport tree, the complete presupport tree which is a presupport tree with some particular properties.

4.2.2 The complete presupport tree

If additional restrictions are applied on the way the nodes are arranged on the presupport tree then we get a category of presupport tree that provides a proof for α. This is the complete presupport tree defined below.

Definition 4.2.3 *Let Δ be a clause knowledgebase and let $\alpha = a_1 \vee \ldots \vee a_n$ be a clause. A* **complete presupport tree** *for Δ, α and $a_k \in \mathsf{Disjuncts}(\alpha)$ is a presupport tree (N, A, f) for Δ, α and a_k such that for any non-root $x \in N$, for every $b \in \mathsf{Disjuncts}(f(x))$ exactly one of the following conditions hold:*

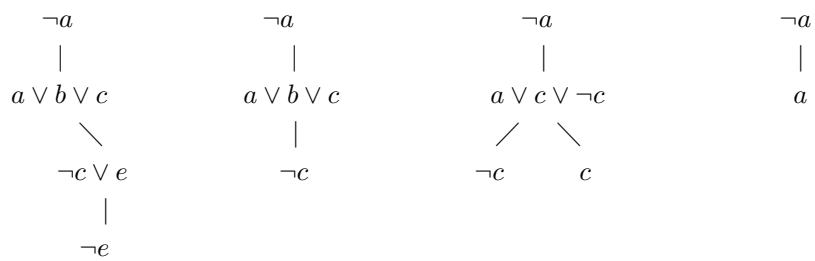
$i)$ $b \in \mathsf{Disjuncts}(\alpha) \setminus \{a_k\}$

$ii)$ *or there is exactly one arc (y, y') where $y' \in \mathsf{Ancestors}(x)$ such that*
$\mathsf{Attacks}(f(y), f(y')) = b$

$iii)$ *or there is exactly one $y \in \mathsf{Children}(x)$ s.t.* $\mathsf{Attacks}(f(y), f(x)) = \overline{b}$

Example 4.2.3 *None of the presupport trees of example 4.2.1 is a complete presupport tree for Δ, α and a. In the first tree, for x with $f(x) = a \vee b \vee c$ and $c \in \mathsf{Disjuncts}(f(x))$ none of the conditions of definition 4.2.3 is satisfied. The second presupport tree is not complete because for node y with $f(y) = a \vee b \vee c$ and $b \in \mathsf{Disjuncts}(f(y))$ both conditions i) and iii) of definition 4.2.3 hold. The third presupport tree is not complete because for node z with $f(z) = b$ and $b \in \mathsf{Disjuncts}(f(z))$ both conditions i) and ii) of definition 4.2.3 hold. The last presupport tree is not complete because for w, with $f(w) = a$ and $a \in \mathsf{Disjuncts}(f(w))$, none of the conditions of the definition holds.*

Continuing example 4.2.1, some complete presupport trees for Δ, α and a are:



With the conditions of the last definition for a complete presupport tree, a unit clause consisting of a unique disjunct can represent a node of the tree only as the root or a leaf node. Moreover for a unit clause α, for a presupport tree for Δ, α and a where $\alpha = a$ conditions $ii)$ and $iii)$ of the above definition are sufficient to provide a complete presupport tree for Δ, α and a.

The idea in building a complete presupport tree (N, A, f) is that in this way the set of clauses produced $(\{f(x) \mid x \in N\})$ is such that for all $x \in N$ and for all $b \in \mathsf{Disjuncts}(f(x)) \setminus \mathsf{Disjuncts}(\alpha)$ there is a $y \in N$ such that $\overline{b} \in \mathsf{Disjuncts}(f(y))$ and the set $\{f(x) \mid x \in N\} \cup \{\neg\alpha\}$ is inconsistent. Apart from ensuring that for all the disjuncts of all $f(z)$ in the set there is a clause $f(y)$ containing their complement, the conditions of definition 4.2.3 help controlling the size of the tree. The fact that a branch is expanded below a node z by adding a node y such that $\mathsf{Attacks}(f(y), f(z)) = \overline{b}$ only when b does not appear in $\mathsf{AncestorLabels}(z)$, controls the length of the branches of the tree, and means that since this disjunct has been already dealt with by another node on the branch, no additional nodes need to be added on the branch to deal with b. The fact that exactly one child of z is added per each such disjunct of $f(z)$ controls the width of the tree and means that every child of z deals with exactly one disjunct of $f(z)$ that has not been considered earlier on the branch.

From the way it is defined, the complete presupport tree has some properties that relate to its structure and are formalised in the next three propositions. These are used later in the chapter when showing how the complete presupport tree is associated with the search for arguments.

Proposition 4.2.1 suggests that apart from the root node of a complete presupport tree for Δ, α and a, no other complement of the disjuncts of α appears in the clauses that represent the tree.

Proposition 4.2.1 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then, there is no $\overline{b} \in \mathsf{Literals}(\{f(x) \mid x \in N\} \setminus \{\overline{a}\})$ such that $b \in \mathsf{Disjuncts}(\alpha)$.*

Proof: Let (N, A, f) be a complete presupport tree for Δ, α and a. If for some $x \in N$, $\overline{b} \in \mathsf{Disjuncts}(f(x))$ is such that $b \in \mathsf{Disjuncts}(\alpha)$ then for $\overline{b} \in \mathsf{Disjuncts}(f(x))$ exactly one of conditions $i)$, $ii)$ and $iii)$ of definition 4.2.3 holds.
$i)$ cannot hold for $\overline{b} \in \mathsf{Disjuncts}(f(x))$ because if it did then according to the condition, $\overline{b} \in \mathsf{Disjuncts}(\alpha)$ and also by assumption $b \in \mathsf{Disjuncts}(\alpha)$ and this contradicts the assumption that α is not a tautology. If condition $ii)$ holds for $\overline{b} \in \mathsf{Disjuncts}(f(x))$ then $\overline{b} \in \mathsf{AncestorLabels}(x)$ and so there is an arc $(w, w') \in A$ where $w' \in \mathsf{Ancestors}(x)$ such that $\mathsf{Attacks}(f(w), f(w')) = \overline{b}$. Then, for w', and $b \in \mathsf{Disjuncts}(f(w'))$ it holds that there is a child w if w' such that $\mathsf{Attacks}(f(w), f(w')) = \overline{b}$ and also that $b \in \mathsf{Disjuncts}(\alpha)$ so both conditions $i)$ and $iii)$ of definition 4.2.3 hold for $b \in \mathsf{Disjuncts}(f(w'))$ which contradicts the assumption that (N, A, f) is a complete presupport tree for Δ, α and a. If condition $iii)$ holds for $\overline{b} \in \mathsf{Disjuncts}(f(x))$ then there is a child x' of x such that $\mathsf{Attacks}(f(x'), f(x)) = b$ and then for x' and $b \in \mathsf{Disjuncts}(f(x'))$ both conditions $i)$ and $ii)$ of definition 4.2.3 hold which contradicts the assumption that (N, A, f) is a complete presupport tree for Δ, α and a. $\square$

According to the next proposition a literal and its complement cannot label arcs on the same branch of a complete presupport tree.

Proposition 4.2.2 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then, there is no $x \in N$ s.t. $b \in \mathsf{AncestorLabels}(x)$ and $\overline{b} \in \mathsf{AncestorLabels}(x)$.*

Proof: For a proof by contradiction let literal b and its complement $\overline{b}$ be such that there are arcs (w, w'), (y, y') on the same branch of (N, A, f) where w' is the parent of w and y' is the parent of y such

that $\mathsf{Attacks}(f(w), f(w')) = b$ and $\mathsf{Attacks}(f(y), f(y')) = \overline{b}$. It is shown that under these conditions (N, A, f) cannot satisfy the conditions for a complete presupport tree. Assume that $w \in \mathsf{Ancestors}(y)$. If w' is the root then $\overline{b} = \overline{a}$ and $b \in \mathsf{Disjuncts}(\alpha)$ and so for $b \in \mathsf{Disjuncts}(f(y'))$, since there is a child y such that $\mathsf{Attacks}(f(y), f(y')) = \overline{b}$ both conditions $i)$ and $iii)$ of definition 4.2.3 hold and so (N, A, f) is not complete. If w' is not the root then $b \notin \mathsf{Disjuncts}(\alpha)$ and so condition $iii)$ of definition 4.2.3 holds for $\overline{b} \in \mathsf{Disjuncts}(f(w'))$ and there is a child w of w' such that $\mathsf{Attacks}(f(w), f(w')) = b$. Then, for $b \in \mathsf{Disjuncts}(f(y'))$ it holds that there is a $w' \in \mathsf{Ancestors}(y')$ such that $\mathsf{Attacks}(f(w), f(w')) = b$ and also a child y such that $\mathsf{Attacks}(f(y), f(y')) = \overline{b}$ and so both conditions $ii)$ and $iii)$ of definition 4.2.3 hold for $b \in \mathsf{Disjuncts}(f(y'))$ and so (N, A, f) is not complete. □

According to the next proposition, for a node x of a complete presupport tree (N, A, f), a literal from $\mathsf{Disjuncts}(f(x))$ cannot have its complement in $\mathsf{AncestorLabels}(x)$.

Proposition 4.2.3 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then, there is no $x \in N$ s.t. $b \in \mathsf{Disjuncts}(f(x))$ and $\overline{b} \in \mathsf{AncestorLabels}(x)$.*

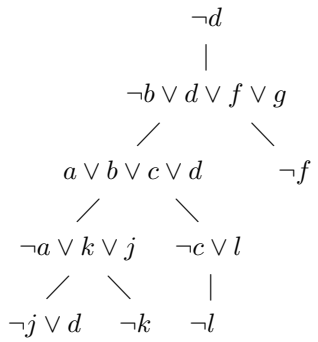
Proof: Let (N, A, f) be a complete presupport tree for Δ, α and a and let $x \in N$ be such that $b \in \mathsf{Disjuncts}(f(x))$ and $\overline{b} \in \mathsf{AncestorLabels}(x)$. Then for $b \in \mathsf{Disjuncts}(f(x))$ at least one of the conditions of definition 4.2.3 holds. By proposition 4.2.1, $b \notin \mathsf{Disjuncts}(\alpha)$ and so condition $i)$ cannot hold for $b \in \mathsf{Disjuncts}(f(x))$. Then for $b \in \mathsf{Disjuncts}(f(x))$ one of conditions $ii)$ or $iii)$ of definition 4.2.3 holds. Condition $ii)$ cannot hold for $b \in \mathsf{Disjuncts}(f(x))$ because then $b \in \mathsf{AncestorLabels}(x)$ and $\overline{b} \in \mathsf{AncestorLabels}(x)$ which contadicts proposition 4.2.2. Condition $iii)$ cannot hold either because then there is a child x' of x such that $\mathsf{Attacks}(f(x'), f(x)) = \overline{b}$ and for x' there is more than one $w \in \mathsf{Ancestors}(x')$ such that $\mathsf{Attacks}(f(w), f(w')) = \overline{b}$. Hence, no condition of definition 4.2.3 can hold for $\overline{b} \in \mathsf{Disjuncts}(f(x'))$ when there is an $x \in N$ such that $b \in \mathsf{Disjuncts}(f(x))$ and $\overline{b} \in \mathsf{AncestorLabels}(x)$ and this contradicts the assumption that (N, A, f) is a complete presupport tree for Δ, α and a. □

There are some theoretical results associated with the definition of a complete presupport tree, concerning entailment of the claim α. Function $\mathsf{SubtreeRes}(z)$ defined below is fundamental for showing how a complete presupport tree provides a set of clauses that entails α.

Definition 4.2.4 *Let (N, A, f) be a complete presupport tree for Δ, α and a. For all $z \in N$, if B is the set of clauses that corresponds to the clauses that represent $\mathsf{Subtree}(z)$ (i.e. $B = \{f(w) \mid w \in \mathsf{Subtree}(z)\}$), then*

$$\mathsf{SubtreeRes}(z) = \bigvee \left\{ \left(\mathsf{Literals}(B) \setminus \{\mathsf{Attacks}(f(w), f(w')) \mid w, w' \in \mathsf{Subtree}(z)\} \right) \right\}$$

Example 4.2.4 *The following is a complete presupport tree for $\Delta = \{\neg b \vee d \vee f \vee g, a \vee b \vee c \vee d, \neg a \vee k \vee j, \neg j \vee d, \neg k, \neg c \vee l, \neg l, \neg f, \neg d \vee b \vee g, \neg g \vee b, \neg b, \neg d \vee \neg j, j, \neg g, c \vee l\}$, $\alpha = d \vee m \vee g$ and d.*



For the subtree rooted at z with $f(z) = a \vee b \vee c \vee d$, let $B = \{f(x) \mid x \in \mathsf{Subtree}(z)\} = \{a \vee b \vee c \vee d, \neg a \vee k \vee j, \neg c \vee l, \neg j \vee d, \neg k, \neg l\}$. Then, $\mathsf{Literals}(B) = \{a, b, c, d, \neg a, k, j, \neg c, l, \neg j, \neg k, \neg l\}$, $\{\mathsf{Attacks}(f(w), f(w')) \mid w, w' \in \mathsf{Subtree}(z)\} = \{\neg a, a, \neg c, c, \neg j, j, \neg k, k, \neg l, l\}$ *and* $\mathsf{Literals}(B) \setminus \{\mathsf{Attacks}(f(w), f(w')) \mid w, w' \in \mathsf{Subtree}(z)\} = \{b, d\}$ *so by definition 4.2.4,* $\mathsf{SubtreeRes}(z) = \bigvee\{b, d\} = b \vee d$.

Proposition 4.2.4 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then for a node $z \in N$, where z' is the parent of z,* $\mathsf{Attacks}(f(z), f(z')) = \mathsf{Attacks}(\mathsf{SubtreeRes}(z), f(z'))$.

Proof: Let (N, A, f) be a complete presupport tree for Δ, α and a. From the way $\mathsf{SubtreeRes}$ is defined follows that for all x, x' where x' is the parent of x, $\mathsf{Attacks}(f(x), f(x')) \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(x))$. Then, in order to prove that $\mathsf{Attacks}(f(x), f(x')) = \mathsf{Attacks}(\mathsf{SubtreeRes}(x), f(x'))$ it is sufficient to prove that it cannot hold that $|\mathsf{Preattacks}(\mathsf{SubtreeRes}(x), f(x'))| > 1$. To give a proof by contradiction assume that (N, A, f) is a complete presupport tree for Δ, α and a and for an arc $(x, x') \in A$ which is such that $\mathsf{Attacks}(f(x), f(x')) = b$, it holds that $|\mathsf{Preattacks}(\mathsf{SubtreeRes}(x), f(x'))| > 1$. Then, $b \in \mathsf{Preattacks}(\mathsf{SubtreeRes}(x), f(x'))$ and there is a $c \neq b$ for which it holds that $c \in \mathsf{Preattacks}(\mathsf{SubtreeRes}(x), f(x'))$. Since $c \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(x))$, there is a $w \in \mathsf{Subtree}(x)$ such that $c \in \mathsf{Disjuncts}(f(w))$ and there is no arc (w, w') where in $w, w' \in \mathsf{Subtree}(x)$ such that $\mathsf{Attacks}(f(w), f(w')) = c$ or $\mathsf{Attacks}(f(w'), f(w)) = \overline{c}$. Because (N, A, f) is a complete presupport tree, for $c \in \mathsf{Disjuncts}(f(w))$ either $c \in \mathsf{AncestorLabels}(w)$ or $c \in \mathsf{Disjuncts}(\alpha)$ holds. If $c \in \mathsf{AncestorLabels}(w)$ then it holds that $c \in \mathsf{AncestorLabels}(x')$ because c does not label any arc in $\mathsf{Subtree}(x')$ and if $c \in \mathsf{AncestorLabels}(x')$ holds it contradicts proposition 4.2.3 because $\overline{c} \in \mathsf{Disjuncts}(f(x'))$. Hence, $c \in \mathsf{AncestorLabels}(w)$ does not hold and so $c \in \mathsf{Disjuncts}(\alpha)$ must hold in order for (N, A, f) to be complete. If x' is not the root node of (N, A, f) then since $\overline{c} \in \mathsf{Disjuncts}(f(x'))$ it holds that $\overline{c} \in \mathsf{Literals}(\{f(x) \mid x \in N\} \setminus \{\overline{a}\})$ and if $c \in \mathsf{Disjuncts}(\alpha)$ then this contradicts proposition 4.2.1. If x' is the root then $f(x')$ is a literal and so it cannot hold that $|\mathsf{Preattacks}(\mathsf{SubtreeRes}(x), f(x'))| > 1$. Hence, w with $c \in \mathsf{Disjuncts}(f(w))$ does not satisfy any of the conditions of definition 4.2.3 which contradicts the assumption that (N, A, f) is a complete presupport tree. So it cannot hold that $|\mathsf{Preattacks}(\mathsf{SubtreeRes}(x), f(x'))| > 1$ and so it follows that $\mathsf{Attacks}(f(x), f(x')) = \mathsf{Attacks}(\mathsf{SubtreeRes}(x), f(x'))$. $\square$

Essentially, for a node z, $\mathsf{SubtreeRes}(z)$ gives a formula at z that is obtained by resolving the formula $f(z)$ with $\mathsf{SubtreeRes}(x_1), \ldots, \mathsf{SubtreeRes}(x_n)$ where $x_1, \ldots, x_n$ are the children of z. In this way, SubtreeRes is used to propagate resolution up the tree.

Proposition 4.2.5 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then for a node $z \in N$ with $\mathsf{Children}(z) = \{x_1, \ldots, x_n\}$,*

$$\mathsf{SubtreeRes}(z) = f(z) \bullet \mathsf{SubtreeRes}(x_1) \bullet \ldots \bullet \mathsf{SubtreeRes}(x_n)$$

Proof: Let (N, A, f) be a complete presupport tree for Δ, α and a and let $z \in N$. Then for every $v_i \in \mathsf{Children}(z)$ it holds that $\mathsf{Attacks}(f(v_i), f(z)) = b_i$ for some $b_i \in \mathsf{Disjuncts}(f(v_i))$ and because of the constraints for a complete presupport tree there is no other arc (w, w') on the branch where v_i belongs such that $\mathsf{Attacks}(f(w), f(w')) = b_i$ or $\mathsf{Attacks}(f(w), f(w')) = \overline{b}_i$ Hence, for neither b_i nor $\overline{b}_i$ are there any $w, w' \in \mathsf{Subtree}(v_i)$ such that $\mathsf{Attacks}(f(w), f(w')) = b_i$ or $\mathsf{Attacks}(f(w), f(w')) = \overline{b}_i$ and so by the definition of $\mathsf{SubtreeRes}(v_i)$, $b_i \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(v_i))$ for all $v_i \in \mathsf{Children}(z)$. Then for $b_i = \mathsf{Attacks}(f(v_i), f(z))$ for all $v_i \in \mathsf{Children}(z)$, $\mathsf{SubtreeRes}(v_1) \bullet \ldots \bullet \mathsf{SubtreeRes}(v_j) \bullet f(z)$ can be re-written to:

$$\bigvee \bigcup_{v_i} \big((\mathsf{Disjuncts}(\mathsf{SubtreeRes}(v_i)) \cup \mathsf{Disjuncts}(f(z))) \setminus \{b_i, \overline{b}_i\}\big)$$

which for $B_i = \{f(w_i) \mid w_i \in \mathsf{Subtree}(v_i)\}$ and $A_i = \{\mathsf{Attacks}(f(w_i), f(w_i')) \mid w_i, w_i' \in \mathsf{Subtree}(v_i)\}$ for all $v_i \in \mathsf{Children}(z)$ can be re-written to

$$\bigvee \bigcup_{v_i} \big(((\mathsf{Literals}(B_i) \setminus A_i) \cup \mathsf{Disjuncts}(f(z))) \setminus \{b_i, \overline{b}_i\}\big) =$$

and $A_i \cap \mathsf{Disjuncts}(f(z)) = \emptyset$ otherwise the conditions for a complete presupport tree would not be satisfied, so the equation above can be re-written to

$$\bigvee \bigcup_{v_i} \big(\mathsf{Literals}(B_i) \cup \mathsf{Disjuncts}(f(z))) \setminus (A_i \cup \{b_i, \overline{b}_i\})\big) =$$

$$\bigvee \big(\mathsf{Literals}(\{f(w) \mid w \in \mathsf{Subtree}(z)\}) \setminus \{\mathsf{Attacks}(f(w), f(w')) \mid w, w' \in \mathsf{Subtree}(z)\}\big)$$

which by definition is equal to $\mathsf{SubtreeRes}(z)$. □

Example 4.2.5 *Continuing example 4.2.4, for z with $f(z) = a \vee b \vee c \vee d$, and x_1, x_2 such that $f(x_1) = \neg a \vee k \vee j$, $f(x_2) = \neg c \vee l$, $\mathsf{SubtreeRes}(x_1) = \neg a \vee d$, $\mathsf{SubtreeRes}(x_2) = \neg c$ and $\mathsf{SubtreeRes}(z) = a \vee b \vee c \vee d \bullet \neg a \vee d \bullet \neg c = b \vee c \vee d \bullet \neg c = b \vee d$.*

From proposition 4.2.5 and the way function SubtreeRes is defined, follows proposition 4.2.6, according to which, if (N, A, f) is a complete presupport tree for Δ, α and a, then for all the nodes z of (N, A, f) there is a deduction of $\mathsf{SubtreeRes}(z)$ from the set of clauses assigned to the $\mathsf{Subtree}(z)$.

Proposition 4.2.6 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then for a node $z \in N$, there is a linear deduction $\{\delta_1, ..., \delta_n\} \in \mathsf{Deductions}(\{f(w) \mid w \in \mathsf{Subtree}(z)\})$ where $\mathsf{SubtreeRes}(z) \equiv \delta_n$ and $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \subseteq \{\delta_1, ..., \delta_n\}$.*

Proof: This is a proof by induction. For the base case, let $z \in N$ be a leaf and let $f(z) = \delta_1$. So $\mathsf{Subtree}(z) = \{z\}$, and $\{f(w) \mid w \in \mathsf{Subtree}(z)\} = \{f(z)\} = \{\delta_1\}$ and by definition, for a leaf node $\mathsf{SubtreeRes}(z) = f(z)$. Therefore, there is a linear deduction $\{\delta_1\} \in \mathsf{Deductions}(\{f(w) \mid w \in \mathsf{Subtree}(z)\}) = \mathsf{Deductions}(\{\delta_1\})$ where $\mathsf{SubtreeRes}(z) = \delta_1$ and $\{f(w) \mid w \in \mathsf{Subtree}(z)\} = \{\delta_1\} \subseteq \{\delta_1\}$. Therefore the proposition holds for the base case.
For the inductive step, let $z \in N$ be a non-leaf node with children $v_1, .., v_j$, and let $f(z) = \delta_k$. Also, assume that the proposition holds for each child $v_i \in \{v_1, ..., v_j\}$. Hence, for each child v_i, assume that there is a linear deduction $\{\delta_1^{v_i}, ..., \delta_{n_i}^{v_i}\} \in \mathsf{Deductions}(\{f(w) \mid w \in \mathsf{Subtree}(v_i)\})$ where $\mathsf{SubtreeRes}(v_i) \equiv \delta_{n_i}^{v_i}$ and $\{f(w) \mid w \in \mathsf{Subtree}(v_i)\} \subseteq \{\delta_1^{v_i}, ..., \delta_{n_i}^{v_i}\}$.
Therefore, each of these linear deductions can be put into a linear deduction for δ_n as follows,

$$\{\delta_1^{v_1}, ..., \delta_{n_1}^{v_1}, ..., \delta_1^{v_j}, ..., \delta_{n_j}^{v_j}, \delta_k, \delta_n\}$$

where $\delta_n \equiv \delta_{n_1}^{v_1} \bullet ... \bullet \delta_{n_j}^{v_j} \bullet \delta_k$. Then by proposition 4.2.5 follows that $\delta_n = \mathsf{SubtreeRes}(z)$. From the structure of the tree, we have that $\{f(w) \mid w \in \mathsf{Subtree}(v_i)\} \subseteq \{f(w) \mid w \in \mathsf{Subtree}(z)\}$. Then, from the constraints on linear deduction, we have the following.

$$\{f(w) \mid w \in \mathsf{Subtree}(z)\} \subseteq \{\delta_1^{v_1}, ..., \delta_{n_1}^{v_1}, ..., \delta_1^{v_j}, ..., \delta_{n_j}^{v_j}, \delta_k, \delta_n\}$$

Therefore, there is a linear deduction, as follows, where $\mathsf{SubtreeRes}(z) \equiv \delta_n$.

$$\{\delta_1^{v_1}, ..., \delta_{n_1}^{v_1}, ..., \delta_1^{v_j}, ..., \delta_{n_j}^{v_j}, ..., \delta_k, \delta_n\} \in \mathsf{Deductions}(\{f(w) \mid w \in \mathsf{Subtree}(z)\})$$

so the proposition holds for the inductive case. □

So, according to proposition 4.2.6, for a node z of a complete presupport tree (N, A, f) for Δ, α and a, the set of clauses assigned to $\mathsf{Subtree}(z)$ entails $\mathsf{SubtreeRes}(z)$.

Corollary 4.2.1 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then for a node $z \in N$, $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \mathsf{SubtreeRes}(z)$.*

Proof: Follows from proposition 4.2.6. □

From the way function SubtreeRes is defined, if z is the unique child of the root, then the only disjuncts b of $\mathsf{SubtreeRes}(z)$ are the ones for which $b \in \mathsf{Disjuncts}(\alpha)$. Then, since $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \mathsf{SResolvents}(z)$ and $\mathsf{Disjuncts}(\mathsf{SResolvents}(z)) \subseteq \mathsf{Disjuncts}(\alpha)$, it holds that $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \alpha$.

Proposition 4.2.7 *Let (N, A, f) be a complete presupport tree for Δ, α and a. If z is the child of the root node of (N, A, f), then $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \alpha$.*

Proof: Follows from proposition 4.2.6. (N, A, f) is a complete presupport tree as a support tree. For z it holds that $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(z)) \subseteq \mathsf{Disjuncts}(\alpha)$. This is because according to the definition for a complete presupport tree, for all $x \in N$ the disjuncts of $\mathsf{Subtree}(x)$ are those b that appear in the clauses of the subtree for which either condition i) or condition ii) of the definition holds. Since z is the child of the root z_0 where $f(z_0) = \overline{a}$, then the only disjunct of $f(z)$ for which condition ii) holds is a, which by definition 4.2.3 is also a disjunct of α and so for $a \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(x))$ it holds that $a \in \mathsf{Disjuncts}(\alpha)$. For the rest of the disjuncts b of $\mathsf{SubtreeRes}(z)$, condition i) of definition 4.2.3 holds and so $b \in \mathsf{Disjuncts}(\alpha)$, hence all the disjuncts of $\mathsf{SubtreeRes}(z)$ are disjuncts of α so $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(z)) \subseteq \mathsf{Disjuncts}(\alpha)$ and if $\Gamma = \{f(x) \mid x \in \mathsf{Subtree}(z)\}$ then by proposition 4.2.6, $\Gamma \vdash \mathsf{SubtreeRes}(z)$ and so $\Gamma \vdash \alpha$. □

So according to proposition 4.2.7, the set of clauses assigned to the non-root nodes of a complete presupport tree (N, A, f) for Δ, α and a provide a proof for α. The next section introduces a category of complete presupport tree that provides a minimal and consistent proof for α.

4.2.3 The support tree

With some additional conditions on the structure of a complete presupport tree (N, A, f) for Δ, α and a, the set of clauses $\{f(x) \mid x \in N\} \setminus \{\overline{a}\} \cup \{\neg\alpha\}$ can be a minimal inconsistent set and so $\{f(x) \mid x \in N\} \setminus \{\overline{a}\}$ can be a support for an argument for α. These conditions are introduced in the definitions of a consistent presupport tree and a minimal presupport tree as follows.

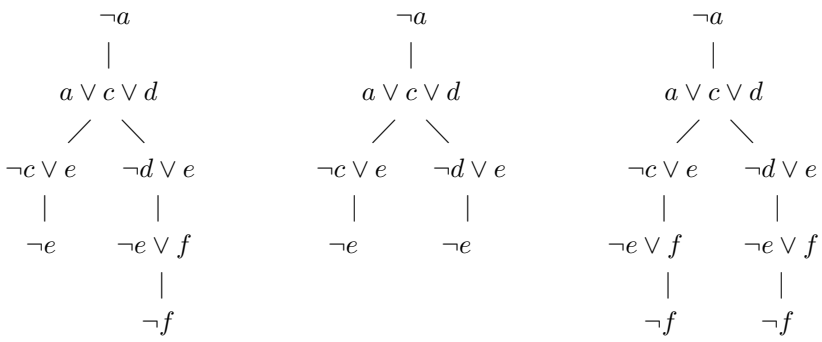
Definition 4.2.5 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then (N, A, f) is a* **consistent presupport tree** *for Δ, α and a iff for any nodes x and y, if x' is the parent of x and y' is the parent of y, $\mathsf{Attacks}(f(x), f(x')) \neq \overline{\mathsf{Attacks}(f(y), f(y'))}$.*

Example 4.2.6 *From the presupport trees of example 4.2.3 only the third is not consistent.*

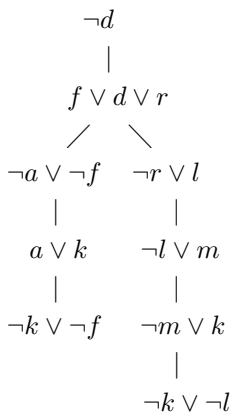
Definition 4.2.6 *Let (N, A, f) be a complete presupport tree for Δ, α and a. Then (N, A, f) is a* **minimal presupport tree** *for Δ, α and a iff there is no complete presupport tree (N', A', f') for Δ, α and a such that $\{f'(x') \mid x' \in N'\} \subset \{f(x) \mid x \in N\}$.*

Example 4.2.7 *All the complete presupport trees of example 4.2.3 are minimal.*

Example 4.2.8 *Let $\alpha = a \vee m$ and $\Delta = \{a \vee c \vee d, \neg b \vee b, \neg b, a \vee d \vee f, \neg e, \neg e \vee f, \neg f, \neg e, \neg d \vee e, \neg c \vee e, r \vee j, j \vee \neg s, \neg s \vee k, p\}$. In the following presupport trees for Δ, α and $a = a$, let (N_1, A_1, f_1) be the first presupport tree, (N_2, A_2, f_2) the second and (N_3, A_3, f_3) the third presupport tree. Then, (N_1, A_1, f_1) is not a minimal presupport tree because (N_2, A_2, f_2) and (N_3, A_3, f_3) are complete presupport trees and the sets of of clauses they consist of are contained in the set of clauses from (N_1, A_1, f_1).*



Example 4.2.9 *The following presupport tree for* $\Delta = \{f \vee d \vee r, \neg a \vee \neg f, \neg r \vee l, a \vee k, \neg l \vee m, \neg k \vee \neg f, \neg k \vee \neg l, \neg m \vee k\}$*,* $\alpha = d \vee q$ *and* d *is a minimal and consistent presupport tree.*



Checking a complete presupport tree (N, A, f) for Δ, α and a for minimality does not necessarily require testing whether each of the subsets of $\{f(x) \mid x \in N\}$ can produce a complete presupport tree for Δ, α and a. In section 4.2.4 I explain how checking some conditions related to the literals that define the arcs of a presupport tree can help in deciding whether the tree satisfies the definition for a minimal presupport tree.

Putting together all the definitions for proof trees given so far in this chapter provides the definition for a support tree for Δ, α and a that follows.

Definition 4.2.7 *A* **support tree** (N, A, f) *for* Δ*,* α *and* $a \in \mathsf{Disjuncts}(\alpha)$ *is a presupport tree* (N, A, f) *for* Δ*,* α *and* a *that is minimal and consistent.*

Example 4.2.10 *Proof trees* (N_2, A_2, f_2) *and* (N_3, A_3, f_3) *of example 4.2.8 are support trees. The proof tree of example 4.2.9 is also a support tree.*

By corollary 4.2.1, the set of clauses that represent a subtree of a complete presupport tree rooted at a node z entails $\mathsf{SubtreeRes}(z)$. According to the next proposition, if (N, A, f) is a support tree then the set of clauses that represent a subtree of a support tree rooted at a node z cannot prove a clause stronger

than $\mathsf{SubtreeRes}(z)$. This property of the support tree is used later for showing the consistency of the proof for α indicated by (N, A, f).

Proposition 4.2.8 *Let (N, A, f) be a support tree for Δ, α and a. Then for all $z \in N$, there is no $\gamma' \in \mathcal{C}$ with $\mathsf{Disjuncts}(\gamma') \subset \mathsf{Disjuncts}(\mathsf{SubtreeRes}(z))$ and $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \gamma'$.*

Proof: Let $\gamma = \mathsf{SubtreeRes}(z)$ and let $B = \{f(w) \mid w \in \mathsf{Subtree}(z)\}$. Then by proposition 4.2.6, $B \vdash \gamma$. To give a proof by contradiction assume $B \vdash \gamma'$ for some $\gamma' \in \mathcal{C}$ such that $\mathsf{Disjuncts}(\gamma') \subset \mathsf{Disjuncts}(\gamma)$. Then there is a literal $m \in \mathsf{Disjuncts}(\gamma) \setminus \mathsf{Disjuncts}(\gamma')$. Also, there is a linear deduction of γ from B, $D_\gamma \in \mathsf{Deductions}(B)$ and a linear deduction of γ' from B, $D_{\gamma'} \in \mathsf{Deductions}(B)$. Then there is a $\rho_m \in D_\gamma$ such that $m \in \mathsf{Disjuncts}(\rho_m)$ and so there is $w \in \mathsf{Subtree}(z)$ such that $f(w) \in D_\gamma$ and $m \in \mathsf{Disjuncts}(f(w))$. Since $m \in \mathsf{Disjuncts}(\gamma)$ then by the way γ is defined there is no arc $(w, w') \in A$ where $w, w' \in \mathsf{Subtree}(z)$ such that $\mathsf{Attacks}(w, w') = m$ or $\mathsf{Attacks}(w, w') = \overline{m}$. Because $m \notin \mathsf{Disjuncts}(\gamma')$ then by lemmas 4.1.1 and 4.1.2 respectively either

(1) There is no $\rho_m \in D_{\gamma'}$ with $m \in \mathsf{Disjuncts}(\rho_m)$ or

(2) There is a $\rho_m \in D_{\gamma'}$ with $m \in \mathsf{Disjuncts}(\rho_m)$ and there is a $\rho_{\overline{m}} \in D_{\gamma'}$ such that $\overline{m} \in \mathsf{Disjuncts}(\rho_{\overline{m}})$

If (1) holds then there is a $w \in \mathsf{Subtree}(z)$ with $m \in \mathsf{Disjuncts}(f(w))$ and $f(w) \notin D_{\gamma'}$. By lemma 4.1.2, for all $k \in \mathsf{Literals}(D_{\gamma'}) \setminus \mathsf{Disjuncts}(\gamma')$ there are $f(y) \in D_{\gamma'} \cap B$ and $f(y') \in D_{\gamma'} \cap B$ such that $k \in \mathsf{Disjuncts}(f(y))$ and $\overline{k} \in \mathsf{Disjuncts}(f(y'))$ and it holds that $\mathsf{Attacks}(f(y), f(y')) = k$. Also for each such y and for the rest of the disjuncts $p \in (\mathsf{Disjuncts}(f(y)) \setminus \{k\}) \setminus \mathsf{Disjuncts}(\gamma')$ because $p \in \mathsf{Literals}(D_{\gamma'}) \setminus \mathsf{Disjuncts}(\gamma')$, then by lemma 4.1.2 there is a $f(y'') \in D \cap B$ with $\overline{p} \in \mathsf{Disjuncts}(f(y''))$ such that $\mathsf{Attacks}(f(y''), f(y)) = \overline{p}$ and the same holds for the rest of the disjuncts of $f(y'')$ and also for all $f(x) \in D_{\gamma'} \cap B$ and the disjuncts of these $f(x)$ that do not appear as disjuncts of γ'. Then, for all $x \in \mathsf{Subtree}(z)$ for which $f(x) \in D_{\gamma'} \cap B$ the conditions for a complete presupport tree hold which means that there is a complete presupport tree (N', A', f') for Δ, α and a such that $\{f'(x') \mid x' \in N'\} \subseteq \{f(x) \mid x \in N\} \setminus \{f(w), f(w')\}$ and so $\{f'(x') \mid x' \in N'\} \subset \{f(x) \mid x \in N\}$ which contradicts the assumption that (N, A, f) is a minimal presupport tree for Δ, α and a.

If (2) holds, there is a $w \in \mathsf{Subtree}(z)$ with $m \in \mathsf{Disjuncts}(f(w))$ and $f(w) \in D_\gamma$ and $f(w) \in D_{\gamma'}$ and there is a $w' \in \mathsf{Subtree}(z)$ such that $f(w') \in D_{\gamma'}$ and $\overline{m} \in \mathsf{Disjuncts}(f(w'))$. Hence there are $w, w' \in \mathsf{Subtree}(z)$ with $m \in \mathsf{Disjuncts}(f(w))$ and $\overline{m} \in \mathsf{Disjuncts}(f(w'))$ but there is no arc (w, w'') where $w'' \in \mathsf{Subtree}(z)$ such that $\mathsf{Attacks}(w, w') = m$ or $\mathsf{Attacks}(w', w'') = \overline{m}$. Hence condition iii) of the definition for a complete presupport tree does not hold for either of $m \in \mathsf{Disjuncts}(f(w))$ and $\overline{m} \in \mathsf{Disjuncts}(f(w'))$. Then, because (N, A, f) is a complete presuport tree for each of $m \in \mathsf{Disjuncts}(f(w))$ and $\overline{m} \in \mathsf{Disjuncts}(f(w'))$ either condition i) or condition ii) of definition 4.2.3 must hold. Condition i) cannot hold for both $m \in \mathsf{Disjuncts}(f(w))$ and $\overline{m} \in \mathsf{Disjuncts}(f(w'))$ because then $\{m, \overline{m}\} \subset \mathsf{Disjuncts}(\alpha)$ and by the definition for a presupport tree α cannot be a tautology. Condition ii) cannot hold for both $m \in \mathsf{Disjuncts}(f(w))$ and $\overline{m} \in \mathsf{Disjuncts}(f(w'))$ because then $\{m, \overline{m}\} \subset \mathsf{AncestorLabels}(z)$ and this contradicts the assumption that (N, A, f) is a consistent presupport tree. If condition i) holds for $m \in \mathsf{Disjuncts}(f(w))$ and condition ii) holds for

$\overline{m} \in \mathsf{Disjuncts}(f(w'))$, then $m \in \mathsf{Disjuncts}(\alpha)$ and there is some $w'' \in N$ (possibly $w = w''$) such that $(w', w'') \in N$ and $\mathsf{Attacks}(f(w'), f(w'')) = \overline{m}$. Then $m \in \mathsf{Disjuncts}(f(w''))$ and for $m \in \mathsf{Disjuncts}(f(w''))$ both conditions i) and iii) hold which contradicts the assumption that (N, A, f) is a complete presupport tree. Similarly if condition ii) holds for $m \in \mathsf{Disjuncts}(f(w))$ and condition i) holds for $\overline{m} \in \mathsf{Disjuncts}(f(w'))$ we get contradiction.

Because either of (1) and (2) lead to contradiction, then it cannot hold that $B \vdash \gamma'$ for some $\gamma' \in \mathcal{C}$ such that $\mathsf{Disjuncts}(\gamma') \subset \mathsf{Disjuncts}(\gamma)$. □

Example 4.2.11 *Continuing example 4.2.4, for* $z = a \vee b \vee c \vee d$, $\mathsf{SubtreeRes}(z) = b \vee d$ *and for* $\gamma' = b$ *and* $\gamma'' = d$, $\{f(x) \mid x \in \mathsf{Subtree}(z)\} \nvdash \gamma'$ *and* $\{f(x) \mid x \in \mathsf{Subtree}(z)\} \nvdash \gamma''$.

The results that follow demonstrate how a support tree is related to a minimal and consistent proof for α. The next lemma is used to prove the completeness of the proposal of using support trees in order to obtain arguments for α from Δ.

Lemma 4.2.1 *Let* $\langle \Phi, \alpha \rangle$ *be an argument. Then, there is a consistent presupport tree* (N, A, f) *for* Δ, α *and* a *such that* $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$.

Proof: Let $\langle \Phi, \alpha \rangle$ be an argument. Then there is a clause α' such that $\Phi \vdash \alpha'$ where $\mathsf{Disjuncts}(\alpha') \subseteq \mathsf{Disjuncts}(\alpha)$ and there is no α'' such that $\mathsf{Disjuncts}(\alpha'') \subset \mathsf{Disjuncts}(\alpha')$ and $\Phi \vdash \alpha''$. Then, $\Phi \cup \{\neg\alpha'\}$ is a minimal inconsistent set. Let C be the set of complements of the disjuncts of α' i.e. $C = \{\overline{b} \mid b \in \mathsf{Disjuncts}(\alpha')\}$. Then, C contains the complements of the literals of α that appear in Φ. $C = \{\overline{b} \mid b \in \mathsf{Disjuncts}(\alpha) \cap \mathsf{Literals}(\Phi)\}$ and for all $\overline{b} \in C$, $\overline{b} \in \mathsf{Conjuncts}(\neg\alpha')$. Because $\Phi \cup \{\neg\alpha'\}$ is a minimal inconsistent set, and by the way C is defined, then if $\Gamma = \Phi \cup C$, it holds that Γ is a minimal inconsistent set of clauses.

By corollary 4.1.1, for all $b \in \mathsf{Literals}(\Gamma)$ there is a $\Gamma' \subset \Gamma$ such that $\langle \Gamma', b \rangle$ is an argument and so there is a linear deduction $D = \{\beta_1, \ldots, \beta_n\}$ such that $\beta_n = b$ and $\Gamma' \subseteq D$. By lemma 4.1.3 there is a $\beta \in \Gamma' \cap D$ such that $b \in \mathsf{Disjuncts}(\beta)$.

Let for $b \in \mathsf{Literals}(\Gamma)$, $\mathcal{D}(b) = \{\beta_1, \ldots, \beta_n\} \in \mathsf{Deductions}(\Gamma')$ be one of these deductions where $\beta_n = b$ and $\beta_{n-1} = \beta \in \Gamma'$ with $b \in \mathsf{Disjuncts}(\beta)$. Also, let for each such $\mathcal{D}(b)$, $\mathsf{Formula}(\mathcal{D}(b))$ be the function that returns β_{n-1} i.e. $\mathsf{Formula}(\mathcal{D}(b)) = \beta_{n-1}$. Hence for all $\mathcal{D}(b)$, $\mathsf{Formula}(\mathcal{D}(b))$ is a clause from the original set of clauses that contains b as a disjunct: $\mathsf{Formula}(\mathcal{D}(b)) \in \Gamma$, and $b \in \mathsf{Disjuncts}(\mathsf{Formula}(\mathcal{D}(b)))$.

Let N be a set of nodes each of which is defined using function $\mathsf{TreeNode}$ as follows. For a node $x \in N$ and a linear deduction $\mathcal{D}(b)$ which is as described above, $\mathsf{TreeNode}(\mathcal{D}(b), x)$ defines a node y that contains $\mathcal{D}(b)$ and has x as its parent. For a node $x \in N$, $\mathsf{Ancestors}(x)$ returns the set of nodes that precede x in a branch. We use this mapping to construct a consistent presupport tree where $\mathsf{Formula}(\mathcal{D}(b))$ will give the clause representation of each node of the tree.

Since Γ is a minimal inconsistent set, there is a $\mathcal{D}(\bot)$ such that $\Gamma \subseteq \mathcal{D}(\bot)$. Because $\mathsf{Disjuncts}(\bot) = \emptyset$ it holds that $\mathsf{Disjuncts}(\bot) \subset \mathsf{Disjuncts}(\gamma)$ for any $\gamma \in \Gamma$ then any clause could be $\mathsf{Formula}(\mathcal{D}(\bot))$. Let

$\mathsf{Formula}(\mathcal{D}(\bot)) = \overline{a}$ for some $\overline{a} \in \Gamma \cap C$. Then, let $x_0 = \mathsf{TreeNode}(\mathcal{D}(\bot), null)$ be the root node of a tree (N, A).

By corollary 4.1.1, for all the disjuncts b of a clause $\gamma \in \Gamma$ there is a $\Gamma_{\overline{b}} \subset \Gamma$ such that $\langle \Gamma_{\overline{b}}, \overline{b} \rangle$ is an argument and so there is a $\mathcal{D}(\overline{b}) \in \mathsf{Deductions}(\Gamma_{\overline{b}})$. Then, for $\mathsf{Formula}(\mathcal{D}(\bot)) = \overline{a} \in \Gamma$ there is a $\mathcal{D}(\overline{\overline{a}}) = \mathcal{D}(a) = \{\alpha_1, \ldots, \alpha_n\}$ where $\alpha_n = \overline{\overline{a}} = a$ and $\mathsf{Formula}(\mathcal{D}(a)) = \alpha_{n-1} \in \Gamma$ is such that $a \in \mathsf{Disjuncts}(a_{n-1})$. Then let $x_a = \mathsf{TreeNode}(\mathcal{D}(a), x_0)$ be the child node of x_0. Then, so far $N = \{x_a, x_0\}$ and $A = \{(x_a, x_0)\}$.

In the same way, for $\mathsf{Formula}(\mathcal{D}(a))$, for all $b_i \in \mathsf{Disjuncts}(\mathsf{Formula}(\mathcal{D}(a))$ there are arguments $\langle \Gamma_{\overline{b}_i}, \overline{b}_i \rangle$ and $\mathcal{D}(\overline{b}_i) \in \mathsf{Deductions}(\Gamma_{\overline{b}_i})$ as described above. Then for all $b_i \in \mathsf{Disjuncts}(\mathsf{Formula}(\mathcal{D}(a))) \setminus \{a\}$, let $x_{\overline{b}_i} = \mathsf{TreeNode}(\mathcal{D}(\overline{b}_i), x_a)$ where $\mathsf{Attacks}(\mathsf{Formula}(\mathcal{D}(\overline{b}_i)), \mathsf{Formula}(\mathcal{D}(a))) = \overline{b}_i$. Then, if $\mathsf{Disjuncts}(\mathsf{Formula}(\mathcal{D}(a))) \setminus \{a\} = \{x_{\overline{b}_1}, \ldots, x_{\overline{b}_m}\}$, the tree so far consists of the set of nodes $N = \{x_{\overline{b}_1}, \ldots, x_{\overline{b}_m}, x_a, x_0\}$ and arcs $A = \{(x_{\overline{b}_1}, x_a), \ldots, (x_{\overline{b}_m}, x_a), (x_a, x_0)\}$.

Continuing constructing the tree in the same way, where for a node $x_{c_i} = \mathsf{TreeNode}(\mathcal{D}(c_i), x_{d_k})$ a child node $x_{\overline{p}_j} = (\mathcal{D}(\overline{p}_j), x_{c_i})$ is created for each disjuct p_j of $\mathsf{Formula}(\mathcal{D}(c_i))$ such that:

(1) there is no $x \in \mathsf{Ancestors}(x_{c_i})$ such that $x = (\mathcal{D}(\overline{p}_j), x')$ for some $x' \in \mathsf{Ancestors}(x_{c_i})$

(2) there is no $x \in \mathsf{Ancestors}(x_{c_i})$ such that $x = (\mathcal{D}(\overline{q}), x')$ for some $x' \in \mathsf{Ancestors}(x_{c_i})$ and $\mathsf{Formula}(\mathcal{D}(\overline{q})) = \mathsf{Formula}(\mathcal{D}(c_i))$

(3) $\mathsf{Attacks}(\mathsf{Formula}(\mathcal{D}(\overline{p}_j)), \mathsf{Formula}(\mathcal{D}(c_i))) = \overline{p}_j$

(4) there is no $x \in N$ such that $x_{p_j} = (\mathcal{D}(p_j), x')$ for some $x' \in N$

we obtain a tree (N, A) which is isomorphic to a complete and consistent presupport tree (N', A', f) for $\Delta \cup C$, $\theta = a$ and $a \in \mathsf{Disjuncts}(\theta)$ where for each $x_{c_i} \in N, x_{c_i} = \mathsf{TreeNode}(\mathcal{D}(c_i), x_{d_k})$ there is a $x \in N'$ such that $f(x) = \mathsf{Formula}(\mathcal{D}(c_i))$ and $\{f(x) \mid x \in N'\} = \Gamma$.

Conditions (1) to (3) in the construction of (N, A) ensure that the conditions of the definition for a presupport tree and conditions $ii)$ and $iii)$ of the definition for a complete presupport tree are satisfied for tree (N', A', f) while condition (4) ensures that the definition for a consistent presupport tree is satisfied for (N', A', f). Because θ is a unit clause, conditions $ii)$ and $iii)$ of definition 4.2.3 are sufficient for (N', A', f) to satisfy the definition for a complete presupport tree. For the nodes $x \in N'$ that are such that $f(x) \in C$ it holds that apart from $\overline{a}$ that represents the root the rest have to be leaf nodes because they are literals and for each $\{x \in N' \mid f(x) \in C\}$, $f(x) = \overline{b}$ where $\overline{b}$ is a literal and $\mathsf{Attacks}(f(x), f(x')) = \overline{b}$ where x' is the parent of x and condition $ii)$ if definition 4.2.3 is satisfied for $\overline{b} \in \mathsf{Disjuncts}(f(x))$ and condition $iii)$ of the definition is satisfied for $b \in \mathsf{Disjuncts}(f(x'))$. Then, if the leaf nodes that have a literal from C assigned as their clause representation i.e. $\{x \in N' \mid f(x) \in C\}$ are removed from the tree, then the resulting tree is a complete presupport tree for Δ, α and a where for the parents x' of each removed leaf x with $f(x) = \overline{b}$ condition $i)$ of the definition for a complete presupport tree holds for $b \in \mathsf{Disjuncts}(f(x'))$ because by the way C is defined it holds that $b \in \mathsf{Disjuncts}(\alpha)$. Apart from the the parents x' of each such removed leaf x no other node in the tree could be affected by

removing these leaf nodes. Then the resulting tree (N'', A'', f) is a complete and consistent presupport tree for Δ, α and a and $\{f(x) \mid x \in N''\} = \{f(y) \mid y \in N'\} \setminus C = \Gamma \setminus C = \Phi$. $\square$

The proposition that follows illustrates the completeness of the proposal for using support trees in order to obtain arguments for α from Δ.

Proposition 4.2.9 *Let $\langle \Phi, \alpha \rangle$ be an argument. Then, there is a support tree (N, A, f) for Δ, α and some $a \in \mathsf{Disjuncts}(\alpha)$ such that $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$.*

Proof: By lemma 4.2.1 there is a consistent presupport tree (N, A, f) for Δ, α and a such that $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$. For a proof by contradiction assume that for all the consistent presupport trees (N, A, f) for Δ, α and a such that $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$ it holds that they are non-minimal. If (N, A, f) is a non-minimal presupport tree then there is a complete presupport tree (N', A', f') for Δ, α and a such that $\{f'(x') \mid x' \in N'\} \subset \{f(x) \mid x \in N\}$. Let be z be the child of the root node of (N', A', f'). Then, because (N', A', f') is a complete presupport tree, from the conditions of definition 4.2.3 the only literals b that appear in the tree for which there are no arcs $(w, w') \in A$ such that $\mathsf{Attacks}(f(w), f(w')) = b$ or $\mathsf{Attacks}(f(w), f(w')) = \overline{b}$ are the ones that are in the disjuncts of α. Also $a \in \mathsf{Disjuncts}(f(z))$ and $a \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(z))$ and by the definition for a presupport tree a is also in the disjuncts of α. Then by the way $\mathsf{SubtreeRes}(z)$ is defined, it holds that for all $b \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(z))$, $b \in \mathsf{Disjuncts}(\alpha)$ and so $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(z)) \subseteq \mathsf{Disjuncts}(\alpha)$. By proposition 4.2.6, for $z \in N'$ it holds that there is a linear deduction $\{\delta_1, \ldots, \delta_n\} \in \mathsf{Deductions}(\{f'(w) \mid w \in \mathsf{Subtree}(z)\})$ where $\delta_n = \mathsf{SubtreeRes}(z)$ and $\{f'(w) \mid w \in \mathsf{Subtree}(z)\} \subseteq \{\delta_1, \ldots, \delta_n\}$. Then, $\{f'(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \mathsf{SubtreeRes}(z)$ and since it holds that $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(z)) \subseteq \mathsf{Disjuncts}(\alpha)$ then it also holds that $\{f'(w) \mid w \in \mathsf{Subtree}(z)\} \vdash \alpha$ and if $\Phi' = \{f'(w) \mid w \in \mathsf{Subtree}(z)\}$ then $\Phi' \subset \Phi$ and $\Phi' \vdash \alpha$ which contradicts the assumption that $\langle \Phi, \alpha \rangle$ is an argument. $\square$

The converse of proposition 4.2.9 also holds, if (N, A, f) is a support tree for Δ, α and some $a \in \mathsf{Disjuncts}(\alpha)$ and Φ is such that $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$, then $\langle \Phi, \alpha \rangle$ is an argument. This result is formalised in the next proposition which illustrates the soundness of the proposal for using support trees for Δ, α and a in order to obtain arguments for α from Δ.

Proposition 4.2.10 *Let Δ be a set of clauses and α be a clause and let (N, A, f) be a support tree for Δ, α and some $a \in \mathsf{Disjuncts}(\alpha)$. Then, $\langle \Phi, \alpha \rangle$ with $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$ is an argument.*

Proof: Let z be the unique child of the root of (N, A, f) where the root z_0 of (N, A, f) is such that $f(z_0) = \overline{a}$. Then $\Gamma = \{f(x) \mid x \in N\} \setminus \{\overline{a}\} = \{f(x) \mid x \in \mathsf{Subtree}(z)\}$.
(1) $\Gamma \vdash \alpha$. Follows from proposition 4.2.7.

(2) $\Gamma \nvdash \bot$. Follows from proposition 4.2.8. It holds that $a \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(z))$, so it follows that $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(z)) \neq \emptyset$. If $\Gamma \vdash \bot$ then for $\gamma' = \bot, \mathsf{Disjuncts}(\gamma') = \emptyset \subset$

$\mathsf{Disjuncts}(\mathsf{SubtreeRes}(z))$ and so by proposition 4.2.8 it cannot hold that $\Gamma \vdash \gamma'$.

(3) There is no $\Gamma' \subset \Gamma$ such that $\Gamma' \vdash \alpha$: For a proof by contradiction assume there is some $\Gamma' \subset \Gamma$ such that $\Gamma' \vdash \alpha$. Without loss of generality assume Γ' is minimal for entailing α. Then, by proposition 4.2.9 there is support tree (N', A', f') for Δ, α and some $b \in \mathsf{Disjuncts}(\alpha)$ such that $\Gamma' = \{f(x') \mid x' \in N'\} \backslash \{\overline{b}\}$. It is shown first that if there is a support tree (N', A', f') for Δ, α and some $b \in \mathsf{Disjuncts}(\alpha)$ then then we can construct a support tree (N'', A'', f'') for $\Delta \cup \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$, $\alpha'' = a$ and a such that $\{f''(x'') \mid x'' \in N''\} = \Gamma' \cup \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$. Then it is proved that using the structure of (N'', A'', f'') we can construct a complete presupport tree for Δ, α and a where the set of clauses assigned to its non-root nodes is equal to Γ'. Then (N, A, f) does not satisfy the definition for a minimal presupport tree and this contradicts the fact that (N, A, f) is a support tree for Δ, α and a.

Let (N', A', f') be a support tree for Δ, α and some $b \in \mathsf{Disjuncts}(\alpha)$ such that $\Gamma' = \{f(x') \mid x' \in N'\} \setminus \{\overline{b}\}$. Then, it holds that $b \neq a$. Otherwise (N', A', f') would be a support tree for Δ, α and a and so (N', A', f') would be a complete presupport tree for Δ, α and a such that $\{f(x') \mid x' \in N'\} \subset \{f(x) \mid x \in N\}$. Then (N, A, f) could not be a minimal presupport tree for Δ, α and a and this contradicts the assumption that (N, A, f) is a support tree for Δ, α and a. So (N', A', f') is a support tree for Δ, α and some $b \in \mathsf{Disjuncts}(\alpha)$ such that $b \neq a$.

Let z be the unique child of the root in (N, A, f). Then, $\Gamma = \{f(x) \mid x \in \mathsf{Subtree}(z)\}$. Let $\gamma = \mathsf{SubtreeRes}(z)$. Then, as explained in the proof of proposition 4.2.7 it holds that $\mathsf{Disjuncts}(\gamma) = \mathsf{Disjuncts}(\alpha) \cap \mathsf{Literals}(\Gamma)$ and also $a \in \mathsf{Disjuncts}(\gamma)$. By corollary 4.2.1, $\Gamma \vdash \gamma$. By proposition 4.2.8 for z, it holds that there is no clause γ'' such $\mathsf{Disjuncts}(\gamma'') \subset \mathsf{Disjuncts}(\gamma)$ and $\Gamma \vdash \gamma''$. Since $\Gamma \nvdash \gamma''$ for any such γ'', then the following holds:

(i) for all $\Gamma'' \subseteq \Gamma$, and for all γ'' such that $\mathsf{Disjuncts}(\gamma'') \subset \mathsf{Disjuncts}(\gamma)$, $\Gamma'' \nvdash \gamma''$.

Let z' be the unique child of the root of (N', A', f'). Then, $\Gamma' = \{f'(x') \mid x' \in \mathsf{Subtree}(z')\}$. Let $\gamma' = \mathsf{SubtreeRes}(z')$. Then, $\mathsf{Disjuncts}(\gamma') = \mathsf{Disjuncts}(\alpha) \cap \mathsf{Literals}(\Gamma')$. Then, $\Gamma' \vdash \gamma'$ and by (i) follows that $\mathsf{Disjuncts}(\gamma') \not\subset \mathsf{Disjuncts}(\gamma)$. Moreover, it cannot hold that $\mathsf{Disjuncts}(\gamma) \not\subset \mathsf{Disjuncts}(\gamma')$ because then it should hold that $\mathsf{Disjuncts}(\alpha) \cap \mathsf{Literals}(\Gamma) \subset \mathsf{Disjuncts}(\alpha) \cap \mathsf{Literals}(\Gamma')$ which cannot hold because $\mathsf{Literals}(\Gamma') \subseteq \mathsf{Literals}(\Gamma)$ (since $\Gamma' \subset \Gamma$). Then it holds that $\mathsf{Disjuncts}(\gamma') = \mathsf{Disjuncts}(\gamma)$ and so $a \in \mathsf{Disjuncts}(\gamma')$. From the way γ' is defined it holds that $\Gamma' \cup \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma')\}$ is a minimal inconsistent set and $\overline{a} \in \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma')\}$. Then if $\Gamma'' = (\Gamma' \cup \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}) \setminus \{\overline{a}\}$, it holds that $\langle \Gamma', a \rangle$ is an argument. Then, by proposition 4.2.9, there is a support tree (N'', A'', f'') for $\Delta \cup \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$, $\alpha'' = a$ and some $a'' \in \mathsf{Disjuncts}(\alpha'')$ such that $\Gamma'' = \{f''(x'') \mid x'' \in N''\} \setminus \{\overline{a''}\}$. Since the unique disjunct of α'' is a, then it follows that $a'' = a$ and so (N'', A'', f'') is a support tree for $\Delta \cup \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$, $\alpha'' = a$ and a. Because α'' is a unit clause, conditions $ii)$ and $iii)$ of the definition for a complete presupport tree are sufficient for (N'', A'', f'') to satisfy the definition for a complete presupport tree. For the nodes $x'' \in N''$ that are such that $f''(x'') \in \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$ it holds that, apart from $\overline{a}$ that represents the root, the rest have to be leaf nodes. Because for each $x'' \in N''$ such that $f''(x'') \in \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\} \setminus \{\overline{a}\}$, it holds that

$f''(x'') = \overline{c}$ for some $\overline{c}$, $f''(x'')$ consists of a unique disjunct. Then, for this $\overline{c} \in \mathsf{Disjuncts}(f''(x''))$, $\mathsf{Attacks}(f''(x''), f(y'')) = \overline{c}$ where y'' is the parent of x'' and condition $ii)$ of definition 4.2.3 is satisfied for $\overline{c} \in \mathsf{Disjuncts}(f''(x''))$ and x'' cannot have a child, so it is a leaf node. For y'' and $c \in \mathsf{Disjuncts}(f''(y''))$ condition $iii)$ of definition 4.2.3 is satisfied. Then, it is shown that if the leaf nodes that have a literal from $\{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\} \setminus \{\overline{a}\}$ assigned as their clause representation are removed from (N'', A'', f''), the resulting tree is a complete presupport tree for Δ, α and a where the set of non-root nodes is equal $\Gamma'' \setminus \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$ which is equal to Γ'. Since $\Gamma' \subset \Gamma$, and $\Gamma = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$ this contradicts the assumption that (N, A, f) is a minimal presupport tree. By removing these leaf nodes of (N'', A'', f'') the resulting tree is complete for Δ, α and a because for the parents y'' of each removed leaf x'' with $f''(x'') = \overline{c}$ condition $i)$ of the definition for a complete presupport tree holds for $c \in \mathsf{Disjuncts}(f''(y''))$ because for all $\{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\}$ it holds that $c \in \mathsf{Disjuncts}(\alpha)$. Apart from the the parents y'' of each such removed leaf x'', no other node in the tree could be affected by removing these leaf nodes of (N'', A'', f'') and the resulting tree is a complete presupport tree for Δ, α and a. Also, the set of clauses assigned to its nodes exluding the root is equal to Γ' because $\mathsf{Disjuncts}(\gamma) \subseteq \mathsf{Disjuncts}(\alpha)$ and so by proposition 4.2.1, $\Gamma' \cap \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\} = \emptyset$ so $\Gamma' \setminus \{\overline{c} \mid c \in \mathsf{Disjuncts}(\gamma)\} = \Gamma'$. Since $\Gamma' \subset \Gamma$ it holds that (N, A, f) is not a minimal presupport tree for Δ, α and a and this contradicts the assumption that (N, A, f) is a support tree for Δ, α and a. So, there cannot be a $\Gamma' \subset \Gamma$ such that $\Gamma' \vdash \alpha$. □

So, according to proposition 4.2.10, if a support tree (N, A, f) is retrieved for Δ, α and some $a \in \mathsf{Disjuncts}(\alpha)$ then an argument $\langle \Phi, \alpha \rangle$ is retrieved for α where Φ is the set of clauses that represent the non-root nodes of (N, A, f). So, proposition 4.2.10 together with proposition 4.2.9 mean that we can have a sound and complete mechanism for generating arguments that is based on support trees.

The fact that we can use different disjuncts of α to determine the root of a support tree does not require comparing results of search based on different disjuncts of α. This is captured in the next proposition.

Proposition 4.2.11 *Let (N, A, f) be a support tree for Δ, α and a. Then, there is no support tree (N', A', f') for Δ, α and some $b \in \mathsf{Disjuncts}(\alpha)$ such that $\{f'(x') \mid x' \in N'\} \setminus \{\overline{b}\} \subset \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$.*

Proof: For a proof by contradiction assume (N, A, f) is a support tree for Δ, α and a and there is a support tree (N', A', f') for Δ, α and some $b \in \mathsf{Disjuncts}(\alpha)$ such that $\{f'(x') \mid x' \in N'\} \setminus \{\overline{b}\} \subset \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$. By proposition 4.2.10 for (N, A, f) it holds that $\langle \{f(x) \mid x \in N\} \setminus \{\overline{a}\}, \alpha \rangle$ is an argument and by proposition 4.2.10 for (N', A', f') it also holds that $\langle \{f'(x') \mid x' \in N'\} \setminus \{\overline{b}\}, \alpha \rangle$ is an argument so $\{f'(x') \mid x' \in N'\} \setminus \{\overline{b}\} \vdash \alpha$ and by assumption $\{f'(x') \mid x' \in N'\} \setminus \{\overline{b}\} \subset \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$ and this contradicts that $\langle \{f(x) \mid x \in N\} \setminus \{\overline{a}\}, \alpha \rangle$ is argument. □

Corollary 4.2.2 *Let Δ be a set of clauses α and be a clause. $\langle \Phi, \alpha \rangle$ is an argument iff there is a support tree (N, A, f) for Δ, α and some $a \in \mathsf{Disjuncts}(\alpha)$ such that $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$.*

Proof: Follows from propositions 4.2.9 and 4.2.10. □

Hence, according to corollary 4.2.2, in order to find all the arguments for α, it is sufficient to find all the support trees for Δ, α and a_i, for all $a_i \in \mathsf{Disjuncts}(\alpha)$. The next section motivates the use of this mechanism by explaining how the structure of a presupport tree can be used in order to check for minimality.

4.2.4 The minimality check

As it was explained earlier, the conditions for a complete presupport tree for Δ, α and a ensure that a proof for α that contains a limited number of clauses is produced. Let (N, A, f) be a complete presupport tree for Δ, α and a. For each node $z \in N$, for each $b \in \mathsf{Disjuncts}(f(z)) \setminus \mathsf{Disjuncts}(\alpha)$ either there is exactly one arc (y, y') such that $y' \in \mathsf{Ancestors}(z)$ and $\mathsf{Attacks}(f(y), f(y')) = b$, and hence there is $y' \in \mathsf{Ancestors}(z)$ with $\overline{b} \in \mathsf{Disjuncts}(f(y'))$, or there is exactly one node $y \in \mathsf{Children}(z)$ such that $\mathsf{Attacks}(f(y), f(z)) = \overline{b}$. This way redundant nodes to resolve with disjunct b of $f(z)$ will be avoided on the branch where z belongs. Avoiding however redundant resolution steps does not necessarily ensure the minimality of the proof indicated by a complete presupport tree. To obtain a minimal and consistent proof for α we need to retrieve a complete presupport tree for Δ, α and a that satisfies the conditions for a minimal and consistent presupport tree. To check whether a complete presupport tree (N, A, f) satisfies the definition for a consistent presupport tree we can simply check whether there are any arcs $(x, x'), (y, y')$ in A where x' is the parent of x and y' is the parent of y such that $\mathsf{Attacks}(f(x), f(x')) = \overline{\mathsf{Attacks}(f(y), f(y'))}$. To check whether (N, A, f) satisfies the definition for a minimal presupport tree, definition 4.2.6 would suggest testing whether subsets of the set of clauses that represent the nodes of (N, A, f) can be used in a complete presupport tree for Δ, α and some $a_i \in \mathsf{Disjuncts}(\alpha)$. In fact the structure of a consistent preupport tree can help deciding whether this indicates a minimal proof for α. So given a consistent presupport tree for Δ, α and a we can decide whether it is minimal by checking some of the properties of the tree.

A complete presupport tree corresponds with a concise linear deduction D from the set of clauses Φ assigned to the non-root nodes. If $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$, then $D \in \mathsf{Deductions}(\Phi)$ where $D = \{\delta_1, \ldots, \delta_n\}$ is such that $\mathsf{Disjuncts}(\delta_n) \subseteq \mathsf{Disjuncts}(\alpha)$ and $\Phi \subseteq D$. D is composed of a limited number of steps, each associated with a node of (N, A, f). Each step uses the preceding steps to calculate $\mathsf{SubtreeRes}(z)$ for a node $z \in N$ and then adds this as a new element of the deduction. With this construction, if z is the child of the root node, then this is the last step of the deduction and $\mathsf{SubtreeRes}(z) = \delta_n$. According to proposition 4.2.5, for a node $z \in N$ with $\mathsf{Children}(z) = \{x_1, \ldots, x_n\}$, it holds that $\mathsf{SubtreeRes}(z) = f(z) \bullet \mathsf{SubtreeRes}(x_1) \bullet \ldots \bullet \mathsf{SubtreeRes}(x_n)$. So, each such step can be regarded as a 'sub-deduction' D_z of D that contributes in obtaining δ_n and is obtained from the set of clauses in $\mathsf{Subtree}(z)$: $D_z = \{\gamma_1, \ldots, \gamma_z\}$ where $\gamma_z = \mathsf{SubtreeRes}(z)$, $D_z \in \mathsf{Deductions}(\{f(w) \mid w \in \mathsf{Subtree}(z)\})$ and $\{f(w) \mid w \in \mathsf{Subtree}(z)\} \subseteq D_z$.

To check whether Φ is minimal for entailing α we can check whether some clauses that contribute to each resolution step can be omitted and as a result get a deduction $D' = \{\delta'_1, \ldots, \delta'_m\}$ such that $D' \in$

$\mathsf{Deductions}(\Phi')$ for some $\Phi' \subset \Phi$ and $\mathsf{Disjuncts}(\delta'_m) \subseteq \mathsf{Disjuncts}(\alpha)$. To decide whether it is possible for this to happen we can look at the presupport tree and examine whether for some node x, there is a node y that plays the same role as x in D and so sub-deduction $D_y \subset D$ can be removed from D and D_x can be used instead in its place. If this is the case, then the set $\Phi' = \{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \setminus \{\overline{a}\}$ is sufficient to entail α. Because the same clauses can be assigned to several nodes of a presupport tree, it can be the case where $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} = \{f(q) \mid q \in N\}$. Then, $\Phi' = \Phi$ and the minimality condition is not affected. If it holds though that $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \neq \{f(q) \mid q \in N\}$, then it means that $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \subset \{f(q) \mid q \in N\}$ and since $\overline{a} \in \{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \cap \{f(q) \mid q \in N\}$, the above can be re-written to $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \setminus \{\overline{a}\} \subset \{f(q) \mid q \in N\} \setminus \{\overline{a}\}$ which is equivalent to $\Phi' \subset \Phi$. So, in the case where $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \neq \{f(q) \mid q \in N\}$, it holds that there is a subset Φ' of Φ such that $\Phi' \vdash \alpha$. Then $\langle \Phi, \alpha \rangle$ is not an argument and by proposition 4.2.10 (N, A, f) is not a support tree so since (N, A, f) is a consistent presupport tree follows that it is not a minimal presupport tree. Hence in such a case where we investigate whether there is some redundancy in the proof for α caused by the steps that correspond to two nodes x, y, we need to see whether $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \neq \{f(q) \mid q \in N\}$ holds. The next paragraph explains which pairs of nodes are likely to cause redundancy and thus need to be investigated.

A pair of nodes x, y can have some overlapping in their role in deduction D if it holds that $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(x)) \cap \mathsf{Disjuncts}(\mathsf{SubtreeRes}(y)) \neq \emptyset$. This is because for a node x and $D_x \subseteq D$ as defined above, $\mathsf{SubtreeRes}(x)$ is the last element of D_x, and some $b \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(x))$ will be used to resolve with some other clause in a later step of the deduction. Hence, if this b appears in $\mathsf{SubtreeRes}(y)$ for some $y \in N$, it is possible for D_y to be subtracted from D and re-use D_x in its place. It can be the case though where $\mathsf{Disjuncts}(\mathsf{SubtreeRes}(x)) \cap \mathsf{Disjuncts}(\mathsf{SubtreeRes}(y)) \subseteq \mathsf{Disjuncts}(\alpha)$ and so the disjuncts that $\mathsf{SubtreeRes}(x)$ and $\mathsf{SubtreeRes}(y)$ have in common are only the ones that are in α. In this case there is no other $b \in \mathsf{Disjuncts}(\mathsf{SubtreeRes}(x)) \cap \mathsf{Disjuncts}(\mathsf{SubtreeRes}(y))$ and so $\mathsf{SubtreeRes}(x)$ is used in the deduction to resolve on disjuncts different than the ones $\mathsf{SubtreeRes}(y)$ is used for. Apart from $a \in \mathsf{Disjuncts}(\alpha)$ which indicates the clause $\overline{a}$ for the root of (N, A, f), the rest of the disjuncts $a_i \in \mathsf{Disjuncts}(\alpha) \cap \mathsf{Literals}(\Phi)$ do not play a role in the proof for α and can make it hard to compare the deduction steps of D. Function Unresolved defined below can be used to deal with this. For an $x \in N$, function $\mathsf{Unresolved}(x)$ gives the clause that consists of the disjuncts of $\mathsf{SubtreeRes}(x)$ excluding the ones that are in α.

Definition 4.2.8 *Let (N, A, f) be a complete presupport tree. For a node $x \in N$,*

$$\mathsf{Unresolved}(x) = \bigvee(\mathsf{Disjuncts}(\mathsf{SubtreeRes}(x)) \setminus \mathsf{Disjuncts}(\alpha))$$

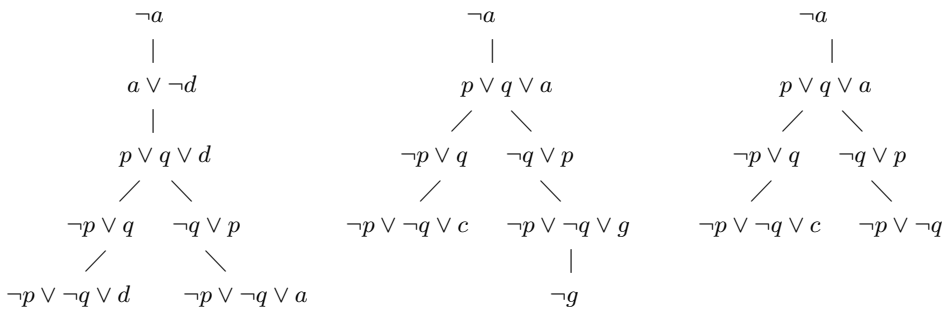
The idea in using function $\mathsf{Unresolved}(x)$ is that it indicates for a node x which literals need to be eliminated at this stage. Checking whether for a pair of nodes $x, y \in N$ there is some $b \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \cap \mathsf{Disjuncts}(\mathsf{Unresolved}(y))$ gives an indication that there is a possibility for D_x to be used instead of D_y in the proof for α or vice versa. So condition $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \cap$

$\mathsf{Disjuncts}(\mathsf{Unresolved}(y)) \neq \emptyset$ can be used to locate which pairs of nodes x, y need to be examined on whether they cause some redundancy in the proof that affects its minimality.

If some $b \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \cap \mathsf{Disjuncts}(\mathsf{Unresolved}(y))$, then replacing D_y by D_x in D can lead to a proof for α if for the rest of the disjuncts b_i of $\mathsf{Unresolved}(x)$ there are clauses in $\Phi \setminus \{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\}$ that can eliminate b_i. From the way the complete presupport tree is built, if (N, A, f) satisfies the definition for a complete presupport tree, then for all $x \in N$, $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \subseteq \mathsf{AncestorLabels}(x)$. If there is a branch on (N, A, f) other than the one where x belongs, where all the disjuncts of $\mathsf{Unresolved}(x)$ are used to label arcs, then it means that there are nodes w_i on this branch for which it holds that $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \cap \mathsf{Disjuncts}(\mathsf{Unresolved}(w_i)) \neq \emptyset$. In particular, there is an arc (y, y') on that branch where y' is the parent of y such that $\mathsf{Attacks}(f(y), f(y')) \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x))$ and $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \subseteq \mathsf{AncestorLabels}(y)$. Then, D_x can replace D_y in D and $\Phi' = \{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\}$ will be sufficient to provide a proof for α. For the parent y' of y, if $\mathsf{Children}(y') = \{y, y_0, \ldots, y_n\}$, then by proposition 4.2.5, $\mathsf{SubtreeRes}(y') = f(y') \bullet \mathsf{SubtreeRes}(y) \bullet \mathsf{SubtreeRes}(y_0) \bullet \ldots \bullet \mathsf{SubtreeRes}(y_n)$. By replacing in this relation $\mathsf{SubtreeRes}(y)$ by $\mathsf{SubtreeRes}(x)$ we get a valid resolution step where no tautologies are involved. This way we obtain a linear deduction $D'_y = \{\gamma_1, \ldots, \gamma_k\}$ where $\gamma_k = f(y') \bullet \mathsf{SubtreeRes}(x) \bullet \mathsf{SubtreeRes}(y_0) \bullet \ldots \bullet \mathsf{SubtreeRes}(y_n)$ and it holds that $\mathsf{Disjuncts}(\gamma_k) \setminus \mathsf{Disjuncts}(\alpha) \subseteq \mathsf{Disjuncts}(\mathsf{SubtreeRes}(y'))$. Then $D'_y \in \mathsf{Deductions}(\Phi'_y)$ where $\Phi'_y = \{f(p) \mid p \in (\mathsf{Subtree}(y') \setminus \mathsf{Subtree}(y)) \cup \mathsf{Subtree}(x)\}$. Continuing the deduction from y by using D'_y we obtain a deduction $D' \in \mathsf{Deductions}(\Phi')$ where $\Phi' = \{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \setminus \{\overline{a}\}$, and $D' = \{\delta'_1 \ldots \delta'_m\}$ is such that $\mathsf{Disjuncts}(\delta'_m) \subseteq \mathsf{Disjuncts}(\alpha)$. For a pair of nodes x, y such that $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \neq \{f(q) \mid q \in N\}$ this would mean that there is a $\Phi' \subset \Phi$ such that $\Phi' \vdash \alpha$ and so (N, A, f) is a non-minimal presupport tree for Δ, α and a.

So, in order to decide whether a consistent presupport tree (N, A, f) is minimal, we need to check first if there are pairs of branches that apart from the arc (z, z') where z' is the root node, they have other arcs labeled by common literals. If they do not, then (N, A, f) is a minimal presupport tree and we do not need to investigate further. If they do, then we check whether there is a pair of nodes x, y such that $\mathsf{Attacks}(f(y), f(y')) \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x))$ (where y' is the parent of y) and $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \subseteq \mathsf{AncestorLabels}(y)$. If this holds and it also holds that $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \neq \{f(q) \mid q \in N\}$ then this means that for $\Phi' = \{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} \setminus \{\overline{a}\}$ it holds that $\Phi' \subset \Phi$ and $\Phi' \vdash \alpha$ and so $\langle \Phi, \alpha \rangle$ is not an argument and (N, A, f) is not a support tree. Since we have assumed that (N, A, f) is a consistent presupport tree, then if (N, A, f) does not satisfy the definition for a support tree it means (N, A, f) does not satisfy the definition for a minimal presupport tree for Δ, α and a.

Example 4.2.12 *Let* $\Delta = \{a \vee \neg d, p \vee q \vee d, p \vee q \vee a, \neg p \vee q, \neg q \vee p, \neg p \vee \neg q, \neg p \vee \neg q \vee a, \neg p \vee \neg q \vee d, \neg p \vee \neg q \vee c, \neg p \vee \neg q \vee g, \neg g\}$ *and* $\alpha = a \vee c$. *All the following consistent presupport trees for* Δ, α *and* a *are non-minimal.*

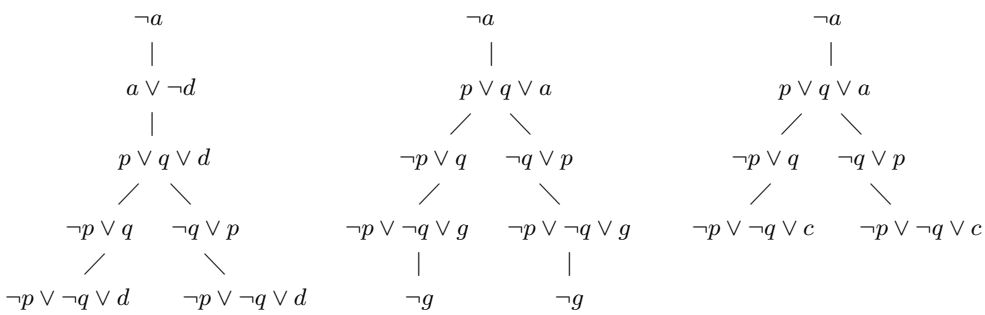


In the first tree, let x be the node with $f(x) = \neg p \vee \neg q \vee d$ and y be the node with $f(y) = \neg p \vee \neg q \vee a$. Then, $\mathsf{Unresolved}(x) = \neg p \vee \neg q \vee d$ *and* $\mathsf{Unresolved}(y) = \neg p \vee \neg q$ *and it holds that* $\mathsf{Attacks}(\neg p \vee \neg q \vee a, \neg q \vee p) \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x))$ *and* $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \subseteq \mathsf{AncestorLabels}(y)$ *and* $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} = \{\neg a, a \vee \neg d, p \vee q \vee d, \neg p \vee q, \neg q \vee p, \neg p \vee \neg q \vee d\} \neq \{f(q) \mid q \in N\}$.

In the second tree, for x such that $f(x) = \neg p \vee \neg q \vee c$ and y such that $f(y) = \neg p \vee \neg q \vee g$, $\mathsf{Unresolved}(x) = \neg p \vee \neg q$ *and* $\mathsf{Unresolved}(y) = \neg p \vee \neg q$ *and it holds that* $\mathsf{Attacks}(\neg p \vee \neg q \vee g, \neg q \vee p) \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x))$ *and* $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \subseteq \mathsf{AncestorLabels}(y)$ *and* $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} = \{\neg a, p \vee q \vee a, \neg p \vee q, \neg q \vee p, \neg p \vee \neg q \vee c\} \neq \{f(q) \mid q \in N\}$.

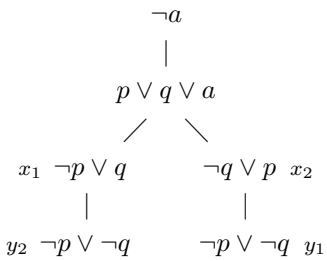
In the third tree, for x such that $f(x) = \neg p \vee \neg q \vee c$ and y such that $f(y) = \neg p \vee \neg q$, $\mathsf{Unresolved}(x) = \neg p \vee \neg q$ *and* $\mathsf{Unresolved}(y) = \neg p \vee \neg q$ *and it holds that* $\mathsf{Attacks}(\neg p \vee \neg q, \neg q \vee p) \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x))$ *and* $\mathsf{Disjuncts}(\mathsf{Unresolved}(x) \subseteq \mathsf{AncestorLabels}(y)$ *and* $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} = \{\neg a, p \vee q \vee a, \neg p \vee q, \neg q \vee p, \neg p \vee \neg q \vee c\} \neq \{f(q) \mid q \in N\}$.

All the following consistent presupport trees are minimal.



Example 4.2.13 *The following presupport tree for $\Delta = \{p \vee q \vee a, \neg p \vee q, \neg q \vee p, \neg p \vee \neg q\}$, $\alpha = a \vee c$ and a is a consistent and minimal presupport tree. For all the pairs of nodes x, y for which it holds that* $\mathsf{Attacks}(f(y), f(y')) \in \mathsf{Disjuncts}(\mathsf{Unresolved}(x))$ *(where y' is the parent of y) and* $\mathsf{Disjuncts}(\mathsf{Unresolved}(x)) \subseteq \mathsf{AncestorLabels}(y)$ *(i.e. the pairs $x = x_1, y = y_1$ or $x = x_2, y = y_2$ or $x = y_1, y = y_2$ or $x = y_2, y = y_1$) it also holds that* $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(y))\} = \{f(q) \mid q \in N\}$

so the minimality condition is not affected for (N, A, f).



4.3 Algorithms for producing proof trees

In this section I present the algorithms that implement the proposal for generating arguments by using proof trees. First I present algorithm SearchTree that generates all the complete presupport trees for a clause α and an $a \in \mathsf{Disjuncts}(\alpha)$ from a knowledgebase Δ. Then, I present algorithm GetSupports which, using the output of algorithm SearchTree, filters out the complete presupport trees that do not satisfy the conditions for being support trees. Then, according to proposition 4.2.10, each of the support trees retrieved gives an argument for α. If this algorithm is applied for all $a_i \in \mathsf{Disjuncts}(\alpha)$, then according to proposition 4.2.9, for all the arguments for α from Δ there is a support tree for Δ, α and some $a_i \in \mathsf{Disjuncts}(\alpha)$ retrieved by the algorithm.

4.3.1 Algorithm for producing complete presupport trees

Algorithm 4.1 builds a depth-first search tree T that represents the steps of the search for arguments for a claim α from a knowledgebase Δ. Every node v' in T is a presupport tree and is an extension of the presupport tree in its parent node v in T. The leaf node of every completed accepted branch is a complete presupport tree.

The search is based on the query graph of α in Δ. The starting point for the search is the complement $\overline{a}$ of one of the disjuncts of α which will also be the root of the retrieved presupport trees. The idea in building presupport trees by using the structure of the query graph, is to start from $\overline{a}$ and walk over the component of the graph that is connected to $\overline{a}$ (i.e. $\mathsf{SubFocus}(\Delta \cup \{\neg\alpha\}, \overline{a})$) by following the links in a way that the conditions for a complete presupport tree indicate until all the complete presupport trees have been generated. The search takes place in a depth-first way. Every step of algorithm SearchTree, which extends the search tree T by a node, consists of extending the presupport tree (N, A, f) that represents the current leaf node v by one level when condition $iii)$ of definition 4.2.3 needs to be satisfied for some of the current leaf nodes of (N, A, f). Function $\mathsf{Extensions}(v)$ gives all the possible extensions of the search tree i.e. all the possible presupport trees that extend the presupport tree in v by one level. Let v be a node in search tree T and let (N, A, f) be the presupport tree that v represents. Let $X = \{x_1, \ldots, x_n\}$ be the set of leaf nodes of (N, A, f) that need to be extended in order to satisfy the conditions for a complete presupport tree. These are the leaf nodes that contain at least one disjunct in their clause representation for which neither condition $i)$ nor condition $ii)$ of definition 4.2.3 holds and hence a child node has to be added for condition $iii)$ of the definition to be satisfied. The

algorithm works by finding all the possible tuples of nodes $Y = (y_1, \ldots, y_n), \ldots, Z = (z_1, \ldots, z_n)$, each of which can be the next level of nodes in (N, A, f) and satisfy the conditions for a complete presupport tree. Hence, each tuple Y retrieved by the algorithm is such that for each x_i in X, for each $b \in \mathsf{Disjuncts}(f(x_i)) \setminus (\mathsf{AncestorLabels}(x_i) \cup \mathsf{Disjuncts}(\alpha))$ such that $\overline{b} \notin \mathsf{AncestorLabels}(x_i)$, there is exactly one y_j in Y such that $\mathsf{Attacks}(f(y_j), f(x_i)) = \overline{b}$, and for each y_j there is a x_i satisfying this condition. So, function $\mathsf{Extensions}(v)$ generates for T all the next possible nodes for its current leaf v where a next possible node v' for the current branch can contain any presupport tree (N', A', f') that is an extension of the presupport tree (N, A, f) of node v produced as described above. If for a node v of the search tree $\mathsf{Extensions}(v) = \emptyset$, then it means that the current branch of the search tree cannot be expanded any further below node v. This happens when for the presupport tree (N, A, f) that is contained in node v

1. either (N, A, f) is a complete presupport tree
2. or for a node x in (N, A, f) a child node has to be added but all the clauses that can be assigned to a child node of x have already been assigned to the ancestors of x
3. or for a node x in (N, A, f) and some $b \in \mathsf{Disjuncts}(f(x))$ there is an arc (w, w'), s.t. $w' \in \mathsf{Ancestors}(x)$ and $\mathsf{Attacks}(f(w), f(w')) = \overline{b}$ and so according to proposition 4.2.3 (N, A, f) cannot be extended to a complete presupport tree.

Algorithm 4.1 $\mathsf{SearchTree}(a)$

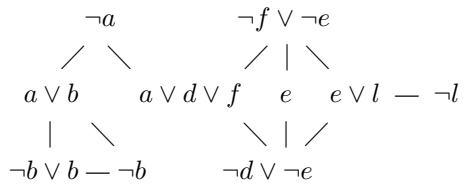
```
Let PresupportTrees = ∅
Let r be a node that contains (N_0, A_0, f_0) s.t. A_0 = ∅ and N_0 = {x} with f_0(x) = ā
Let S be an empty stack
Push r onto S
while S is non-empty do
  Let v be the top of S
  pop S
  if Extensions(v) ≠ ∅ then
    for all y ∈ Extensions(v) do
      push y onto S
    end for
  else
    if Accept(v) then
      PresupportTrees = PresupportTrees ∪ {v}
    end if
  end if
end while
return PresupportTrees
```

In such a case where a leaf node v of the search tree has been reached, the algorithm uses boolean function $\mathsf{Accept}(v)$ which either stores or rejects the presupport tree (N, A, f) in the leaf node v of the currently built branch. $\mathsf{Accept}(v)$ rejects v in cases 2 and 3 given above. If case 1 holds, the presupport tree (N, A, f) that represents v is a complete presupport tree for Δ, α and a and so is stored in the set $PresupportTrees$ of presupport trees that will be returned in the end. Because function $\mathsf{Extensions}(v)$ is extending the presupport tree of each next new node in T according to the conditions of definition

4.2.3, all the non-leaf nodes of a presupport tree (N, A, f) of a node v from T satisfy the conditions for a complete presupport and $\mathsf{Accept}(v)$ only tests the leaf nodes of (N, A, f). After $\mathsf{Accept}(T)$ has either rejected the current branch of T, or stored the result of its leaf node, the algorithm backtracks and continues to the next node of T to be expanded.

After this algorithm has been applied for all $a_i \in \mathsf{Disjuncts}(\alpha)$, and a search tree has been produced for each a_i, all the complete presupport trees for Δ, α and all a_i are obtained.

Example 4.3.1 *Let $\alpha = a \vee m$ and $\Delta = \{a \vee b, \neg b \vee b, \neg b, a \vee d \vee f, \neg f \vee \neg e, e, \neg d \vee \neg e, e \vee l, \neg l, m \vee k, \neg s \vee g, r \vee j, j \vee \neg s, \neg s \vee k, p\}$. The query graph of α in Δ is given below and contains the negation $\neg a$ of $a \in \mathsf{Disjuncts}(\alpha)$ which is the unique starting point for the search for arguments for α, since the negation $\neg m$ of $m \in \mathsf{Disjuncts}(\alpha)$ does not appear in the graph. Figure 4.1 illustrates how algorithm 4.1 traverses this query graph starting from $\neg a$.*

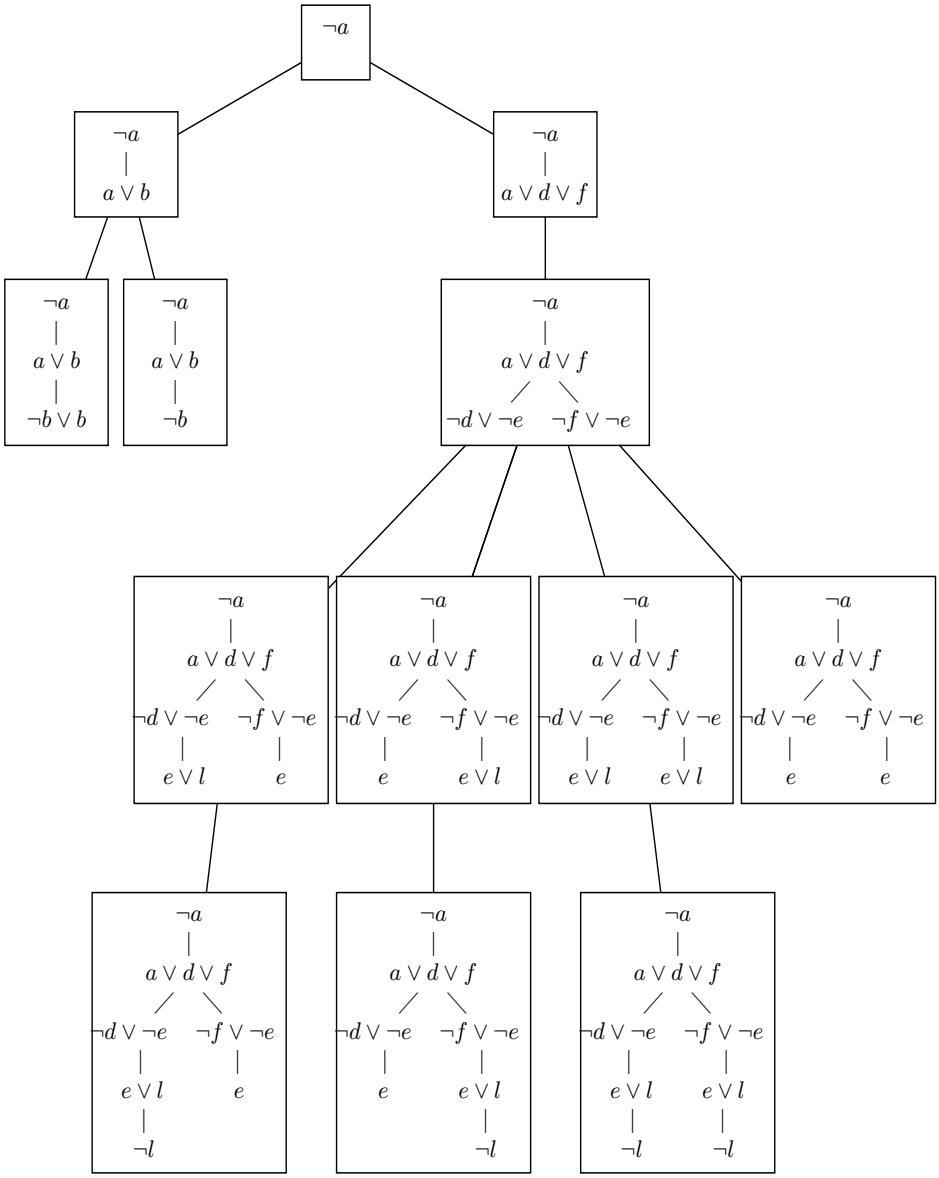


4.3.2 Algorithm for selecting the support trees

Algorithm SearchTree presented in the previous section returns all the complete presupport trees for Δ, α and α_i for some $a_i \in \mathsf{Disjuncts}(\alpha)$. By applying the algorithm for all $a_i \in \mathsf{Disjuncts}(\alpha)$ we obtain all the complete presupport trees for Δ, α and all $a_i \in \mathsf{Disjuncts}(\alpha)$. By following this process, all the arguments for α are generated, but along with the arguments other results that do not correspond to arguments are also produced. A complete presupport tree does not necessarily indicate an argument. In order to obtain all the arguments for α, we need to select for each a_i the complete presupport trees returned by algorithm SearchTree that satisfy the conditions for a consistent and minimal presupport tree.

Selecting the presupport trees (N, A, f) that satisfy the definition for a consistent presupport tree simply requires testing whether there are any arcs on the tree that are labeled by complementary disjuncts. The simplest way to decide for a complete presupport tree whether it satisfies the consistency condition is to store the set of literals that label the arcs of the tree in a set and check whether this set contains a literal and its complement. This holds iff there is pair of arcs (w, w'), (v, v') in A where w' is the parent of w and v' is the parent of v such that $\mathsf{Attacks}(f(w), (w')) = \overline{\mathsf{Attacks}(f(v), (v'))}$. Function $\mathsf{IsConsistent}((N, A, f))$ tests this way whether a complete presupport tree (N, A, f) is consistent.

After rejecting the presupport trees that do not satisfy the consistency condition, the remaining consistent trees have to be tested on whether they satisfy the minimality condition. Algorithm $\mathsf{IsMinimal}((N, A, f))$ tests a consistent presupport tree (N, A, f) on whether it is minimal, by applying the method described in section 4.2.4. The algorithm first tests whether there are branches in the tree, that have arcs labeled by common literals. For this, it tests whether there are pairs of



The result of applying algorithm searchTree for $\alpha = a \vee m$ where the starting node v consists of $\neg a$. The leaf node of the first branch is rejected because for y with $f(y) = \neg b \vee b$, $b \in \mathsf{Disjuncts}(f(y))$ and $\neg b \in \mathsf{AncestorLabels}(y)$ so this cannot be expanded to a complete presupport tree. The results of the other branches are accepted and each of them is a complete presupport tree for Δ, α and a.

Figure 4.1: Applying algorithm SearchTree

nodes x, y from the set *Leaves* of leaf nodes of (N, A, f) for which it holds that the intersection $\Lambda = \mathsf{AncestorLabels}(x) \cap \mathsf{AncestorLabels}(y)$ is non-empty. If $\Lambda = \emptyset$, then those branches do not need to be tested further and the algorithm proceeds to next pair of branches to be tested. If $\Lambda \neq \emptyset$, then the algorithm tests whether there is some node w on the branch where x belongs (where w may be equal to x) and some z on the branch where y belongs (where z may be equal to y) such that $\mathsf{Subtree}(w)$ can replace $\mathsf{Subtree}(z)$ and result to a complete presupport tree. As explained in section 4.2.4, this can happen if $\mathsf{Unresolved}(w) \subseteq \mathsf{AncestorLabels}(z)$. If this holds, then as explained in section 4.2.4, if it is the case where $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(z))\} = \{f(q) \mid q \in N\}$, then the minimality condition is not affected by removing $\mathsf{Subtree}(z)$ from the tree. Because $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(z))\} \subseteq \{f(q) \mid q \in N\}$, in order to check whether $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(z))\} = \{f(q) \mid q \in N\}$ holds, it is sufficient to check whether the cardinality of the two sets is the same. So, the algorithm tests whether $|\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(z))\}| \neq |\{f(q) \mid q \in N\}|$ holds. If this holds then it means that $\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(z))\} \subset \{f(q) \mid q \in N\}$ and so by replacing $\mathsf{Subtree}(z)$ by $\mathsf{Subtree}(w)$ we obtain a complete presupport tree whose set of clauses is a subset of the clauses from (N, A, f). Then, the definition for a minimal presupport tree is not satisfied for (N, A, f) so the algorithm returns false. If $|\{f(p) \mid p \in (N \setminus \mathsf{Subtree}(z))\}| \neq |\{f(q) \mid q \in N\}|$ does not hold, it means that by replacing $\mathsf{Subtree}(z)$ by $\mathsf{Subtree}(w)$ we obtain a complete presupport tree whose set of clauses is equal to the set of the clauses from (N, A, f) and so the minimality of the tree is not affected by nodes w, z and the algorithm proceeds to the next pair of nodes that is to be checked.

Algorithm 4.2 $\mathsf{IsMinimal}((N, A, f))$

```
for all x, y ∈ Leaves do
  Let Λ = AncestorLabels(x) ∩ AncestorLabels(y)
  if Λ ≠ ∅ then
    Let w = x
    while AncestorLabels(w) ∩ Λ ≠ ∅ do
      if Attacks(w, Parent(w)) ∈ Λ then
        Node z = y
        while AncestorLabels(z) ∩ Λ ≠ ∅ do
          if Unresolved(w) ⊆ AncestorLabels(z) then
            if |{f(p) | p ∈ (N \ Subtree(z))}| ≠ |{f(q) | q ∈ N}| then
              return false
            end if
          end if
          z = Parent(z)
        end while
      end if
      w = Parent(w)
    end while
  end if
end for
return true
```

Example 4.3.2 *All the accepted results of the search tree in figure 4.1 are consistent presupport trees. From this tree, algorithm* IsMinimal *would reject the ones from the third and fourth branches.*

clauses-to-variables ratio	$\|\Delta\| = 15$	$\|\Delta\| = 20$	$\|\Delta\| = 25$	$\|\Delta\| = 30$
1	3.000	6.000	9.000	13.00
2	3.000	6.000	11.00	17.00
3	2.000	6.000	12.50	238.0
4	2.000	5.000	14.00	466.5
5	2.000	4.000	8.000	178.0
6	1.000	3.000	6.500	71.00
7	1.000	5.000	4.000	9.000
8	0.000	1.000	4.000	6.000
9	1.000	1.000	2.000	6.000
10	1.000	2.000	2.000	7.000

Table 4.1: Experimental data on generating presupport trees

4.4 Experimental results

This section covers a preliminary experimental evaluation on generating arguments using the idea of support trees using a prototype implementation programmed in java running on a modest PC (Core2 Duo 1.8GHz).

The experimental data were obtained using randomly generated clause knowledgebases according to the fixed clause length model K-SAT ([70, 46]) where the chosen length (i.e. K) for each clause was 3 literals and the claim was a literal. The 3 disjuncts of each clause were chosen out of a set of N distinct variables (i.e. atoms). Each variable was randomly chosen out of the N available and negated with probability 0.5. For a fixed number of clauses, the number of distinct variables that occur in the disjuncts of all the clauses determines the size of the query graph which in turn determines the size of the search space and hence influences the performance of the system. For this reason, 10 different clauses-to-variables ratios were used for each of the different cardinalities tested (where this ratio varied from 1 to 10). For the definition of the ratio the integer part of the division of the number of clauses in Δ by the number of variables N was taken(i.e. $\lfloor |\Delta|/|N| \rfloor$).

The evaluation was based on the time consumed by the system when searching for all the arguments for a given literal claim and the randomly generated knowledgebases of 15 to 30 clauses. Hence, for the results presented the smallest number of variables used was 1 and so for the case of a 15 clause knowledgebase, the clauses-to-variables ratio is 10. The largest number of variables used was 30 and so for the case of a 30 clause knowledgebase, and clauses-to-variables ratio is 1.

The preliminary results are presented in Table 4.1 which contains the median time consumed in milliseconds for 100 repetitions of running the system for each different cardinality and each ratio from 1 to 10. In other words, each field of the table is the median time obtained from finding all the arguments for a random claim in 100 different knowledgebases of fixed cardinality where the cardinality is determined by the column of the table and the different clauses-to-variables ratios is determined by the row.

From the preliminary results in Table 4.1, we see that for a low clauses-to-variables ratio (≤ 2) the number of variables is large enough to allow a distribution of the variables amongst the clauses such

that it is likely for a literal to occur in a clause without its opposite occurring in another clause from the set. As a result, the query graph tends to contain a small subset of the knowledgebase and the system perfoms relatively quickly. The query graph tends also to be small in the case when a relatively small number of variables are distributed amongst the clauses of the knowledgebase (i.e. when the ratio is high) and this makes the occurrence of a variable and its negation in different clauses more frequent. As a result, it is likely for a pair of clauses ϕ, ψ from Δ to be such that $|\mathsf{Preattacks}(\phi, \psi)| > 1$ which will then allow the $\mathsf{Attacks}$ relation to be defined among a small number of clauses and therefore the attack graph will involve only a small subset of the knowledgebase. Hence, a large clauses-to-variables ratio also makes the system perform quickly. From these preliminary results the worst case occurs for ratio 4, and this appears to be because the size of the query graph tends to be maximized. This indicates that the clauses-to-variables ratio, rather than the cardinality of the knowledgebase is the dominant factor determining the time performance for the system.

Overall, the system's behaviour on this experiment is encouraging for further experimentation. The datasets used for this experiment were designed so as to generate hard satisfiability problems that make the system's performance deteriorate. The fact that in the worst case, the median time for generating all the arguments from a knowledgebase of 30 clauses was a fraction of a second gives positive evidence on the efficiency of the underlying algorithms.

4.5 Discussion

This chapter introduced a proposal for generating arguments for claims that are clauses from knowledgebases that are clauses. This proposal is based on resolution and the arguments are produced by appying a search on the query graph defined in chapter 3. By walking over the query graph minimal and consistent proofs for the claim are obtained and the constraints according to which the steps of the search take place are represented by proof trees. Theoretical results supporting the correctness of the proposal were given along with some preliminary experimental results that motivate the practical use of the algorithms that implement this theory.

The next chapter introduces algorithms that generate canonical undercuts based on the theory of this chapter.

Chapter 5

Searching for canonical undercuts

The previous chapter provided a method for generating arguments for claims that are disjunctive clauses. In order to produce argument trees that depict series of arguments and counterarguments we need a mechanism than can support arguments whose claim is not necessarily a clause. Assume we want to find a canonical undercut for an argument $\langle \Phi, \alpha \rangle$. This requires generating an argument with $\neg \bigwedge \Phi$ as the claim and this is not supported by the proposal presented in the previous chapter. In this chapter I show how the definitions and theoretical results of chapter 4 can be extended to generate canonical undercuts for an argument.

The chapter starts by describing how the properties of the language of clauses $\mathcal{C}$ can be taken into account to make the search for undercuts more efficient. It shows how when looking for canonical undercuts for an argument that has Φ for support, we can use a subset Γ of $\mathsf{Resolvents}(\Phi)$ that contains the strongest clauses from $\mathsf{Resolvents}(\Phi)$ and try to find arguments for $\neg \bigwedge \Gamma$ instead. I explain how this alternative can make the search for canonical undercuts more effective. Also, I explain how traversing a support tree that contains the elements of Φ as its non-root nodes can be useful in generating this set Γ of strong resolvents of Φ efficiently, and helps avoiding redundancy in producing resolvents. In addition, I explain how we can use the support tree for producing supports for canonical undercuts. I provide theoretical results and algorithms for the corresponding ideas. Finally, I introduce an algorithm which by putting together the algorithms presented so far in chapters 3, 4 and 5, produces argument trees, and an algorithm for implementing the warrant check introduced in definition 2.2.11 page 30.

5.1 Reducing the search space for canonical undercuts

Let $A = \langle \Phi, \alpha \rangle$ be an argument where $\Phi = \{\phi_1, \ldots, \phi_n\}$ is a set of clauses and α is a clause. Finding counterarguments (i.e. canonical undercuts) for A requires finding arguments with $\neg \bigwedge \Phi$ as the claim. Taking into account that the clauses from Φ are connected through the $\mathsf{Attacks}$ relation, we can focus the search for counterarguments on negating $\bigwedge \Gamma$ where Γ is a particular subset of the resolvents of Φ rather than on negating $\bigwedge \Phi$. This set Γ is the set of strong resolvents defined next and has some interesting properties that can make the search for canonical undercuts effective.

5.1.1 The set of strong resolvents

Definition 5.1.1 *Let Φ be a set of clauses. Then, the set of* **strong resolvents** *of Φ, denoted* $\mathsf{SResolvents}(\Phi)$ *is defined as follows.*

$$\mathsf{SResolvents}(\Phi) = \{\psi \in \mathsf{Resolvents}(\Phi) \mid \neg\exists\psi' \in \mathsf{Resolvents}(\Phi) \textit{ s.t. } \psi' \neq \psi \textit{ and } \mathsf{Disjuncts}(\psi') \subseteq \mathsf{Disjuncts}(\psi)\}$$

Example 5.1.1 *Let* $\Phi = \{\neg e, e \vee \neg k, \neg j \vee k, \neg l \vee j \vee f\}$. *Then,* $\mathsf{Resolvents}(\Phi) = \{\neg e, e \vee \neg k, \neg j \vee k, \neg l \vee j \vee f, \neg k, e \vee \neg j, k \vee \neg l \vee f, \neg j, e \vee \neg l \vee f, \neg l \vee f\}$ *and* $\mathsf{SResolvents}(\Phi) = \{\neg e, \neg j, \neg k, \neg l \vee f\}$.

A set of clauses Φ is equivalent to its set of strong resolvents $\mathsf{SResolvents}(\Phi)$. From this, the next proposition follows where $\equiv$ denotes equivalence.

Proposition 5.1.1 *For a set of clauses Φ, $\neg \bigwedge \Phi \equiv \neg \bigwedge \mathsf{SResolvents}(\Phi)$.*

Proof: Clearly, $\Phi \equiv \mathsf{Resolvents}(\Phi)$ and $\mathsf{Resolvents}(\Phi) \equiv \mathsf{SResolvents}(\Phi)$. Therefore, we get $\bigwedge \Phi \equiv \bigwedge \mathsf{SResolvents}(\Phi)$ and $\neg \bigwedge \Phi \equiv \neg \bigwedge \mathsf{SResolvents}(\Phi)$. □

Example 5.1.2 *Continuing example 5.1.1, it holds that* $\mathsf{SResolvents}(\Phi) = \{\neg e, \neg j, \neg k, \neg l \vee f\}$ *and* $\neg \bigwedge \mathsf{SResolvents}(\Phi) = \neg\big(\neg e \wedge \neg j \wedge \neg k(\neg l \vee f)\big) = (e \vee k \vee j \vee l) \wedge (e \vee k \vee j \vee \neg f)$ *which is the conjunctive normal form of* $\neg\big(\neg e \wedge (e \vee \neg k) \wedge (\neg j \vee k) \wedge (\neg l \vee j \vee f)\big) = \neg \wedge \Phi$

As a consequence of proposition 5.1.1 we obtain the following corollary.

Corollary 5.1.1 *Let Φ, Ψ be sets of clauses. Then $\langle \Psi, \neg \bigwedge \Phi\rangle$ is an argument iff $\langle \Psi, \neg \bigwedge \mathsf{SResolvents}(\Phi)\rangle$ is an argument.*

Proof: Follows from proposition 5.1.1. □

Assume we want to find a canonical undercut for an argument $\langle \Phi, \alpha\rangle$. $\langle \Psi, \diamond\rangle$ is a canonical undercut for $\langle \Phi, \alpha\rangle$ iff $\langle \Psi, \neg \bigwedge\{\phi_1, \ldots, \phi_n\}\rangle$ is an argument, where $\{\phi_1, \ldots, \phi_n\}$ is the canonical enumeration of Φ. According to corollary 5.1.1, searching for a support for an argument with claim $\neg \bigwedge \Phi$ is equivalent to searching for a support for an argument with claim $\neg \bigwedge \mathsf{SResolvents}(\Phi)$. Therefore, searching for a canonical undercut for an argument $\langle \Phi, \alpha\rangle$ is equivalent to searching for an argument for $\neg \bigwedge \mathsf{SResolvents}(\Phi)$. Later in the chapter I explain why using $\neg \bigwedge \mathsf{SResolvents}(\Phi)$ instead of $\neg \bigwedge \Phi$ as the claim for an argument can be useful in generating canonical undercuts for $\langle \Phi, \alpha\rangle$. In order to explain this I first present some propositions. According to the proposition that follows, in the search for canonical undercuts for $\langle \Phi, \alpha\rangle$ we can omit the clauses from Δ that are subsumed by some clause from Φ.

Proposition 5.1.2 *Let $A = \langle \Phi, \alpha\rangle$ be an argument and let $\langle \Psi, \diamond\rangle$ be a canonical undercut for A. Then, $\forall\psi \in \Psi$, $\forall\phi \in \Phi$, $\mathsf{Disjuncts}(\phi) \not\subseteq \mathsf{Disjuncts}(\psi)$.*

Proof: Let $\langle \Psi, \diamond\rangle$ be a canonical undercut for an argument A where $A = \langle \Phi, \alpha\rangle$. Then $\Psi \vdash \neg \bigwedge \Phi$ and $\Psi \cup \{\bigwedge \Phi\}$ is a minimal inconsistent set. To show a contradiction suppose for some $\psi \in \Psi$ and $\phi \in \Phi$,

$\mathsf{Disjuncts}(\phi) \subseteq \mathsf{Disjuncts}(\psi)$ (and hence $\phi \vdash \psi$). Then, $(\Psi \setminus \{\psi\}) \cup \{\phi\} \cup \{\bigwedge \Phi\} \vdash \bot$ but because $\phi \in \mathsf{Conjuncts}(\bigwedge \Phi)$, then $(\Psi \setminus \{\psi\}) \cup \{\bigwedge \Phi\} \vdash \bot$ which contradicts the fact that $\Psi \cup \{\bigwedge \Phi\}$ is a minimal inconsistent set. □

In addition, for the clauses of $\mathsf{SResolvents}(\Phi)$, the next corollary holds.

Corollary 5.1.2 *Let $A = \langle \Phi, \alpha \rangle$ be a argument and let $\langle \Psi, \diamond \rangle$ be a canonical undercut for A. Then, $\forall \psi \in \Psi$, $\forall \rho \in \mathsf{SResolvents}(\Phi)$, $\mathsf{Disjuncts}(\rho) \not\subseteq \mathsf{Disjuncts}(\psi)$.*

Proof: Let $\langle \Psi, \diamond \rangle$ be a canonical undercut for $\langle \Phi, \alpha \rangle$. By Corollary 5.1.1, $\langle \Psi, \neg \bigwedge \mathsf{SResolvents}(\Phi) \rangle$ is an argument and so $\Psi \cup \{\bigwedge \mathsf{SResolvents}(\Phi)\}$ is a minimal inconsistent set. Let for some $\psi \in \Psi$ and $\rho \in \mathsf{SResolvents}(\Phi)$, $\mathsf{Disjuncts}(\rho) \subseteq \mathsf{Disjuncts}(\psi)$ (and hence $\rho \vdash \psi$). Then, $\big((\Psi \setminus \{\psi\}) \cup \{\rho\}\big) \cup \{\bigwedge \mathsf{SResolvents}(\Phi)\} \vdash \bot$ and $\rho \in \mathsf{Conjuncts}(\bigwedge \mathsf{SResolvents}(\Phi))$, so $(\Psi \setminus \{\psi\}) \cup \{\bigwedge \mathsf{SResolvents}(\Phi)\} \vdash \bot$ which contradicts the fact that $\Psi \cup \{\bigwedge \mathsf{SResolvents}(\Phi)\}$ is a minimal inconsistent set. □

Hence, by Corollary 5.1.2, when looking for canonical undercuts for an argument $\langle \Phi, \alpha \rangle$ we can remove from the knowledgebase the clauses that are subsumed by some clause from $\mathsf{SResolvents}(\Phi)$ since these cannot be in the premises for a canonical undercut $\langle \Psi, \diamond \rangle$ for $\langle \Phi, \alpha \rangle$. Then, the next corollary also holds.

Corollary 5.1.3 *Let $\langle \Phi, \alpha \rangle$ be an argument and $\langle \Psi, \diamond \rangle$ be a canonical undercut for $\langle \Phi, \alpha \rangle$. Then, if $\Delta' = \{\delta \in \Delta \mid \exists \rho \in \mathsf{SResolvents}(\Phi)$ s.t. $\rho \vdash \delta\}$ it holds that $\Psi \subseteq \Delta \setminus \Delta'$.*

Proof: Follows from corollary 5.1.2. □

5.1.2 Using the strong resolvents to find canonical undercuts

The definition for a canonical undercut suggests that generating canonical undercuts for an argument $\langle \Phi, \alpha \rangle$ (i.e. finding arguments with claim $\neg \bigwedge \Phi$) requires converting $\neg \bigwedge \Phi$ to its CNF $\overline{\Phi}$ and find arguments for $\overline{\Phi}$. However, conversion to CNF would be an inefficient way to deal with the problem as the length of the CNF can be exponential to the length of the original formula.

As an alternative we can find the sets of all arguments A_i for each $\neg \rho_i$, $\rho_i \in \mathsf{SResolvents}(\Phi)$ using $\Delta \cup \mathsf{SResolvents}(\Phi)$ as the background knowledge. Because $\bigwedge \mathsf{SResolvents}(\Phi)$ is equivalent to $\bigwedge \Phi$, if for some $\Psi \subset \Delta$, $\Psi \cup \{\bigwedge \mathsf{SResolvents}(\Phi)\}$ is a minimal inconsistent set then $\Psi \cup \{\bigwedge \Phi\}$ is also a minimal inconsistent set. Since $\Psi \cup \{\bigwedge \mathsf{SResolvents}(\Phi)\}$ is a minimal inconsistent set then there is a $\Gamma'_i \subseteq \mathsf{SResolvents}(\Phi)$, $\Gamma_i \neq \emptyset$ such that $\Psi \cup \Gamma'_i$ is a minimal inconsistent set and for some $\rho_i \in \Gamma'_i$, if $\Gamma_i = \Gamma'_i \setminus \{\rho_i\}$, then $\langle \Psi \cup \Gamma_i, \neg \rho_i \rangle$ is an argument.

Using an element ρ_i of $\mathsf{SResolvents}(\Phi)$ to find arguments for $\neg \rho_i$ from $\Delta \cup \mathsf{SResolvents}(\Phi)$ helps reducing the number of non-minimal proofs for $\neg \wedge \Phi$ that may be produced during the search. The fact that the premises in Φ are clauses linked with each other by resolution can be used when looking for canonical undercuts $\langle \Psi, \diamond \rangle$ for $\langle \Phi, \alpha \rangle$ to avoid using premises in Ψ that are subsumed in Φ but are

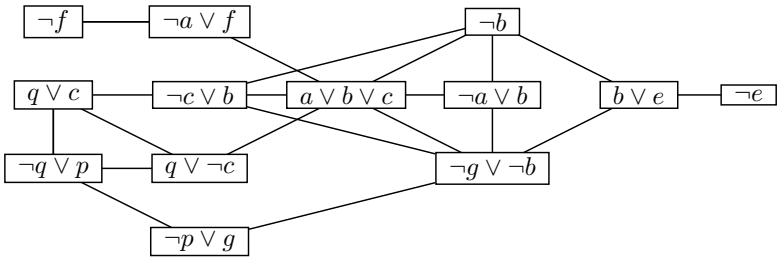
not possible to detect before producing the set $\mathsf{SResolvents}(\Phi)$. For example, assume we are looking for canonical undercuts for $\langle\Phi,\alpha\rangle = \langle\{a\vee b\vee c, \neg c\vee b, \neg b\}, a\vee d\rangle$, hence we are looking for arguments for $\neg\big((a\vee b\vee c)\wedge(\neg c\vee b)\wedge\neg b\big)$. It holds that $\mathsf{SResolvents}(\Phi) = \{a, \neg c, \neg b\}$ so in order to find all the arguments for $\neg\big((a\vee b\vee c)\wedge(\neg c\vee b)\wedge\neg b\big)$ we can alternatively try to find all the arguments for $\neg\big(a\wedge\neg c\wedge\neg b\big)$. For this we can use the negation $\neg\rho_i$ of one of the elements ρ_i of $\mathsf{SResolvents}(\Phi)$ and try to find arguments for $\neg\rho_i$. Not all the arguments for these $\neg\rho_i$ are necessarily canonical undercuts for $\langle\Phi,\alpha\rangle$, but all the canonical undercuts for $\langle\Phi,\alpha\rangle$ are retrieved this way. Although this method generates non-canonical undercuts besides canonical undercuts, it has the advantages that it does not require converting $\neg\wedge\Phi$ to CNF and also that it narrows down the search to what actually needs to be proved. Later I describe how we can decide which of these arguments for $\neg\rho_i$ are canonical undercuts.

Besides the benefits mentioned above, using this method when looking for canonical undercuts for $\langle\Phi,\alpha\rangle$ can help in reducing the cardinality of the knowledgebase that needs to be considered during this search. Any clause $\delta\in\Delta$ that is subsumed by some $\rho\in\mathsf{SResolvents}(\Phi)$ can be removed from the knowledgebase since, by Corollary 5.1.2, δ cannot be included in the premises of a canonical undercut for $\langle\Phi,\alpha\rangle$. For instance, back to the example where we are looking for canonical undercuts for $\langle\Phi,\alpha\rangle = \langle\{a\vee b\vee c, \neg c\vee b, \neg b\}, a\vee d\rangle$, if we use the negation of $a\in\mathsf{SResolvents}(\Phi)$ as the claim for an argument, then $\Psi = \{\neg a\vee g, \neg g\vee\neg b, b\vee e, \neg e\}$ is a support for an an argument for $\neg a$ but it is not a support for a canonical undercut for $\langle\Phi,\alpha\rangle$ because $\Psi' = \{b\vee e, \neg e\}$ is a subset of Ψ sufficient to entail $\neg\wedge\Phi$. Removing $\neg g\vee\neg b$ from Δ because it is subsumed in $\neg b$ where $\neg b\in\mathsf{SResolvents}(\Phi)$ not only reduces the number of clauses considered during the search, but also means Ψ cannot be retrieved during the search for canonical undercuts for $\langle\Phi,\alpha\rangle$.

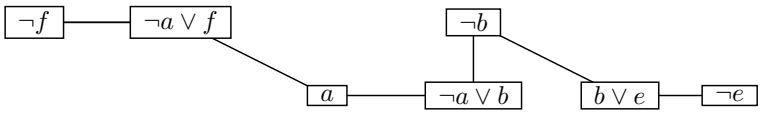
Apart from the fact that removing from Δ the clauses that are subsumed by some clause from $\mathsf{SResolvents}(\Phi)$ can reduce the cardinality of the background knowledge, it can result in a substantially reduced closed graph as the search space for canonical undercuts. Let $\Delta' = \{\delta\in\Delta \mid \exists\rho\in\mathsf{SResolvents}(\Phi)$ s .t. $\rho\vdash\delta\}$. Then, by corollary 5.1.3, $\Delta\setminus\Delta'$ contains the premises for all the canonical undercuts for $\langle\Phi,\alpha\rangle$. The clauses from $\Delta\setminus\Delta'$ can be linked in the closed graph for Δ to a number of clauses from Δ' that would not satisfy the connectivity condition for belonging to $\mathsf{Closed}(\Delta)$ if it had not been for the clauses of Δ'. As an effect, $\mathsf{Closed}((\Delta\setminus\Delta')\cup\mathsf{SResolvents}(\Phi))$ can contain a smaller number of clauses than $\mathsf{Closed}(\Delta)$. Taking into account the fact that each $\delta\in\Delta'$ contains a larger number of disjuncts than at least one $\rho\in\mathsf{SResolvents}(\Phi)$, removing Δ' from the knowledgebase affects the connectivity of the graph and can result to a substantially reduced graph as the search space for canonical undercuts for $\langle\Phi,\alpha\rangle$. This is demonstrated in the following example.

Example 5.1.3 *Let* $\Delta = \{q\vee c, \neg q\vee p, q\vee\neg c, \neg p\vee g, \neg c\vee b, a\vee b\vee c, \neg g\vee\neg b, \neg a\vee b, \neg b, b\vee e, \neg e, \neg a\vee f, \neg f, s\vee t, \neg s\vee m, \neg m\vee n, h\vee j, \neg j\vee i\}$. *Then, for* $\Phi = \{a\vee b\vee c, \neg c\vee b, \neg b\}$ *and* $\alpha = a\vee d$, $\langle\Phi,\alpha\rangle$ *is an argument. Assume we want to find all the canonical undercuts for* $\langle\Phi,\alpha\rangle$. *It holds that* $\mathsf{SResolvents}(\Phi) = \{a, \neg c, \neg b\}$. *Let* $\Delta' = \{\delta\in\Delta \mid \exists\rho\in\mathsf{SResolvents}(\Phi)$ *s .t.* $\rho\vdash\delta\}$. *Then,* $\Delta' = \{\neg c\vee b, a\vee b\vee c, q\vee\neg c, \neg g\vee\neg b, \neg b\}$ *and* $\Delta\setminus\Delta' = \{q\vee c, \neg f, \neg a\vee f, \neg a\vee b, b\vee e, \neg e, \neg q\vee p, \neg p\vee g, s\vee t, \neg s\vee m, \neg m\vee n, h\vee j, \neg j\vee i\}$ *contains the supports for all the canonical undercuts*

for $\langle \Phi, \alpha \rangle$. *In order to retrieve them we can use the closed graph for* $\Delta'' = (\Delta \setminus \Delta') \cup \mathsf{SResolvents}(\Phi)$ $= \{q \vee c, \neg f, \neg a \vee f, \neg a \vee b, b \vee e, \neg e, \neg q \vee p, \neg p \vee g, s \vee t, \neg s \vee m, \neg m \vee n, h \vee j, \neg j \vee i, a, \neg b, \neg c\}$. *The closed graph for* Δ *is*



and the closed graph for Δ'' *is*



The closed graph for Δ'' *contains the supports for* $A_1 = \langle \{\neg f, \neg a \vee f\}, \diamond \rangle$, $A_2 = \langle \{b \vee e, \neg e\}, \diamond \rangle$ *and* $A_3 = \langle \{\neg a \vee b\}, \diamond \rangle$ *that are all the canonical undercuts for* $\langle \Phi, \alpha \rangle$ *from* Δ.

In order to find all the canonical undercuts $\langle \Psi_i, \diamond \rangle$ for $\langle \Phi, \alpha \rangle$ we can find arguments for each $\neg \rho_i$, where $\rho_i \in \mathsf{SResolvents}(\Phi)$, using $\Delta \setminus \{\delta \in \Delta \mid \exists \rho \in \mathsf{SResolvents}(\Phi) \text{ s.t. } \rho \vdash \delta\} \cup \mathsf{SResolvents}(\Phi)$ as the background knowledge. We obtain in this way arguments $\langle \Psi_i \cup \Gamma_i, \neg \rho_i \rangle$ for each $\neg \rho_i$, where $\Psi_i \subseteq \Delta$ and $\Gamma_i \subset \mathsf{SResolvents}(\Phi)$ (where Γ_i can be the empty set). This is illustrated in the next example and formalised in the proposition that follows after the example.

Example 5.1.4 *In order to obtain the canonical undercuts of example 5.1.3, by using the negation of* $\rho_1 = a$ *and* $\rho_2 = \neg b$ *from* $\mathsf{SResolvents}(\Phi)$ *as claims for arguments we obtain*

$$A'_1 = \langle \Psi_1 \cup \Gamma_1, \neg \rho_1 \rangle = \langle \{\neg f, \neg a \vee f\} \cup \emptyset, \neg a \rangle,$$
$$A'_2 = \langle \Psi_2 \cup \Gamma_2, \neg \rho_2 \rangle = \langle \{b \vee e, \neg e\} \cup \emptyset, b \rangle,$$
$$A'_3 = \langle \Psi_3 \cup \Gamma_3, \neg \rho_1 \rangle = \langle \{\neg a \vee b\} \cup \{\neg b\}, \neg a \rangle$$
$$A'_4 = \langle \Psi_4 \cup \Gamma_4, \neg \rho_2 \rangle = \langle \{\neg a \vee b\} \cup \{a\}, b \rangle$$

The part Ψ_i *(i=1...4) of each the support sets* $\Psi_i \cup \Gamma_i$ *of the arguments above, which contains clauses from* Δ, *gives a support for a canonical undercut for* $\langle \Phi, \alpha \rangle = \langle \{a \vee b \vee c, \neg c \vee b, \neg b\}, a \vee d \rangle$.

It is not always the case though that an argument $\langle \Psi_i \cup \Gamma_i, \neg \rho_i \rangle$ where $\Psi_i \subseteq \Delta$, $\Gamma_i \subset \mathsf{SResolvents}(\Phi)$ and $\rho_i \in \mathsf{SResolvents}(\Phi)$ indicates a support Ψ_i for a canonical undercut $\langle \Psi_i, \diamond \rangle$ for $\langle \Phi, \alpha \rangle$. For Ψ_i, it holds that it is consistent and $\Psi_i \vdash \neg \bigwedge \mathsf{SResolvents}(\Phi)$ (and hence $\Psi_i \vdash \neg \bigwedge \Phi$). It can be the case though where Ψ_i is not minimal for entailing $\neg \bigwedge \mathsf{SResolvents}(\Phi)$ so $\langle \Psi_i, \neg \bigwedge \mathsf{SResolvents}(\Phi) \rangle$ is not an argument and there is a $\Psi_j \subset \Psi_i$ such that $\langle \Psi_j, \neg \bigwedge \mathsf{SResolvents}(\Phi) \rangle$ is an argument. In this case, there is a $\Gamma_j \subset \mathsf{SResolvents}(\Phi)$ such that $\Psi_j \cup \Gamma_j$ is a minimal inconsistent set. Then, for some $\rho_j \in \Gamma_j$ (where possibly $\rho_j = \rho_i$) it holds that

$\langle(\Psi_j \cup \Gamma_j) \setminus \{\rho_j\}, \neg\rho_j\rangle$ is an argument. After having found all the arguments for all the $\neg\rho_i$ we can decide which ones indicate canonical undercuts for $\langle\Phi, \alpha\rangle$ by comparing them and checking whether for some $\langle\Psi_i \cup \Gamma_i, \neg\rho_i\rangle$ there is some $\langle\Psi_j \cup \Gamma_j, \neg\rho_j\rangle$ (where ρ_j may or may not be equal to ρ_i) such that $\Psi_j \subset \Psi_i$. The Ψ_i for which this condition does not hold for any Ψ_j are canonical undercuts for $\langle\Phi, \alpha\rangle$. The correctness of using strong resolvents for finding arguments is captured in the following result.

Proposition 5.1.3 *Let $\langle\Phi, \alpha\rangle$ be an argument. $\langle\Psi, \diamond\rangle$ is a canonical undercut for $\langle\Phi, \alpha\rangle$ iff there is a $\rho_i \in \mathsf{SResolvents}(\Phi)$ and a $\Gamma_i \subset \mathsf{SResolvents}(\Phi)$ (possibly the empty set) such that $\langle\Psi \cup \Gamma_i, \neg\rho_i\rangle$ is an argument and there is no $\Psi' \subset \Psi$ and $\Gamma_j \subset \mathsf{SResolvents}(\Phi)$ and $\rho_j \in \mathsf{SResolvents}(\Phi)$ (where ρ_j can be equal to ρ_i) such that such that $\langle\Psi' \cup \Gamma_j, \neg\rho_j\rangle$ is an argument.*

Proof: ($\rightarrow$) Let $A = \langle\Phi, \alpha\rangle$ be an argument and $\langle\Psi, \diamond\rangle$ be a canonical undercut for A. Then $\langle\Psi, \neg\bigwedge\Phi\rangle$ is an argument and $\langle\Psi, \neg\bigwedge\mathsf{SResolvents}(\Phi)\rangle$ is also an argument. Then, $\Psi \cup \{\bigwedge\mathsf{SResolvents}(\Phi)\}$ is a minimal inconsistent set and so there is an $\Gamma'_i \subseteq \mathsf{SResolvents}(\Phi)$ such that $\Psi \cup \Gamma'_i$ is a minimal inconsistent set and $\Gamma'_i \neq \emptyset$ because otherwise $\Psi \vdash \bot$ would hold and $\langle\Psi, \diamond\rangle$ would not be a canonical undercut for A. Then, there is a $\rho_i \in \Gamma'_i$, and if $\Gamma_i = \Gamma'_i \setminus \{\rho_i\}$ then $\langle\Psi \cup \Gamma_i, \neg\rho_i\rangle$ is an argument, so there exists a $\rho_i \in \mathsf{SResolvents}(\Phi)$ and a $\Gamma_i \subset \mathsf{SResolvents}(\Phi)$ (possibly the empty set) such that $\langle\Psi \cup \Gamma_i, \neg\rho_i\rangle$ is an argument. Moreover, there is no $\Psi' \subset \Psi$, $\Gamma_j \subset \mathsf{SResolvents}(\Phi)$ and $\rho_j \in \mathsf{SResolvents}(\Phi)$ such that $\langle\Psi' \cup \Gamma_j, \neg\rho_j\rangle$ is an argument because then $\Psi' \cup \Gamma_j \cup \{\rho_j\}$ would be a minimal inconsistent set and $\Psi' \vdash \neg\bigwedge(\Gamma_j \cup \{\rho_j\})$ from which follows that $\Psi' \vdash \neg\bigwedge(\mathsf{SResolvents}(\Phi))$ and so $\Psi' \vdash \neg\bigwedge\Phi$ and this contradicts the assumption that $\langle\Psi, \diamond\rangle$ is a canonical undercut for $\langle\Phi, \alpha\rangle$.

($\leftarrow$) Assume that for some $\Psi \subseteq \Delta$, $\rho_i \in \mathsf{SResolvents}(\Phi)$, $\Gamma_i \subset \mathsf{SResolvents}(\Phi)$ (possibly the empty set) $\langle\Psi \cup \Gamma_i, \neg\rho_i\rangle$ is an argument. Then Ψ is consistent and $\Psi \vdash \neg\bigwedge(\Gamma_i \cup \{\rho_i\})$ from which follows that $\Psi \vdash \neg\bigwedge\mathsf{SResolvents}(\Phi)$. Also assume and there are no $\Psi' \subset \Psi$, $\Gamma_j \subset \mathsf{SResolvents}(\Phi)$ and $\rho_j \in \mathsf{SResolvents}(\Phi)$ (where ρ_j can be equal to ρ_i) such that $\langle\Psi' \cup \Gamma_j, \neg\rho_j\rangle$ is an argument. Then, $\Psi \cup \Gamma_i \cup \{\rho_i\}$ is a minimal inconsistent set and there is no $\Psi' \subset \Psi$ and $\Gamma'_j \subseteq \mathsf{SResolvents}(\Phi)$ such that $\Psi' \cup \Gamma'_j$ is a minimal inconsistent set, so there is no $\Psi' \subset \Psi$ such that $\Psi' \cup \{\bigwedge\mathsf{SResolvents}(\Phi)\} \vdash \bot$, and hence there is no $\Psi' \subset \Psi$ such that $\Psi' \vdash \neg\bigwedge\mathsf{SResolvents}(\Phi)$, from which follows that $\langle\Psi, \neg\bigwedge\mathsf{SResolvents}(\Phi)\rangle$ is an argument and $\langle\Psi, \diamond\rangle$ is a canonical undercut for $\langle\Phi, \alpha\rangle$. □

Example 5.1.5 *Let $\Delta = \{e \vee q \vee f, \neg f \vee e, \neg e \vee \neg f, f, e \vee \neg q\}$. Then $\langle\Phi, \alpha\rangle = \langle\{e \vee q \vee f, \neg f \vee e\}, e \vee q\rangle$ is an argument and $\mathsf{SResolvents}(\Phi) = \{\neg f \vee e, e \vee q\}$. Moreover, the following are arguments:*

$$A_1 = \langle\Psi_1 \cup \Gamma_1, \neg\rho_1\rangle = \langle\{f, \neg e \vee \neg f, e \vee \neg q\} \cup \emptyset, \neg(e \vee q)\rangle$$

$$A_2 = \langle\Psi_2 \cup \Gamma_2, \neg\rho_2\rangle = \langle\{f, \neg e \vee \neg f\} \cup \emptyset, \neg(\neg f \vee e)\rangle,$$

and $\Psi_2 \subset \Psi_1$ so $\langle\Psi_1, \diamond\rangle$ is not a canonical undercut for $\langle\Phi, \alpha\rangle$. $\langle\Psi_2, \diamond\rangle$ is a canonical undercut for $\langle\Phi, \alpha\rangle$.

To sum up, in order to generate all the canonical undercuts for an argument $\langle\Phi, \alpha\rangle$ from a knowledgebase Δ we can start by generating the set $\mathsf{SResolvents}(\Phi)$ and producing arguments for the negation $\neg\rho_i$ of each $\rho_i \in \mathsf{SResolvents}(\Phi)$ using $\Delta \setminus \{\delta \in \Delta \mid \exists\rho \in \mathsf{SResolvents}(\Phi) \text{ s.t. } \rho \vdash \delta\} \cup \mathsf{SResolvents}(\Phi)$ as the knowledgebase. This process produces for each $\neg\rho_i$, arguments $\langle\Psi_i \cup \Gamma_i, \neg\rho_i\rangle$ where $\Psi_i \subseteq \Delta$ and

$\Gamma_i \subset \mathsf{SResolvents}(\Phi)$ (where Γ_i may be the empty set). After having generated all these arguments for all $\neg\rho_i$ we can decide which ones indicate canonical undercuts for $\langle\Phi, \alpha\rangle$ by rejecting the ones that do not satisfy proposition 5.1.3. Then, the part Ψ_i of the support of the remaining arguments $\langle\Psi_i \cup \Gamma_i, \neg\rho_i\rangle$ is a support for a canonical undercut for $\langle\Phi, \alpha\rangle$.

Example 5.1.3 demostrated how producing the set $\mathsf{SResolvents}(\Phi)$ and removing from Δ the clauses that are subsumed by clauses in $\mathsf{SResolvents}(\Phi)$ can provide a reduced search space for canonical undercuts for $\langle\Phi, \alpha\rangle$. Producing the set of strong resolvents though can be inefficient since the number of resolvents produced by applying resolution exhaustively on a set of clauses can be large. So, we need a mechanism that produces this set efficiently. In Section 5.3, I provide an algorithm that by traversing a support tree that contains the clauses from Φ in its non-root nodes produces the set $\mathsf{SResolvents}(\Phi)$ in a way that controls the number of resolvents produced. Another point to address at this stage is that in order to generate arguments for each $\neg\rho_i$ we need a method that can generate an argument for a negated clause and so far the theory and algorithms presented in previous chapters deal with generating arguments for disjunctive clauses. The topic of the next section is how we can use the support tree in order to generate arguments that have a negated clause as their claim.

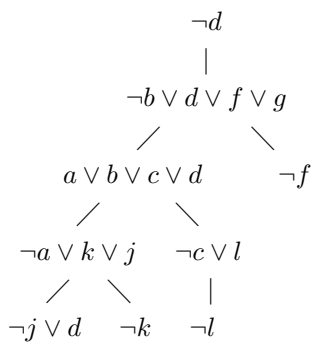
5.2 Using a support tree to generate canonical undercuts

In section 5.1.2 it was demonstrated how using the clauses from $\mathsf{SResolvents}(\Phi)$ and finding arguments for each $\neg\rho_i$ of each $\rho_i \in \mathsf{SResolvents}(\Phi)$ can be used in order to generate undercuts for an argument $\langle\Phi, \alpha\rangle$. This requires finding arguments for the negated clause $\neg\rho_i$. The support tree has been proposed as a way to find arguments for a claim that is a disjunctive clause. In this section I describe how the support tree can be used again as the structure to provide the arguments for $\neg\rho_i$, which in this case is a conjunctive clause.

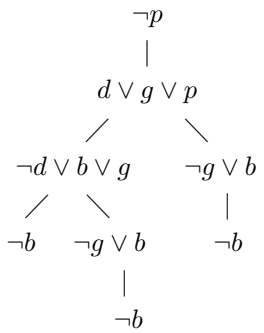
Let (N, A, f) be a support tree for Δ, α and a where α is a clause consisting of a unique literal. Then, $\mathsf{Disjuncts}(\alpha) = \{a\}$, and $\overline{a}$ represents the root node of (N, A, f). If Γ' is the set of non-root nodes of (N, A, f), then by proposition 4.2.10 $\langle\Gamma', \alpha\rangle$ is an argument and for $\Gamma = \Gamma' \cup \{\neg\alpha\}$ it holds that Γ is a minimal inconsistent set. Because $\neg\alpha = \overline{a}$, then if p_0 is the root of (N, A, f) it holds that $f(p_0) = \neg\alpha$ and so Γ is equal to the set of clauses assigned to the nodes of the tree, i.e. $\Gamma = \{f(x) \mid x \in N\}$. Then, since Γ is a minimal inconsistent set, for all $\gamma \in \Gamma$ it holds that $\langle\Gamma \setminus \{\gamma\}, \neg\gamma\rangle$ is an argument. Hence, in order to find an argument for a negated clause $\neg\gamma$, we can find a support tree (N, A, f) for a claim α that consists of a unique disjunct, where γ is assigned to a node in (N, A, f) and obtain in this way a minimal inconsistent set that contains γ. For this we can construct a clause γ' that consists of the literals of γ together with an arbitrary literal p whose atom does not appear anywhere in the knowledgebase or the disjuncts of γ. i.e. $\gamma' = \bigvee(\mathsf{Disjuncts}(\gamma) \cup \{p\})$ where $p \notin \mathsf{Literals}(\Delta \cup \{\gamma\})$ and $\overline{p} \notin \mathsf{Literals}(\Delta \cup \{\gamma\})$. Using $\alpha' = p$ as the clause for which a support tree will be constructed from $\Delta \cup \{\gamma'\}$, ensures that γ' will be contained in all the minimal inconsistent sets that also contain $\overline{p}$ and all the minimal inconsistent sets that contain γ' are the ones that also contain $\overline{p}$. If (N, A, f) is a support tree for $\Delta \cup \{\gamma'\}$, $\alpha' = p$ and $p \in \mathsf{Disjuncts}(\alpha')$, then $\{f(x) \mid x \in N\}$ is a minimal inconsistent set and $\{\gamma', \neg p\} \subset \{f(x) \mid x \in N\}$. From the way γ' is constructed it holds that $\gamma' \wedge \neg p \equiv \gamma$. Then,

$\{f(x) \mid x \in N\} \setminus \{\gamma', \neg p\} \cup \{\gamma\}$ is a minimal inconsistent set, so $\langle \{f(x) \mid x \in N\} \setminus \{\gamma', \neg p\}, \neg\gamma \rangle$ is an argument and $\{f(x) \mid x \in N\} \setminus \{\gamma', \neg p\} \subseteq \Delta$. Therefore, the nodes of (N, A, f) that are located below the node represented by γ' correspond to a support for an argument for $\neg\gamma$.

Example 5.2.1 *The support tree (N, A, f) of example 4.2.4 from chapter 4 gives an argument for $\alpha = d \vee m \vee g$ from $\Delta = \{\neg b \vee d \vee f \vee g, a \vee b \vee c \vee d, \neg a \vee k \vee j, \neg j \vee d, \neg k, \neg c \vee l, \neg l, \neg f, \neg d \vee b \vee g, \neg g \vee b, \neg b, \neg d \vee \neg j, j, \neg g, c \vee l\}$.*



The set of clauses corresponding to the non-root nodes of (N, A, f) is $\Phi = \{\neg b \vee d \vee f \vee g, a \vee b \vee c \vee d, \neg a \vee k \vee j, \neg j \vee d, \neg k, \neg c \vee l, \neg l, \neg f\}$. Then, $\langle \Phi, \alpha \rangle$ is an argument and $\mathsf{SResolvents}(\Phi) = \{\neg f, \neg k, \neg a \vee j, d \vee \neg a, d \vee g, d \vee b, \neg c, \neg l, \neg j \vee d\}$. Assume we want to find a canonical undercut for $\langle \Phi, \alpha \rangle$ and for this we look for an argument for the negation of $\gamma = d \vee g \in \mathsf{SResolvents}(\Phi)$. If $\gamma' = d \vee g \vee p$ and $\alpha' = p$, and $\Delta' = \Delta \cup \{\gamma'\} \cup \mathsf{SResolvents}(\Phi)$, the following is a support tree for Δ', α' and p.



For z such that $f(z) = \gamma' = d \vee g \vee p$, if $\Psi = \{f(x) \mid x \in \mathsf{Subtree}(z) \setminus \{z\}\}$, then $\Psi = \{\neg b, \neg g \vee b, \neg d \vee b \vee g\}$, and $\langle \Psi, \neg d \wedge \neg g \rangle$ is an argument and is equal to $\langle \Psi, \neg\gamma \rangle$.

5.3 Algorithms

In this section I present the algorithms that generate counterarguments and argument trees by using the algorithms introduced in chapters 3 and 4 and additional algorithms that implement the theory from this chapter that concerns strong resolvents. First I give the algorithm that generates the set of strong resolvents of a set Φ that corresponds to the non-root nodes of a support tree (N, A, f) by traversing (N, A, f). Then, I introduce the algorithm that produces canonical undercuts for an argument $\langle \Phi, \alpha \rangle$ by making use of the set of strong resolvents of Φ. Finally, I give the algorithm which, putting together all the algorithms given so far in chapters 3-5, generates an argument tree.

5.3.1 Algorithm for generating resolvents

Let $\langle \Phi, \alpha \rangle$ be an argument, where α and the formulae from Φ and are clauses. In chapter 4 it was proved that there is an $a \in \mathsf{Disjuncts}(\alpha)$ such that (N, A, f) is a support tree for Δ, α and a, such that $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$. In previous paragraphs of this chapter it was described how using the negation of each of the clauses from $\mathsf{SResolvents}(\Phi)$ as claims for arguments can be useful for generating canonical undercuts for $\langle \Phi, \alpha \rangle$. This section introduces an algorithm that generates $\mathsf{SResolvents}(\Phi)$ based on the structure of the suppport tree (N, A, f) that corresponds to Φ.

Apart from providing a minimal and consistent proof for a claim α, a support tree can also be used to generate $\mathsf{SResolvents}(\Phi)$ efficiently. The way the support tree is built helps minimizing the number of disjuncts of the resolvents produced when the order in which the resolution of clauses happens is indicated by a post-order traversal of (N, A, f).

Given for an argument $\langle \Phi, \alpha \rangle$ a support tree (N, A, f) for Δ, α and a that corresponds to Φ, i.e. $\Phi = \{f(x) \mid x \in N\} \setminus \{\overline{a}\}$, algorithm $\mathsf{GetSResolvents}((N, A, f))$ returns the set $\mathsf{SResolvents}(\Phi)$. Using function $\mathsf{ResolveSubtree}(y)$ the algorithm assigns a set of clauses to each node y, that represents the set of strong resolvents produced by $\{f(x) \mid x \in \mathsf{Subtree}(y)\}$. Initially, before the traversal takes place, for each non-root node y from N, $\mathsf{ResolveSubtree}(y)$ is initialized to be equal to $\{f(y)\}$ and for the root node it is initialized to be equal to the empty set (since $\overline{a} \notin \Phi$ and $\overline{a}$ will not be considered when applying resolution in order to produce the set $\mathsf{SResolvents}(\Phi)$). Each node is visited after all its children have been visited. Every time a child x of node y is visited, the value of $\mathsf{Resolvents}(\mathsf{ResolveSubtree}(x) \cup \mathsf{ResolveSubtree}(y))$ is assigned to $\mathsf{ResolveSubtree}(y)$. Every time set $\mathsf{ResolveSubtree}(y)$ is updated, the clauses for which there is a stronger clause in $\mathsf{ResolveSubtree}(y)$ are removed using function $\mathsf{RemoveSubsumed}$. In this way, the number of resolvents produced at each step is controlled and the comparison for subsumed clauses is focused on each node and happens locally. Hence, for each node y that has been visited and processed during the traversal, $\mathsf{ResolveSubtree}(y)$ gives the set of strong resolvents of the set of clauses that represent its subtree. When the root node $root$ is reached during the traversal, the algorithm outputs $\mathsf{ResolveSubtree}(root)$ which is the set $\mathsf{SResolvents}(\{f(x) \mid x \in N\} \setminus \{\overline{a}\}) = \mathsf{SResolvents}(\Phi)$. In order to traverse the tree, the algorithm uses functions $\mathsf{FirstChild}(x)$, $\mathsf{Parent}(x)$, and $\mathsf{NextSibling}(x)$ which, given a node x, return respectively the first child of x, the parent of x and the next sibling of x in (N, A, f). Boolean function $\mathsf{Visited}(x)$ indicates whether node x has been visited so far during the traversal.

In the case where (N, A, f) is a support tree that has been retrieved while looking for a canonical undercut $\langle \Psi, \diamond \rangle$ for some argument that has a support Ψ', then apart from clauses from Ψ, (N, A, f) can also contain clauses from $\mathsf{SResolvents}(\Psi')$. As explained in section 5.1.2, in order to find a canonical undercut for an argument with support Ψ' we look for arguments $\langle \Psi \cup \Gamma, \neg\rho \rangle$ for all the $\rho \in \mathsf{SResolvents}(\Psi')$ from $(\Delta \setminus \Delta') \cup \mathsf{SResolvents}(\Psi')$ where Δ' is the set of clauses that are subsumed by some clauses from $\mathsf{SResolvents}(\Psi')$, $\Psi' \subset \Delta$ and $\Gamma \subset \mathsf{SResolvents}(\Psi')$. In this case, the presupport tree contains the clauses of $\Psi \cup \Gamma$, where Γ may or may not be equal to the empty set. Then, the traversal would produce $\mathsf{SResolvents}(\Psi \cup \Gamma)$ rather than $\mathsf{SResolvents}(\Psi)$. In order to avoid this, the nodes z of

(N, A, f) for which $f(z) \in \Gamma$, have the value for ResolveSubtree(z) initialized to be the empty set. This way, no clauses from Γ are involved in the resolvents produced by the traversal of (N, A, f). For the same reason, in all support trees the root node has its value for ResolveSubtree initialized to be equal to the empty set since it does not represent a clause from the set for which the set of strong resolvents has to be generated.

Algorithm 5.1 GetSResolvents$((N, A, f))$

```
1:  x = root
2:  while x ≠ null do
3:     if FirstChild(x) ≠ null & Visited(x) == false then
4:        Visited(x) = true
5:        x = FirstChild(x)
6:     else
7:        y = Parent(x)
8:        ResolveSubtree(y) = Resolvents(ResolveSubtree(y) ∪ ResolveSubtree(x))
9:        ResolveSubtree(y) = RemoveSubsumed(ResolveSubtree(y))
10:       if NextSibling(x) ≠ null then
11:          x = NextSibling(x)
12:       else
13:          x = Parent(x)
14:       end if
15:    end if
16:    if x == root then
17:       return ResolveSubtree(x)
18:    end if
19: end while
```

The following example illustrates how algorithm GetSResolvents works on one of the support trees of example 5.2.1. The clause representation of each node x is used to denote x, and each corresponding line of the algorithm operation is given on the right hand side of each step. The tree on which the algorithm is applied represents the search for an argument $\langle \Phi, \alpha \rangle$ for a clause α and not the search for a canonical undercut, hence the set of non-root nodes from (N, A, f) is equal to Φ and all the non-root nodes x have their value for ResolveSubtree(x) initialized to be equal to $\{f(x)\}$. For the root node, since it does not have its clause representation in Φ, its value for ResolveSubtree is initialised to be equal to the empty set.

Example 5.3.1 *Let (N, A, f) be the first support tree of example 5.2.1 where $\Phi = \{\neg b \vee d \vee f \vee g, a \vee b \vee c \vee d, \neg a \vee k \vee j, \neg j \vee d, \neg k, \neg c \vee l, \neg l, \neg f\}$. The following represents the sequence of operations that corresponds to the application of algorithm 5.1 on (N, A, f).*

GetSResolvents$((N, A, f))$

$x = \neg d$ *(1)*

Visited$(\neg d) = true$, $x = \neg b \vee d \vee f \vee g$ *(4),(5)*

Visited$(\neg b \vee d \vee f \vee g) = true$, $x = a \vee b \vee c \vee d$ *(4),(5)*

Visited$(a \vee b \vee c \vee d) = true$, $x = \neg a \vee k \vee j$ *(4),(5)*

Visited$(\neg a \vee k \vee j) = true$, $x = \neg j \vee d$ *(4)(5)*

$y = \neg a \vee k \vee j$ *(7)*

$\mathsf{ResolveSubtree}(\neg a \vee k \vee j) = \{\neg a \vee k \vee j, \neg j \vee d, \neg a \vee k \vee d\}$ *(8),(9)*

$x = \neg k$ *(11)*

$y = \neg a \vee k \vee j$ *(7)*

$\mathsf{ResolveSubtree}(\neg a \vee k \vee j) = \{\neg a \vee k \vee j, \neg j \vee d, \neg a \vee k \vee d, \neg k, \neg a \vee j, \neg a \vee d\}$ *(8)*

$\mathsf{ResolveSubtree}(\neg a \vee k \vee j) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d\}$ *(9)*

$x = \neg a \vee k \vee j$ *(13)*

$y = a \vee b \vee c \vee d$ *(7)*

$\mathsf{ResolveSubtree}(a \vee b \vee c \vee d) = \{a \vee b \vee c \vee d, \neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, b \vee c \vee d \vee j, b \vee c \vee d\}$ (8)

$\mathsf{ResolveSubtree}(a \vee b \vee c \vee d) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, b \vee c \vee d\}$ *(9)*

$x = \neg c \vee l$ *(11)*

$\mathsf{Visited}(\neg c \vee l) = true$, $x = \neg l$ *(4),(5)*

$y = \neg c \vee l$ *(7)*

$\mathsf{ResolveSubtree}(\neg c \vee l) = \{\neg c \vee l, \neg l, \neg c\}$ *(8)*

$\mathsf{ResolveSubtree}(\neg c \vee l) = \{\neg l, \neg c\}$ *(9)*

$x = \neg c \vee l$ *(13)*

$y = a \vee b \vee c \vee d$ *(7)*

$\mathsf{ResolveSubtree}(a \vee b \vee c \vee d) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, b \vee c \vee d, \neg l, \neg c, b \vee d\}$ *(8)*

$\mathsf{ResolveSubtree}(a \vee b \vee c \vee d) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d\}$ *(9)*

$x = a \vee b \vee c \vee d$ *(13)*

$y = \neg b \vee d \vee f \vee g$ *(7)*

$\mathsf{ResolveSubtree}(\neg b \vee d \vee f \vee g) = \{\neg b \vee d \vee f \vee g, \neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d, d \vee f \vee g\}$ (8)

$\mathsf{ResolveSubtree}(\neg b \vee d \vee f \vee g) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d, d \vee f \vee g\}$ (9)

$x = \neg f$ *(11)*

$y = \neg b \vee d \vee f \vee g$ *(7)*

$\mathsf{ResolveSubtree}(\neg b \vee d \vee f \vee g) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d, d \vee f \vee g, \neg f, d \vee g\}$ *(8)*

$\mathsf{ResolveSubtree}(\neg b \vee d \vee f \vee g) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d, \neg f, d \vee g\}$ (9)

$x = \neg b \vee d \vee f \vee g$ *(13)*

$y = root$ *(7)*

$\mathsf{ResolveSubtree}(root) = \{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d, \neg f, d \vee g\}$ *(8),(9)*

return $\mathsf{ResolveSubtree}(root)$ *(17)*

The set of clauses returned by the algorithm is $\{\neg j \vee d, \neg k, \neg a \vee j, \neg a \vee d, \neg l, \neg c, b \vee d, \neg f, d \vee g\}$ *and is the set* $\mathsf{SResolvents}(\Phi)$ *of strong resolvents of* Φ.

5.3.2 Algorithms for generating counterarguments

As described in paragraph 5.1.2, given an argument $\langle \Phi, \alpha \rangle$, and the requirement to find all the canonical undercuts for $\langle \Phi, \alpha \rangle$ from Δ, we can start by looking for all the arguments

$$\mathsf{Args}(\neg\rho_1) = \langle \Psi_1^1 \cup \Gamma_1^1, \neg\rho_1\rangle, \ldots \langle \Psi_k^1 \cup \Gamma_k^1, \neg\rho_1\rangle$$
$$\vdots$$
$$\mathsf{Args}(\neg\rho_n) = \langle \Psi_1^n \cup \Gamma_1^n, \neg\rho_n\rangle, \ldots \langle \Psi_m^n \cup \Gamma_m^n, \neg\rho_n\rangle$$

for all $\rho_1 \ldots \rho_n \in \mathsf{SResolvents}(\Phi)$ from $\Delta \cup \mathsf{SResolvents}(\Phi)$ where $\Psi_i \subseteq \Delta$ and $\Gamma_i \subset \mathsf{SResolvents}(\Phi)$. As explained in paragraph 5.1.2, we can use the set $\Delta'' = (\Delta \setminus \{\delta \in \Delta \mid \exists\rho \in \mathsf{SResolvents}(\Phi) \text{ s .t. } \rho \vdash \delta\}) \cup \mathsf{SResolvents}(\Phi)$ as the knowledgebase from which all these arguments are generated. Also, as explained in section 5.2, although the support tree is defined with respect to a disjunctive clause and is related to finding arguments for claims that are clauses, we can use it to find arguments for a negated clause $\neg\rho_i$. For this we can construct a clause ρ_i' consisting of the literals of ρ_i together with an arbitrary literal p whose atom does not appear anywhere in Δ or α, and hence anywhere in Δ''.

Algorithm 5.2 $\mathsf{GetCounterarguments}((N, A, f))$

$Undercuts = \emptyset$, $Canonical = \emptyset$
$\mathsf{SResolvents}(\Phi) = \mathsf{GetSResolvents}((N, A, f))$
$\Delta'' = \mathsf{RemoveSubsumedOf}(\Delta, \mathsf{SResolvents}(\Phi))$
$\Delta'' = \Delta'' \cup \mathsf{SResolvents}(\Phi)$
Let v be node that contains (N, A, f) s.t. $N = \{x\}, A = \emptyset, \{f(x) \mid x \in N\} = \{p\}$
for all $\rho_i \in \mathsf{SResolvents}(\Phi)$ **do**
  $\rho_i' = \mathsf{Augment}(\rho_i, p)$
  $\Delta'' = \Delta'' \cup \{\rho_i'\}$
  $PresupportTrees_i = \mathsf{SearchTree}(v)$
  for all $(N_j, A_j, f_j) \in PresupportTrees_i$ **do**
    if $\neg\mathsf{IsConsistent}((N_j, A_j, f_j))$ **then**
      $PresupportTrees_i = PresupportTrees_i \setminus \{(N_j, A_j, f_j)\}$
    else
      if $\neg\mathsf{IsMinimal}((N_j, A_j, f_j))$ **then**
        $PresupportTrees_i = PresupportTrees_i \setminus \{(N_j, A_j, f_j)\}$
      end if
    end if
  end for
  $Undercuts = Undercuts \cup PresupportTrees_i$
  $\Delta'' = \Delta'' \setminus \{\rho_i'\}$
end for
$Canonical = \mathsf{getCanonical}(Undercuts)$
return $Canonical$

Then, a support tree (N, A, f) for $\Delta'' \cup \{\rho_i'\}$, $\alpha' = p$ and p indicates an argument for $\neg\rho_i$ where the support of this argument is the set of clauses that represent (N, A, f) excluding clauses $\alpha' = p$ and ρ_i'. Let function $\mathsf{Augment}(\rho_i, p)$ return ρ_i' as described above, and let $\mathsf{RemoveSubumedOf}(\Delta, \mathsf{SResolvents}(\Phi))$ return the set $\Delta \setminus \{\delta \in \Delta \mid \exists\rho \in \mathsf{SResolvents}(\Phi) \text{ s .t. } \rho \vdash \delta\}$. Then, given the support tree (N, A, f) that corresponds to argument $\langle \Phi, \alpha\rangle$, algorithm $\mathsf{GetCounterarguments}((N, A, f))$ returns the set of all canonical undercuts for $\langle \Phi, \alpha\rangle$. For this, for each $\rho_i \in \mathsf{SResolvents}(\Phi)$, a search tree T_i is generated by algorithm SearchTree of chapter 4 for $\alpha' = p$ where the knowledgebase is $\Delta'' \cup \{\rho_i'\}$. Each of the results of T_i is then examined on whether it satisfies the conditions for a consistent and minimal presupport tree by using function IsConsistent and algorithm IsMinimal introduced in chapter 4. The ones that do satisfy these conditions are added to set $Undercuts$ and the algorithm proceeds in the same way by building a tree T_j associated to the next $\rho_j \in \mathsf{SResolvents}(\Phi)$ using $\Delta'' \cup \{\rho_j'\}$ (where $\rho_j' = \mathsf{Augment}(\rho_j, p)$) as

the knowledgebase until all the clauses from $\mathsf{SResolvents}(\Phi)$ have been examined.

After having generated all these arguments, the algorithm refines which indicate canonical undercuts for $\langle\Phi,\alpha\rangle$ by comparing them with function getCanonical. The $\langle\Psi_s^i \cup \Gamma_s^i, \neg\rho_i\rangle$ for which there is no $\langle\Psi_l^j \cup \Gamma_l^j, \neg\rho_j\rangle$ (where ρ_j may be equal to ρ_i) such that $\Psi_l^j \subset \Psi_s^i$, indicate a canonical undercut $\langle\Psi_s^i, \diamond\rangle$ for $\langle\Phi,\alpha\rangle$. Because the literals from each ρ_i are linked to the literals of the clauses of each support $\Psi_s^i \cup \Gamma_s^i$ for $\langle\Psi_s^i \cup \Gamma_s^i, \neg\rho_i\rangle$, it is more likely for a Ψ_l^j from $\langle\Psi_l^j \cup \Gamma_l^j, \neg\rho_j\rangle$ to be contained in Ψ_s^i from $\langle\Psi_s^i \cup \Gamma_s^i, \neg\rho_i\rangle$ when these are arguments for the same claim i.e. when it holds that $\rho_i = \rho_j$. For this reason, function getCanonical first compares the supports for arguments that correspond to each $\rho_i \in \mathsf{SResolvents}(\Phi)$ individually and reject the ones that cannot indicate a canonical undercut for $\langle\Phi,\alpha\rangle$ as described above. Then, it compares the supports for the remaining arguments of different ρ_i, ρ_j and similarly it rejects the ones that according to proposition 5.1.3 do not indicate a canonical undercut for $\langle\Phi,\alpha\rangle$.

5.3.3 Algorithm for argument trees

In this section I introduce an algorithm which, putting together the algorithms given so far generates an argument tree. Given the support tree (N_0, A_0, f_0) for Δ, α and a, that corresponds to an argument for α, algorithm $\mathsf{ArgumentTree}((N_0, A_0, f_0))$ generates an argument tree that has $\langle\{f_0(x) \mid x_0 \in N_0\} \setminus \{\overline{a}\}, \alpha\rangle$ as its root. It uses function $\mathsf{Node}((N, A, f), parentNode)$ which, given a support tree (N, A, f), creates an argument tree node structure that is identified by (N, A, f) and has $parentNode$ as its parent in the argument tree.

Algorithm 5.3 $\mathsf{ArgumentTree}((N_0, A_0, f_0))$

$Arcs = \emptyset$, $Nodes = \emptyset$
$root = Node((N_0, A_0, f_0), null)$
S is an empty Stack
push $root$ onto S
while S is not empty **do**
  $topNode = ((N_t, A_t, f_t), t')$ is the top of S
  pop S
  $Canonical = \mathsf{GetCounterarguments}((N_t, A_t, f_t))$
  $\Gamma = \mathsf{GetSResolvents}((N_t, A_t, f_t))$
  for all $(N, A, f) \in Canonical$ **do**
    $\Psi = (\{f(x) \mid x \in N \setminus root\} \cap \Delta) \setminus \Gamma$
    if $\Psi \not\subseteq \mathsf{BranchClauses}(topNode)$ **then**
      $newNode = Node((N, A, f), topNode)$
      $\mathsf{BranchClauses}(newNode) = \mathsf{BranchClauses}(topNode) \cup \Psi$
      $Arcs = Arcs \cup \{(newNode, topNode)\}$
      $Nodes = Nodes \cup \{newNode\}$
      push $newNode$ onto S
    end if
  end for
end while
return $(Arcs, Nodes)$

With the method described in section 5.1.2, when looking for canonical undercuts for an argument that has support Ψ', a canonical undercut $\langle\Psi, \diamond\rangle$ is retrieved for this argument by finding arguments $\langle\Psi \cup \Gamma, \neg\rho_i\rangle$ for some $\rho_i \in \mathsf{SResolvents}(\Psi)$ where $\Gamma \subset \mathsf{SResolvents}(\Psi')$. Therefore, apart from the support

tree (N_0, A_0, f_0) of the root argument, the rest of the nodes of the argument tree are identified by support trees (N, A, f) that may contain clauses from the set of strong resolvents of their parent node in the argument tree. So, the support for each undercut $\langle \Psi, \diamond \rangle$ is given by the relation $\Psi = (\{f(x) \mid x \in N \setminus root\} \cap \Delta) \setminus \Gamma$ where $root$ is the root of (N, A, f).

The algorithm generates the tree in a depth-first way and a branch stops expanding when either there is no canonical undercut for the current leaf from the knowledgebase, or the canonical undercuts for the current leaf have supports whose clauses have all been used in the supports of the arguments of the branch and so the branch cannot be extended further because this would violate condition (2) of the definition of the argument tree. Function BranchClauses gives for each node of the argument tree the set of clauses that appear in the supports of its ancestor nodes in the branch where it belongs and is used by the algorithm to monitor whether a new node can be added on the tree without violating condition (2) of the definition of the argument tree.

The next paragraph provides an algorithm for checking whether an argument tree is warranted.

5.3.4 Algorithm for the warrant check

As discussed in chapter 2, paragraph 2.2.3, we can assess whether an argument tree is warranted by marking its nodes as defeated or undefeated by a recursive mechanism. All the leaf nodes are marked as undefeated and then all the nodes that have at least one undefeated child are marked as defeated. If through this process the root node is marked as defeated then the argument tree (and the root) is not warranted while if the root node is marked as undefeated then the argument tree is warranted.

Algorithm 5.4 Mark$((N, A))$

```
Node next = theRoot
while next ≠ null do
  if FirstChild(next) ≠ null && Visited(next) == false then
    Visited(next) = true
    next = FirstChild(next)
  else
    if Parent(next) == null then
      return
    end if
    if Defeated(next) == false then
      Defeated(Parent(next)) = true
    end if
    if NextSibling(next) ≠ null then
      next = NextSibling(next)
    else
      next = Parent(next)
    end if
  end if
end while
```

Given an argument tree (N, A), algorithm Mark$((N, A))$ assigns a boolean value to the nodes of (N, A) which is set to true for a node that is marked as defeated and false for the opposite. The algorithm assigns these values by traversing the tree. Boolean function Defeated keeps track of the value assigned

to a node. Initially all the nodes x have the value for $\mathsf{Defeated}(x)$ initialised to be equal to $false$.

The algorithm uses functions $\mathsf{FirstChild}(x)$, $\mathsf{Parent}(x)$, and $\mathsf{NextSibling}(x)$ which, given a node x, return respectively the first child of x, the parent of x and the next sibling of x in (N, A). Boolean function $\mathsf{Visited}(x)$ indicates whether node x has been visited so far during the traversal. Every node in the tree is visited after all its children have been visited and its value for defeated is assigned with respect to the values assigned to its children nodes.

After $\mathsf{Mark}((N, A))$ has assigned the values for defeated to the nodes of the tree, the tree can be evaluated on whether it is warranted by checking the value assigned to its root node.

5.4 Discussion

This chapter provided a proposal for generating canonical undercuts for arguments whose supports are from $\mathcal{C}$. The theory of chapters 3 and 4 was extended to deal with canonical undercuts, and algorithms that implement this theory were introduced. Finally, an algorithm that generates argument trees based on the algorithms of chapters 3, 4 and 5 was given, along with an algorithm that assigns the values defeated or undefeated to the nodes of an argument tree, providing this way the means for evaluating whether the argument tree is warranted.

In the next chapter I present the software implementation of an argumentation system that is based on these algorithms and empirical evaluation of the algorithms through experimentation on the system.

Chapter 6

Implementation

This chapter introduces JArgue, an argumentation engine that implements argumentation based on classical propositional logic using the algorithms introduced in chapters 3-5. It is developed in Java and can be used with propositional clauses, where a claim for an argument, which is a disjunctive clause, and a knowledgebase that consists of disjunctive clauses are given as input.

The input is given through a simple graphical user interface from which a text file containing the knowledgebase is loaded to the system, and a claim for arguments is typed in the appropriate field. The system first generates all the supports for arguments for the given claim from the given knowledgebase and displays them in a list. The user can then select one of the supports from this list, and the system generates an argument tree where the root of the tree is the argument with the selected support. This tree can then be evaluated on whether it is warranted (according to the warrant mechanism introduced in [12] and discussed in chapter 2), providing this way an evaluation for the root argument.

The chapter starts by describing the architecture of the software and proceeds by presenting the functionality of the system's graphical user interface. It closes with an experimental evaluation on generating argument trees using JArgue.

6.1 System architecture

JArgue is developed in Java and implements the theory and algorithms introduced in chapters 3-5. Some of the most important classes of the system are described below.

Class `Clause` constructs objects that represent propositional clauses given the String representation of a clause. It implements operations like resolution between pairs of clauses, negation of a literal, comparison of clauses on which is stronger than the other or whether two clauses are equivalent or equal and the Attacks and Preattacks functions as these are defined in chapter 3.

Class `ClauseKb` constructs objects that represent sets of propositional clauses given either the String representation of a set or an array of `Clause` objects that are the elements of the set. It stores `Clause` objects in its instance variable `elements` and implements the standard set operations of union, intersection, subtraction and equality check as well as consistency check and resolution in between its elements.

Class `GraphNode` constructs objects that represent nodes for a graph. Each `GraphNode` object

is initialised given a `Clause` object and is used by classes that represent graphs. It contains boolean instance variables that keep track of its state, i.e. whether it has been visited or not in a search of a graph where it may belong.

Class `AttackGraph` constructs an attack graph for a `ClauseKb` object as this is defined in chapter 3. For each `Clause` contained in the input `ClauseKb`, a `GraphNode` object is created and stored in a 2-dimensional array that acts as an adjacency matrix. `AttackGraph` implements operations for traversing the graph and checking connectivity of nodes.

Class `FocalGraph` constructs a focal graph object for a given `ClauseKb` and a `Clause` as the epicentre. It contains an `AttackGraph` instance variable where `GraphNode` objects costructed for the input clauses are stored. This `AttackGraph` is the main structure that is used in order to traverse and isolate the connected component that corresponds to the focal graph for the given epicentre in the given knowledgebase as this is defined in chapter 3. The class also implements operations that are used by other classes in order to walk over the graph and retrieve supports for arguments.

Class `PresupportSupertree` represents a search tree as introduced in chapter 4 for a given claim and knowledgebase (i.e. `Clause` and `ClauseKb` objects respectively). It has a `FocalGraph` object as an instance variable that represents the corresponding query graph. Method `getPresupportTrees` carries out a search for arguments for the given claim using the structure of the query graph and based on algorithm SearchTree introduced in chapter 4. It creates during the search `Node` objects, each of which is identified by a `GraphNode` from the query graph. A `Node` object differs from a `GraphNode` object in that it has properties that make it suitable for a tree node rather than a simple graph node, for example instance variables and functions related to its ancestors and children. Tuples of `Node` objects that represent an extension to a node of the search tree are stored in `PresupportTreeLevel` objects each of which represents an element of set Extensions introduced in the algorithm. All the leaves of this search tree that correspond to complete presupport trees are stored in an array.

Given one the results of function `getPresupportTrees` of class `PresupportSupetree`, class `PresupportTree` constructs an object that has the properties of a complete presupport tree. It implements methods that check whether a presupport tree is minimal and consistent as described in chapter 4 and methods that carry out traversals like for instance post-order traversals that compute the value $\mathsf{SubtreeRes}(x)$ for a node x.

By the results of chapter 4 and 5, a minimal and consistent presupport tree can be associated to a support for an argument. Class `ArgumentTreeNode` constructs objects that are initialised given a `PresupportTree` object. In addition, it contains instance variables that keep track of the parent and the children on an `ArgumentTreeNode` object, a boolean value that keeps track of the state defeated/undefeated during the warrant check process of an argument tree where this node may belong, as well as the union of supports of all its ancestor nodes so that condition (2) of the definition for an argument tree can be ensured for each newly created argument tree node.

Finally, class `ArgumentTree` represents an argument tree that consists of `ArgumentTreeNode`

objects. Function `getTree(ClauseKb,PresupportTree)` generates an argument tree where the root `ArgumentTreeNode` is initialized by the given `presupportTree` and the tree expands by applying algorithm ArgumentTree from paragraph 5.3.3. The class implements algorithm 5.4 for marking the nodes of an `ArgumentTree` object as defeated or undefeated and using function `isWarranted` determines whether an argument tree is warranted.

Figure 6.1 shows a diagram of the major clases that compose JArgue.

6.2 Using the system

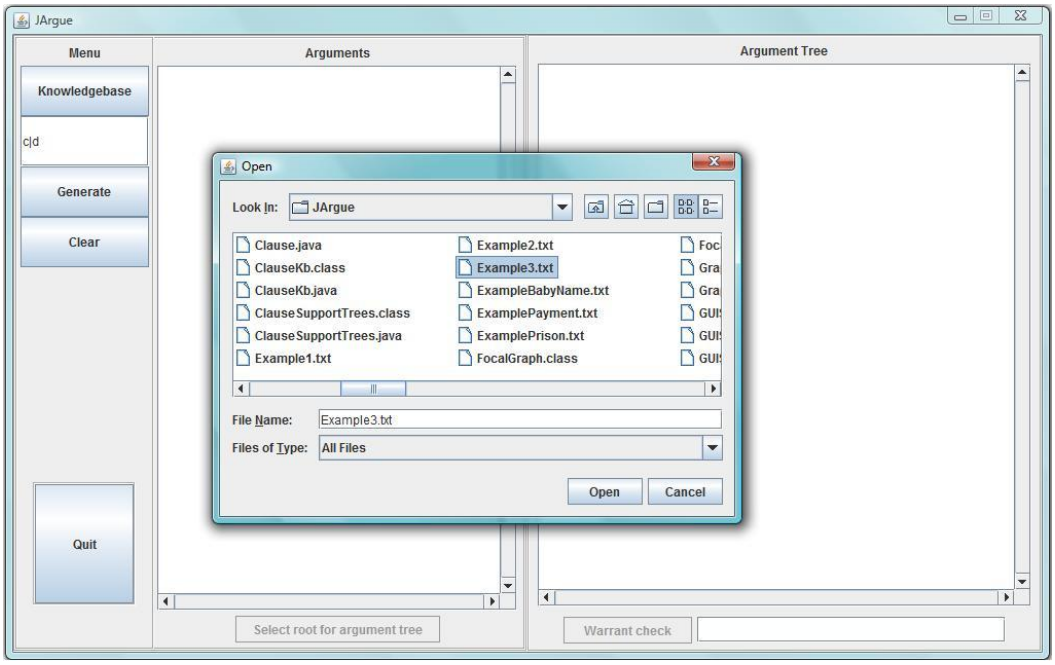
This section starts by describing the input format for JArgue and continues by demonstrating the functionality of the application.

6.2.1 Giving the input

The language of the input is composed as follows: '!' is used for the symbol of negation $\neg$, '|' is used for the symbol of disjunction $\vee$, and a propositional atom can be a string of characters other than the symbols introduced above and ',' or '&'. A clause is a disjunction of positive/negative atoms (i.e. literals) composed as described above. Some examples of clauses in this syntax are: $a, !a, a|b|c, !a|b, !c|!d$, $bird(Tweety)$, $!bird(Tweety)|flies(Tweety)$. A knowledgebase is written as a string of clauses where each clause is separated from its previous by ','. The string of the input knowledgebase has to be written in one line and saved in a text file located in the directory with the software (for example C:\workspace\JArgue \Example3.txt).

6.2.2 Functionality

A knowledgebase and a claim for arguments can be loaded to the system through the application window. The knowledgebase is loaded by a selection window that opens when pressing the 'knowledgebase' button. The claim is typed in the field provided.



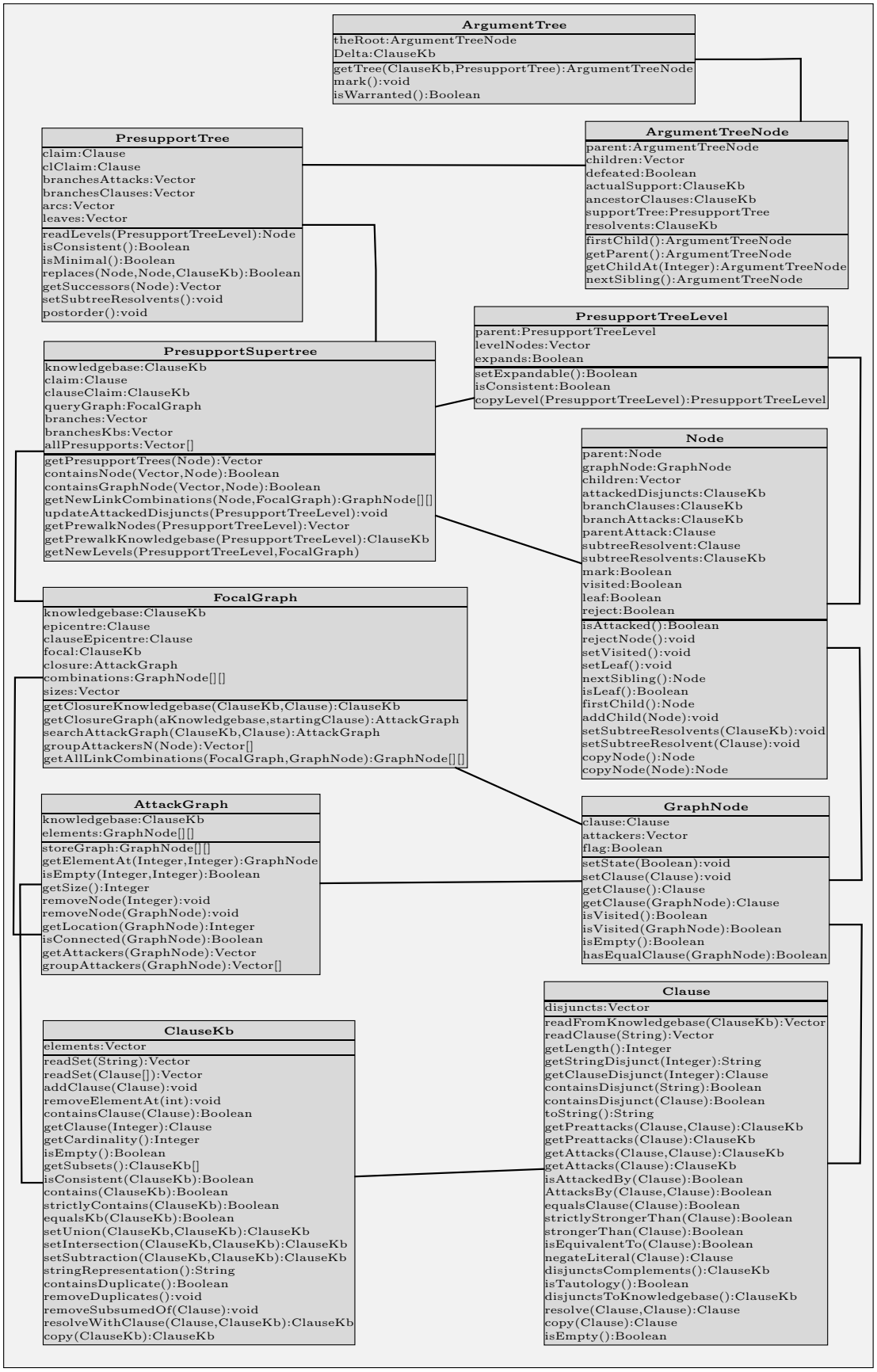
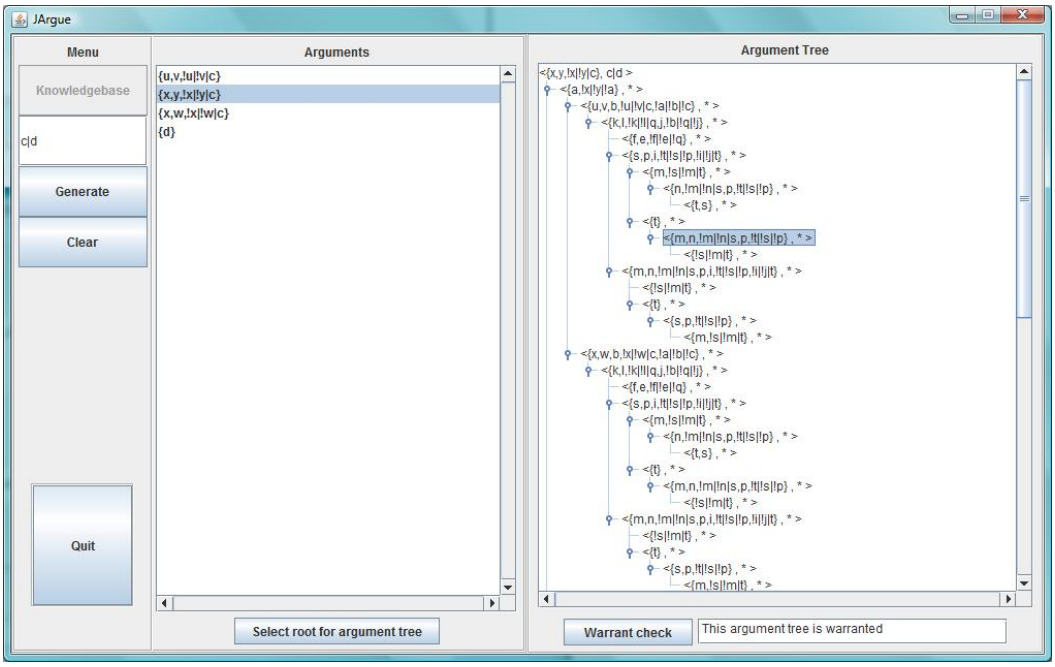
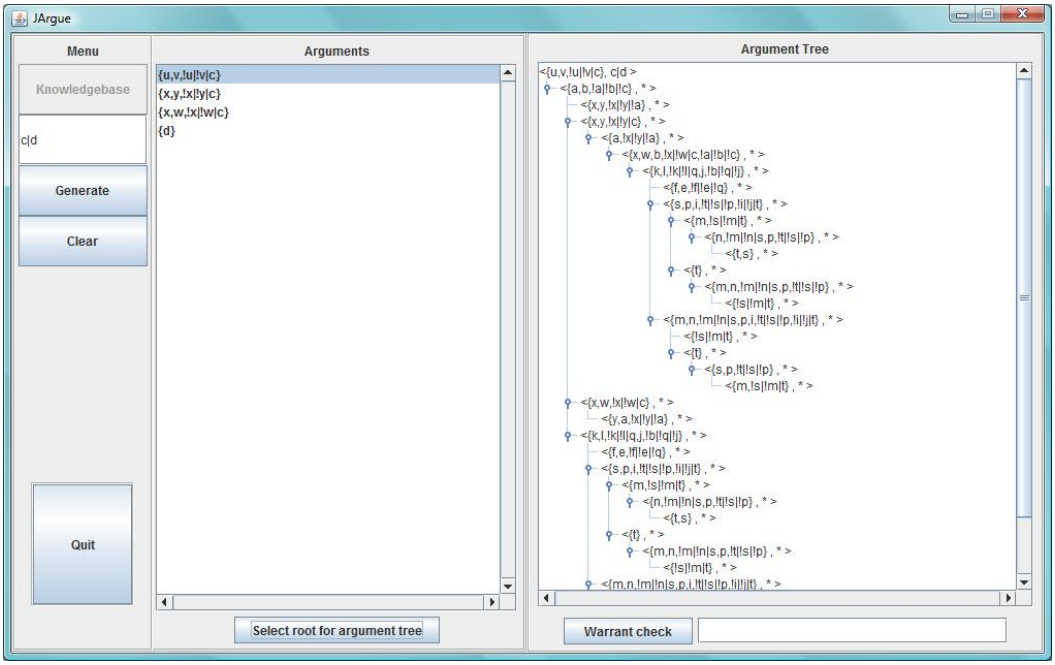


Figure 6.1: Diagram with the major classes of JArgue

By pressing the ‘Generate’ button, all the supports for arguments for the given query from the loaded knowledgebase are displayed in a list. By selecting one of these sets and pressing the ‘Select root for argument tree’ button, the argument tree with the chosen root is displayed on the right hand side of the window. Finally, by pressing the ‘Warrant check’ button the system tests whether this argument tree is warranted.

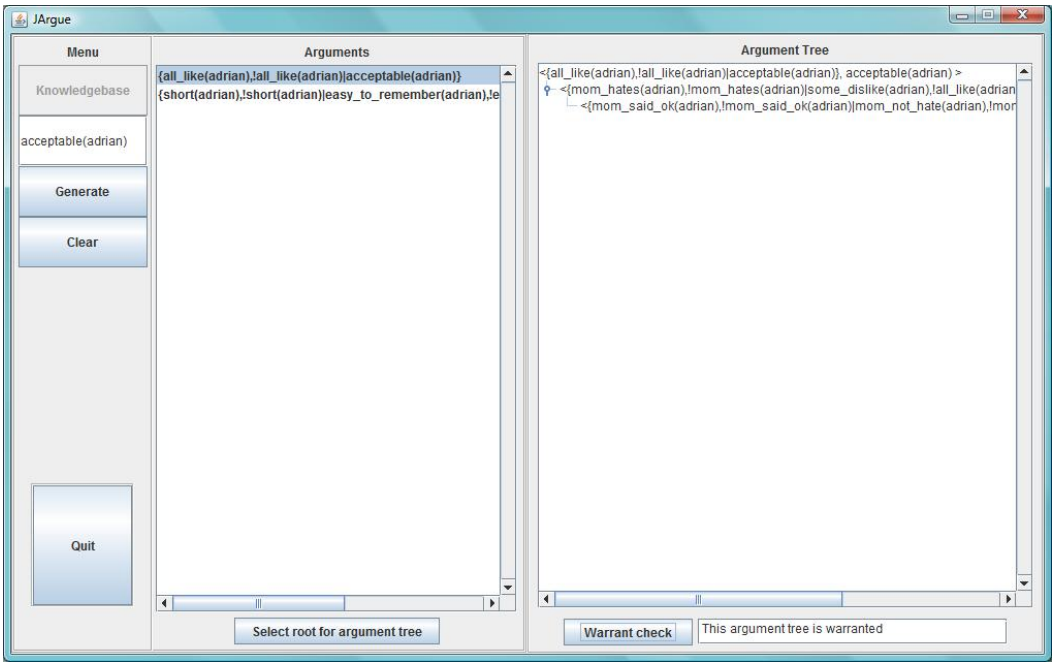


By selecting a different root argument from the list, a new argument tree can be generated for the newly selected root.



In addition, a different claim for arguments from the same knowledgebase can be loaded through the application window by deleting the old claim and retyping the new one in the claim field. By pressing the 'Clear' button all the inputs and output are cleared off the system and a new knowledgebase can be loaded. The knowledgebase of the following example is adapted from [42]. and concerns arguments for and against choosing a baby name. The claim for this example is: $acceptable(adrian)$ and the knowledgebase is

$$\Delta = \{\neg easy\_to\_remember(adrian) \vee acceptable(adrian), \neg all\_like(adrian) \vee acceptable(adrian),$$
$$\neg mom\_hates(adrian) \vee some\_dislike(adrian), \neg short(adrian) \vee easy\_to\_remember(adrian),$$
$$\neg dad\_hates(adrian) \vee some\_dislike(adrian), \neg too\_commom(adrian) \vee dad\_hates(adrian),$$
$$\neg uncle\_has(adrian) \vee dad\_hates(adrian), \neg mom\_said\_ok(adrian) \vee mom\_not\_hate(adrian),$$
$$\neg all\_like(adrian) \vee \neg some\_dislike(adrian), all\_like(adrian) \vee some\_dislike(adrian),$$
$$\neg mom\_hates(adrian) \vee \neg mom\_not\_hate(adrian), mom\_hates(adrian) \vee mom\_not\_hate(adrian),$$
$$mom\_said\_ok(adrian), short(adrian), all\_like(adrian), mom\_hates(adrian)\}$$



Apart from the tree representation displayed or the right hand side of the window the text representation of the last tree generated is stored in a text file which is located in the directory with the code. The content of this file is updated every time new argument tree is generated and replaced by the newly created argument tree.

6.3 Experimental evaluation

This section covers experimental evaluation on JArgue using randomly generated knowledgebases of fixed cardinality. Two different experiments were conducted where different kinds of input sets were used.

The first experiment involved generating 1000 non-empty argument trees using randomly generated knowledgebases of 50 distinct clauses each. 25 of these clauses were 2-place clauses (i.e. clauses that consist of 2 disjuncts) and the remaining 25 were 1-place clauses (literals). The claim of the root of each argument tree was a literal.

The knowledgebases were constructed out of 30 distinct atoms. Each atom was selected out of 30 available using java class `Random`, and was negated with probability 0.5, constructing this way a literal. Tuples of 2 literals that were generated with this method were then used to construct a 2-place clause. The number of 30 atoms was chosen for this experiment because for the fixed number of 25 2-place and 25 1-place input clauses, it gives a variation of results that range from very small single node trees to large trees that consist of hundreds of nodes. The largest argument tree produced in this experiment consists of 9137 nodes.

Table 6.1 presents the results of the experiment. The 1000 non-empty trees generated are separated in 8 different groups according to the number of nodes they contain. The first column identifies the interval in which lie the trees associated with the corresponding row of results. For example, a tree that consists of 6 nodes is associated with the first row of the table while a tree that consists of 10 nodes is associated with the second row of the table. The second column lists the percentage of trees that belong to the corresponding interval and the third column lists the average number of nodes in these trees. So for example, according to the values given in the first row of the table, 36.6% of the trees produced (i.e. 366 out of the 1000 produced overall) contain a number of nodes that is greater or equal to 1 and smaller than 10. The average number of nodes in these trees is 3. The fourth column of the table lists the average size of the query graph of the claim of the root in the knowledgebase. The fifth column lists the average support cardinality in the nodes of the corresponding argument trees. The sixth column lists the average number of branches of the trees in the corresponding interval and the seventh column gives the average depth of these trees, where the root node is considered to be at depth 0, every child of the root at depth 1 and from then on the depth increments by 1 for each level. The eigth column lists the average time taken in minutes in order to generate a tree in the interval by running the software on an ordinary PC. So, for example, with the values given in the first row, the average time taken to generate an argument tree with an average of 3 nodes is 1 to 4 milliseconds (which is rounded to zero in the table), the time taken in order to generate an argument tree with an average of 245 nodes is 0.15 mins (i.e. 9 seconds) and so on. The associated per node times in milliseconds are listed in the last column.

The average cardinality of the supports of the nodes of the trees in the different intervals varies between 1.56 and 2.35 with the lowest average of 1.56 corresponding to the smaller trees in interval $[1-10)$ and the highest average of 2.35 corresponding to the larger trees in the interval $[7500, 10000)$. The depth and width of an argument tree tend to increase concurrently, and as a result increase the overall

nodes interval	% trees	$\frac{\text{nodes}}{\text{tree}}$	|Query|	|support|	width	depth	$\frac{\text{mins}}{\text{tree}}$	$\frac{\text{msecs}}{\text{node}}$
$[1, 10)$	36.6	3	9.7	1.52	1.68	0.86	0.00	29.69
$[10, 100)$	32.2	39	20.3	2.04	19	3.57	0.02	34.02
$[100, 500)$	12.8	245	24.7	2.30	118	5.18	0.15	36.81
$[500, 1000)$	6.7	695	23.3	2.32	323	6.13	0.48	39.79
$[1000, 2500)$	4.8	1619	24.0	2.32	802	6.46	1.24	44.77
$[2500, 5000)$	4.4	3586	28.3	2.26	1812	7.57	3.64	59.02
$[5000, 7500)$	1.9	5958	28.9	2.31	3125	7.74	7.73	77.00
$[7500, 10000)$	0.6	8667	29.8	2.35	4332	7.83	17.63	120.5

Table 6.1: Experimental data on generating argument trees with knowledgebases of 1 and 2-place clauses

number of nodes in the tree. Increased depth of an argument tree indicates that there is a relatively high level of inconsistency in the search space and this is also reflected in the width of tree. This also suggests an increased density in the query graph where the clauses of the search space can be combined in many different ways together and give different sequences of arguments and counterarguments. By observing the results that concern the query graph sizes in the fourth column of the table it seems that the levels of inconsistency play a more important role in the size of an argument tree than the size of the search space. So for instance, it could be expected that trees in intervals $[100, 500)$ where the average tree size is 245 nodes would have their nodes generated from substantially reduced search spaces in comparison to trees in interval $[500, 1000)$ where the average tree size is 695 nodes. In the results of the experiment though this is not the case and in contrary the average query graph that corresponds to trees in interval $[100, 500)$ is larger that the average query graph that corresponds to trees in interval $[500, 1000)$. The fact that the size of the search space is not the dominant factor in determining the size of an argument tree can also be observed by comparing the results in the last three rows of the table where the average query graph sizes vary only from 28.3 to 29.8 for trees with average sizes from 3586 to 8667 nodes. So graphs that vary in size by one node on average produce argument trees that vary by more than 5000 nodes on average. The results in the last column of the table demonstrate how the average per node time increases with the size of the argument tree where a node belongs. This could be explained as follows: high levels of inconsistency indicate dense graphs which in turn cause an increased number of cycles that makes the search for arguments more complex.

A similar experiment was conducted that involved 3-place instead of 2-place clauses. Randomly generated knowledgebases of 25 identical 3-place and 25 identical 1-place clauses were constructed out of 30 identical atoms with the method described in the previous experiment. These knowledgebases were then used in order to construct argument trees for literal claims. Because the trees produced with this kind of input sets tend to contain a relatively small number of nodes (less than 10 nodes) and the sample of 1000 argument trees did not provide sufficient variation on the results, this experiment involved producing 2000 argument trees. The results of this experiment are displayed in table 6.2.

For the results of table 6.2 the same comments with the previous experiment apply in terms of how the values of different parameters vary. Again the time per node increases as the size of the argument

nodes interval	**% trees**	$\frac{\textbf{nodes}}{\textbf{tree}}$	$\lvert\mathsf{Query}\rvert$	$\lvert$**support**$\rvert$	**width**	**depth**	$\frac{\textbf{mins}}{\textbf{tree}}$	$\frac{\textbf{msecs}}{\textbf{node}}$
$[1, 10)$	47.85	3	21.66	2.22	2	0.92	0.01	113.29
$[10, 100)$	28.7	33	26.94	2.72	18	2.8	0.09	139.07
$[100, 500)$	14.25	241	30.73	3.06	136	4.76	0.6	147.62
$[500, 1000)$	4.4	719	29.42	3.15	405	5.65	1.86	158.16
$[1000, 2500)$	3.55	1584	29.65	3.34	895	6.08	4.2	162.45
$[2500, 5000)$	0.6	3375	28.75	3.19	1892	6.83	10.04	167.25
$[5000, 7500)$	0.65	6217	32.23	3.17	3835	7	38.5	392.92
$[7500, 10000)$	0	-	-	-	-	-	-	-

Table 6.2: Experimental data on generating argument trees with knowledgebases of 1 and 3-place clauses

tree increases, the depth and width of the tree increase together and the size of the search space does not seem to be the major factor in influencing the size of the trees produced.

By comparing tables 6.1 and 6.2 we can see that for trees within same ranges in the two tables, with knowledgebases that contain 3-place clauses the average support sizes are larger, and the trees are wider and less deep than with knowledgebases that contain 2-place clauses instead. The average supports are larger because having a larger number of disjuncts in the input clauses means that there is a larger number of literals that need to be resolved in order to produce a proof and hence additional clauses need to be added in a set in order to apply resolution. Larger supports in turn, tend to make the search for counterarguments more complicated and therefore the average time per node taken when producing argument trees is increased in this experiment compared to the previous experiment for trees with similar sizes. In addition, arguments with larger supports tend to have a larger number of canonical undercuts resulting this way in trees that are wider on average compared to trees from the previous experiment that have a smaller number of clauses in their supports. The average depth of trees in this experiment on the other hand tends to be smaller than in the first experiment for trees within same ranges. This could be explained as follows. Assume $A_n = \langle \Phi_n, \diamond \rangle$ is a canonical undercut and therefore a potential child of node A_{n-1} which is the current leaf on a branch with nodes $A_1, \ldots, A_{n-1}$ that have supports $\Phi_1, \ldots, \Phi_{n-1}$ respectively. Then, increasing number of elements in each of $\Phi_1, \ldots, \Phi_{n-1}$ has as a consequence for condition (2) of the definition of the argument tree ($\Phi_1 \cup \ldots \cup \Phi_{n-1} \subseteq \Phi_n$) to be reached at smaller depths than if supports $\Phi_1, \ldots, \Phi_{n-1}$ had smaller cardinality on average. Therefore, the argument trees in this experiment tend to stop extending in depth earlier that the ones in equivalent ranges from the previous experiment.

Figure 6.2 shows how the time per node varies with the size of the argument tree where a node belongs. Each curve corresponds to one of the experiments described above, one curve is for test sets that contain 3 and 1-place clauses and the other for test sets that contain 2 and 1-place clauses. Axis x contains the values for the average per node time in milliseconds for trees whose overall number of nodes is identified by the associated value on axis y. Each of the values on axis y is the upper limit of the interval where the trees that contain the corresponding nodes belong.

The fact that with this artificial data, argument trees that consist of hundreds of nodes can be ge-

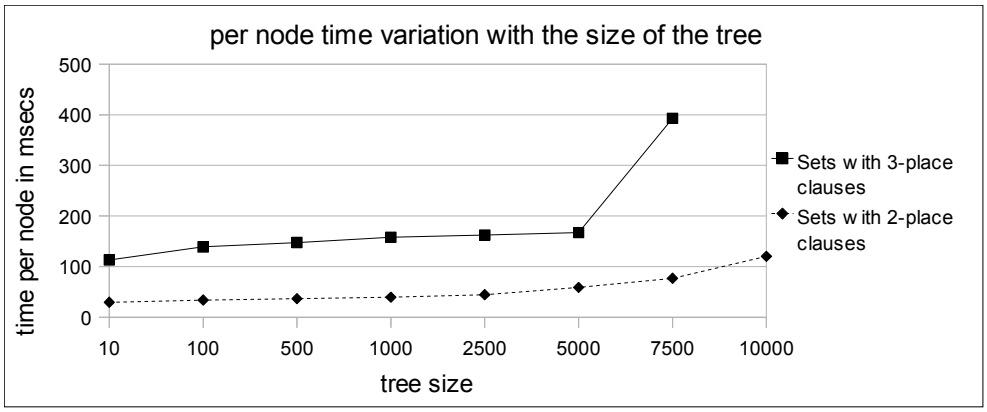


Figure 6.2: Comparison in time per argument tree node for knowledgebases with 2-place and 3-place clauses

nerated in less than a minute motivates for further experimentation and use of the system with actual data from real world applications. The system has also been tested with knowledgebases adapted from examples of hypothetical cases in published papers [41, 9, 42] and actual data from the medical domain [52]. These knowledgebases are listed in appendix A. The argument trees produced with these datasets were relatively small (5 nodes at most). Most of these trees were generated at fractions of a second while the trees produced using the medical knowledgebases where the length of the input string was large (8-55 characters for each literal) was a few seconds for a knowledgebase of 107 elements.

6.4 Discussion

This chapter presented JArgue, a software system developed in java that implements the theoretical work presented in chapters 3-5. Empirical evaluation of the system demonstrated the viability of the underlying algorithms. For knowledgebases containing 1 and 2-place clauses, argument trees consisting of 1 to 999 nodes were produced in less than 0.5 minutes on average, trees consisting of 1000 to less than 7500 nodes were produced in 1.24 to 7.73 minutes on average, while trees with 7500 up to 10000 nodes were produced on an average of 17.6 minutes. With knowledgebases of 1 and 3-place clauses, trees consisting of 1 to 999 nodes were produced in less than 2 minutes on average, trees consisting of 1000 to less than 5000 nodes were produced in 4.2 to 10 minutes on average and trees with 5000 to 7500 nodes were produced on an average of 38 minutes.

The performance of the system with this artificial data where argument trees of up to 1000 nodes were produced in less that 2 minutes motivates further experimentation and practical use of the software.

Chapter 7

Extending to first-order logic

Although classical propositional logic provides a language strong enough in many cases to represent detailed information and simulate human reasoning, it lacks some important features. In propositional logic information is represented as propositions that are simple declarative statements. First-order logic, on the other hand, involves predicates that can be used to express relations between individuals (e.g binary predicates) and quantification, that provides the means for generalising statements that can be instantianted by particular constants. In this way, it offers further advantages in representing knowledge, in comparison to propositional logic.

This chapter extends the work presented in chapters 3-4 for generating arguments in propositional logic to generating arguments in a first-order language of clauses. The chapter starts by defining this language and proceeds by introducing relations on its elements. It continues with definitions of graphs equivalent to the graphs for propositional clauses introduced in chapter 3 and then introduces trees that represent a search for arguments by walking over the graphs. It closes with an algorithm that implements this search for arguments in the language of first-order clauses.

7.1 Argumentation for a language of quantified clauses

For a first-order language $\mathcal{F}$, the set of formulae that can be formed is given by the usual inductive definitions of classical logic. In equivalence to the work on propositional logic presented so far which involves disjunctive clauses, the work of this chapter involves first-order disjunctive clauses. The language used here is a restricted function-free first-order language of quantified clauses $\mathcal{F}$ consisting of n-ary predicates ($n \geq 1$) where both existential and universal quantifiers are allowed. The arguments considered have claims that consist of one disjunct (i.e. unit clauses). This language is composed of the set of n-ary ($n \geq 1$) predicates $\mathcal{P}$, a set of constant symbols $\mathcal{C}$, a set of variables $\mathcal{V}$, the quantifiers $\forall$ and $\exists$, the connectives $\neg$ and $\vee$ and the bracket symbols ( ). The clauses of $\mathcal{F}$ are in prenex normal form, consisting of a quantification string followed by a disjunction of literals. Literals are trivially defined as positive or negative atoms where an atom is an n-ary predicate. The quantification part consists of a sequence of quantified variables that appear as parameters of the predicates of the clause. These need not follow some ordering, that is any type of quantifier (existential or universal) can preceed any type of quantifier.

Example 7.1.1 *If $\{a, b, d, e\} \subset \mathcal{C}$ and $\{x, y, z, w\} \subset \mathcal{V}$, then each of the following formulae is a clause in $\mathcal{F}$.*

$$\forall x \exists z (P(x) \vee \neg Q(z, a))$$
$$\exists x \exists z (P(x) \vee \neg Q(z, a)$$
$$\forall w \exists x \exists z (P(x) \vee \neg Q(z, a) \vee P(b, w, x, z))$$
$$\forall w (\neg Q(w, b, a))$$
$$\neg Q(e, b, a) \vee R(d), \neg P(a, d)$$

In addition $\forall w(\neg Q(w, b, a))$ is a unit clause, $\neg Q(e, b, a) \vee R(d)$ is a ground clause and $\neg P(a, d)$ is a ground unit clause.

Although the language used in this chapter is different to the one used in the rest of the thesis, the conditions for the definition for an argument are identical with the only difference that the knowledgebases and claims considered here are first-order clauses. In addition, for simplicity the claims are first-order unit clauses.

Example 7.1.2 *Consider the following knowledgebase where each element is from $\mathcal{F}$. $\Delta = \{\forall x(\neg P(x) \vee Q(x)),\ P(a), \forall x \forall y(P(x, y) \vee \neg P(x)), R(a, b),\ \exists x(R(x, b)),\ \exists x(\neg S(x, b))\}$. Some arguments from Δ are:*

$$\langle \{\forall x(\neg P(x) \vee Q(x)), P(a)\}, Q(a) \rangle \qquad \langle \{R(a, b)\}, \exists x(R(x, b)) \rangle$$
$$\langle \{\forall x \forall y(P(x, y) \vee \neg P(x)), P(a)\}, \forall y(P(a, y)) \rangle \qquad \langle \{P(a)\}, \exists y(P(y)) \rangle$$

7.2 Relations on first-order clauses

Applying resolution on first-order clauses raises difficulties and complications that are not introduced in the propositional case. In the propositional case, it is simple to identify complemenary literals and determine whether resolution can be applied for a pair of clauses. In the first-order case it is not as simple to distinguish complementary literals. Given a pair of first-order literals, in order to decide whether they are complementary, apart from their predicate and their sign, their quantification needs also to be taken into account. Considering these factors and by unifying a pair of literals we can then decide whether these are complementary or not. Moreover, in order to avoid producing tautologies when applying resolution to a pair of clauses, these clauses must have exactly one pair of complementary literals in between them. Since deciding whether a pair of literals are complementary requires considering the issues discussed above, then deciding whether resolution can be applied to a pair of clauses also requires considering these factors, and hence is much more complicated than in the propositional case. Defining the equivalent Preattacks and Attacks relations for first-order clauses requires some additional definitions introduced in this section.

In the functions on $\mathcal{F}$ defined in this chapter, the terms 'variable' and 'constant' are used in the usual way. In addition, a 'term' is either a variable or a constant. Functions $\mathsf{Variables}(X)$ and $\mathsf{Constants}(X)$ return the set of all variables and constants respectively that appear in a literal or a clause or a set X.

The set of bindings defined below for a language $\mathcal{L}$ denotes the set of all substitutions of a variable

x by the terms t in the language.

Definition 7.2.1 *For a language $\mathcal{L}$, with variables $\mathcal{V}$ and constant symbols $\mathcal{C}$, the set of all bindings $\mathcal{B}$ is $\{x/t \mid x \in \mathcal{V}$ and $t \in \mathcal{V} \cup \mathcal{C}\}$.*

A set of bindings B can be applied to a clause ϕ in order to create an instance of ϕ by function $\mathsf{Assign}(\phi, B)$ defined next.

Definition 7.2.2 *For a clause ϕ and a set of all bindings $B \subseteq \mathcal{B}$, $\mathsf{Assign}(\phi, B)$ returns clause ϕ with the values of B assigned to the terms of ϕ. So, for each x/t, if x is a variable in ϕ, then x is replaced by t and the quantifier of x is removed.*

Example 7.2.1 *Let $\phi = \exists x \forall y \exists z \forall w (P(c, x) \vee Q(x, y, z) \vee R(y, c, w))$. Some assignments for ϕ with the corresponding values from the given sets of bindings assigned to its variables are:*

$$B_1 = \{x/a, z/b\},\ \mathsf{Assign}(\phi, B_1) = \forall y \forall w (P(c, a) \vee Q(a, y, b) \vee R(y, c, w))$$
$$B_2 = \{x/a, y/b, z/b\},\ \mathsf{Assign}(\phi, B_2) = \forall w (P(c, a) \vee Q(a, b, b) \vee R(b, c, w))$$
$$B_3 = \{x/a, z/b, w/b\},\ \mathsf{Assign}(\phi, B_3) = \forall y (P(c, a) \vee Q(a, y, b) \vee R(y, c, b))$$
$$B_4 = \{x/a, y/a, z/b, w/b\},\ \mathsf{Assign}(\phi, B_4) = P(c, a) \vee Q(a, a, b) \vee R(a, c, b)$$
$$B_5 = \{w/z\},\ \mathsf{Assign}(\phi, B_5) = \exists x \forall y \exists z (P(c, x) \vee Q(x, y, z) \vee R(y, c, z))$$

Function $\mathsf{Assign}(\phi, B)$ gives a specific instance of ϕ, indicated by the bindings in B. The two functions defined next return all the possible instances for a clause ϕ and the elements of a set of clauses Ψ respectively. In the following, the symbol $\wp(B)$ denotes the power set of a set B.

Definition 7.2.3 *For a clause ϕ, $\mathsf{Assignments}(\phi)$ returns the set of all the possible instances of ϕ, while for a set of clauses Ψ, $\mathsf{SetAssignments}(\Psi)$ returns the set of all the possible instances of all the clauses in Ψ.*

$$\mathsf{Assignments}(\phi) = \{\mathsf{Assign}(\phi, B_i) \mid B_i \in \wp(\mathcal{B})\}.$$
$$\mathsf{SetAssignments}(\Psi) = \bigcup_{\phi \in \Psi} \{\mathsf{Assignments}(\phi)\}.$$

The assignment functions are used to create partial instances of the clauses from the knowledgebase during the search for arguments. Assuming instead a universal prenex form of the clauses as the basis of the search would make it hard to distinguish pairs of atoms that could be unified in between clauses. This would make it hard to distinguish pairs of complementary literals, and therefore to locate clauses on which resolution could be applied. As no restrictions apply to the order of the quantifiers in the quantification of a clause from $\mathcal{F}$, the scope of interchanging universal and existential quantifiers in a clause ϕ is taken into account when a partial instance of ϕ is created, like in Skolemization. For this, function $\mathsf{Prohibited}(\phi)$ is defined to return the sets of bindings that would not be allowed for ϕ in Skolemization.

Definition 7.2.4 *Let ϕ be a clause. Then, $\mathsf{Prohibited}(\phi) \subseteq \wp(\mathcal{B})$ returns the set of sets of bindings such that for each $B \in \mathsf{Prohibited}(\phi)$ there is at least one $y_i/t_i \in B$ such that y_i is a universally quantified*

variable which is in the scope of an existentially quantified variable x_i for which either $x_i = t_i$ or $x_i/t_i \in B$.

Example 7.2.2 *For the sets of bindings of example 7.2.1, $B_3, B_4, B_5 \in$ Prohibited(ϕ) and $B_1, B_2 \notin$ Prohibited(ϕ).*

Function ExistentialGrounding(ϕ, B) defined next, gives a partial instance of a clause ϕ where each of the existentially quantified variables is replaced by a distinct arbitrary constant from $\mathcal{C} \setminus$ Constants(ϕ). For simplicity, for the definition of this function, the set of bindings B that is assigned to ϕ in order to produce the partial instance of ϕ is from $\wp(\mathcal{B})$, including the bindings from Prohibited(ϕ). Later in the chapter, where the function is used in the search for arguments, the sets of bindings considered for ϕ are refined using function Prohibited(ϕ).

Definition 7.2.5 *For a clause ϕ,* ExistentialGrounding$(\phi, B) =$ Assign(ϕ, B) *where $B \in \wp(\mathcal{B})$ is such that:*

(1) $x_i/t_i \in B$ *iff* $x_i \in$ Variables(ϕ) *and x_i is existentially quantified*

(2) $t_i \in \mathcal{C} \setminus$ Constants(ϕ) *and*

(3) *for all $x_j/t_j \in B$, if $x_j \neq x_i$ then $t_j \neq t_i$.*

If $\phi' =$ ExistentialGrounding(ϕ, B) *for some $\phi \in \mathcal{F}$ and $B \in \wp(\mathcal{B})$, ϕ' is an* **existential instance** *of ϕ.*

Example 7.2.3 *For $\phi = \exists x \forall y \exists z \forall w (P(c, x) \vee Q(x, y, z) \vee R(y, c, w))$,* Constants$(\phi) = \{c\}$ *and so each of the elements of the set of constants $I = \{a, b\} \subset \mathcal{C} \setminus$* Constants$(\phi)$ *can be used for the substitution of each of the existentially bound variables x, z. For $B = \{x/a, z/b\}$,* ExistentialGrounding(ϕ, B) $= \forall y \forall w (P(c, a) \vee Q(a, y, b) \vee R(y, c, w))$.

The next definition introduces function Disjuncts(ϕ) for a first-order clause ϕ, that returns the set of literals (i.e. predicates) that appear as disjuncts in ϕ.

Definition 7.2.6 *For a clause $\phi = \mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m (p_1 \vee \ldots \vee p_k)$,* Disjuncts$(\phi)$ *returns the set of disjuncts of ϕ.* Disjuncts$(\phi) = \{p_1, \ldots, p_k\}$.

For a literal $p_i \in$ Disjuncts(ϕ), function Unit(ϕ, p_i) returns the unit clause that consists of p_i along with its corresponding quantification in ϕ.

Definition 7.2.7 *Let $\phi = \mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m (p_1 \vee \ldots \vee p_k)$. Then, for each $p_i \in$* Disjuncts(ϕ)*, function* Unit(ϕ, p_i) *returns the unit clause that has p_i as its unique disjunct and the part of the quantification $\mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m$ of ϕ that involves the variables that occur in p_i as its quantification:* Unit$(\phi, p_i) =$ $\mathsf{Q}_j x_j \ldots \mathsf{Q}_l x_l (p_i)$ *where $\{\mathsf{Q}_j x_j, \ldots, \mathsf{Q}_l x_l\} \subseteq \{\mathsf{Q}_1 x_1, \ldots, \mathsf{Q}_m x_m\}$ and $\{x_j, \ldots, x_l\} =$* Variables(p_i).

Example 7.2.4 *Let $\phi = \forall x \forall y \exists z (P(x) \vee Q(a) \vee \neg R(x, y, z, b) \vee S(a, b, c))$ and let $p = P(x), q = Q(a), r = \neg R(x, y, z, b)$ and $s = S(a, b, c)$. Then,* Disjuncts$(\phi) = \{p, q, r, s\}$ *and*

Unit$(\phi, p) = \forall x (P(x))$

Unit$(\phi, q) = Q(a)$

$\mathsf{Unit}(\phi, r) = \forall x \forall y \exists z(\neg R(x, y, z, b))$
$\mathsf{Unit}(\phi, s) = S(a, b, c)$

Function $\mathsf{Units}(\phi)$ returns the set of unit clauses produced by applying function $\mathsf{Unit}(\phi, p_i)$ to all the literals $p_i \in \mathsf{Disjuncts}(\phi)$.

Definition 7.2.8 *For a clause ϕ,* $\mathsf{Units}(\phi) = \{\mathsf{Unit}(\phi, p_i) \mid p_i \in \mathsf{Disjuncts}(\phi)\}$.

Example 7.2.5 *Continuing example 7.2.4, for $\phi = \forall x \forall y \exists z(P(x) \vee Q(a) \vee \neg R(x, y, z, b) \vee S(a, b, c))$,* $\mathsf{Units}(\phi) = \{\forall x(P(x)), Q(a), \forall x \forall y \exists z(\neg R(x, y, z, b)), S(a, b, c)\}$

The next definition formalises the contradiction relation between a pair of first-order unit clauses. This definition together with the Units function is used later in order to apply resolution on first-order clauses.

Definition 7.2.9 *Let ϕ and ψ be unit clauses. Then, ϕ and ψ* **contradict** *each other iff $\phi \vdash \neg\psi$. Then ψ is a complement of ϕ and this is denoted $\phi = \overline{\psi}$.*

Example 7.2.6 *According to definition 7.2.9, the following hold.*

$\neg P(a, b) = \overline{P(a, b)}$
$\forall x(Q(x, a)) = \overline{\neg Q(c, a)}$
$\exists x \exists y(S(x, y)) = \overline{\forall x \forall y(\neg S(x, y))}$

The two definitions that follow redefine the Attacks and Preattacks relations for pairs of first-order clauses.

Definition 7.2.10 *Let ϕ, ψ be clauses. Then,*

$$\mathsf{Preattacks}(\phi, \psi) = \{\alpha_i \in \mathsf{Units}(\phi) \mid \exists \alpha_j \in \mathsf{Units}(\psi) \text{ s.t. } \alpha_i = \overline{\alpha}_j\}.$$

Example 7.2.7 *According to definition 7.2.10, the following relations hold.*
7.2.7.1) $\mathsf{Preattacks}(\forall x(\neg N(x) \vee R(x)), N(a) \vee \neg R(b)) = \{\forall x(\neg N(x)), \forall x(R(x))\}$
7.2.7.2) $\mathsf{Preattacks}(\forall x(\neg N(x) \vee R(x)), N(a) \vee \neg R(a)) = \{\forall x(\neg N(x)), \forall x(R(x))\}$
7.2.7.3) $\mathsf{Preattacks}(P(a) \vee \neg Q(b), \neg P(a) \vee Q(b)) = \{P(a), \neg Q(b)\}$
7.2.7.4) $\mathsf{Preattacks}(\forall x(P(x) \vee \neg Q(a, x)), \exists x(\neg P(a) \vee Q(x, b))) = \{\forall x(P(x))\}$
7.2.7.5) $\mathsf{Preattacks}(\exists x(\neg P(a) \vee Q(x, b)), \forall x(P(x) \vee \neg Q(a, x)) = \{\neg P(a)\}$

Like in the propositional case, the Attacks relation is defined for a pair of clauses in the special case where the Preattacks relation returns a singleton set.

Definition 7.2.11 *For a pair of clauses ϕ, ψ, if for some $\alpha \in \mathsf{Units}(\phi)$,* $\mathsf{Preattacks}(\phi, \psi) = \{\alpha\}$ *then* $\mathsf{Attacks}(\phi, \psi) = \alpha$*, otherwise* $\mathsf{Attacks}(\phi, \psi) = \mathsf{Attacks}(\psi, \phi) = null$.

Example 7.2.8 *For examples 7.2.7.1, 7.2.7.2 and 7.2.7.3,* $\mathsf{Attacks}(\phi, \psi) = null$. *For example 7.2.7.4 it holds that* $\mathsf{Attacks}(\phi, \psi) = \forall x(P(x))$ *and for example 7.2.7.5, it holds that* $\mathsf{Attacks}(\phi, \psi) = \neg P(a)$.

Although the Attacks relation might be $null$ for a pair of clauses ϕ, ψ, it can sometimes have a non-$null$ value for instances of ϕ and ψ.

Example 7.2.9 *In Example 7.2.7.1, let* $\phi = \forall x(\neg N(x) \vee R(x))$ *and* $\psi = N(a) \vee \neg R(b)$. *Then* $|\mathsf{Preattacks}(\phi, \psi)| > 1$ *and so,* $\mathsf{Attacks}(\phi, \psi) = null$. *There are instances* ϕ' *of* ϕ *though for which* $\mathsf{Attacks}(\phi', \psi) \neq null$. *Let* $B_1 = \{x/a\}$, *and* $B_2 = \{x/b\}$. *Then for* $\phi_1 = \mathsf{Assign}(\phi, B_1) = \neg N(a) \vee R(a)$ *and* $\phi_2 = \mathsf{Assign}(\phi, B_2) = \neg N(b) \vee R(b)$, $\mathsf{Attacks}(\phi_1, \psi) = \neg N(a)$ *and* $\mathsf{Attacks}(\phi_2, \psi) = R(b)$. *For all other instances* ϕ' *of* ϕ $\mathsf{Attacks}(\phi', \psi) = null$.

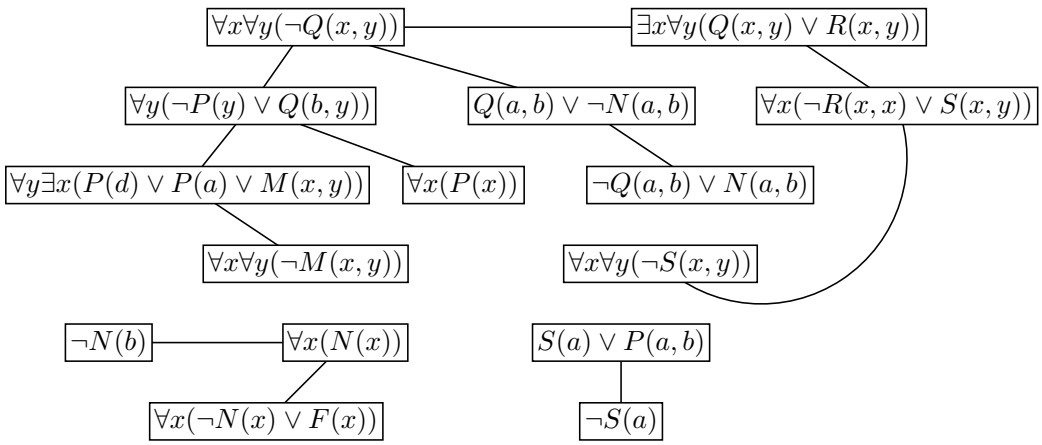
Example 7.2.10 *In example 7.2.7.2, let* $\gamma = \forall x(\neg N(x) \vee R(x))$ *and* $\delta = N(a) \vee \neg R(a)$. *Then, for all the instances* γ' *of* γ, $|\mathsf{Preattacks}(\gamma', \delta)| \neq 1$ *and so there is no instance* γ' *of* γ *for which* $\mathsf{Attacks}(\gamma', \delta) \neq null$.

7.3 Connection Graphs for first-order clauses

Similarly to the propositional case, given a set of first-order clauses it is possible to define graphs where the nodes are clauses from Δ and the arcs are defined using the Preattacks and Attacks functions for pairs of nodes. The definition for the connection graph in the first-order case remains the same with the definition for the propositional case.

Definition 7.3.1 *Let* Δ *be a clause knowledgebase of first-order clauses. The* **connection graph** *for* Δ, *denoted* $\mathsf{Connect}(\Delta)$, *is a graph* (N, A) *where* $N = \Delta$ *and* $A = \{(\phi, \psi) \mid \mathsf{Preattacks}(\phi, \psi) \neq \emptyset\}$.

Example 7.3.1 *Let* $\Delta = \{\forall x \forall y(\neg Q(x, y)), \exists x \forall y(Q(x, y) \vee R(x, y)), \forall y(\neg P(y) \vee Q(b, y)), Q(a, b) \vee \neg N(a, b), \forall x(\neg R(x, x) \vee S(x, y)), \neg Q(a, b) \vee N(a, b), \forall x(P(x)), \forall y \exists x(P(d) \vee P(a) \vee M(x, y)), \neg S(a), \forall x \forall y(\neg M(x, y)), \forall x \forall y(\neg S(x, y)), \forall x(N(x)), \neg N(b), \forall x(\neg N(x) \vee F(x)), S(a) \vee P(a, b)\}$. *Then, the following is the connection graph for* Δ.

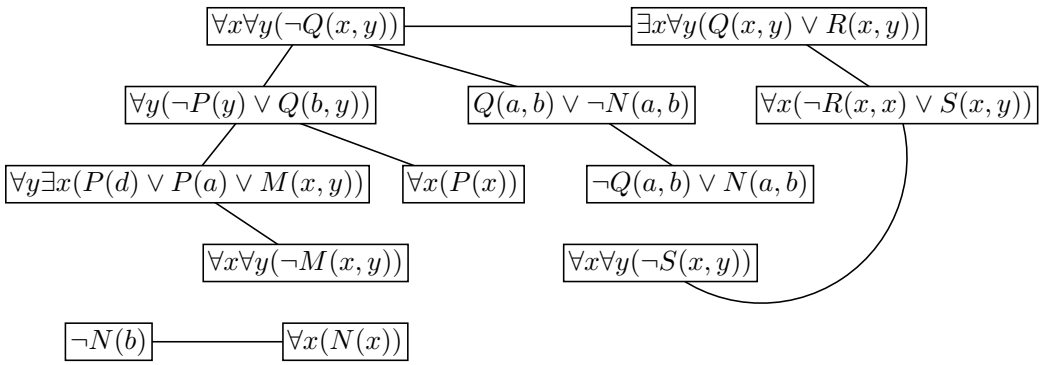


As demonstrated in example 7.2.9, for a pair of first-order clauses ϕ, ψ the Attacks relation might be $null$ but for instances ϕ', ψ' of ϕ, ψ the attack relation can be non-$null$. So, it is not always possible to identify whether for a pair of clauses that is such that $\mathsf{Preattacks}(\phi, \psi) \neq \emptyset$ there can be instances

ϕ', ψ' for which the Attacks relation is not null unless specific instances of the clauses are produced. For this reason, in the first-order case the closed graph is defined in terms of the Preattacks relation.

Definition 7.3.2 *Let Δ be a clause knowledgebase. The* **closed graph** *for Δ, denoted* $\mathsf{Closed}(\Delta)$*, is the largest subgraph (N, A) of* $\mathsf{Connect}(\Delta)$*, such that for each $\phi \in N$, for each $\beta \in$* $\mathsf{Disjuncts}(\phi)$ *there is a $\psi \in N$ with $\beta \in$* $\mathsf{Preattacks}(\phi, \psi)$.

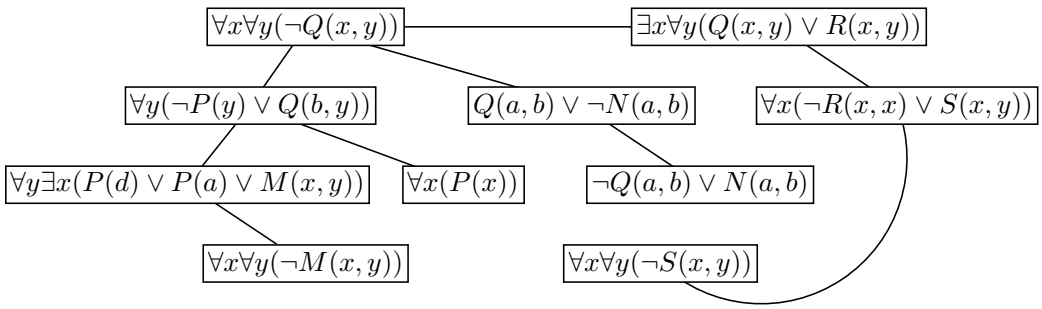
Example 7.3.2 *Continuing example 7.3.1, the following is the closed graph for Δ.*



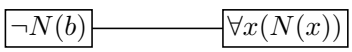
Given a clause ϕ from Δ, the focal graph of ϕ in Δ is the component of $\mathsf{Closed}(\Delta)$ that contains ϕ.

Definition 7.3.3 *Let Δ be a clause knowledgebase and $\phi \in \Delta$. The* **focal graph** *of ϕ in Δ denoted* $\mathsf{Focal}(\Delta, \phi)$ *is defined as follows: If there is a component X in* $\mathsf{Closed}(\Delta)$ *containing node ϕ, then* $\mathsf{Focal}(\Delta, \phi) = X$*, otherwise* $\mathsf{Focal}(\Delta, \phi)$ *is the empty graph. Clause ϕ is called the* **epicentre** *of the focal graph.*

Example 7.3.3 *Continuing example 7.3.1, the following is the focal graph of $\psi = \forall x \forall y(\neg Q(x,y))$ in Δ.*



Moreover, the following is the focal graph of $\phi = \forall x(N(x))$ in Δ.



The query graph of a clause α in Δ is defined below. For simplicity, in the first-order case consideration is limited to queries that are unit clauses. Then, the negation $\neg\alpha$ of α is also a disjunctive clause consisting of a unique disjunct and the query graph of α in Δ consists of a unique component.

Definition 7.3.4 *Let Δ be a set of first-order clauses and α be a first-order unit clause. Then, the* **query graph** *of α in Δ, denoted $\mathsf{Query}(\Delta, \alpha)$, is the focal graph of $\neg\alpha$ in $\Delta \cup \{\neg\alpha\}$.*

Example 7.3.4 *Let $\Delta' = \{\exists x \forall y(Q(x,y) \vee R(x,y)), \forall y(\neg P(y) \vee Q(b,y)), Q(a,b) \vee \neg N(a,b), \forall x P(x), \forall x(\neg R(x,x) \vee S(x,y)), \neg Q(a,b) \vee N(a,b), \forall y \exists x(P(d) \vee P(a) \vee M(x,y)), \neg S(a), \forall x \forall y(\neg M(x,y)), \forall x \forall y(\neg S(x,y)), \forall x(\neg N(x) \vee S(x)), \forall x(N(x)), S(a) \vee P(a,b)\}$. Then, $\Delta' = \Delta \setminus \{\forall x \forall y(\neg Q(x,y))\}$ where Δ is the knowledgebase of example 7.3.1. Moreover, if $\alpha = \exists x \exists y(Q(x,y))$, then $\neg\alpha = \forall x \forall y(\neg Q(x,y))$ which is equal to ψ from example 7.3.3 and $\Delta = \Delta' \cup \{\neg\alpha\}$. Since the query graph of α in Δ' is by definition equal to the focal graph of $\neg\alpha$ in $\Delta' \cup \{\neg\alpha\}$, then $\mathsf{Query}(\Delta', \alpha)$ is equal to $\mathsf{Focal}(\Delta, \neg\alpha)$ which is the graph of example 7.3.3.*

For the query graph, the following proposition holds with respect to the search for arguments for a claim α that is a unit clause .

Proposition 7.3.1 *Let α be a first-order unit clause and let $(N, A) = \mathsf{Query}(\Delta, \alpha)$. If $\langle \Phi, \alpha \rangle$ is an argument (where $\Phi \subseteq \Delta$), then $\Phi \subset N$.*

Proof: Since $\langle \Phi, \alpha \rangle$ is an argument, then $\Phi \cup \{\neg\alpha\}$ is a minimal inconsistent set and so there exists a closed tableau for this set, and there is no $\Gamma \subset \Phi \cup \{\neg\alpha\}$ such that Γ is inconsistent, so there is no $\Gamma \subset \Phi \cup \{\neg\alpha\}$ that has a closed tableau. In order for a tableau of $\Phi \cup \{\neg\alpha\}$ to be closed, for each $\phi \in \Phi \cup \{\neg\alpha\}$, for each $\beta \in \mathsf{Units}(\phi)$, there should be at least one clause $\phi' \in \Phi \cup \{\neg\alpha\}$ with $\overline{\beta} \in \mathsf{Units}(\phi')$. Otherwise, if for some ϕ this does not hold, then either there should be at least one branch of the tableau which is open, or the tableau for $\Phi \cup \{\neg\alpha\} \setminus \{\phi\}$ is closed and this contradicts the assumption that $\Phi \cup \{\neg\alpha\}$ is a minimal inconsistent set. Since it holds that for each $\phi \in \Phi \cup \{\neg\alpha\}$, for each $\beta \in \mathsf{Units}(\phi)$, there is at least one clause $\phi' \in \Phi \cup \{\neg\alpha\}$ such that $\overline{\beta} \in \mathsf{Units}(\phi')$, then it holds that for each $\phi \in \Phi \cup \{\neg\alpha\}$, for each $\beta \in \mathsf{Units}(\phi)$, there is at least one clause $\phi' \in \Phi \cup \{\neg\alpha\}$ with $\beta \in \mathsf{Preattacks}(\phi, \phi')$ so for all $\phi \in \Phi \cup \{\neg\alpha\}$ and for all $\beta \in \mathsf{Units}(\phi)$ there is a $\phi' \in \Phi \cup \{\neg\alpha\}$ such that $\beta \in \mathsf{Preattacks}(\phi, \phi')$ hence for all $\phi \in \Phi \cup \{\neg\alpha\}$, it holds that $\phi \in \mathsf{Closed}(\Phi \cup \{\neg\alpha\})$. Since this holds for all $\phi \in \Phi \cup \{\neg\alpha\}$ it also holds for $\neg\alpha$ that $\neg\alpha \in \mathsf{Closed}(\Phi \cup \{\neg\alpha\})$. Moreover, since $\Phi \cup \{\neg\alpha\} \subseteq \Delta \cup \{\neg\alpha\}$ and $\neg\alpha \in \mathsf{Closed}(\Phi \cup \{\neg\alpha\})$, it holds that $\mathsf{Closed}(\Phi \cup \{\neg\alpha\})$ is a subgraph of $\mathsf{Focal}(\Delta \cup \{\neg\alpha\}, \neg\alpha)$ and by definition $\mathsf{Focal}(\Delta \cup \{\neg\alpha\}, \neg\alpha) = \mathsf{Query}(\Delta, \alpha)$ and so it also holds that $\mathsf{Closed}(\Phi \cup \{\neg\alpha\})$ is a subgraph of $\mathsf{Query}(\Delta, \alpha)$. Therefore, $\Phi \cup \{\neg\alpha\} \subseteq N$ and also $\neg\alpha \notin \Phi$ since Φ is a support for an argument for α (otherwise it would hold that $\Phi \cup \{\neg\alpha\} = \Phi$ and $\Phi \vdash \bot$ since $\Phi \cup \{\neg\alpha\} \vdash \bot$), so $\Phi \subset N$. □

Example 7.3.5 *Consider the query graph (N, A) of $\alpha = \exists x \exists y(Q(x,y))$ in Δ from example 7.3.4. Some arguments for α from Δ are:*

$$\langle \Phi_1, \alpha \rangle = \langle \{\forall y(\neg P(y) \vee Q(b,y)), \forall x(P(x))\}, \exists x \exists y(Q(x,y)) \rangle$$
$$\langle \Phi_2, \alpha \rangle = \langle \{\forall y(\neg P(y) \vee Q(b,y)), \forall y \exists x(P(d) \vee P(a) \vee M(x,y)), \forall x \forall y(\neg M(x,y))\}, \exists x \exists y(Q(x,y)) \rangle$$

For Φ_1 and Φ_2 it holds that $\Phi_1 \subset N$ and $\Phi_2 \subset N$.

Like in the propositional case, the query graph of α in Δ can be used as the search space when looking for arguments for α from Δ. By applying search algorithms based on the structure of the graph it is possible to obtain arguments for α. This is the topic of the next section.

7.4 Proof trees for first-order clauses

Given a first-order unit clause α as a claim for an argument, it is possible to retrieve a support for an argument for α by walking over the query graph of α in Δ and applying resolution. This requires a unification process, between complementary literals that indicate paths traversed during the search. The selection of the paths to be traversed takes place in a similar way to that in the propositional case. An assignment tree depicts the process of walking over the query graph of α in Δ and unifying literals that indicate paths in the search for arguments for α, and hence represents a tentative proof of an argument for α.

Definition 7.4.1 *Let Δ be a clause knowledgebase and α be a unit clause and let $\Delta' = \Delta \cup \{\neg\alpha\}$. An* **assignment tree** *for Δ and α is tuple (N, A, e, f, g, h) where N is a set of nodes and A is a set of arcs such that (N, A) is a tree and e, f, g, h are functions such that: $e : N \mapsto \Delta'$, $f : N \mapsto \mathsf{SetAssignments}(\Delta')$, $g : N \mapsto \wp(\mathcal{B})$, $h : N \mapsto \mathsf{SetAssignments}(\Delta')$ and*

(1) *if p is the root of the tree, then $e(p) = \neg\alpha$*
(2) *f(p) is an existential instance of $e(p)$ s.t. $\mathsf{Constants}(f(p)) \subseteq \mathsf{Constants}(g(p))$*
(3) *for any nodes p, q in the same branch, if $e(p) = e(q)$ then $g(p) \neq g(q)$*
(4) *for all $p \in N, g(p) \cap \mathsf{Prohibited}(e(p)) = \emptyset$*
(5) *for all $p \in N, h(p) = \mathsf{Assign}(f(p), g(p))$*
(6) *for all $p, q \in N$, if p is the parent of q, then $\mathsf{Attacks}(h(q), h(p)) \neq null$*
(7) *for all $p, q \in N$, $(\mathsf{Constants}(f(p)) \setminus \mathsf{Constants}(e(p))) \bigcap \mathsf{Constants}(\Delta') = \emptyset$, &*
$(\mathsf{Constants}(f(p)) \setminus \mathsf{Constants}(e(p))) \bigcap (\mathsf{Constants}(f(q)) \setminus \mathsf{Constants}(e(q))) = \emptyset$

Each of the functions e, f, g, h for a node p gives the state of the tentative proof for an argument for α. Function $e(p)$ identifies for p the clause ϕ from $\Delta \cup \{\neg\alpha\}$ associated with node p and $f(p)$ is an existential instance of $e(p)$. $g(p)$ is a set of bindings that when assigned to $e(p)$ creates the instance $h(p)$ of $e(p)$. Hence, $g(p)$ contains the set of bindings that create the existential instance $f(p)$ of $e(p)$ together with the bindings that unify atoms of contradictory literals connected with arcs on the tree as condition 6 indicates. Condition 7 ensures that the existential instances used in the proof are created by assigning to the existentially quantified variables of a clause $e(p)$ constants that do not appear anywhere else in $\Delta \cup \{\neg\alpha\}$ or the other instances of the clauses of the tentative proof. Finally, condition 3 ensures that an infinite sequence of identical nodes on a branch will be avoided in an assignment tree.

In all the examples that follow, assignment trees are represented by the value $h(p)$ for each node p, where no existentially bound variables appear. Hence, all the variables that appear in a tree representation are universally quantified and so universal quantifiers are omitted for simplicity.

Example 7.4.1 *Let* $\Delta = \{\forall y(\neg P(y) \vee Q(b,y)), \forall y \exists x(P(d) \vee P(a) \vee M(x,y)), R(c), \forall x \forall y(\neg M(x,y)), \exists x \forall y(Q(x,y) \vee R(x,y)), Q(a,b) \vee \neg N(a,b), \forall x \forall y(L(x,y,a)), \forall x(\neg R(x,x) \vee S(x,y)), \neg Q(a,b) \vee N(a,b), \forall x \forall y(\neg S(x,y)), \neg L(c,d,a), \forall x(P(x)), \neg R(c,a)\}$. *The following is an assignment tree for* Δ *and* $\alpha = \exists x \exists y(Q(x,y))$.

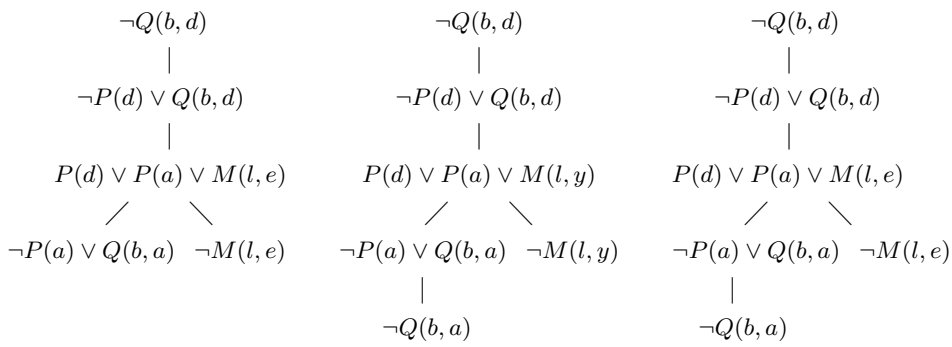
$$
\begin{array}{ccc}
\neg Q(b,d)_{\ p_0} & & e(p_0) = f(p_0) = \forall x \forall y(\neg Q(x,y)), g(p_0) = \{x/b, y/d\} \\
| & & e(p_1) = f(p_1) = \forall y(\neg P(y) \vee Q(b,y)), g(p_1) = \{y/d\} \\
\neg P(d) \vee Q(b,d)_{\ p_1} & & e(p_2) = \forall y \exists x(P(d) \vee P(a) \vee M(x,y)), g(p_2) = \{x/l\} \\
| & & f(p_2) = \forall y(P(d) \vee P(a) \vee M(l,y)) \\
P(d) \vee P(a) \vee M(l,y)_{\ p_2} & & e(p_3) = f(p_3) = \forall y(\neg P(y) \vee Q(b,y)), g(p_3) = \{y/a\} \\
/ \qquad \backslash & & e(p_4) = f(p_4) = \forall x \forall y(\neg M(x,y)), g(p_4) = \{x/l\} \\
\neg P(a) \vee Q(b,a)_{\ p_3} \quad \neg M(l,y)_{\ p_4} & &
\end{array}
$$

An assignment tree for Δ and α does not necessarily indicate a proof for α. It provides a structure which, with additional constraints does provide a proof for α. Some of these constraints are introduced in the definition for a complete assignment tree.

Definition 7.4.2 *A* **complete assignment tree** (N, A, e, f, g, h) *is an assignment tree such that for any* $x \in N$ *if* y *is a child of* x *then there is a* $\overline{b}_i \in \mathsf{Units}(h(x))$ *such that* $\mathsf{Attacks}(h(y), h(x)) = b_i$ *and for each* $b_j \in \mathsf{Units}(h(y)) \setminus \{b_i\}$
(1) either there is exactly one child z *of* y *s.t.* $\mathsf{Attacks}(h(z), h(y)) = \overline{b}_j$
(2) or there is a node w *in the branch containing* y *s.t.* $b_j = \mathsf{Attacks}(h(y), h(w))$

Definition 7.4.3 *A* **grounded assignment tree** (N, A, e, f, g, h) *is an assignment tree such that for any* $x \in N$, $h(x)$ *is a ground clause.*

Example 7.4.2 *The assignment tree of example 7.4.1 is neither complete nor grounded. It is not a complete assignment tree because for* $Q(b,a) \in \mathsf{Units}(h(p_3))$ *the conditions of definition 7.4.2 do not hold. Adding a node* p_5 *as a child of* p_3 *with* $e(p_5) = f(p_5) = \forall x \forall y(\neg Q(x,y))$, $g(p_5) = \{x/b, y/a\}$ *for which* $h(p_5) = \neg Q(b,a)$ *gives a complete assignment tree. It is not a grounded assignment tree because for nodes* p_2 *and* p_4 $h(p_2) = P(d) \vee P(a) \vee M(l,y)$ *and* $h(p_4) = \neg M(l,y)$ *are non-ground clauses. If we substitute the non-ground term* y *in* $h(p_2)$ *and* $h(p_4)$ *with the same arbitrary constant value (*$e \in \mathcal{C}$ *for instance), the resulting tree still satisfies the conditions for an assignment tree and it is also a grounded assignment tree.*



assignment tree 1 **(grounded)** **assignment tree 2** **(complete)** **assignment tree 3** **(grounded & complete)**

For a complete grounded assignment tree the following holds regarding the entailment of the grounded clause represents the root.

Lemma 7.4.1 *If (N, A, e, f, g, h) is a complete grounded assignment tree for Δ and α, where p_0 is the root, then $\{h(p) \mid p \in N \setminus \{p_0\}\} \vdash \neg h(p_0)$.*

Proof: This proof proceeds as follows. It is proved first that a complete grounded assignment tree (N, A, e, f, g, h) is isomorphic to a support tree (N', A', f') and then by regarding the ground clauses of (N, A, e, f, g, h) as propositional clauses and using the properties of a complete presupport tree presented in chapter 4, the lemma is shown to hold.

Let (N, A, e, f, g, h) be a complete grounded assignment tree for Δ and α and let p_0 be its root (i.e. $e(p_0) = \neg\alpha$). Then, (N, A, e, f, g, h) is isomorphic to a complete presupport tree (N', A', f') for $\Delta' = \{h(p) \mid p \in N\}$, $\alpha' = \neg h(p_0)$ and $a' = \neg h(p_0)$ where for each $p \in N$, there is a $p' \in N'$ such that $h(p) = f'(p')$ and for each node $p' \in N'$, there is a $p \in N$ such that $f'(p') = h(p)$ (and hence $\{h(p) \mid p \in N\} = \{f'(p') \mid p' \in N'\}$). Let p'_0 be the root of (N', A', f') (i.e. $f'(p_0) = h(p_0)$)

Then, (N', A', f') satisfies the conditions of definition 4.2.1 for Δ', α' and a':
Condition (1) of definition 4.2.1 holds for (N', A', f') because condition (1) of definition 7.4.1 and condition (1) of definition 7.4.2 hold for (N, A, e, f, g, h).
Condition (2) of definition 4.2.1 holds for (N', A', f') because condition (3) of definition 7.4.1 holds for (N, A, e, f, g, h).
Condition (3) of definition 4.2.1 holds for (N', A', f') because condition (6) of definition 7.4.1 holds for (N, A, e, f, g, h).

Moreover, (N', A', f') satisfies the conditions of definition 4.2.3. For each non-root $x' \in N'$, for every $b \in \mathsf{Disjuncts}(f'(x'))$:
Condition (i) of definition 4.2.3 does not hold because α' consists of a unique disjunct so $\mathsf{Disjuncts}(\alpha') \setminus \{a'\} = \{\neg h(p_0)\} \setminus \{\neg h(p_0)\} = \emptyset$ and hence $b \notin \emptyset$. Then, for $b \in \mathsf{Disjuncts}(f'(x'))$, if

$x \in N$ is such that $f'(x') = h(x)$
either condition (ii) of definition 4.2.3 holds for $b \in \mathsf{Disjuncts}(f'(x'))$ in (N', A', f') because condition (2) of definition 7.4.2 holds for $b \in \mathsf{Units}(h(x))$ in (N, A, e, f, g, h), or
condition (iii) of definition 4.2.3 holds for $b \in \mathsf{Disjuncts}(f'(x'))$ in (N', A', f') because condition (1) of definition 7.4.2 holds for $b \in \mathsf{Units}(h(x))$ in (N, A, e, f, g, h).
So, (N', A', f') satisfies definition 4.2.3 and (N', A', f') is a complete presupport tree for Δ', α' and a' where $\{f'(p') \mid p' \in N'\} = \{h(p) \mid p \in N\}$. Then, by proposition 4.2.7, $\{f'(p') \mid p' \in N' \setminus \{p'_0\}\} \vdash \alpha'$ and because $\{f'(p') \mid p' \in N' \setminus \{p'_0\}\} = \{h(p) \mid p \in N \setminus \{p_0\}\}$, then $\{h(p) \mid p \in N \setminus \{p_0\}\} \vdash \alpha'$ and since $\alpha' = \neg h(p_0)$, then $\{h(p) \mid p \in N \setminus \{p_0\}\} \vdash \neg h(p_0)$. □

Using the last lemma, the following proposition can be proved.

Proposition 7.4.1 *If (N, A, e, f, g, h) is a complete grounded assignment tree for Δ and α, where p_0 is the root then $\{e(p) \mid p \in N\} \setminus \{e(p_0)\} \vdash \alpha$.*

Proof: By lemma 7.4.1, $\{h(p) \mid p \in N \setminus \{p_0\}\} \vdash \neg h(p_0)$ where by the definition for an assignment tree $h(p_0)$ is an instance of $e(p_0)$ and $e(p_0) = \neg\alpha$. Then, $(\{h(p) \mid p \in N \setminus \{p_0\}\}) \cup \{h(p_0)\}$ is an unsatisfiable grounding of $(\{e(p) \mid p \in N \setminus \{p_0\}\}) \cup \{e(p_0)\} = \{e(p) \mid p \in N\}$. Hence, since $e(p_0) = \neg\alpha$, it follows that $\neg\alpha \in \{e(p) \mid p \in N\}$ and since $\{e(p) \mid p \in N\} \vdash \bot$ then $\{e(p) \mid p \in N\} \setminus \{\neg\alpha\} \vdash \alpha$ which is equivalent to $\{e(p) \mid p \in N\} \setminus \{e(p_0)\} \vdash \alpha$. □

Example 7.4.3 *For the complete and grounded assignment tree of example 7.4.2, $\{e(p) \mid p \in N\} \setminus \{\neg\alpha\} = \{\forall y(\neg P(y) \vee Q(b, y)), \forall y \exists x(P(d) \vee P(a) \vee M(x, y)), \forall x \forall y(\neg M(x, y))\} \vdash \alpha$ where $\alpha = \exists x \exists y(Q(x, y))$.*

Although all the assignment trees in example 7.4.2 correspond to the same subset of clauses $e(p)$ from Δ, it is not always the case that a non-grounded or non-complete assignment tree is sufficient to indicate a proof for α.

The following definitions introduce additional constraints on the definition of a complete assignment tree (N, A, e, f, g, h) for Δ and α that give properties related to the minimality and the consistency of the proof for α indicated by the set of nodes in the assignment tree. In the following, for an arc (q, p) of an assignment tree (N, A, e, f, g, h), p denotes the parent of q.

Definition 7.4.4 *Let (N, A, e, f, g, h) be a complete assignment tree for Δ and α. Then, (N, A, e, f, g, h) is* **a minimal assignment tree** *if for any arcs (q, p), (q', p') where*

$\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q))$ *for some* $\beta \in \mathsf{Units}(e(q))$*, and*
$\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$ *for some* $\beta' \in \mathsf{Units}(e(q'))$

$\beta \vdash \beta'$ holds iff $e(q) = e(q')$.

Example 7.4.4 *The following (N, A, e, f, g, h) is a complete assignment tree for a knowledgebase Δ and $\alpha = \exists x(\neg M(x))$, with $\{e(p) \mid p \in N\} = \{\forall x(M(x)), \forall x(\neg S(x) \vee \neg M(x) \vee \neg T(x)), \forall x(S(x) \vee N(x)), \forall x(T(x) \vee N(x)), \forall x(\neg N(x)), \forall x(\neg N(x) \vee R(x)), \forall x(\neg R(x))\} \subseteq \Delta \cup \{\neg\alpha\}$. (N, A, e, f, g, h)*

is not minimal because of arcs (q,p), (q',p'). *In this tree,* $g(q) = \emptyset$ *and* $g(q') = \emptyset$. *For* $\beta = \forall x(\neg N(x)) \in \mathsf{Units}(e(q))$ *and* $\beta' = \forall x(\neg N(x)) \in \mathsf{Units}(e(q'))$ *it holds that:*

$\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q)) = \forall x(\neg N(x))$ *and*

$\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q')) = \forall x(\neg N(x))$

and $\beta \vdash \beta'$ *but* $e(q) \neq e(q')$.

$$
\begin{array}{c}
M(x) \\
| \\
\neg S(x) \vee \neg M(x) \vee \neg T(x) \\
\diagup \qquad \diagdown \\
S(x) \vee N(x)_p \qquad\qquad T(x) \vee N(x)_{p'} \\
| \qquad\qquad\qquad | \\
\neg N(x)_q \qquad\qquad \neg N(x) \vee R(x)_{q'} \\
\qquad\qquad\qquad\qquad | \\
\qquad\qquad\qquad\qquad \neg R(x)
\end{array}
$$

If a copy of the subtree rooted at q' *in* (N, A, e, f, g, h) *is substituted by the subtree rooted at* q, *a minimal assignment tree* (N', A', e', f', g', h') *with* $\{e(p) \mid p \in N'\} = \{\forall x(M(x)), \forall x(\neg S(x) \vee \neg M(x) \vee \neg T(x)), \forall x(S(x) \vee N(x)), \forall x(T(x) \vee N(x)), \forall x(\neg N(x))\}$ *is obtained. Similarly, if a copy of the subtree rooted at* q *in* (N, A, e, f, g, h) *is substituted by the subtree rooted at* q', *another minimal assignment tree* $(N'', A', e'', f'', g'', h'')$ *with* $\{e(p) \mid p \in N''\} = \{\forall x(M(x)), \forall x(\neg R(x)), \forall x(\neg S(x) \vee \neg M(x) \vee \neg T(x)), \forall x(S(x) \vee N(x)), \forall x(T(x) \vee N(x)), \forall x(\neg N(x) \vee R(x))\}$ *is obtained.*

$$
\begin{array}{c}
M(x) \\
| \\
\neg S(x) \vee \neg M(x) \vee \neg T(x) \\
\diagup \qquad \diagdown \\
S(x) \vee N(x) \qquad\qquad T(x) \vee N(x) \\
| \qquad\qquad\qquad | \\
\neg N(x) \qquad\qquad \neg N(x)
\end{array}
\qquad\qquad
\begin{array}{c}
M(x) \\
| \\
\neg S(x) \vee \neg M(x) \vee \neg T(x) \\
\diagup \qquad \diagdown \\
S(x) \vee N(x) \qquad\qquad T(x) \vee N(x) \\
| \qquad\qquad\qquad | \\
\neg N(x) \vee R(x) \qquad\qquad \neg N(x) \vee R(x) \\
| \qquad\qquad\qquad | \\
\neg R(x) \qquad\qquad \neg R(x)
\end{array}
$$

The next definition introduces the notion of a consistent assignment tree.

Definition 7.4.5 *Let* (N, A, e, f, g, h) *be a complete assignment tree for* Δ *and* α. *Then,* (N, A, e, f, g, h) *is* **a consistent assignment tree** *if for any arcs* (q,p), (q',p') *where*

$\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q))$ *for some* $\beta \in \mathsf{Units}(e(q))$ *and*

$\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$ *for some* $\beta' \in \mathsf{Units}(e(q'))$

$\beta \vdash \overline{\beta}'$ *holds iff* $e(q) = e(p')$.

Example 7.4.5 *The following complete (and minimal) assignment tree* (N, A, e, f, g, h) *for some* Δ *and* $\alpha = \exists x \exists y(Q(x,y))$ *with* $\{e(p) \mid p \in N\} = \{\forall x \forall y(\neg Q(x,y)), \forall x \forall y(P(a) \vee \neg P(b) \vee$

$Q(x,y)), \forall x(\neg P(x)), \forall x(P(x))\} \subseteq \Delta \cup \{\neg\alpha\}$ *is not consistent because of arcs* (q,p), (q',p'). *In this tree,* $g(q) = \{x/a\}$ *and* $g(q') = \{x/b\}$. *For* $\beta = \forall x(\neg P(x)) \in \mathsf{Units}(e(q))$ *and* $\beta' = \forall x(P(x)) \in \mathsf{Units}(e(q'))$, *it holds that:*

$\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q)) = \neg P(a)$ *and*

$\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q')) = P(b)$

and $\beta \vdash \overline{\beta'}$ *but* $e(q) \neq e(p')$.

$$\begin{array}{ccccc}
 & & \neg Q(x,y) & & \\
 & & | & & \\
 & & P(a) \vee \neg P(b) \vee Q(x,y)_{p\,=\,p'} & & \\
e(q) = \forall x(\neg P(x)) & & / \qquad \backslash & & e(q') = \forall x(P(x)) \\
g(q) = \{x/a\} & & \neg P(a)_q \qquad P(b)_{q'} & & g(q') = \{x/b\}
\end{array}$$

An assignment tree (N', A', e', f', g', h') *with the same representation as the above can be formed from the set of clauses* $\{e(p) \mid p \in N'\} = \{\forall x \forall y(\neg Q(x,y)), \forall x \forall y(P(a) \vee \neg P(b) \vee Q(x,y)), \neg P(a), P(b)\} \subseteq \Delta \cup \{\neg\alpha\}$. *In this case,* (N', A', e', f', g', h') *satisfies the conditions of definition 7.4.5. Although the two trees have identical representations, in the tree below where* $e(q) = \neg P(a)$ *and* $e(q') = P(b)$, *for* $\beta = \neg P(a) \in \mathsf{Units}(e(q))$ *and* $\beta' = P(b) \in \mathsf{Units}(e(q'))$, $\neg P(a) \nvdash \overline{P(b)}$ *so arcs* (q,p), (q',p') *do not violate the conditions for a consistent assignment tree.*

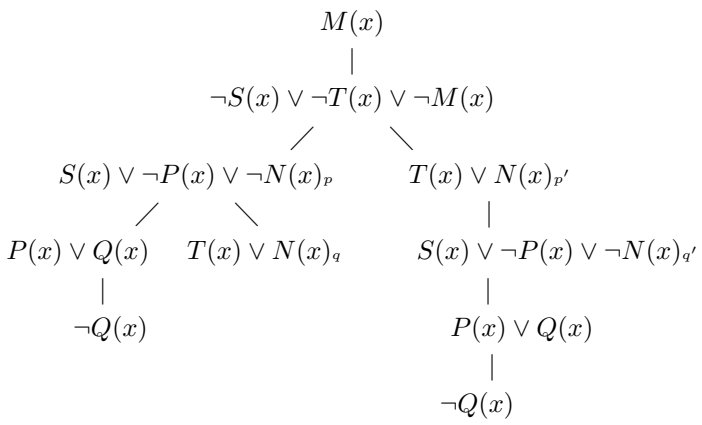
$$\begin{array}{ccccc}
 & & \neg Q(x,y) & & \\
 & & | & & \\
 & & P(a) \vee \neg P(b) \vee Q(x,y)_{p\,=\,p'} & & \\
e(q) = \neg P(a) & & / \qquad \backslash & & e(q') = P(b) \\
g(q) = \emptyset & & \neg P(a)_q \qquad P(b)_{q'} & & g(q') = \emptyset
\end{array}$$

Example 7.4.6 *The following assignment tree* (N, A, e, f, g, h) *for* Δ *and* $\alpha = \exists x(\neg M(x))$ *where* $\{e(p) \mid p \in N\} = \{\forall x(M(x)), \forall x(\neg S(x) \vee \neg T(x) \vee \neg M(x)), \forall x(S(x) \vee \neg P(x) \vee \neg N(x)), \forall x(P(x) \vee Q(x)), \forall x(\neg Q(x)), \forall x(T(x) \vee N(x))\} \subseteq \Delta \cup \{\neg\alpha\}$ *is a consistent assignment tree. For* q, q', *where* $g(q) = \emptyset$ *and* $g(q') = \emptyset$, *for* $\beta = \forall x(N(x)) \in \mathsf{Units}(e(q))$, $\beta' = \forall x(\neg N(x)) \in \mathsf{Units}(e(q'))$

$\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q)) = \forall x(N(x))$ *and*

$\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q')) = \forall x(\neg N(x))$

and it holds that $\beta \vdash \overline{\beta}'$ *and* $e(q) = e(p')$.



For a minimal and consistent grounded assignment tree the following result holds.

Proposition 7.4.2 *Let (N, A, e, f, g, h) be a complete, minimal and consistent grounded assignment tree for Δ and α. Then $\langle \Phi, \alpha \rangle$ with $\Phi = \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}$ is an argument.*

Proof: Let (N, A, e, f, g, h) be a complete, minimal and consistent grounded assignment tree for Δ and α where p_0 is the root node. In order to prove that $\langle \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}, \alpha \rangle$ is an argument, it is proved that $\{e(p) \mid p \in N\}$ is a minimal inconsistent set. By proposition 7.4.1, $\{e(p) \mid p \in N\} \setminus \{e(p_0)\} \vdash \alpha$ from which follows that $\{e(p) \mid p \in N\} \setminus \{e(p_0)\} \cup \{\neg\alpha\} \vdash \bot$ which is equivalent to $\{e(p) \mid p \in N\} \vdash \bot$. By the definitions for a minimal and consistent assignment tree follows that for each $\phi \in \{e(p) \mid p \in N\}$, for each $\beta \in \mathsf{Units}(\phi)$, there is exactly one clause $\phi' \in \{e(p) \mid p \in N\}$ with $\overline{\beta} \in \mathsf{Units}(\phi')$, otherwise the conditions of either definition 7.4.4 or definition 7.4.5 would be violated. Using this remark and assuming that $\{e(p) \mid p \in N\}$ is not a minimal inconsistent set leads to contradiction. Assume there is a $\Psi \subset \{e(p) \mid p \in N\}$ such that $\Psi \vdash \bot$ and without loss of generality assume this Ψ is a minimal inconsistent set. Then, Ψ has a closed tableau and for all $\psi \in \Psi$, for all $\gamma \in \mathsf{Units}(\psi)$ there exist $\psi' \in \{e(p) \mid p \in N\} \setminus \{\psi\}$ with $\overline{\gamma} \in \mathsf{Units}(\psi')$. Otherwise, if for some ψ this does not hold, then either there is at least one branch of the tableau which is open, or the tableau for $\Psi \setminus \{\psi\}$ is closed and this contradicts the assumption that Ψ is a minimal inconsistent set. Since $\Psi \subset \{e(p) \mid p \in N\}$ and the clauses of $\{e(p) \mid p \in N\}$ are all linked with each other through complementary units, there is at least one $\chi \in \Psi$ with some $\delta \in \mathsf{Units}(\chi)$ for which there is a $\chi' \in \{e(p) \mid p \in N\} \setminus \Psi$ such that $\overline{\delta} \in \mathsf{Units}(\chi')$. Otherwise Ψ and $\{e(p) \mid p \in N\} \setminus \Psi$ would be disjoint components in the query graph of α in Δ and they could not represent nodes in the same assignment tree. Then, since Ψ is a minimal inconsistent set and has closed tableau, for this $\chi \in \Psi$ and $\delta \in \mathsf{Units}(\chi)$ there is some $\chi'' \in \Psi$ such that $\overline{\delta} \in \mathsf{Units}(\chi'')$, and since $\Psi \subset \{e(p) \mid p \in N\}$, it follows that $\{\chi'', \chi'\} \subset \{e(p) \mid p \in N\}$ and $\chi'' \neq \chi'$ since $\chi' \in \{e(p) \mid p \in N\} \setminus \Psi$ and $\chi'' \in \Psi$. This contradicts the fact that that for $\chi \in \{e(p) \mid p \in N\}$ and $\delta \in \mathsf{Units}(\chi)$ there is exactly one clause in $\{e(p) \mid p \in N\}$ with $\overline{\delta}$ in its units since this holds for both χ' and χ''. Therefore, there is no $\Psi \subset \{e(p) \mid p \in N\}$ such that $\Psi \vdash \bot$ and so $\{e(p) \mid p \in N\}$ is a minimal inconsistent set and $\langle \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}, \alpha \rangle$ is an argument. $\square$

Example 7.4.7 *The following is a grounded assignment tree for Δ and $\alpha = \exists x \exists y(Q(x,y))$, derived from the consistent assignment tree of example 7.4.5 where the remaining variables have been instantiated by arbitrary constants.*

$$\begin{array}{c} \neg Q(c,d) \\ | \\ P(a) \vee \neg P(b) \vee Q(c,d) \\ / \qquad \backslash \\ \neg P(a) \qquad P(b) \end{array}$$

This (N, A, e, f, g, h) is a complete minimal and consistent grounded assignment tree and the set of clauses from Δ that correspond to the non-root nodes i.e. $\{e(p) \mid p \in N\} \setminus \{\neg\alpha\} = \{\forall x \forall y(P(a) \vee \neg P(b) \vee Q(x,y)), \neg P(a), P(b)\}$ is a support for an argument for α.

The converse of the last proposition does not always hold. If $\langle \Phi, \alpha \rangle$ is an argument, where $\Phi \subseteq \Delta$, it does not necessarily mean that there exists a minimal and consistent grounded assignment tree (N, A, e, f, g, h) for Δ and α such that $\Phi = \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}$.

Example 7.4.8 *Consider the argument $\langle \Phi, \alpha \rangle$ where $\Phi \subseteq \Delta$ is such that $\Phi = \{\forall x(P(x) \vee Q(x) \vee S(x)), \forall x(\neg P(x) \vee Q(x)), \forall x(\neg Q(x) \vee P(x)), \forall x(\neg P(x) \vee \neg Q(x))\}$ and $\alpha = \exists x(S(x))$. The only complete assignment trees (N, A, e, f, g, h) for Δ and α for which $\Phi = \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}$ are the ones below.*

$$\begin{array}{c} \neg S(x) \\ | \\ P(x) \vee Q(x) \vee S(x) \\ / \qquad \backslash \\ \begin{array}{cc} \neg P(x) \vee Q(x) & \neg Q(x) \vee P(x) \\ | & | \\ \neg P(x) \vee \neg Q(x) & \neg P(x) \vee \neg Q(x) \end{array} \end{array} \qquad \begin{array}{c} \neg S(x) \\ | \\ P(x) \vee Q(x) \vee S(x) \\ / \qquad \backslash \\ \begin{array}{cc} \neg Q(x) \vee P(x) & \neg P(x) \vee Q(x) \\ | & | \\ \neg P(x) \vee \neg Q(x) & \neg P(x) \vee \neg Q(x) \end{array} \end{array}$$

By substituting variable x by an arbitrary constant we obtain two complete grounded assignment trees neither of which is minimal or consistent. So for $\langle \Phi, \alpha \rangle$ there there is no minimal and consistent assignment tree (N, A, e, f, g, h) for Δ and α such that $\Phi = \{e(p) \mid p \in N\} \setminus \{\neg\alpha\}$. [1]

Because of the complicated nature of first-order logic, the work presented in this chapter is limited to the sound, but not complete solution based on assignment trees for searching for arguments. The next section presents algorithms which, given a knowledgebase of first-order clauses and a unit clause α as a claim for an argument, generates complete minimal and consistent assignment trees for Δ and α.

[1] These trees are in fact identical and the algorithm introduced in the next paragraph would only produce one of the two. They are both included in the example in order to demonstrate exhaustive consideration of all the possible complete assignment trees for this set and root.

7.5 Algorithms

This section introduces the algorithms that search for all the complete minimal and consistent assignment trees for a unit clause α from a knowledgebase of first-order clauses Δ. If a grounded version of a complete minimal and consistent assignment tree exists, then according to proposition 7.4.2 this indicates an argument for α. The search is based on the structure of the query graph of α in Δ. The query graph can be retrieved by using the algorithm for the focal graph introduced in chapter 3, adapted for first-order clauses where the connectivity criterion is as in the definition for the closed graph for first-order clauses given in this chapter. Since α in a unit clause, $\mathsf{Query}(\Delta, \alpha)$ consists of a unique component. Then, algorithm $\mathsf{GetFocal}(\Delta \cup \{\neg\alpha\}, \neg\alpha)$ of chapter 3 page 49 (algorithm 3.1) returns the query graph of α in Δ. The query graph is then used in order to apply search algorithms that produce assignment trees as described in the next section.

7.5.1 Algorithm for producing complete assignment trees

By using the structure of the query graph of α in Δ it is possible to generate supports for arguments for α by a walking over the graph and producing assignment trees. Algorithm 7.1 builds a depth-first search tree T that represents the steps of the search for arguments for α from Δ. The algorithm proceeds in a way similar to that of the propositional case presented in chapter 4. Every node in T is an assignment tree, every node is an extension of the assignment tree in its parent node. The leaf node of every complete accepted branch is a complete assignment tree.

The idea in building assignment trees by using the structure of the query graph, is to start from the negation of the claim and walk over the graph by following the links and unifying the atoms of pairs of contradictory literals connected with arcs. Hence, the algorithm at the same time follows the arcs of the graph and also produces partial instances of the clauses it visits as the unification of atoms indicates.

The order in which the clauses in the graph are visited is established by the conditions of definitions 7.4.1 and 7.4.2. For a node v of T, function $\mathsf{Extensions}(v)$ gives all the possible extensions of the search tree below node v i.e. all the possible assignment trees that extend the assignment tree (N, A, e, f, g, h) contained in node v by one level. Each such extension v' contains an assignment tree (N', A', e', f', g', h') that extends (N, A, e, f, g, h) with respect to the conditions of the definition for a complete assignment tree and by using the paths the query graph indicates. If for a node v of the search tree $\mathsf{Extensions}(v) = \emptyset$, then it means that the current branch of the search tree cannot be extended any further below node v. This happens either because the assignment tree (N, A, e, f, g, h) that is contained in v is a complete assignment tree or because it cannot be extended further because for some node y in (N, A, e, f, g, h) a child node y' needs to be added but no clause from the query graph can be assigned to y' without violating the conditions of the definition for an assignment tree. In order to decide whether it is the first or the second case, function $\mathsf{Accept}(v)$ checks whether a solution has been found. Hence, $\mathsf{Accept}(v)$ tests whether the assignment tree in the leaf node v of the currently built branch of T is a complete assignment tree. If (N, A, e, f, g, h) cannot be extended further because it is a complete assignment tree, $\mathsf{Accept}(v)$ returns true and node v is stored in the set $AssignmentTrees$ that is returned in the end, otherwise the node is rejected. Then the algorithm backtracks and continues to the next node

of tree T to be expanded.

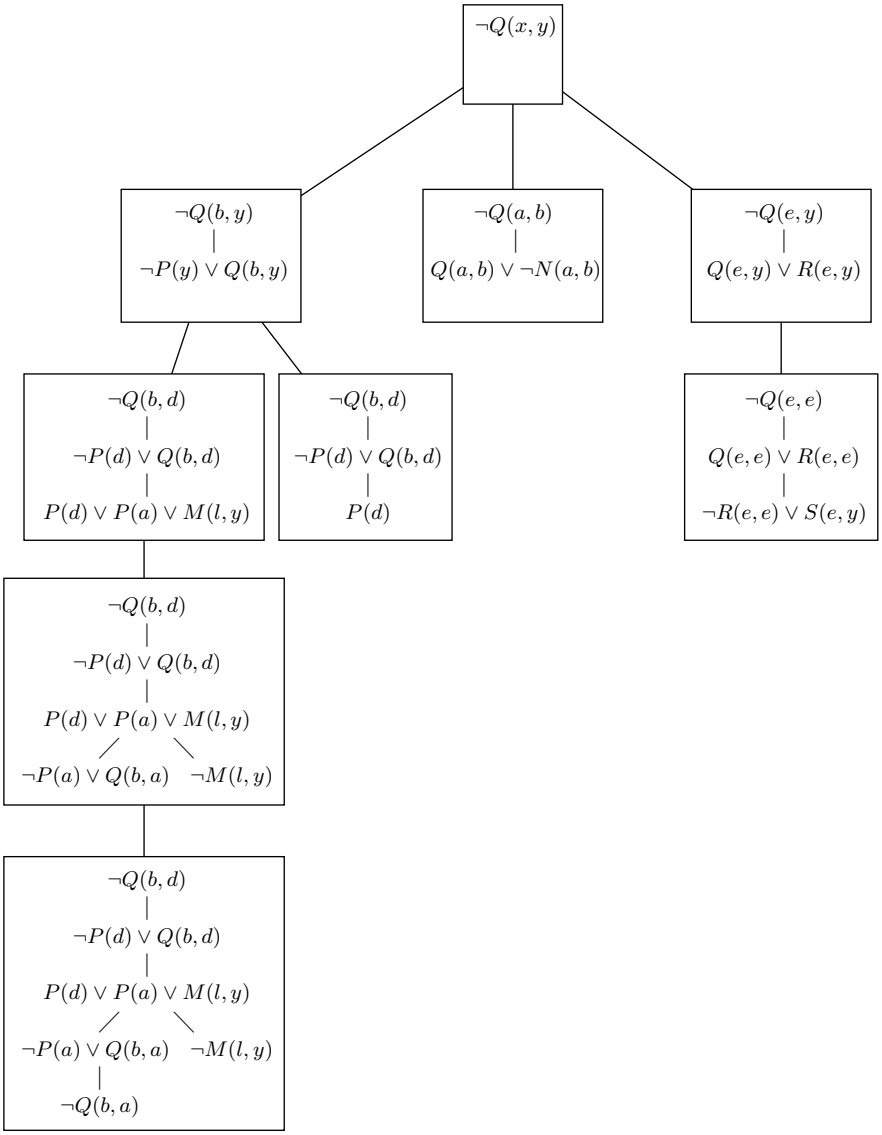
The partial instances produced while walking over the graph are generated with respect to the conditions of definition 7.4.1. Every time a clause ϕ on the graph is visited, a node q for an assignment tree is created with $e(q) = \phi$. For this node, an existential-free instance of $e(q)$ is generated by substituting each of its existentially quantified variables with an arbitrary constant that does not appear anywhere in $\Delta \cup \{\neg\alpha\}$ or in the instances already created during the search. This instantiation initializes the value $g(q)$ and sets the value $f(q)$ for q: $f(q) = \mathsf{Assign}(e(q), g(q))$. The value of $h(q)$ is also initialized at this stage to be equal to $f(q)$. After node q has been initialized as an assignment tree node, another instantiation process follows, which is based on unifying the atoms of the contradictory units in $h(q)$ and its parent with their most general unifier. This updates values $g(q)$ and $h(q)$. Let p be the parent of q in an assignment tree. If $\theta \subset \wp(\mathcal{B})$ is the most general unifier of the atoms of a pair of contradictory units from $h(q)$ and $h(p)$, then $g(q) = g(q) \cup \theta$ and $h(q) = \mathsf{Assign}(h(q), g(q))$. Every time such a unification binding is retrieved for a current leaf node q of (N', A', e', f', g', h'), its values may affect the links in the rest of the assignment tree. Any of the corresponding clauses that can be associated through a sequence of arcs in the assignment tree with the variables of $e(q)$ can be affected by the bindings in θ. For this reason, for each node v' in $\mathsf{Extensions}(v)$, function $\mathsf{PropagateAssignments}(v')$ propagates for each newly created node q in the assignment tree (N', A', e', f', g', h') that is contained in v', the values of $g(q)$ to the nodes that are affected by this assignment. It updates this way the values of $g(r)$ and hence $h(r)$ for the nodes r whose variables are associated with the bindings of q. There are cases where this propagation of values causes violation of the conditions of the definition of the assignment tree. For instance, when updating the value $g(r)$ for a node r in (N', A', e', f', g', h'), the updated $g(r)$ may be from $\mathsf{Prohibited}(e(r))$, violating this way condition (4) of definition 7.4.1. This means that the currently built assignment tree cannot develop into a complete assignment tree. For this, after every time $\mathsf{PropagateAssignments}(v')$ is applied, another function, $\mathsf{Reject}(v')$ checks whether $\mathsf{PropagateAssignments}(v')$ has caused such a problem in the assignment tree contained in v'. If the function returns false for a node v', the algorithm then continues recursively for v'.

Algorithm 7.1 $\mathsf{FirstOrderSearchTree}(v)$

```
if Extensions(v) ≠ ∅ then
  for all w ∈ Extensions(v) do
    PropagateAssignments(w)
    if ¬Reject(w) then
      FirstOrderSearchTree(w)
    end if
  end for
else
  if Accept(v) then
    AssignmentTrees = AssignmentTrees ∪ {v}
  end if
end if
return AssignmentTrees
```

Figure 7.1 represents the result of searching for arguments for $\alpha = \exists x \exists y(Q(x, y))$ using the query graph of α in Δ which is the first of the components of the graph from example 7.3.3. The results

of the first and second branch of the search tree (at the leaf) are complete assignment trees. The third branch is rejected because for node p with $e(p) = f(p) = h(p) = Q(a,b) \vee \neg N(a,b)$ there is only one arc in the graph that connects $e(p)$ with a clause that contains a complement of $\neg N(a,b)$. This is clause $\neg Q(a,b) \vee N(a,b)$ but a child q of p with $e(q) = \neg Q(a,b) \vee N(a,b)$ cannot be created because there is no assignment $g(q)$ for which $\mathsf{Attacks}(h(q), h(p)) \neq null$. The last branch of the search tree is rejected because adding node s with $e(s) = \forall x(\neg R(x,x) \vee S(x,y))$ as a child of r with $h(r) = Q(e,y) \vee R(e,y)$ requires unifying $R(x,x)$ with $R(e,y)$ which updates the value of $g(r)$ to $g(r) = \{x/e, y/e\} \in \mathsf{Prohibited}(e(r))$ and so condition 4 of the definition for an assignment tree is violated.



A search tree generated using algorithm FirstOrderSearchTree. Each node of this search tree represents an assignment tree which extends the assignment tree contained in its parent node by one level. For this, the algorithm adds clauses each of which attacks its parent clause on a different unit. The atoms of the contradictory units between a parent and a child are unified and the bindings of the unification are passed on to any other clauses that may be affected in the assignment tree by this node.

Figure 7.1: Applying algorithm FirstOrderSearchTree

7.5.2 Algorithms for selecting the minimal and consistent assignment trees

Given the accepted results of algorithm 7.1 (i.e. the complete assignment trees generated by the algorithm), algorithm $\mathsf{IsMinimal}(N, A, e, f, g, h)$ tests whether the complete assignment tree (N, A, e, f, g, h) satisfies the conditions for a minimal assignment tree.

Algorithm 7.2 $\mathsf{IsMinimal}(N, A, e, f, g, h)$

for all $(q, p), (q', p') \in A$ **do**
  Let $\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q)), \mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$
  if $\beta \vdash \beta'$ && $e(q) \neq e(q')$ **then**
    return false
  end if
end for
return true

Given a complete assignment tree (N, A, e, f, g, h), the next algorithm tests whether (N, A, e, f, g, h) satisfies the conditions for a consistent assignment tree.

Algorithm 7.3 $\mathsf{IsConsistent}(N, A, e, f, g, h)$

for all $(q, p), (q', p') \in A$ **do**
  Let $\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q)), \mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$
  if $\beta \vdash \overline{\beta'}$ && $e(q) \neq e(p')$ **then**
    return false
  end if
end for
return true

Example 7.5.1 *The following assignment tree is the first of the accepted results of the search tree in figure 7.1.*

$$\begin{array}{c}
\neg Q(b,d) \\
| \\
\neg P(d) \vee Q(b,d)_w \\
| \\
P(d) \vee P(a) \vee M(l,y)_t \\
/ \qquad \backslash \\
\neg P(a) \vee Q(b,a)_v \quad \neg M(l,y) \\
| \\
\neg Q(b,a)
\end{array}$$

In this (N, A, e, f, g, h) *there are no* $(q, p), (q', p') \in A$ *where* $\mathsf{Attacks}(h(q), h(p)) = \mathsf{Assign}(\beta, g(q))$, $\mathsf{Attacks}(h(q'), h(p')) = \mathsf{Assign}(\beta', g(q'))$ *for which* $\beta \vdash \beta'$ *so the tree is a minimal assignment tree and algorithm 7.2 returns true. There are arcs* $(v, t), (t, w) \in A$ *where* $e(w) = \forall y(\neg P(y) \vee Q(b, y))$ *with* $g(w) = \{y/d\}$, $e(t) = \forall y \exists x(P(d) \vee P(a) \vee M(x, y))$ *with* $g(t) = \{x/l\}$ *and* $e(v) = \forall y(\neg P(y) \vee Q(b, y))$ *with* $g(v) = \{y/a\}$ *such that for* $\beta' = P(d) \in \mathsf{Units}(e(t))$ *and for* $\beta = \forall y(\neg P(y)) \in \mathsf{Units}(e(v))$ *the following hold.*

$\mathsf{Attacks}(h(v), h(t)) = \mathsf{Assign}(\beta, g(v)) = \neg P(a)$ *and*

$\mathsf{Attacks}(h(t), h(w)) = \mathsf{Assign}(\beta', g(t)) = P(d)$

and $\forall y(\neg P(y)) \vdash \neg P(d)$ *so* $\forall y(\neg P(y)) \vdash \overline{P(d)}$ *which means that* $\beta \vdash \overline{\beta'}$ *so this pair of nodes is examined further by the algorithm. It holds though in addition that* $e(v) = e(w) = \forall y(\neg P(y) \vee Q(b, y))$ *so these arcs do not affect the conditions for a consistent assignment tree and the algorithm returns true. By substituting in this tree variable* y *in literals* $M(l, y)$ *and* $\neg M(l, y)$ *by the same arbitrary constant, a minimal and consistent grounded assignment tree is obtained. The set of clauses* $\{\forall y(\neg P(y) \vee Q(b, y)), \forall y \exists x(P(d) \vee P(a) \vee M(x, y)), \forall x \forall y(\neg M(x, y))\}$ *that corresponds to the set of non-root nodes of this tree is a support for an argument for* $\alpha = \exists x \exists y(Q(x, y))$.

The second of the accepted results of the search tree in figure 7.1 (i.e. the leaf of the second branch of T*) is a complete assignment tree that also satisfies the condition for a minimal and consistent assignment tree and is already grounded. The set of clauses* $\{\forall y(\neg P(y) \vee Q(b, y)), \forall x(P(x))\} \subset \Delta$ *that corresponds to the non-root nodes of this tree is also a support for an argument for* α

7.6 Discussion

Classical first-order logic has many advantages for representing and reasoning with knowledge. However, it is well established that first-order logic is undecidable [24]. There is no algorithm that can take as input an arbitrary set of clauses Ψ and determine whether or not Ψ is satisfiable. This raises limitations on the method presented in this chapter for generating arguments in first-order logic by using resolution and unification because the process can loop forever generating new resolvents. In general, it is impossible to determine whether or not this generation will be productive in arriving at a refutation.

In this chapter I proposed a method for retrieving arguments in a rich first-order language. This method was partly based on the ideas for propositional logic presented in Chapter 4 and it was proved to be sound. Developing a solution that is both sound and complete is an interesting point for further investigation.

Chapter 8

Conclusions

8.1 Argumentation overview

Computational argumentation has become a topic of increasing interest in AI research over the last decade. It provides a context for resolving conflicts that features human behaviour. These conflicts may arise in the form of arguments and counterarguments in a monological presentation of pros and cons when evaluating a situation, or as a dialogue where two or more agents are interacting. Therefore, it provides the means for analysing and assessing information, which is a daily task for humans and consequently its automation can be useful in various applications. Substantial work has been done on argumentation based on specific applications, some of these include

1. Medical applications where argumentation can be useful in assessing the strengths and weakness of alternative solutions to a clinical problem [52, 48, 38].

2. Legal applications where reasoning often takes place in a context of dispute and disagreement and argumentation offers the grounds for logical analysis of legal arguments [66]. Further applications of argumentation in political philosophy involve e-democracy where argumentation can be useful for communicating information and facilitating structured debate. [6].

3. Sharing information e.g. for professionals that do collaborative work in distributed teams and at asynchronous timings, or for instance for idea linking in social networks [71, 3].

The use of argumentation in various applications encourages further research on the topic and creates the necessity of developing theory and tools to automate the process.

Various existing formalisations for argumentation provide different approaches to representing information and evaluating the credibility of arguments according to different criteria. The objective of this thesis was to develop theory that can make computational argumentation based on classical propositional logic viable for practical applications. This problem is interesting since classical logic is a strong tool for representing and reasoning with knowledge, and hence is an appealing option for argumentation. It provides a powerful language strong enough to represent in detail complicated information. It has a simple and intuitive syntax and semantics, and it is supported by well established proof theory and extensive foundational results. However, it has the tradeoff that generating arguments in classical logic

is a computationally challenging task. Not much theoretical work has been done with respect to this issue, while no implemented working system for argumentation based on classical logic existed before this thesis. For these reasons, it was interesting to develop algorithms that implement argumentation in classical logic and produce, based on these algorithms, a prototype software system.

8.2 Contribution of this work

The work of this thesis contributed in the area of computational argumentation by providing the theoretical background for algorithms that search for arguments in classical propositional logic efficiently. It adapted widely used automated theorem proving techniques and extended them to fit the requirements of argumentation (i.e. minimal and consistent proofs). The problem was approached in three directions:

1. **Reducing the number of formulae considered when searching for arguments.** Chapter 3 introduced algorithms which, given a claim in CNF and a knowledgebase that contains formulae in CNF, isolate a part of the knowledgebase that contains all the formulae that may be premises for arguments for the given claim. It was proved that using this part of the knowledgebase rather than the initial knowledgebase for generating arguments still provides a complete method. Experimentation using hard satisfiability examples gave positive evidence on the efficiency of the approach. In addition to decreasing the number of clauses considered in a search for arguments, this technique retrieves this reduced search space in the form of a connection graph which provides the basis for the next step which is applying algorithms for generating arguments.

2. **Generating arguments using the reduced search space.** By using the results of chapter 3, chapter 4 provided a sound and complete proposal for generating arguments for claims that are disjunctive clauses from knowledgebases that consist of disjunctive clauses. The corresponding algorithms that are based on the structure of the graphs introduced in chapter 3, search for arguments by traversing the corresponding isolated part of the knowledgebase. The algorithms were evaluated by using randomly generated datasets that were designed to make the process of generating arguments computationally challenging.

3. **Generating counterarguments efficiently.** The theory of chapter 4 concerns generating arguments for claims that are disjunctive clauses. Generating canonical undercuts, which is the form of counterargument employed in this work, is a more complicated task. Chapter 5 extended the work of chapter 4 to deal with generating canonical undercuts. The algorithms for generating canonical undercuts were in turn adapted by an algorithm that generates argument trees which was implemented in a software system and evaluated by experimentation on randomly generated datasets.

Although the work of this thesis is mainly based on classical propositional logic, an extension of the work in chapters 3 and 4 that involves classical first-order logic is introduced in chapter 7. Chapter 7 presents a proposal for generating arguments in a rich first-order language of clauses. Like in the propositional case, initially a part of the knowledgebase that contains all the clauses that can play a role

for an argument for the given claim is isolated using connection graphs. This reduced search space, which is in the form of a connection graph, is then traversed using a search algorithm in a way that the set of clauses visited during the search indicates a proof for the given claim. Additional subsidiary algorithms then check this proof for minimality and consistency. This was proved to be a sound method for generating arguments in this first-order language of clauses.

In addition to the theoretical work on algorithms, this thesis contributed in the area of computational argumentation by providing the first software system that implements argumentation based on classical propositional logic. This system, presented in chapter 6, has been tested thoroughly using complex example sets. It comes as a simple standalone application with comprehensive functionality that can be used by other research groups.

8.3 Assessment and further work

The aim of this thesis was to develop theory and software for argumentation in classical propositional logic which, by addressing the computational issues involved, provide an efficient solution for practical argumentation. These goals have been accomplished since both the theoretical and the practical part have been developed. Experimental evaluation of the implemented software has shown positive evidence of the usefulness of the underlying theory, especially in comparison with an earlier naive software implementation that did not involve any study on the algorithmic efficiency of the problem [29]. Apart from the empirical evaluation, the quality of the theoretical work of this thesis has been assessed through peer reviews in conferences and journals where it has been submitted and accepted for publication.

Further improvements that can be done to extend the existing work involve overcoming the language restrictions involved. This could help in using the algorithms and their implementation directly with real world data that may involve propositions that make use of conjunction. Part of this work involves formulae in conjunctive normal form, while the rest is restricted to propositional clauses. The language of propositional clauses is rich enough to express rules which are widely used in logic programming, while sets of clauses are sufficient to express conjunctions of clauses. Still, a good point to address in further work would be extension of the algorithms presented in this thesis to the full syntax of propositional logic.

It would also be interesting to see the system's performance in generating argument trees using different kinds of undercuts. Using direct undercuts for example (undercuts that negate exactly one of the premises of the argument attacked) would make the search for counterarguments less complex than when searching for canonical undercuts but it could possibly cause an explosion on the number of nodes of an argument tree. Using the implemented software would allow for a comparison of the two approaches.

Another point that could be addressed is duplication of results returned by the algorithms that generate arguments. The fact that the algorithms produce search trees that may have the same set of clauses arranged in different structures causes redundancy that could possibly be avoided by some technique that would predict this by investigation on the structure of the graphs.

Another topic where further work could be conducted is the algorithms for first-order logic pre-

sented in chapter 7. Because of the complicated nature of first-order logic, the proposed mechanism for generating arguments in first-order is relatively simple but sufficient to provide a sound solution to the problem. With additional constraints, a complete mechanism could possibly be developed. In addition, an interesting idea would be the software implementation of a system that builds argument trees in first-order logic.

Additional work could be done in testing the system with applications that involve domains with large datasets and integration with other resources (e.g. databases). It would be interesting to assess the system's usability in conjunction with external applications and have access to actual data. The datasets used in the thesis for benchmarking the algorithms were designed so as to generate hard satisfiability problems that possibly do not depict the level of inconsistency that appears in real world applications. The fact that the software responded to this kind of data and produced argument trees that consist of hundreds of nodes at reasonable time is encouraging for using the system with non-artificial data.

8.4 Discussion

The aim of this thesis was to develop algorithms that address the computational issues that arise when constructing arguments in classical propositional logic. The approach to the problem was based on adapting existing automated theorem proving techniques and extending them in the argumentation context. The viability of the proposal was assessed by empirical studies undertaken on a software system that implements the algorithms introduced in the thesis. Part of the theoretical work was extended to classical first-order logic, which provides an interesting topic for further work.

Appendix A

Example knowledgebases

The following knowledgebase is adapted from the medical domain of breast cancer where rules are developed from the results of published clinical trials [52].

```
!hasPosIntent(p.q)|hasTreatment(p.q), !hasNegIntent(p.q)|notHasTreatment(p.q),
!Women(p)||!hasDisease(p.q)||!EarlyBreastCancer(q)||!ERPos(q)|
!hasTreatment(p.s)||!Tamoxifen5YrCourse(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.5),
!Women(p)||!hasDisease(p.q)||!EarlyBreastCancer(q)||!ERPos(q)||!hasTreatment(p.s)|
!Tamoxifen2YrCourse(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.21),
!Women(p)||!hasDisease(p.q)||!EarlyBreastCancer(q)||!ERPos(q)||!hasTreatment(p.s)|
!Tamoxifen2YrCourse(s)|hasDeltaRisk(p.IncreasedOS1.2),
!Women(p)||!hasDisease(p.q)||!EarlyBreastCancer(q)||!ERPos(q)||!hasTreatment(p.s)|
!ChemoTamRegimeTypes(s)|hasDeltaRisk(p.IncreasedOS1.2),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)||!LNNeg(q)||!hasTreatment(p.s)|
!TamoxifenMore5YrCourse(s)|hasDeltaRisk(p.DecreasedBrCaDFS0.8),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)||!hasTreatment(p.s)|
!TamoxifenMore5YrCourse(s)||!NoChangeBrCaDFS(t)|
!refersDisease(t.q)|hasDeltaRisk(p.t),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)||!hasTreatment(p.s)|
!TamoxifenMore5YrCourse(s)||!NoChangeOS(t)||!refersDisease(t.q)|
hasDeltaRisk(p.t),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)||!LNPos(q)||!hasTreatment(p.s)|
!TamoxifenMore5YrCourse(s)||!NoChangeOS(t)||!refersDisease(t.q)|
hasDeltaRisk(p.t),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)|
!LNPos(q)||!hasTreatment(p.s)||!TamoxifenMore5YrCourse(s)||!NoChangeBrCaDFS(t)|
!refersDisease(t.q)|hasDeltaRisk(p.t),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)||!LNNeg(q)|
!TamoxifenMore5YrCourse(s)|notHasTreatment(p.s),
!Women(p)||!hasDisease(p.q)||!BreastAdenoCa(q)||!LNPos(q)||!Aged50Plus(p)|
```

```
!hasTreatment(p.s)|!ACTChemoTam(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.2),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!LNPos(q)|!Aged50Plus(p)
|!hasTreatment(p.s)|!ACTChemoTam(s)|hasDeltaRisk(p.IncreasedOS1.1),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!LNPos(q)|!Aged50Plus(p)|
!hasTreatment(p.s)|!PAFTChemoTam(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.3),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!LNPos(q)|!Aged50Plus(p)|
!hasTreatment(p.s)|!PFTChemoTam(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.1),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!ERPos(q)|!LNNeg(q)|
!hasTreatment(p.s)|!MFTChemoTam(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.05),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!ERPos(q)|!LNNeg(q)|
!hasTreatment(p.s)|!CMFTChemoTam(s)|hasDeltaRisk(p.IncreasedBrCaDFS1.05),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!ERPos(q)|!LNNeg(q)|
!hasTreatment(p.s)|!MFTChemoTam(s)|hasDeltaRisk(p.IncreasedOS1.03),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!ERPos(q)|!LNNeg(q)|
!hasTreatment(p.s)|!CMFTChemoTam(s)|hasDeltaRisk(p.IncreasedOS1.02),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|!LNPos(q)|
!hasTreatment(p.s)|!EarlyCMFTChemoTam(s)|
hasDeltaRisk(p.IncreasedBrCaDFS1.1),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|!LNPos(q)|
!hasTreatment(p.s)|!LateCMFTChemoTam(s)|!NoChangeBrCaDFS(t)|
!refersDisease(t.q)|hasDeltaRisk(p.t),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|!LNPos(q)|
!ERPos(q)|!hasTreatment(p.s)|!CMFTChemoTam(s)|
hasDeltaRisk(p.IncreasedBrCaDFS1.1),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|!LNPos(q)|
!HRPos(q)|!hasTreatment(p.s)|!CMFTChemoTam(s)|
hasDeltaRisk(p.IncreasedRiskSideEffects23),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|!LNPos(q)|
!HRPos(q)|!hasTreatment(p.s)|!CMFTChemoTam(s)|!NoChangeOS(t)|
hasDeltaRisk(p.t),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|
!hasTreatment(p.s)|!Tamoxifen(s)|hasDeltaRisk(p.DecreasedRiskBrCa0.55),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!Postmenopausal(p)|
!hasTreatment(p.s)|!Tamoxifen(s)|
hasDeltaRisk(p.IncreasedRiskEndometrialCancer6.4),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasDisease(p.r)|
!EndometrialCancer(r)|!hasTreatment(p.s)|!Tamoxifen(s)|!PoorPrognosis(t)|
hasPrognosis(r.t), !Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|
```

```
!hasDisease(p.r)|!EndometrialCancer(r)|!hasTreatment(p.s)|!Tamoxifen(s)|
hasDeltaRisk(p.DecreasedEndoCaDSS0.66),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!LNNeg(q)|!ERPos(q)|
!hasTreatment(p.s)|!Tamoxifen5YrCourse(s)|
hasDeltaRisk(p.IncreasedRiskEndometrialCancer7.5),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.IncreasedRiskEndometrialCancer2.3),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasDisease(p.r)|
!EndometrialCancer(r)|!hasTreatment(p.s)|!Tamoxifen(s)|!PoorPrognosis(t)|
notHasPrognosis(r.t),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|hasRisk(p.RiskEndometrialCa0.005),
!Women(p)|!hasTreatment(p.s)|!Tamoxifen(s)|
hasRisk(p.RiskEndometrialAbnormality3.9),
!Women(p)|!hasTreatment(p.s)|!Tamoxifen(s)|
hasDeltaRisk(p.NoChangeRiskEndometrialAbnormality1),
!Women(p)|!hasDisease(p.q)|!EarlyBreastCancer(q)|!hasTreatment(p.s)|
!Tamoxifen40mg(s)|hasDeltaRisk(p.DecreasedRiskBreastCancer0.6),
!Women(p)|!hasDisease(p.q)|!EarlyBreastCancer(q)|!hasTreatment(p.s)|
!Tamoxifen40mg(s)|hasDeltaRisk(p.IncreasedRiskEndometrialCancer6.0),
!Women(p)|!hasDisease(p.q)|!EarlyBreastCancer(q)|!hasTreatment(p.s)|
!Tamoxifen40mg(s)|hasDeltaRisk(p.IncreasedRiskGastricCa3.0),
!Women(p)|!hasDisease(p.q)|!EarlyBreastCancer(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.IncreasedRiskEndometrialCancer4.1),
!Women(p)|!hasDisease(p.q)|!EarlyBreastCancer(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.IncreasedRiskCRC1.9),
!Women(p)|!hasDisease(p.q)|!EarlyBreastCancer(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.IncreasedRiskGastricCa3.2),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasTreatment(p.s)|
!AdjuvantTreatment(s)|hasDeltaRisk(p.IncreasedRiskVenousThrombosis3.4),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.IncreasedRiskVenousThrombosis3.4),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|
!Postmenopausal(p)|hasDeltaLevel(p.DecreasedPlasmaFibrinogenLevel0.85),
!Women(p)|!hasDisease(p.q)|!BreastAdenoCa(q)|!hasTreatment(p.s)|
!Tamoxifen(s)|!Postmenopausal(p)|
hasDeltaLevel(p.DecreasedPlasmaPlateletCount0.92),
```

```
!Women(p)||!hasTreatment(p.s)||!Tamoxifen(s)|
hasDeltaRisk(p.IncreasedRiskCataracts1.14),
!Women(p)||!hasTreatment(p.s)||!Tamoxifen(s)|
hasDeltaRisk(p.IncreasedRiskEndometrialCancer2.53),
!Women(p)||!hasTreatment(p.s)||!Tamoxifen20mg(s)||!RiskOvarianCyst(t)|
hasRisk(p.t),
!Women(p)||!hasTreatment(p.s)||!Tamoxifen20mg(s)|
hasRisk(p.RiskOvarianCyst0.06),
!Women(p)||!Postmenopausal(p)||!hasTreatment(p.s)||!Tamoxifen(s)|
hasDeltaRisk(p.IncreasedRiskPersistentSideEffects2.3),
!Women(p)||!hasSymptoms(p.q)||!HotFlushes(q)||!hasDisease(p.u)|
!BreastAdenoCa(u)||!hasTreatment(p.s)||!ClonidineRegime(s)|
hasDeltaRisk(p.DecreasedRiskHotFlushes0.8),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!hasTreatment(p.s)|
!Tamoxifen(s)||!IncreasedRiskRetinopathy(r)|hasDeltaRisk(p.r),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!hasTreatment(p.s)|
!Tamoxifen(s)|hasRisk(p.RiskRetinopathy0.12),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!LNNeg(p)|
!hasTreatment(p.s)||!Tamoxifen(s)|
hasDeltaLevel(p.DecreasedCholesterol0.88),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!LNNeg(p)|
!hasTreatment(p.s)||!Tamoxifen(s)|
hasDeltaLevel(p.DecreasedLDL0.88),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.DecreasedRiskFatalMI0.37),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.DecreasedRiskCardiacDisease0.68),
!Women(p)||!hasDisease(p.u)||!BreastAdenoCa(u)||!hasTreatment(p.s)|
!Tamoxifen(s)|hasDeltaRisk(p.IncreasedCardiacDSS1.52),
!Women(p)||!Postmenopausal(p)||!hasDisease(p.u)||!BreastAdenoCa(u)|
!LNNeg(u)||!hasTreatment(p.s)||!Tamoxifen(s)|
hasDeltaLevel(p.IncreasedLumbarBMD1.006),
!Women(p)||!Postmenopausal(p)||!hasDisease(p.u)||!BreastAdenoCa(u)|
!LNNeg(u)||!hasTreatment(p.s)||!Tamoxifen30mg(s)|
!IncreasedLumbarBMD(r)|hasDeltaLevel(p.r),
!Women(p)||!Postmenopausal(p)||!hasDisease(p.u)||!BreastAdenoCa(u)|
!LNNeg(u)||!DecreasedRadialBMD(r)|hasDeltaLevel(p.r),
!Women(p)||!Postmenopausal(p)||!hasDisease(p.u)||!BreastAdenoCa(u)|
```

```
!LNNeg(u)|!hasTreatment(p.s)|!Tamoxifen30mg(s)|!NoChangeRadialBMD(r)|
hasDeltaLevel(p.r),
!Women(p)|!Postmenopausal(p)|!hasDisease(p.u)|!BreastAdenoCa(u)|
!LNNeg(u)|!hasTreatment(p.s)|!Tamoxifen5YrCourse(s)|
hasDeltaLevel(p.IncreasedLumbarBMD1.008),
!Women(p)|!Postmenopausal(p)|!hasDisease(p.u)|!BreastAdenoCa(u)|
!LNNeg(u)|hasDeltaLevel(p.DecreasedLumbarBMD0.993),
!Women(p)|!Postmenopausal(p)|!hasTreatment(p.s)|!Tamoxifen20mg(s)|
hasDeltaLevel(p.IncreasedLumbarBMD1.012),
!Women(p)|!Premenopausal(p)|!hasTreatment(p.s)|!Tamoxifen20mg(s)|
Women(p),hasDeltaLevel(p.DecreasedLumbarBMD0.986), hasPosIntent(p.q),
hasNegIntent(p.q), hasDisease(p.q), EarlyBreastCancer(q),hasTreatment(p.s),
Tamoxifen5YrCourse(s), Tamoxifen2YrCourse(s),ChemoTamRegimeTypes(s),
TamoxifenMore5YrCourse(s), BreastAdenoCa(q),LNNeg(q), refersDisease(t.q),
NoChangeOS(t),LNPos(q), NoChangeBrCaDFS(t),Aged50Plus(p),ACTChemoTam(s),
PAFTChemoTam(s),PFTChemoTam(s), MFTChemoTam(s),EarlyCMFTChemoTam(s),
LateCMFTChemoTam(s),CMFTChemoTam(s),HRPos(q),Tamoxifen(s),LNNeg(u),
PoorPrognosis(t),EndometrialCancer(r),AdjuvantTreatment(s),ERPos(q),
Postmenopausal(p),Tamoxifen20mg(s),RiskOvarianCyst(t),hasSymptoms(p.q),
hasDisease(p.u),BreastAdenoCa(u),ClonidineRegime(s),Tamoxifen30mg(s),
IncreasedRiskRetinopathy(r),NoChangeRadialBMD(r)
```

The following knowledgebase describes the payment example adapted from [41].

```
!goodJob|payment,!tooLateJob|badJob,!incompleteJob|badJob,
!not(accordingToSpec)|!delivered|incompleteJob,!reqA|!reqB|accordingToSpec,
delivered,not(reqB),goodJob,not(accordingToSpec),reqA,reqB,goodJob|badJob,
!goodJob|!badJob,!not(accordingToSpec)|!accordingToSpec,
not(accordingToSpec)|accordingToSpec,
!reqA|!not(reqA),reqA|not(reqA),!reqB|!not(reqB),reqB,not(reqB)
```

The next knowledgebase describes the baby name example adapted from [42].

```
!easy_to_remember(adrian)|acceptable(adrian),
!all_like(adrian)|acceptable(adrian),
!short(adrian)|easy_to_remember(adrian),short(adrian),
!mom_hates(adrian)|some_dislike(adrian),mom_hates(adrian),
!dad_hates(adrian)|some_dislike(adrian),
```

```
!too_commom(adrian)|dad_hates(adrian),
!uncle_has(adrian)|dad_hates(adrian),
!mom_said_ok(adrian)|mom_not_hate(adrian), mom_said_ok(adrian),
!all_like(adrian)|!some_dislike(adrian),all_like(adrian),
all_like(adrian)|some_dislike(adrian),
!mom_hates(adrian)|!mom_not_hate(adrian),
mom_hates(adrian)|mom_not_hate(adrian)
```

The next knowledgebase contains the prison example adapted from [9].

```
!prison|punish,!fine|punish,!service|punish,!prison|deter,!prison|protect,
!prison|!rehab,!fine|deter,!service|!deter,!service|rehab,prison,fine,
service
```

Bibliography

[1] Aspartix answer set programming argumentation reasoning tool. http://www.dbai.tuwien.ac.at/proj/argumentation/systempage.

[2] The aspic argumentation system. http://www.cossac.org/projects/aspic.

[3] cohere ››› make the connection. http://cohere.open.ac.uk/.

[4] L. Amgoud and C. Cayrol. A model of reasoning based on the production of acceptable arguments. *Annals of Math. and A.I.*, 34:197–216, 2002.

[5] P. B. Andrews. Resolution with merging. *J. ACM*, 15(3):367–381, 1968.

[6] K. Atkinson, T. Bench-Capon, and P. McBurney. Parmenides: Facilitating democratic debate. pages 313–316, 2004.

[7] L. Bachmair, H. Ganzinger, and W. Snyder Ch. Lynch. Basic paramodulation and superposition. pages 462–476, 1992.

[8] T. Bench-Capon and P. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.

[9] T. Bench-Capon and H. Prakken. Justifying actions by accruing arguments. In *Proceeding of the 2006 conference on Computational Models of Argument*, pages 247–258, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.

[10] S. Benferhat, D. Dubois, and H. Prade. Argumentative inference in uncertain and inconsistent knowledge bases. In *Proceedings of the 9th Annual Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 1449–1445, 1993.

[11] Ph. Besnard and A.Hunter. Argumentation based on classical logic. *Argumentation in Artificial Intelligence*, pages 133–152, 2009.

[12] Ph. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128:203–235, 2001.

[13] Ph. Besnard and A. Hunter. Practical first-order argumentation. In *Proceedings of the 20th American National Conference on Artificial Intelligence (AAAI'2005)*, pages 590–595. MIT Press, 2005.

[14] Ph. Besnard and A. Hunter. Knowledgebase compilation for efficient logical argumentation. In *Proceedings of the 10th International Conference on Knowledge Representation (KR 2006)*, pages 123–133. AAAI Press, 2006.

[15] Ph. Besnard and A. Hunter. *Elements of Argumentation*. MIT Press, 2008.

[16] Ph. Besnard, A. Hunter, and S. Woltran. Encoding deductive argumentation in quantified boolean formulae. *Artif. Intell.*, 173(15):1406–1423, 2009.

[17] Wolfgang Bibel. On matrices with connections. *J. ACM*, 28(4):633–645, 1981.

[18] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93(1-2):63 – 101, 1997.

[19] D. Bryant. Argue tuprolog. `http://www.danielbryant.co.uk/research.htm`.

[20] D Bryant and P. Krause. A review of current defeasible reasoning implementations. *The Knowledge Engineering Review*, 23(03):227–260, 2008.

[21] D. Bryant, P. Krause, and G. Vreeswijk. Argue tuProlog: A lightweight argumentation engine for agent applications. In *Computational Models of Argument (Comma'06)*, pages 27–32. IOS Press, 2006.

[22] C. Cayrol, S. Doutre, and J. Mengin. Dialectical proof theories for the credulous preferred semantics of argumentation frameworks. In *Quantitative and Qualitative Approaches to Reasoning with Uncertainty*, volume 2143 of *LNCS*, pages 668–679. Springer, 2001.

[23] C. Chesñevar, A. Maguitman, and R. Loui. Logical models of argument. *ACM Computing Surveys*, 32:337–383, 2000.

[24] A. Church. A note on the entscheidungsproblem. *J. of Symbolic Logic*, 1:101–102, 1936.

[25] Y. Dimopoulos, B. Nebel, and F. Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence*, 141:57–78, 2002.

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

[27] P. Dung, R. Kowalski, and F. Toni. Dialectical proof procedures for assumption-based admissible argumentation. *Artificial Intelligence*, 170:114–159, 2006.

[28] P. E. Dunne and M. Wooldridge. Complexity of abstract argumentation. *Argumentation in Artificial Intelligence*, pages 85–104, 2009.

[29] V. Efstathiou. A logic-based argumentation system for decision support. MSc Thesis, University College London, 2006.

[30] V. Efstathiou and A.Hunter. Jargue: An implemented argumentation system for classical propositional logic (software demo). `http://www.ing.unibs.it/comma2010/demos/Efstathiou_etal.pdf`, 2010.

[31] V. Efstathiou and A. Hunter. Algorithms for effective argumentation in classical propositional logic: A connection graph approach. In *FoIKS*, pages 272–290. Springer, 2008.

[32] V. Efstathiou and A. Hunter. Focused search for arguments from propositional knowledge. In *Proceedings of the Second International Conference on Computational Models of Argument (COMMA'08)*. IOS Press, 2008.

[33] V. Efstathiou and A. Hunter. An algorithm for generating arguments in classical predicate logic. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'09)*, pages 119–130. Springer, 2009.

[34] V. Efstathiou and A. Hunter. Algorithms for generating arguments and counterarguments in propositional logic. *International Journal of Approximate Reasoning*, (accepted for publication), 2010.

[35] U. Egly, S. A. Gaggl, and S. Woltran. Aspartix: Implementing argumentation frameworks using answer-set programming. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 734–738, Berlin, Heidelberg, 2008. Springer-Verlag.

[36] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42:3–42, 1995.

[37] M. Elvang-Gøransson, P. Krause, and J. Fox. Dialectic reasoning with classically inconsistent information. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 114–121. Morgan Kaufmann, 1993.

[38] J. Fox, D. Glasspool, D. Grecu, S. Modgil, M. South, and V. Patkar. Argumentation-based inference and decision making–a medical perspective. *IEEE Intelligent Systems*, 22(6):34–41, 2007.

[39] D. Gaertner and F. Toni. Casapi - credulous and sceptical argumentation. `http://www.doc.ic.ac.uk/~dg00/casapi.html`.

[40] D. Gaertner and F. Toni. Casapi: a system for credulous and sceptical argumentation. In *Proc. Workshop on Argumentation for Non-monotonic Reasoning. (2007)*, pages 80–95, 2007.

[41] D. Gaertner and F. Toni. Computing arguments and attacks in assumption-based argumentation. *IEEE Intelligent Systems*, 22(6):24–33, 2007.

[42] D. Gaertner and F. Toni. Hybrid argumentation and its properties. In *Proceeding of the 2008 conference on Computational Models of Argument*, pages 183–195, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[43] J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Wiley, 1986.

[44] A. García and G. Simari. Delp client. `http://lidia.cs.uns.edu.ar/delp_client/index.php`.

[45] A. García and G. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.

[46] I. P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70(1-2):335–345, 1994.

[47] T. Gordon. Carneades argumentation system. `http://carneades.berlios.de/`.

[48] N. Gorogiannis, A. Hunter, and M. Williams. An argument-based approach to reasoning with clinical knowledge. *International Journal of Approximate Reasoning*, 51(1):1 – 22, 2009.

[49] R. Hirsch and N. Gorogiannis. The Complexity of the Warranted Formula Problem in Propositional Argumentation. *J Logic Computation*, page exp074, 2009.

[50] A. Hunter. Contouring of knowledge for intelligent searching for arguments. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 250–254. IOS Press, 2006.

[51] A. Hunter. Real arguments are approximate arguments. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 66–71. AAAI Press, 2007.

[52] A. Hunter and M. Williams. Harnessing ontologies for argument-based decision-making in breast cancer. In *ICTAI '07: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, pages 254–261, Washington, DC, USA, 2007. IEEE Computer Society.

[53] P. Krause J. Fox and M. Elvang-Gøransson. Argumentation as a general framework for uncertain reasoning. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence (UAI 1993)*, pages 428–434. Morgan Kaufmann, 1993.

[54] A. Kakas and F. Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9:515–562, 1999.

[55] R. Kowalski. A proof procedure using connection graphs. *Journal of the ACM*, 22:572–595, 1975.

[56] R. Kowalski. *Logic for problem solving*. North-Holland Publishing, 1979.

[57] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3-4):227 – 260, 1971.

[58] D. W. Loveland. Mechanical theorem-proving by model elimination. *J. ACM*, 15(2):236–251, 1968.

[59] D. W. Loveland. A linear format for resolution. In *Symposium on Automatic Demonstration*, pages 147–162. Springer, 1970.

[60] D. W. Loveland. *Automated theorem proving: A logical basis (Fundamental studies in computer science)*. sole distributor for the U.S.A. and Canada, Elsevier North-Holland, 1978.

[61] N. V. Murray and E. Rosenthal. Inference with path resolution and semantic graphs. *J. ACM*, 34(2):225–254, 1987.

[62] D. Nute. *Defeasible logics*, volume 3: Nonmonotonic Reasoning and Uncertainty Reasoning. Oxford University Press, 1994.

[63] S. Parsons, M. Wooldridge, and L. Amgoud. Properties and complexity of some formal inter-agent dialogues. *Journal of Logic and Computation*, 13(3):347–376, 2003.

[64] J. L. Pollock. Defeasible reasoning. *Cognitive Science*, 11:481–518, 1987.

[65] J. L. Pollock. How to reason defeasibly. *Artif. Intell.*, 57(1):1–42, 1992.

[66] H. Prakken. *Logical Tools for Modelling Legal Argument*. Law and Philosophy Library. Springer, 1997.

[67] H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.

[68] H. Prakken and G. Vreeswijk. Logical systems for defeasible argumentation. In D. Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer, 2000.

[69] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[70] B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[71] S. Buckingham Shum. Cohere: Towards web 2.0 argumentation. In *Proceeding of the 2008 conference on Computational Models of Argument*, pages 97–108, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[72] G. Simari and R. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53:125–157, 1992.

[73] M. South, G. Vreeswijk, and J. Fox. Argkit. `http://www.argkit.org`.

[74] M. South, G. Vreeswijk, and J. Fox. Dungine: a java dung reasoner. In *Proceeding of the 2008 conference on Computational Models of Argument*, pages 360–368, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[75] M. E. Stickel. Resolution theorem proving. *Annual Review of Computer Science*, 3(1):285–316, 1988.

[76] W. Visser. The epistemic and practical reasoner. `http://www.wietskevisser.nl/research/epr/`, 2008.

[77] G. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90:225–279, 1997.

[78] G. Vreeswijk. An algorithm to compute minimally grounded and admissible defence sets in argument systems. In *Computational Models of Argument (Comma'06)*, pages 109–120. IOS Press, 2006.

[79] D. Walton. *Informal logic: a handbook for critical argumentation*. Cambridge University Press, 1989.

[80] M. Wooldridge, P.E. Dunne, and S. Parsons. On the complexity of linking deductive and abstract argument systems. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*, pages 299–304. AAAI Press, 2006.

[81] L. Wos, R. Overbeck, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. Prentice Hall Professional Technical Reference, 1984.

[82] L. Wos, G. A. Robinson, and D. F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.

[83] T. Yuan and J. Schulze. Arg!draw 0.1 drawing argument graphs and playing argument games. `http://staff.unak.is/yuan/games/Arg!Draw/Arg!Draw.htm`.

Index