Applications of Games to Propositional Proof Complexity

by

Alexander Hertel

A Thesis Submitted in Conformity With the Requirements
For the Degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Created on May 13, 2008.

Abstract

Applications of Games to Propositional Proof Complexity

Alexander Hertel
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2008

In this thesis we explore a number of ways in which combinatorial games can be used to help prove results in the area of propositional proof complexity.

The results in this thesis can be divided into two sets, the first being dedicated to the study of Resolution space (memory) requirements, whereas the second is centered on formalizing the notion of 'dangerous' reductions.

The first group of results investigate Resolution space measures by asking questions of the form, 'Given a formula F and integer k, does F have a [Type of Resolution] proof with [Type of Resource] at most k?'. We refer to this as a proof complexity resource problem, and provide comprehensive results for several forms of Resolution as well as various resources. These results include the PSPACE-Completeness of Tree Resolution clause space (and the Prover/Delayer game), the PSPACE-Completeness of Input Resolution derivation total space, and the PSPACE-Hardness of Resolution variable space. This research has theoretical as well as practical motivations: Proof complexity research has focused on the size of proofs, and Resolution space requirements are an interesting new theoretical area of study. In more practical terms, the Resolution proof system forms the underpinnings of all modern SAT-solving algorithms, including clause learning. In practice, the limiting factor on these algorithms is memory space, so there is a strong motivation for better understanding it as a resource.

With the second group of results in this thesis we investigate and formalize what it means for a reduction to be 'dangerous'. The area of SAT-solving necessarily employs reductions in order to translate from other domains to SAT, where the power of highly-optimized algorithms can be brought to bear. Researchers have empirically observed that it is unfortunately possible for reductions to map easy instances from the input domain to hard SAT instances. We develop a non-Hamiltonicity proof system and combine it with additional results concerning the Prover/Delayer game from the first part of this thesis as well as proof complexity results for intuitionistic logic in order to provide the first formal examples of harmful and beneficial reductions, ultimately leading to the development of a framework for studying and comparing translations from one language to another.

Acknowledgements

First and foremost, I would like to thank Tanya for all of her love, effort, and support that she gave me during my time in graduate school at the University of Toronto.

Another major debt which I owe is to my supervisor Alasdair Urquhart. I could not have asked for a better mentor, and many of the results in this thesis would not have been possible without him. His keen insights, uncanny research sense, and encyclopedic knowledge of all areas of computer science and logic were truly invaluable and taught me a great deal about how to be a good researcher. I really enjoyed all of the good times we spent together and the many interesting conversations we had. KXW!

I would like to thank my brother Philipp for his contributions to the 'dangerous reductions' Chapter in this thesis, and for being such a great teammate during the many classes we took, both at U of T and UVic.

I am also very grateful to my mother for doing such a good job raising us and making so many sacrifices to give us the good education which ultimately led us here, and to my father for buying us our first computers so long ago and stimulating our interest in computer science.

Many thanks to all of the members of my Ph.D. committee for their helpful suggestions, and especially to Toni Pitassi for all of our very useful discussions and for helping with the NP-Hardness of approximating optimal Input Resolution proof size.

Thank you to Steve Cook who brought the Resolution width problem back with him from a conference in England, and for his helpful suggestions about where to publish our Tree Resolution clause space results.

Thank you to Charlie Rackoff for chairing every single one of my checkpoint meetings and for the many stimulating discussions we had. Charlie is without a doubt one of the most creative and insightful thinkers which I have had the fortune of knowing.

And thank you to Mike Molloy for his suggestions in one of the earlier checkpoints which definitely shaped the direction of this research and without a doubt improved the quality of this thesis a great deal.

One of the most significant lessons which I learned during my time in graduate school at the University of Toronto was from just being part of the Theory Group, which is simply saturated with exceptional researchers and role models. I was fortunate to work and interact with some of the very top researchers in Canada (or anywhere else, for that matter), and I will surely look back on my years here with a strong sense of intellectual reminiscence.

I would like to thank Fahiem Bacchus for his many valuable comments and suggestions regarding the 'dangerous reductions' chapter and François Pitt for his thesis template and willingness to answer Latex questions. I would also like to thank Sam Buss for editing his proof tree package at my request to allow upside-down proofs, and Moshe Vardi for his stimulating conjecture concerning the complexity of Resolution width.

My sincerest gratitude goes out to Charles Morgan, Valerie King, Ulrike Stege, and Hausi Müller in Victoria for getting us started in research and encouraging us to go to grad school, and especially to Ulrike for supporting us in our directed studies research and for the work she did with us leading to our first publication.

Many thanks also to our family friends Cliff and Liz in Victoria for all of their love and support.

Thanks to all of the friends we met during our studies including Marc, Alan, Kleoni, Vlad, Daniela, Nazanin, and Mark. Marc, I have fond memories of staying up all night working on assignments, going skating, playing squash, and working on derivations with you. I also have very fond memories of all of the different characters Philipp and I worked with and ran into during our time at U of T and UVic.

I am also grateful to Sara Burns, who valiantly came to my rescue on more than one occasion after I had been defeated by various dangerous creatures such as jammed staplers, photocopiers, and fax machines.

Last but not least, I would like to thank the Department of Computer Science at the University of Toronto as well as NSERC for supporting me financially. NSERC in particular helped me tremendously with their many generous scholarships, and I will always be grateful.

Contents

List of Figures

Part I

Preliminaries

Chapter 1

Thesis Overview

1.1 Summary & Motivation

This thesis falls in the area of propositional proof complexity, and also contains topics in computational complexity, graph theory, and circuit complexity. It consists of three parts: Part I provides a thesis summary as well as an overview of propositional proof complexity. Part II contains research pertaining to Resolution space measures. Finally, Part III contains results concerning the more traditional resource of proof complexity size and is focused on the study and formalization of 'dangerous reductions'. Apart from proof complexity, which is the topic of every chapter in this thesis, its main recurring theme is the use of combinatorial games and game characterizations as a technique for proving new results. Our main games of focus are the Prover/Delayer game, as well as several versions of the black and black-white pebbling games on DAGs and monotone circuits. An additional motivation tying together Parts II and III is that both sets of results have strong applications to research in the area of SAT-solvers and automated theorem proving.

Part I of this thesis includes this introductory chapter as well as Chapter 2, in which we give a broad overview of propositional proof complexity including its most important definitions, results, and descriptions of the most well-studied proof systems. Results and definitions more specific to Parts II and III are located in their respective introductory chapters.

Some conventions worth mentioning immediately are that throughout this thesis we use separate fonts to refer to proof systems (e.g. RES, Frege) and complexity classes (e.g. $\mathcal{NP}$, $\mathcal{PSPACE}$). Formal languages such as SAT are capitalized in accordance with the usual practice in complexity theory. We assume that the reader is already somewhat familiar with the basics of proof complexity, complexity theory, graph theory, circuit complexity, and the Resolution proof system. We shall use [CK01] as our reference for standard proof complexity terminology, and to [GJ79, Pap94] as our references for computational complexity.

The second part of this thesis contains some of its most important results, namely those pertaining to the complexity of Resolution space measures. Roughly speaking, space refers to the memory requirements needed in order to compute a proof. Propositional proofs have many resources associated with them such as size, width, and several different measures of space. All of these resources are worth studying, and proof complexity resource problems ask questions of the form, 'Given a formula F and integer k, does F have a [Proof System] proof of [Resource] at most k?'. More specifically, we are interested in the complexity of determining the answer to these questions, and in these chapters we prove that a number of such resource problems for various forms of Resolution are $\mathcal{PSPACE}$-Complete by applying pebbling formulas in order to reduce from pebbling games whose complexities are already known. This research is located contiguously in Chapters 4 through 7, and our main results in this part are the $\mathcal{PSPACE}$-Completeness of Tree Resolution clause space, the $\mathcal{PSPACE}$-Completeness of Input Resolution total space, and the $\mathcal{PSPACE}$-Hardness of Resolution variable space. In proving the $\mathcal{PSPACE}$-Completeness of Tree Resolution clause space, we also prove the $\mathcal{PSPACE}$-Completeness of the Prover/Delayer game, an important combinatorial game which is closely related to the Tree Resolution proof system.

There are both academic as well as practical motivations for studying Resolution space and other resources such as width. Theoretically, Resolution proof size has been well-studied but its space complexity is a relatively new area of research with many stimulating open problems that have generated interest in the theory community. However, apart from its relevance to proof complexity, circuit complexity, and general complexity theory, Resolution space research also has important implications for the areas of automated theorem proving and SAT-solving. This is because previous results show that Resolution space and width lower bounds imply size lower bounds. For example, in [BSW01], Ben-Sasson and Wigderson show that for any unsatisfiable formula, its minimal Resolution refutation size is exponential in its minimum width. Similarly, by combining the results in [BSIW04] and [ET03], we get a result showing that for any CNF formula, the size of its minimum Tree Resolution refutation is exponential in its minimum Tree Resolution clause space. Large minimum refutation size immediately implies long refutation search runtimes, and since virtually all modern SAT-solving algorithms are based on some form of Resolution, these relationships could be of great practical value. If efficient algorithms existed for determining Resolution width or Tree Resolution clause space, researchers would be able to use them as preprocessing steps on formulas to determine whether or not to even attempt running their SAT-solvers; if a formula has a large minimum width or tree clause space, then there is no sense in running the SAT-solver because it is guaranteed to take too much time. Unfortunately, there is little hope for this plan of attack because our results show that determining clause space requirements for DPLL and clause learning algorithms is $\mathcal{PSPACE}$-Complete. In other words, instead of trying to use space solvers as preprocessing to predict failure, researchers would be better off simply solving the formula.

The possibility of approximation algorithms for these problems still exists, but since the relationship between size and tree clause space is exponential, even an approximation to within a constant factor would probably be of limited use in practice, since being off by even a tiny amount in the exponent would give dramatically different answers.

Another practical motivation behind this research has to do with combined resources for SAT-solvers: The limiting factor on most modern SAT-solving algorithm tends to be memory space, but on the other hand, if one is too frugal with memory, then proofs can become intractably large. Algorithms must therefore carefully balance attempts to save memory with a potentially massive increase in minimum proof size and running time if they are too aggressive in their savings. We hope that future researchers will be able to build on the results in this part of the thesis in order to better understand the tension between size and space.

The third and final part of this thesis contains a series of interrelated results dedicated to the study and formalization of a phenomenon from the area of automated theorem proving and SAT-solving called 'dangerous reductions'. The entire strategy behind SAT-solvers and propositional reasoning is to translate a diverse range of $\mathcal{NP}$-Hard problems to SAT, where highly-optimized SAT algorithms can be brought to bear. Researchers have discovered empirically that there is a danger in this strategy, namely that not all translations are equal, and in fact it is possible for a reduction to map easy instances from the input domain to very hard SAT instances. In this part of the thesis we combine a variety of research topics including the development of a new proof system for non-Hamiltonicity, Prover/Delayer game upper bounds, and the proof complexity of intuitionistic logic to give the first formal examples of dangerous and beneficial reductions, ultimately allowing us to develop a framework for comparing competing encodings. These results are located contiguously in Chapters 9 through 12 and are important mainly because they highlight and formalize dangers and opportunities of which researchers in the area of SAT-solving should be aware.

The other results supporting this main one are also interesting in their own rights. For example, the Non-Hamiltonicity Proof System is a novelty because it is the first of its kind and it is graphical, whereas most proof systems for co$\mathcal{NP}$-Complete languages deal with propositional logic. The fact that the Prover/Delayer game from Part II can be used to prove Tree Resolution size upper bounds is interesting because it is the final result needed to show that the Prover/Delayer game completely captures both the size and space aspects of Tree Resolution, and can be used to simplify proofs. Finally, the results concerning intuitionistic logic are important because it is the most-studied non-classical logic, is currently the subject of a fair amount of research interest, and it provides another formal example of a dangerous reduction.

For more detailed summaries of Parts II and III, please refer to Chapters 3 and 8 respectively.

1.2 Relationship To Games

The results in this thesis explore several ways in which combinatorial games can be used to help prove results in the area of propositional proof complexity. For instance, in Part II we reduce from the black pebbling game on monotone circuits, the black pebbling game on DAGs, and the black-white pebbling game on monotone circuits in order to respectively determine the complexities of calculating Tree Resolution clause space, Input Resolution total space, and Resolution variable space.

However, the application of games in this thesis runs deeper than simply using them as good starting points for reductions. For example, in Chapter 4 we do not reduce from the black pebbling game on monotone circuits directly to the Tree Resolution clause space problem, but rather to the Prover/Delayer game. However, we know from [ET03] that the Prover/Delayer game is an exact characterization of Tree Resolution clause space, so our desired results follow immediately. In this case it is much easier to prove complexity results for the corresponding game than to attack the Resolution resource problem directly. This suggests that researchers attempting to prove hardness results for future resource problems may find it useful to first develop game characterizations for the quantities in question and reduce to those games instead.

In Part III of this thesis we make further use of the Prover/Delayer game by showing that it can be used to give simplified Tree Resolution size upper bounds. We then apply this technique in order to prove polynomial upper bounds for various formulas. These facts are vital to the main results concerning our 'dangerous reductions' research.

In this thesis we therefore make three distinct uses of games in order to prove results in the area of propositional proof complexity:

1. As starting points for reductions,
2. As game characterizations for problems, making them easier to attack, and
3. As a simplified method for proving proof size upper bounds.

1.3 Organization

As already mentioned, the results in this thesis are organized into two main parts, which are self-contained but related by the common themes of propositional proof complexity, the use of games and game characterizations as proof techniques, and their relevance to automated theorem proving and SAT-solvers.

Each part begins with an introductory chapter containing preliminaries such as an overview and important definitions which shall be used throughout that part. In both parts, this introductory chapter is followed by four chapters containing the research and main results of this thesis.

Each of the research chapters has a layout similar to a research paper: It starts with a detailed introduction and motivation for the research, as well as additional definitions, if necessary. This is followed by proofs of the results, and finally a list of open problems.

As of this writing, two of the chapters have been published, and another has been submitted: Chapter 4 was presented at CSL 2007 in Lausanne [HU07], Chapter 11 was presented at SAT 2007 in Lisbon [HHU07], while Chapter 5 has been accepted for publication by the Journal on Satisfiability, Boolean Modeling and Computation [HU08a].

The research paper version of Chapter 7 was accepted by Theory of Computing [HU08b], but unfortunately we discovered a subtle yet fatal flaw in one of the main proofs and were compelled to withdraw its submission. The corresponding chapter in this thesis has been updated accordingly, and we hope to soon correct the problem and resubmit it.

1.4 Future Research

Several reviewers have commented that many of the results and ideas in this thesis can be viewed as initial steps to potentially interesting areas of future research, and lend themselves very well to being extended. Indeed, it is our hope that this work will indeed be continued. Once again, the primary motivation for this future research comes from the area of automated theorem proving and SAT-solving. For example:

- The rigorous treatment of space and other Resolution resource problems in Part II of this thesis has brought many open problems to light (for example, see the problems in the table of Section 3.4.2 for which a gap still exists). In addition, these results can be viewed as initial steps into the investigation and understanding of the important tension between size and space for Resolution-based proof systems, described in Section 4.9.7.

- Another important line of research from Part II of this thesis comes from Section 5.4 in which we prove that Input Resolution is automatizable. Although in practice this is somewhat limited by the fact that this proof system is not even complete, the area of automated theorem proving would certainly benefit from more results of this kind.

- The results from Part III also suggest several open problems and future avenues of research. Although we were able to formalize the intuitive notions of dangerous and beneficial reductions and use them to establish a framework for comparing competing SAT encodings, these results are geared towards classification rather than prediction. As described in Section 11.5, the problem of predicting which reductions are beneficial or harmful remains an important open problem.

- In addition, the notion of effective p-simulation came up in Part III in relation to dangerous reductions, but this idea has a great deal of potential, both as a type of preprocessing algorithm for SAT-solvers, but also as an entirely new way of comparing proof systems. For more information, please refer to Section 11.5.

These are just a few of the many different research paths the work in this thesis might take. Several other open problems are described at the end of each chapter.

Chapter 2

An Overview of Proof Complexity & Broad Definitions

2.1 Introduction & Motivation

In order to provide a context for the later chapters and a motivation for our main results, we begin this thesis by giving a high-level description of propositional proof complexity, including major definitions as well as descriptions of important proof systems.

Propositional proof complexity is the study of the lengths of propositional proofs in various different proof systems and is ultimately aimed at settling the open problem of whether the complexity class $\mathcal{NP}$ is closed under complement (i.e. whether $\mathcal{NP}$ and co$\mathcal{NP}$ are equal). Although the history of logic and proofs dates back to antiquity, the formal study of proof complexity is relatively new. Tseitin completed the first major proof complexity research in the 1960s [Tse70]. This work predated complexity theory by a few years, but it nevertheless contains many of the fundamental ideas of modern proof complexity. After Cook's seminal 1971 paper [Coo71] which established complexity theory, he and Reckhow did the groundbreaking work [CR74, Rec76, CR79] which also established proof complexity as an important field of research.

Throughout this thesis we will closely follow their definitions, formalizations, and terminology. Unless otherwise stated, formulas are in conjunctive normal form and interpreted as sets of clauses. We typically refer to formulas using the letters F or Σ. Similarly, given a formula, we use the letters n and m to respectively refer to the number of distinct variables and clauses, and N to refer to its symbol length. We refer to graphs using the letter G and in this context use the letters n and m to refer to $|V|$ and $|E|$ respectively.

If the $\mathcal{P}$ vs. $\mathcal{NP}$ question is the most important open problem in theoretical computer science, then the $\mathcal{NP}$ vs. co$\mathcal{NP}$ question is probably the second most important one. Despite decades of very focused research, both of these problems remain very much open, and many researchers in the area believe that new ideas and techniques will be required in order to make any progress. The motivation for studying propositional proof complexity goes beyond the $\mathcal{NP}$ vs. co$\mathcal{NP}$ question and could potentially affect both of these open problems. If $\mathcal{NP} \neq co\mathcal{NP}$, then $\mathcal{P} \neq \mathcal{NP}$ follows as an immediate corollary because $\mathcal{P}$ is closed under complement, so if $\mathcal{NP}$ is not, then they cannot be the same sets. These would of course be tremendous results.

Another motivation for studying proof complexity comes from the area of automated theorem proving and propositional reasoning. More specifically, the results in proof complexity can translate into lower bound results for algorithms. Creating better SAT-solvers has become a field of research in its own right, and many of the algorithms used in practice have corresponding proof systems for which exponential lower bounds have been proven. These lower bounds give us a sense of the inherent limitations of the current implementations. The motivations for studying proof complexity are therefore both theoretical and practical.

2.2 Proof Complexity Definitions & Terminology

Propositional proof systems have been traditionally defined by logicians in a very structural way. For instance, one way to define a proof system is to provide a set of simple logical axioms together with a set of inference rules. The Cook-Reckhow definition of a propositional proof system (with good reason) is somewhat different:

Definition 2.2.1 (Proof System [CR74, Rec76, CR79]). *A proof system for a language $L \subseteq \Sigma^*$ is a polytime computable onto function $f : \Sigma_p^* \to L$ where Σ_p is some alphabet.*

Intuitively, Σ_p is an alphabet of 'proof symbols', and f maps proofs to the elements in L that they prove. One advantage of this definition is that it immediately brings propositional proof systems into the realm of computer science by formally requiring f to be feasibly computable. Even more importantly, the words 'polytime computable' enforce our intuitive notion that when given a proof, we should be able to tell whether a proposed proof of a claim is correct or not within a reasonable amount of time. This rules out the possibility of absurd proof systems which map arbitrary strings to elements in L.

Regardless of whether we follow the Cook-Reckhow definition or the traditional structural definition, proof systems must be sound; for example, a proof system for tautologies should not be able to prove a non-tautology. Similarly, proof systems must be complete; for example, a proof system for unsatisfiability must be capable of proving the unsatisfiability of every possible unsatisfiable formula. Another characteristic of the Cook-Reckhow definition is that it implicitly demands soundness and completeness. Soundness is guaranteed because the range of f is contained in L. The fact that f is onto guarantees completeness.

Loosely speaking, complexity theory is focused on the study of the complexity class $\mathcal{NP}$ whereas proof complexity is focused on the study of its complement, the complexity class co$\mathcal{NP}$. The following are equivalent definitions of $\mathcal{NP}$:

Definition 2.2.2 ($\mathcal{NP}$). $\mathcal{NP} = \{\mathcal{L} \mid \mathcal{L}$ *is decidable by a nondeterministic Turing Machine in polynomial time*$\}$

Definition 2.2.3 ($\mathcal{NP}$). $\mathcal{NP} = \{\mathcal{L} \mid$ *Each* $x \in \mathcal{L}$ *has a poly-length certificate which can be verified by a deterministic Turing Machine in polynomial time*$\}$

For our purposes, the latter definition is more intuitive because certificates are synonymous with proofs. The other major class that we are interested in is co$\mathcal{NP}$. It is defined as containing the complements of the languages in $\mathcal{NP}$:

Definition 2.2.4 (co$\mathcal{NP}$). *co*$\mathcal{NP} = \{\overline{\mathcal{L}} \mid \mathcal{L} \in \mathcal{NP}\}$

2.3 Polynomially-Bounded Proof Systems & co$\mathcal{NP}$

By the definition of $\mathcal{NP}$, we know that every language in $\mathcal{NP}$ has polynomial-length certificates (proofs). However, it is totally unclear whether the same can be said for the languages in co$\mathcal{NP}$. This leads us to another fundamental definition:

Definition 2.3.1 (Polynomially-Bounded Proof System). *The proof system $f : \Sigma_p^* \to L$ is said to be polynomially-bounded if for all $y \in L$ there exists an $x \in \Sigma_p^*$ such that $y = f(x)$ and $|x| \leq p(|y|)$, where $p(x)$ is some polynomial.*

For example, SAT, the canonical $\mathcal{NP}$-Complete language, does have a polynomially-bounded proof system: Just take f to map (F, α) to F, where F is a satisfiable formula and α is a truth assignment which satisfies it. All syntactically incorrect proofs are mapped to an arbitrary formula outside of SAT. By contrast, nobody knows whether SAT's complement, $\overline{SAT}$ has a polynomially-bounded proof system; proofs that a formula is not satisfiable seem to be inherently longer than proofs for satisfiability. Note that in practice, we are not actually interested in the language $\overline{SAT}$, but rather we are interested in UNSAT. There is a subtle distinction between these languages. Technically speaking, since the complement of a language L is defined to be $\Sigma^* - L$, $\overline{SAT}$ contains many strings that do not even encode formulas. UNSAT is $\overline{SAT}$ without these 'garbage' strings; i.e. it contains only the unsatisfiable formulas. In any case, these 'garbage' strings are decidable by a DTM in polytime, so this is not a problematic distinction.

The two co$\mathcal{NP}$-Complete languages that we are most interested in are UNSAT (the set of all logically false, or unsatisfiable formulas) and TAUT (the set of all logically true formulas, or tautologies). Just as SAT is $\mathcal{NP}$-Complete, so is UNSAT co$\mathcal{NP}$-Complete. FALSIFIABILITY (FAL for short), the language consisting of all falsifiable formulas, is another $\mathcal{NP}$-Complete language. Its counterpart in co$\mathcal{NP}$ is TAUT. TAUT and UNSAT are the most studied co$\mathcal{NP}$-Complete languages, and most propositional proof systems in existence were devised for proving tautologies or refuting unsatisfiable formulas. The relationship between these languages is shown in the Venn diagram in Figure 2.1 below.

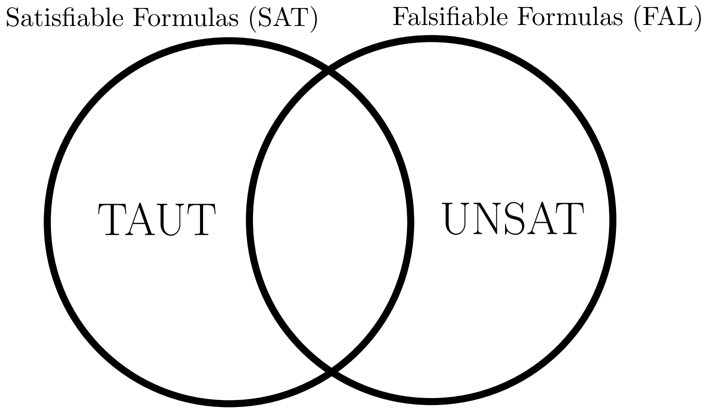

Figure 2.1: The relationship between SAT, FAL, UNSAT, and TAUT

One of the fundamental theorems of propositional proof complexity formalizes the relationship between the $\mathcal{NP}$ vs. co$\mathcal{NP}$ problem and polynomially-bounded proof systems:

Theorem 2.3.2 ([CR79]). *$\mathcal{NP} = co\mathcal{NP}$ if and only if there exists a polynomially-bounded proof system for some co$\mathcal{NP}$-Complete language.*

A proof that there are no polynomially-bounded or 'super' proof systems for TAUT would therefore immediately imply that $\mathcal{NP} \neq co\mathcal{NP}$, and would as already stated therefore also imply that $\mathcal{P} \neq \mathcal{NP}$. In contrast, a positive answer to the $\mathcal{NP}$ vs. co$\mathcal{NP}$ question would not necessarily imply that $\mathcal{P} = \mathcal{NP}$, since it may be the case that all tautologies have short certificates, but they may take exponentially long to find (similarly, all formulas in SAT have short certificates, but they may take exponentially long to find in the worst case).

2.4 A Description of Major Propositional Proof Systems

In this section we shall describe a number of the more important proof systems. Many propositional proof systems have been studied and categorized into a hierarchy based on their relative efficiencies. A portion of this hierarchy is shown in Figure 2.2 at the end of this section and is based on the concept of p-simulation, which allows us to objectively compare two proof systems with respect to efficiency.

Informally, a proof system α is said to p-simulate another proof system β if for every unsatisfiable CNF formula F, there exists an α refutation of F which is at most a polynomial factor larger than F's smallest β refutation.

A proof system α is said to be exponentially separated from another proof system β if there exists some set of formulas S such that for some constant c and all $F \in S$, $|\pi_\beta(F)| \geq 2^{c|\pi_\alpha(F)|}$, where $\pi_\alpha(F)$ and $\pi_\beta(F)$ are respectively the shortest α and β proofs of F. Such a separation of course implies that β cannot p-simulate α. If α p-simulates β and β does not p-simulate α, then we say that α is strictly stronger than β. More formally p-simulation is defined as follows:

Definition 2.4.1 (P-Simulation & P-Equivalence). *Let $f_1 : \Sigma_1^* \to L$ and $f_2 : \Sigma_2^* \to L$ be proof systems for L. If for all $x_1 \in \Sigma_1^*$, there exists an $x_2 \in \Sigma_2^*$ such that $f_1(x_1) = f_2(x_2)$ where $|x_2| \leq p(|x_1|)$ for some polynomial p, then we say that f_2 weakly p-simulates f_1.*

In addition, if there exists a polytime computable function $t : \Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$, $f_1(x) = f_2(t(x))$, then we say that f_2 strongly p-simulates f_1.

Proof systems which strongly p-simulate each other are said to be p-equivalent.

The distinction between weak p-simulation and strong p-simulation is that strong p-simulation is more constructive in that it requires a 'proof translating function' t. Since it is therefore a stronger form of simulation, it is seen more often in positive results, whereas weak p-simulation is seen when referring to negative results. In the remainder of this thesis, we omit the adjectives 'strong' and 'weak'. Instead, whenever referring one proof system p-simulating another, we shall implicitly be referring to strong p-simulation, and whenever referring to a separation between two proof systems, we shall implicitly be referring to a lack of weak p-simulation.

In fact, we can even compare proof systems over different languages by using a more general definition of p-simulation called 'effective p-simulation':

Definition 2.4.2 (Effective P-Simulation). *Let $f_1 : \Sigma_1^* \to L_1$ and $f_2 : \Sigma_2^* \to L_2$ be proof systems. If there exists a k and a polytime reduction $r : L_1 \to L_2$ such that $y \in L_1$ if and only if $r(y) \in L_2$ and for all $x_1 \in \Sigma_1^*$ there exists an $x_2 \in \Sigma_2^*$ such that $r(f_1(x_1)) = f_2(x_2)$ and $|x_2| \leq |x_1|^k$, then we say that f_2 effectively weakly p-simulates f_1.*

If there also exists a polytime computable function $t : \Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$, $r(f_1(x)) = f_2(t(x))$, then f_2 effectively strongly p-simulates f_1.

The idea of p-simulation by proof systems for different languages dates back to [Rec76], and shall be further explored when we discuss our Non-Hamiltonicity Proof System in Chapter 9 and dangerous reductions in Chapter 11. Intuitively, when dealing with proof systems over different languages, one must necessarily use a reduction. Unfortunately, this reduction may affect the proof complexity of the formulas being translated, so the definition of effective p-simulation must take the reduction into account.

However, the applicability of effective p-simulation is not limited to proof systems on different languages. In fact, it is a potentially powerful tool which can be used to relate proof systems on the same language. Please refer to Sections 11.3.3 and 11.4 for a description of how effective p-simulation can yield very useful reductions and preprocessing algorithms for SAT-solvers.

2.4.1 Truth Tables

Truth tables (TT) are perhaps the most obvious way of proving formulas to be tautologies or unsatisfiable. A truth table for a formula F on n sentence letters is a table in which each row contains one of the 2^n possible truth assignments along with the truth value of F under that assignment. The disadvantage of TT is that as a proof system they can be extremely inefficient. Since a formula of length N can contain up to $O(N)$ sentence letters, truth tables require $\Omega(2^N)$ rows. In effect, TT cannot weakly p-simulate any proof system that has polynomially-bounded proofs for some family of formulas in which the number of sentence letters grows linearly with the length of the formula. As such, this proof system is considered to be one of the weakest, and is often described as a 'brute-force' system.

2.4.2 Analytic Tableaux

Also called 'truth trees', Analytic Tableaux (AT) is another 'weak' proof system. It is a refutation system that is typically used to show that formulas in conjunctive normal form (CNF) are unsatisfiable. When dealing with CNF formulas, it is equivalent and often simpler to view the formula in question as a set of clauses. The underlying structure of AT is a tree, and the basic idea is to choose a clause at each node in the tree and branch on it so that each child is labelled with one of the literals of that clause. The root node is unique in that it does not contain any literals. As soon as a node v contains a literal such that its negation is found in one of its ancestors in the tree, then the branch ending in v is closed off and becomes a leaf. Once all the paths from the root to the leaves are closed off, the proof is complete. The tree that is produced is called a 'tableau'.

The size of a tableau refutation is defined to be its number of interior nodes. It should be noted that in a tableau refutation of minimal size, no path contains repeated literals. The pruning technique used to eliminate duplicate literals is described in [Urq95].

Exponential lower bounds for AT are due to Cook [Coo75], and further described in [APU01, Urq95]. The idea is to build sets of contradictory clauses based on complete binary trees. These clauses force any tableau refutation to contain an enormous amount of necessary repetition. A testament to the weakness of AT is that it cannot even p-simulate TT [D'A92, Urq95]. Similarly, TT cannot p-simulate AT.

2.4.3 Resolution

Resolution in all of its many forms is perhaps the best-studied propositional proof system and it forms the basis of almost all modern SAT-solving algorithms. Much like AT, Resolution is a refutation system for unsatisfiable CNF formulas. The resolution rule is as follows: Given two clauses $(A \vee x)$ and $(B \vee \overline{x})$, we resolve on the variable x to derive the new clause $(A \vee B)$. A resolution refutation of a contradictory set of clauses consists of the application of a sequence of resolution steps until the empty clause $\emptyset$ is derived. This proof can be written as a sequence of clauses in which each is either an initial clause or follows by the resolution rule from two previous ones. It is also possible to represent a resolution proof as a directed graph instead of as a sequence of clauses. Each vertex in these graphs is labelled with a clause. Initial clauses have in-degree 0, whereas all other clauses have in-degree 2, with the edges indicating which clauses they were derived from.

The size of a Resolution refutation is sometimes used to refer to its length encoded as a string, and is sometimes used to refer to the number of clauses it contains.

Tree Resolution

If the underlying graph of a resolution proof forms a tree, then we say that the proof is 'tree-like'. Intuitively, this implies that each clause may only be resolved on once, so if a clause needs to be used again, it must be re-derived. Tree Resolution (T-RES) is a very limited form of Resolution, and its exponential lower bounds are easily understood.

As already mentioned, there is considerable interest in SAT-solvers. One such SAT algorithm, called DPLL after its authors [DLL62], is widely-used and successful. It turns out that DPLL corresponds almost exactly to T-RES as a proof system. The DPLL algorithm works by building a binary tree in which the root contains the original formula, and each edge represents a restriction leading to a node containing the resulting restricted formula. The algorithm terminates once every leaf contains a clause that has been falsified.

Ordered Resolution

In Ordered Resolution (O-RES), each variable x is systematically eliminated by performing all possible resolutions involving it. The formula to be solved is therefore reduced from containing n to $n-1$ sentence letters. Whereas the graph structure underlying T-RES is a tree, the graph structure underlying O-RES is a directed acyclic graph (DAG). This means that clauses may be used as inputs for any arbitrary number of resolutions. The algorithmic implementation of O-RES is called DP, again after its authors [DP60]. As a

SAT-solver, it motivated the invention of DPLL, and the exponential lower bounds for O-RES correspond to algorithmic lower bounds for DP.

Regular Resolution

Regular Resolution (R-RES) is resolution in which the underlying structure is a DAG, but irregularities are disallowed, so once a variable has been eliminated along a branch of the refutation, it may not be introduced again. Irregularities are formally defined as follows:

Definition 2.4.3 (Irregularity). *Let $C_1, C_2, ..., C_{k-1}, C_k$ be the clauses along a path in the DAG underlying a refutation π. If C_1 and C_k contain a literal l but $C_2, ..., C_{k-1}$ do not, then that path is said to contain an 'irregularity', and π is said to be irregular.*

In what was to become the first major proof complexity lower bound, Tseitin proved superpolynomial size lower bounds for R-RES [Tse70]. This lower bound is based on families of formulas that are constructed using odd-charged square grid graphs. These graphs and their corresponding formulas are described well in [Urq87]. Galil used similar arguments involving bipartite expander graphs to improve this lower bound to exponential [Gal77].

In much the same way that no branch in an AT refutation contains a repeated literal, all T-RES refutations of minimal size are regular. That is to say, given an irregular T-RES refutation, it is always possible to remove all irregularities while shortening the proof [Tse70, Urq95], and clearly a regular proof has a depth of at most n, since a formula only contains n distinct variables. Since every Regular Tree Resolution (RT-RES) proof is a T-RES proof, this shows that RT-RES and T-RES are p-equivalent. Similarly, O-RES systematically eliminates variables one at a time, and therefore trivially produces R-RES proofs. In effect, R-RES subsumes both T-RES and O-RES, so exponential R-RES lower bounds immediately translate into exponential lower bounds for these other two Resolution-based systems.

Algorithmically, RT-RES is very important. In the previous section, we commented about how T-RES is almost identical to DPLL. In fact, DPLL is the exact algorithmic incarnation of RT-RES. This is easy to see, since all DPLL proofs are binary trees, and they are of course regular, since a variable which has been eliminated by restriction cannot reappear along a branch. Furthermore, the tree built by the DPLL algorithm forms an inverted RT-RES proof for exactly the same formula. Size lower bounds for T-RES (such as the exponential lower bounds for the Pigeonhole Principle) therefore translate directly into algorithmic size (and time) lower bounds for DPLL (and all other Resolution-based SAT-solving algorithms). This is a good example of how proof complexity lower bounds can yield algorithmic lower bounds.

General Resolution

General Resolution (RES) is DAG-like resolution without any restrictions such as regularity. For many years, researchers tried unsuccessfully to apply the Tseitin and Galil techniques in order to extend the R-RES lower bounds to RES. The breakthrough was made in [Hak85] by Haken. Instead of using graphs to build contradictory formulas, Haken used the pigeonhole principle (PHP). The PHP is a combinatorial principle which states that it is impossible to put n pigeons into $n-1$ holes such that no two pigeons share the same hole. The formula encoding the negated PHP is called PHP^n_{n-1}, and its variables are of the form $m_{i,j}$, with the intended meaning that $m_{i,j}$ being set to true means that pigeon i is mapped to hole j. The formula PHP^n_{n-1} consists of the following clauses:

$\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{n-1} m_{i,j})$ *i.e. The mapping is total; every pigeon maps to at least one hole.*

$\bigwedge_{j=1}^{n-1}\bigwedge_{i_1=1}^{n}\bigwedge_{\substack{i_2=1\\ i_1\neq i_2}}^{n}(\neg m_{i_1,j} \vee \neg m_{i_2,j})$ *i.e. The mapping is 1-1; no hole has more than one pigeon mapped to it.*

Since the PHP is clearly true, its negated form PHP^n_{n-1} must be unsatisfiable. Haken showed that any RES proof of PHP^n_{n-1} requires exponentially many clauses. Although ingenious, his argument was somewhat complicated. It has since been simplified and improved by Beame & Pitassi [BP96] as well as Urquhart [Urq03]. A third very clear simplified exposition is given in [Pit02a]. Although slightly improved, Haken's result remains essentially the same:

Theorem 2.4.4. *For sufficiently large n, any* RES *proof of PHP^n_{n-1} requires at least $2^{\frac{n}{20}}$ clauses.*

At a high-level the proof is relatively straightforward: First prove that every RES refutation of PHP^n_{n-1} contains a clause with at least $\frac{2}{9}n^2$ literals. Define a 'large' clause as being one that has at least $\frac{1}{10}n^2$ literals. Assume that there exists a refutation of PHP^n_{n-1} that contains fewer than $2^{\frac{n}{20}}$ clauses. Repeatedly restrict this proof and make all obvious simplifications so as to eliminate all large clauses. The resulting restricted proof is a refutation of $PHP^{n'}_{n'-1}$, where n' is the number of variables in the restricted proof. However, $\frac{2}{9}{n'}^2$ equates to a little bit more than $\frac{1}{10}n^2$, so the resulting proof must contain a large clause. However, we eliminated all large clauses, a contradiction.

Variations of the PHP also exist; for example, we can insist that the mapping from pigeons to holes be a function. The formula encoding the functional PHP is denoted $fPHP^n_{n-1}$ and includes the following clauses in addition to those mentioned above:

$$\bigwedge_{i=1}^{n}\bigwedge_{j_1=1}^{n-1}\bigwedge_{\substack{j_2=1\\ j_1\neq j_2}}^{n-1}(\neg m_{i,j_1}\vee\neg m_{i,j_2})\ \textit{i.e. The mapping is a function; no pigeon is mapped more than once.}$$

The onto PHP includes all of the above clauses, but also requires that the mapping be onto [CK01]. It is denoted $ontoPHP^n_{n-1}$ and includes the following clauses:

$$\bigwedge_{j=1}^{n-1}(\bigvee_{i=1}^{n} m_{i,j})\ \textit{i.e. The mapping is onto; every hole has at least one pigeon mapped to it.}$$

Intuitively, since these variants contain more clauses than the 'normal' PHP, and extra clauses could only help when trying to find shorter proofs, $fPHP$ and $ontoPHP$ should be easier for proof systems to deal with. Nevertheless, Haken's original proof generalizes easily to show that even $fPHP$ and $ontoPHP$ require exponentially long RES proofs.

Exponential lower bounds did not close the book on RES research. In [Urq87], Urquhart applied ideas from Haken's argument in order to find essentially optimal graph-based exponential lower bounds for RES. He showed that RES has $2^{\Omega(|\Sigma|)}$ size lower bounds, thereby completing the line of research started by Tseitin.

Considerable research has also gone into optimizing the separation between the different versions of Resolution. In [AJPU07], Alekhnovich, Johannsen, Pitassi, and Urquhart prove an exponential separation between RES and R-RES. Similarly, in [BSIW04], Ben-Sasson, Impagliazzo, and Wigderson use pebbling graphs to prove a near-optimal separation between RES and T-RES. More information on this last result and pebbling graphs can be found in Section 3.3.

Limited Extension

One final concept relevant to Resolution is 'limited extension'. Apart from his superpolynomial R-RES lower bound, Tseitin's other major contribution in [Tse70] was limited extension. It is not hard to see that proof systems for UNSAT can easily be turned into systems for TAUT (and vice-versa) by simply negating the formula in question. For example, RES can be used to prove tautologies, even if they use logical connectives which it cannot handle; just take a tautology F and negate it to produce the unsatisfiable formula $\neg F$. Convert this formula to its CNF equivalent, and RES will be able to refute it, thereby proving that the original formula F was a tautology. Unfortunately, some formulas grow exponentially when converted to

CNF. Tseitin realized that this problem can be overcome by turning $\neg F$ into a new formula F' that is not equivalent to $\neg F$, but rather is satisfiable if and only if $\neg F$ is. The trick is to introduce a new 'limited extension' variable for every non-atomic subformula of $\neg F$. For example, let us assume that we want to use RES to show that the biconditional tautology $F = (p \equiv (q \equiv (p \equiv q)))$ is in fact a tautology. First we negate it to get $\neg(p \equiv (q \equiv (p \equiv q)))$. Next we recursively create extension variables, one for each subformula:

1. $a \equiv (p \equiv q)$
2. $b \equiv (q \equiv a)$
3. $c \equiv (p \equiv b)$
4. $\neg c$

Finally, we turn each of these equivalences into clauses by observing that $x \equiv (y \equiv z)$ is logically equivalent to $(x \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee z)$. Of course, $\neg c$ remains a singleton clause. By taking a conjunction of all these clauses we have produced the desired CNF formula which is only linearly longer than F. We therefore have a feasible translation procedure that allows us to use RES to prove arbitrary tautologies, regardless of the basis. Of course, the introduction of extension variables is by no means restricted to RES.

2.4.4 Gentzen's System LK

Gentzen systems refer to the logical proof systems due to Gerhard Gentzen [Gen35a, Gen35b]. Originally published in German, his work has since been translated into English [Sza69]. Good overviews can be found in both [Coo02] and [CK01]. Gentzen's original system, called LK, was formulated for first-order logic, where LK stands for 'Logischer Kalkül'. He also formulated a similar system called LJ for intuitionistic logic, which we shall use extensively in Chapter 12. Interestingly, Gentzen never seems to explain what LJ stands for; possibly the letter 'J' was chosen simply because it precedes 'K'. Technically speaking, LK is a formulation for first-order logic, but we are only interested in its propositional fragment. There has been some effort to distinguish this propositional part of LK by calling it PK, but we shall use the more standard form, since it is clear that we are dealing with propositional logic.

The LK proof system is called a sequent calculus because every line in a LK proof is a 'sequent'. A sequent is a logical implication of the form $\Gamma \mapsto \Delta$ where Γ and Δ are sets of formulas. Sequents are interpreted as meaning that a conjunction of all the formulas on the left of the $\mapsto$ symbol imply a disjunction of all the formulas on the right. For example, if $\Gamma = \{A_1, A_2, ..., A_k\}$ and $\Delta = \{B_1, B_2, ..., B_k\}$ then $\Gamma \mapsto \Delta$ is equivalent to the formula $(A_1 \wedge A_2 \wedge ... \wedge A_k) \supset (B_1 \vee B_2 \vee ... \vee B_k)$.

LK is typically used as a proof system for TAUT, and its axioms are all sequents of the form $p \mapsto p$, where p is some sentence letter. The following are the standard LK inference rules (with the quantifier rules omitted):

Weakening:

$$\textbf{Left}\quad \frac{\Gamma \mapsto \Delta}{A, \Gamma \mapsto \Delta} \qquad \text{and} \qquad \textbf{Right}\quad \frac{\Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, A}$$

$\neg$ Introduction:

$$\textbf{Left}\quad \frac{\Gamma \mapsto \Delta, A}{\neg A, \Gamma \mapsto \Delta} \qquad \text{and} \qquad \textbf{Right}\quad \frac{A, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, \neg A}$$

$\wedge$ Introduction:

$$\textbf{Left}\quad \frac{A, B, \Gamma \mapsto \Delta}{A \wedge B, \Gamma \mapsto \Delta} \qquad \text{and} \qquad \textbf{Right}\quad \frac{\Gamma \mapsto \Delta, A \quad \Gamma \mapsto \Delta, B}{\Gamma \mapsto \Delta, A \wedge B}$$

$\vee$ Introduction:

$$\textbf{Left} \quad \frac{A, \Gamma \mapsto \Delta \quad B, \Gamma \mapsto \Delta}{A \vee B, \Gamma, \mapsto \Delta} \qquad \text{and} \qquad \textbf{Right} \quad \frac{\Gamma \mapsto \Delta, A, B}{\Gamma \mapsto \Delta, A \vee B}$$

Cut:

$$\frac{\Gamma \mapsto \Delta, A \quad A, \Gamma \mapsto \Delta}{\Gamma \mapsto \Delta}$$

Although LK is normally formulated over the basis $\wedge, \vee, \neg$, it is possible to formulate a Gentzen system which includes rules for any logical connectives. Much like RES, LK can be characterized as being either DAG-like or tree-like depending on whether derived sequents are allowed to be reused or not. Unlike RES, however, tree-like and DAG-like LK are p-equivalent, provided that they both include the cut rule [CK01, p.267]. If the cut rule is not included, then the picture changes dramatically. Both tree-like and DAG-like LK with cut are p-equivalent to Frege systems, whereas cut-free DAG-like LK can be p-simulated by RES. This last result is non-trivial, since the resolution rule is so similar to cut, and there are some issues about the languages being discussed. An interested reader should refer to [Urq92, Urq95]. In addition, cut-free tree-like LK is very weak; it is only p-equivalent to AT. Whether cut-free LK can p-simulate RES is an open problem.

2.4.5 Frege Systems

Frege systems (Frege), also called Hilbert-style systems, are a group of robust and powerful proof systems that are of particular interest in the area of proof complexity. Any sound and complete proof system that includes a finite number of axiom schemes instead of axioms is considered to be a Frege system. Axiom schemes are axiom templates that allow for a simultaneous and uniform substitution of any arbitrary formulas for sentence letters. For example, if a Frege system has an axiom scheme of the form $p \rightarrow p$, then a substitution of the formula $(q \vee r)$ for p yields the axiom $(q \vee r) \rightarrow (q \vee r)$, which can be introduced at any point in a proof. In effect, Frege systems have an infinite number of axioms. It appears that tree-like incarnations of proof systems are weaker than their DAG-like counterparts only among the proof systems low down in the proof complexity hierarchy, because much like Gentzen's system LK with cut, tree-like and DAG-like Frege systems are p-equivalent [CK01, p.372].

It turns out that Frege systems are a bit of a misnomer; Frege's original system did not use axiom schemes, but rather defined axioms and explicitly allowed for substitution into those axioms by way of a substitution rule. Ironically, his original system was therefore not technically a Frege system. The use of axiom schemes rather than the substitution rule is due to Von Neumann [Neu27].

Related to Frege systems, Natural Deduction systems (ND) are sound and complete proof systems that include the 'deduction theorem' as an inference rule. The deduction theorem takes the form, If $\Gamma, A \vdash B$ then $\Gamma \vdash A \supset B$, where '$\supset$' is standard implication, or some logically equivalent form. In his Ph.D. thesis [Rec76], Reckhow proves that all Frege systems and ND systems as well as Gentzen's system with cut are p-equivalent.

All Frege systems and their p-equivalents are quite powerful, and no superpolynomial (let alone exponential) lower bounds are known. These systems are widely believed to have exponential lower bounds, and proving this remains a major open problem.

2.4.6 Bounded-Depth Frege Systems

We do have lower bounds for a special weakened class of Frege systems, however. When we place a restriction on Frege systems requiring that all formulas can have a depth of at most a constant d, we create a new class of proof systems called Bounded-Depth Frege systems (AC^0-Frege).

Definition 2.4.5 (Formula Depth). *The depth of a formula F is defined as follows: Convert F into a tree in the obvious way and push all negations to the leaves. Next, allow unbounded fan-in and produce the tree-like circuit C by merging all blocks of $\wedge$ and $\vee$ gates so that no gates of the same type are adjacent. The depth of F is the number of $\wedge, \vee$ alternations along the longest path from the root of C to a leaf.*

For example, all CNF and all DNF formulas have depth 2.

Bounded-Depth Frege systems are alternatively called AC^0-Frege Systems. This is a reference to the circuit class AC_0, which contains poly-sized bounded (constant) depth circuits. Exponential lower bounds for AC^0-Frege Systems were proved using techniques based on circuit complexity lower bounds for parity [Ajt83, FSS84]. The relationship between these results is chronicled in [CK01] and [UF96].

The key idea is that of a switching lemma, which allows us to efficiently convert a restricted formula from DNF to CNF and vice-versa. Håstad's switching lemma, published in [Hås87], but described particularly clearly in both [Bea94] and [Pit02b], is stated as follows:

Lemma 2.4.6. *Let F be any arbitrary DNF formula on n variables with terms of length at most r. Then for all $s \geq 0$, all $p \leq \frac{1}{7}$, and all $l = pn$,*

$$\frac{|B_n^l|}{|R_n^l|} < (7pr)^s$$

Where R_n^l is the set of all restrictions on n variables that leave l unset, and B_n^l is the subset of R_n^l such that each $\alpha \in B_n^l$ causes the canonical decision tree of $F\lceil_\alpha$ to have height $\geq s$.

Intuitively, the switching lemma is saying that the number of 'bad' restrictions in B is insignificant when compared with the total number of restrictions, so there are many 'good' length $n-l$ restrictions $\beta \in R_n^l - B_n^l$ which allow us to convert $F\lceil_\beta$ from DNF to CNF according to the following argument: Since paths in the canonical decision tree of $F\lceil_\beta$ leading to 0-leaves correspond to falsifying assignments of $F\lceil_\beta$'s terms, these paths tell us the terms in the DNF of $\neg F\lceil_\beta$. If we push another negation through this formula, we are left with $F\lceil_\beta$ in CNF. In effect, the height of the tree is a bound on the length of the terms in the DNF of $\neg F\lceil_\beta$ as well as the length of the clauses in the CNF of $F\lceil_\beta$. By controlling the height of the tree, we control the lengths of the terms and clauses.

Håstad's switching lemma can be applied to prove that parity has no bounded-depth circuits [Bea94], which as already stated was the inspiration for the proof of AC^0-Frege exponential lower bounds.

2.4.7 Extended Frege Systems

Frege systems can be augmented to create proof systems that are potentially even more powerful. One way of augmenting Frege systems is to add an extension rule that is similar to Tseitin's idea of limited extension, except that it can be applied arbitrarily to any formula at any point in the Frege proof. More specifically, extension allows for the arbitrary introduction of the formula $p \equiv G$, where p is a sentence letter not found in G, nor any previous location in the Frege proof, nor in the conclusion. Intuitively, the extension rule allows for formulas to be abbreviated. The proof systems resulting from augmenting Frege systems with extension are called Extended Frege (E-Frege) systems, and they trivially p-simulate Frege systems. Much like Frege systems, all E-Frege systems are p-equivalent [Rec76].

2.4.8 Substitution Frege Systems

Another augmented class of Frege systems are called Substitution Frege (S-Frege) systems. These systems are simply Frege systems augmented with a substitution rule which allows us to take any formula F in the proof and uniformly substitute any arbitrary formula G for all occurrences of an arbitrary sentence letter p in F. This rule is written as

$$\frac{F}{F(G/p)}$$

In effect, S-Frege systems regard every newly derived formula as being an axiom scheme. Again, S-Frege systems trivially p-simulate Frege systems. Reckhow also proved that S-Frege systems p-simulate E-Frege systems [Rec76]. E-Frege systems were later shown to p-simulate S-Frege systems [CK01, p.396], thereby showing them to be p-equivalent.

2.4.9 Extended Resolution

Another powerful proof system, Extended Resolution (E-RES) is RES with the extension rule. E-RES was formulated by Tseitin in his landmark paper [Tse70]. The distinction between E-RES and RES with limited extension is that while the latter only allows for the use of extension variables on subformulas in the formula to be refuted, E-RES allows extension to be used on any subformulas within the proof. E-RES is p-equivalent to E-Frege and S-Frege [Rec76].

2.5 Non-Propositional Proof Systems

It turns out that the phrase 'Propositional Proof Complexity' has become a bit of a misnomer because proof systems are not necessarily proof systems for propositional logic. For instance, a great deal of work has gone towards understanding the so-called 'algebraic proof systems'. Cutting Planes (CP) is a proof system for refuting inconsistent systems of linear inequalities [BP01, CK01]. The systems Nullstellensatz (NS) and Polynomial Calculus (PC) [CK01, p.306] are also of great interest to proof complexity researchers. Exponential lower bounds are known for all of these proof systems.

2.5.1 Cutting Planes

Motivated by integer programming, Cutting Planes (CP) is a proof system that is used to refute a set of linear inequalities. A linear inequality is of the form $\Sigma_i a_i x_i \geq k$, where a_i and k are integers, and the underlying variables are x_i.

A CP refutation of a set of linear inequalities $L = \{C_1, C_2, \ldots, C_q\}$ is a sequence of inequalities, S_1, $S_2, \ldots, S_m$ where each S_i is either from L or is an axiom or follows from two previous inequalities by a valid rule, and the final inequality is $S_m = (0 \geq 1)$. A CP proof may equivalently be viewed as a DAG. A more restricted form of CP is tree-like CP (TCP) in which the underlying structure of the proof is required to be a tree. The leaves of the tree each correspond to one of the initial linear inequalities, the root corresponds to the final inequality S_m, and the edges correspond to inference rules.

In [BPR97], Bonet, Pitassi, and Raz describe a restricted form of TCP which we shall call TCP*. It is identical to TCP except that coefficients must be bounded by a polynomial.

2.5.2 Polynomial Calculus & Nullstellensatz

Although beyond the scope of this thesis, Nullstellensatz (NS) [CK01, p.308] and Polynomial Calculus (PC) [CK01, p.316] are algebraic refutation systems. We mention them here briefly because like our Non-Hamiltonicity Proof System in Chapter 9, they are non-propositional systems. PC is strictly stronger than NS [CK01, p.316]. PC is a refutation system for unsatisfiable sets of polynomial equations. A PC refutation of a polynomial P is a derivation of 1 from P. For more information on these systems, we refer an interested reader to [CK01].

2.5.3 Hajós Calculus

Another interesting non-propositional proof system which also has nothing to do with algebra or polynomials is the Hajós Calculus (HC). It is a graph theoretic proof system for proving non-3-colourability, which is co$\mathcal{NP}$-Complete. Its complexity was an open problem for many years until Pitassi and Urquhart showed that it is effectively p-equivalent to E-Frege [PU95].

2.5.4 Relating Propositional & Non-Propositional Systems

Some relationships between proof systems are immediately obvious. For example, AC^0-Frege systems p-simulate RES, which in turn p-simulates T-RES. Similarly, CP obviously p-simulates TCP, which p-simulates TCP*. It is also not hard to see that CP p-simulates RES (and that TCP p-simulates T-RES) [CK01, p.345].

Furthermore, CP has poly-sized refutations of PHP formulas, which shows that even AC^0-Frege cannot p-simulate it [CK01, p.348].

Perhaps not as obvious is that TCP does not p-simulate RES [CK01, p.365]. Furthermore, AC^0-Frege and TCP^* do not p-simulate each other [BPR97]. As mentioned before, PC p-simulates NS, but the algebraic proof systems seem to be incomparable with the others, since AC^0-Frege systems cannot p-simulate NS, and NS cannot even p-simulate T-RES [BP01].

It is not hard to see that since AC^0-Frege systems cannot p-simulate TCP^* or NS, none of the proof systems below AC^0-Frege in the middle column of the Proof System Hierarchy shown in Figure 2.2 can p-simulate any of the algebraic or polynomial proof systems in the left- or right-most columns.

2.6 Summary of the Relationships Between Major Proof Systems

The following diagram gives a summary of the relationships between the major proof systems discussed in this chapter. An arrow from proof system A to proof system B indicates that A p-simulates B. Whenever a slash appears on an arrow, it means that an exponential separation between the two systems is known. Question marks indicate open problems, and when multiple proof systems appear in the same box, it means that they are all p-equivalent. In order to minimize clutter, simulation and separation arrows which are implied by transitivity have been omitted. For more details on the simulations and separations, please refer to the proceeding sections describing the relevant proof systems. For a more detailed diagram showing the relationship between Resolution-based proof systems, please refer to Figure 11.1.

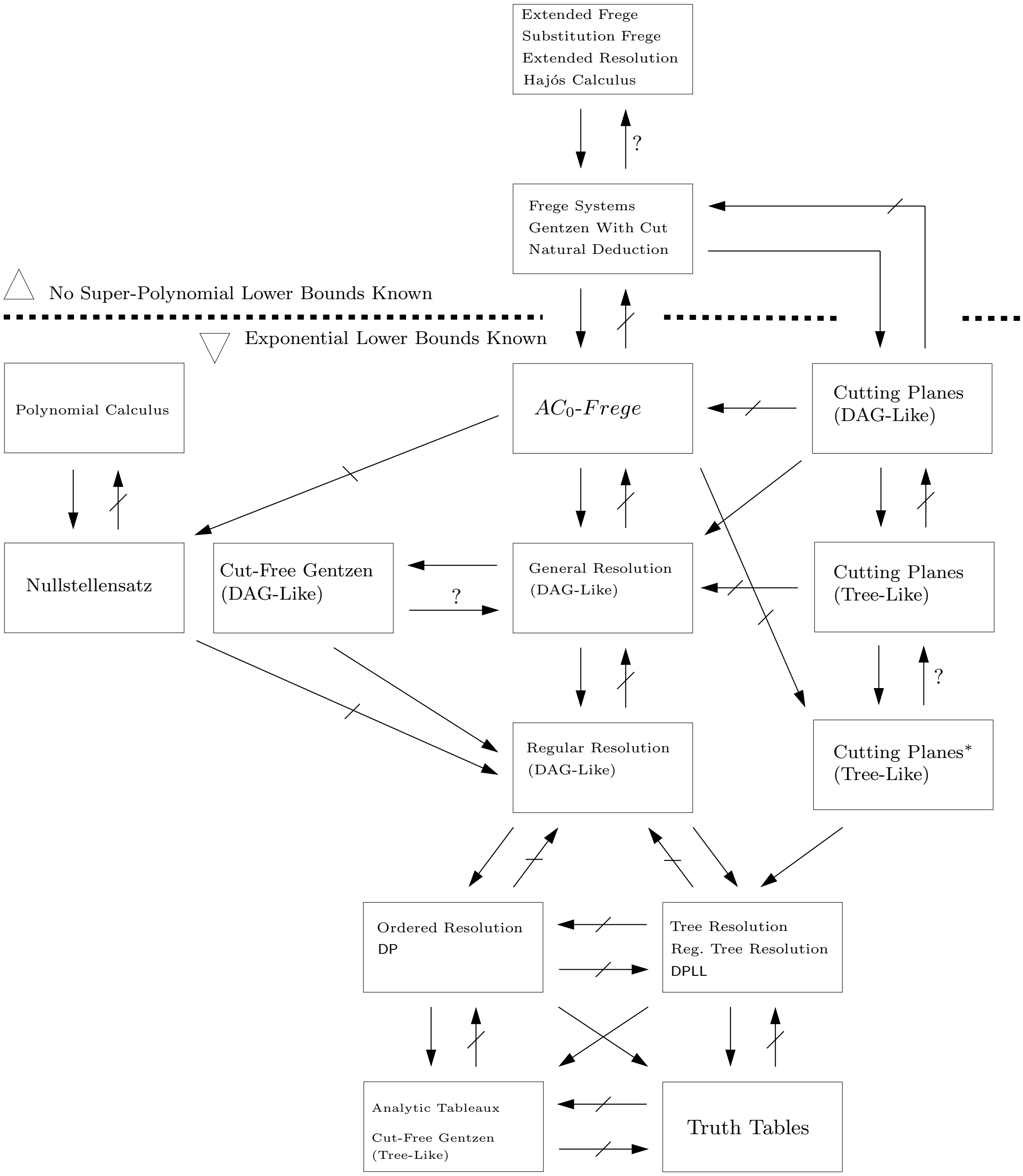

Figure 2.2: The Proof Complexity p-Simulation Hierarchy

Part II

The Complexity of Resolution Space Measures

Chapter 3

Introduction to Part II

This chapter serves as an introduction to Part II of this thesis, which is focused on the study of proof complexity resource problems (see Definition 3.2.7), and especially on space resources for Resolution-based proof systems, where space intuitively is the amount of memory required to compute a proof. We will start in Section 3.1 below by giving a brief history of Resolution space research. This is followed by Section 3.2, which contains definitions specific to this part of the thesis, and Section 3.3, which describes important previous results. Finally, in Section 3.4 we shall give a summary of the next four chapters, which contain this part's research results.

3.1 History of Resolution Space Research

The investigation of proof complexity space requirements is a surprisingly new research area which seems to have originated independently as separate threads on opposite sides of the Atlantic ocean.

In Europe, the first serious space research was carried out by Juan Luis Esteban and Jacobo Torán, who were the first to use the term 'clause space'. However, in fairness they did not invent the concept entirely on their own, but rather adapted and perfected a previous (but slightly inadequate) definition due to Kleine Büning and Lettman [BL94]. Their space research begins with Esteban's M.Sc. [Est95] and Ph.D. [Est03] thesis work, and also encompasses several papers including [Tor99], and [ET01]. Esteban and Torán were also the first to observe the intimate and important relationship between Resolution space and pebbling games.

Meanwhile, in North America, general questions about the space requirements of computing propositional proofs were first raised by Armin Haken during the 1998 'Complexity Lower Bounds' workshop held at the Fields Institute in Toronto [ABSRW01]. The first paper to address Haken's questions was [ABSRW01] by Alekhnovich, Ben-Sasson, Razborov and Wigderson, who were aware of the research done by Esteban and Torán, and were the first to define our notion of 'total space'. In a related paper, Ben-Sasson proved several tradeoff results between Resolution size and space [BS02], re-emphasizing the connection between space and pebbling.

At this point the two threads merge with a third from the area of finite model theory. In [AD03], Atserias and Dalmau investigate RES width in the context of the $(\exists, k)$-pebble game of Kolaitis and Vardi [KV95] and us it to relate Resolution width and space.

Other important space results include [ET03], which equated the Prover/Delayer game from [PI00, BSIW04] with $\mathsf{T\text{-}RES}$ clause space, and Nordström's more recent work [Nor06], which won the best student paper award at STOC 2006, showing that Resolution space research does have a mainstream appeal as well.

This provides the research setting framing the work in this part of the thesis, and it is easy to see how our research extends but also continues to unify these previous threads. As for the specific history of the research in this thesis, we had been investigating the area of Resolution space resource problems for about six months when we were further motivated by a conjecture made by Moshe Vardi which was relayed to Toronto by Stephen Cook in the early part of July 2006. Vardi conjectured that the problem of determining whether an arbitrary unsatisfiable formula F has a RES refutation of width k is $\mathcal{EXPTIME}$-Complete. This stimulating conjecture inspired us to redouble our research efforts into Resolution resource problems.

3.2 Definitions Specific To Part II

As opposed to Chapter 2 in which we gave broad, high-level definitions and major results, this section contains many of the important recurring definitions and previous results which are relevant specifically to the research in this part of the thesis, including definitions, concepts such as pebbling formulas, pebbling games, and Resolution space. Many of these definitions and results are somewhat non-standard and are not contained in textbooks or survey articles. This section should therefore be used as a reference for the remainder of this part of the thesis. Note that this section mainly contains recurring definitions and previous results; those relevant to only a single chapter tend to be included in them rather than here.

Since it is the main focus of this thesis as well as the most generalized form of Resolution, most of our definitions are given for the RES proof system. However, the majority of these definitions apply equally to all Resolution-based proof systems such as T-RES and I-RES, and efforts have been made to point out when significant differences exist.

3.2.1 Resolution, Size, & Width

The notions of size and width can be formalized using a fairly simple definition of what constitutes a RES proof, whereas the notions of space require slightly more complicated definitions.

Definition 3.2.1 (Resolution Proof). *A clause C is a set of literals. We use the notation $C \vee D$ for the clause $C \cup D$, and write $C \vee l$ for $C \cup \{l\}$, where l is a literal. If $C \vee x$ and $D \vee \neg x$ are clauses, then the resolution rule allows us to derive the clause $C \vee D$, by resolving on the variable x. If F is a set of clauses, then the sequence of clauses $\pi = C_1, C_2, ..., C_k$ is a* RES *proof of C_k from F if each C_i in π appears in F (i.e. is an input, or initial clause) or follows from two previous clauses in π by the resolution rule.*

*If the graph underlying the structure of π is a tree (i.e. each clause in π is a premise for at most one application of the resolution rule), then the proof is said to be a Tree-Like Resolution (*T-RES*) proof. Otherwise it is said to be DAG-Like. A* RES *refutation of F is a* RES *proof from F in which $C_k = \emptyset$ (the empty clause).*

This allows us to define size and width:

Definition 3.2.2 (Resolution Size & Width). *Given a refutation proof system α, if an α proof π of a formula F contains k occurrences of clauses, then the size of π, denoted $Size(\pi)$, is k. Similarly, the size of the smallest α refutation of F is denoted $Size(F \vdash_\alpha \emptyset)$. The width of a clause C refers to how many literals it contains, and is denoted $w(C)$. The width of a formula F is the width of the widest clause in F, and is denoted $w(F)$. The width of an α proof π is equal to the width of its widest clause, and is denoted $w(\pi)$. Finally, the minimum width of any α refutation of F is denoted $w(F \vdash_\alpha \emptyset)$.*

3.2.2 Resolution Space

The definitions of the various different measures of space that we shall be discussing requires an alternative definition of RES proof which depends on the notion of configuration. Intuitively, a configuration is the set of clauses stored in memory at any time, and a proof can be viewed as a sequence of configurations.

Definition 3.2.3 (Configuration-Style RES, T-RES, RT-RES, I-RES, and U-RES Proof). *A configuration $\mathbb{C}$ is a set of clauses. If F is a formula (set of clauses), then the sequence of configurations $\pi = \mathbb{C}_0, \mathbb{C}_1, ..., \mathbb{C}_k$ is a* RES *proof of C from F if $\mathbb{C}_0 = \emptyset$, $C \in \mathbb{C}_k$, and for each $i < k$, $\mathbb{C}_{i+1}$ is obtained from $\mathbb{C}_i$ by one of the following rules:*

1. *Deleting one or more of its clauses,*

2. *Adding the resolvent of two clauses of $\mathbb{C}_i$, or*

3. *Adding one or more of the clauses of F (initial clauses).*

*The proof π is said to be a Tree Resolution (*T-RES*) proof if we replace rule 2 with the following:*

2. *Adding the resolvent of two clauses of $\mathbb{C}_i$* ***and deleting both parent clauses****.*

*Similarly, π is said to be a Regular Tree Resolution (*RT-RES*) proof if it is a* T-RES *proof in which the underlying proof tree does not contain any irregularities.*
*In addition, π is said to be an Input Resolution (*I-RES*) proof if at least one input to every instance of the resolution rule is an input clause, and a Linear Resolution (*L-RES*) if the output of each application of the resolution rule is one of the inputs to the next application.*

*Finally, if at least one input to every instance of the resolution rule is a unit clause (i.e. a clause which contains just one literal), then we say that π is a Unit Resolution (*U-RES*) proof.*

In any type of Resolution proof, the first two clauses resolved on are called the 'top clauses' of π. The final result of the refutation, C_k is called the 'Goal Clause'. If $\emptyset \in C_k$, then we refer to π as a refutation.

This leads us to our different definitions of space. Intuitively, space is the amount of memory required in order to compute a proof π. For each of these measures, it is important to distinguish between space for RES, T-RES, RT-RES, and I-RES.

Note that our terminology differs from that introduced by Alekhnovich, Ben-Sasson, Razborov and Wigderson in [ABSRW01], and Ben-Sasson in [BS02]. Our notions of 'clause space' are identical, but what Ben-Sasson refers to as 'variable space', we refer to as 'total space'. Our notion of 'variable space' is not explicitly defined in [BS02], but does appear implicitly in one of the results (see Theorem 3.3.5 and Corollary 3.3.6 below).

Definition 3.2.4 (Clause Space). *Let F be a set of clauses and π be a configuration-style α proof of clause C from F, where α is* RES *or one of its refinements. The clause space of a configuration $\mathbb{C}$ in π, denoted $CS(\mathbb{C})$, is the number of clauses in $\mathbb{C}$. The clause space of π, denoted $CS(\pi)$, is the maximum $CS(\mathbb{C})$ over all $\mathbb{C}$ in π. Finally, the clause space of resolving C from F, denoted $CS(F \vdash_\alpha C)$, is the minimum $CS(\pi)$ over any α proof π of C from F.*

Definition 3.2.5 (Total Space). *Let F be a set of clauses and π be a configuration-style α proof of clause C from F, where α is* RES *or one of its refinements. The total space of a configuration $\mathbb{C}$ in π, denoted $TS(\mathbb{C})$, is defined as $\sum_{C\in\mathbb{C}} w(C)$. The total space of π, denoted $TS(\pi)$, is the maximum $TS(\mathbb{C})$ over all $\mathbb{C}$ in π. Finally, the total space of resolving C from F, denoted $TS(F \vdash_\alpha C)$, is the minimum $TS(\pi)$ over any α proof π of C from F.*

Definition 3.2.6 (Variable Space). *Let F be a set of clauses and π be a configuration-style α proof of clause C from F, where α is* RES *or one of its refinements. The variable space of a configuration $\mathbb{C}$ in π, denoted $VS(\mathbb{C})$, is the number of distinct variables in $\mathbb{C}$. The variable space of π, denoted $VS(\pi)$, is the maximum $VS(\mathbb{C})$ over all $\mathbb{C}$ in π. Finally, the variable space of resolving C from F, denoted $VS(F \vdash_\alpha C)$, is the minimum $VS(\pi)$ over any α proof π of C from F.*

For example, given the formula $F = \{\{x_1, x_2\}, \{\neg x_2, x_1\}, \{\neg x_1, x_3\}, \{\neg x_3, \neg x_1\}\}$, the following table shows a step-by-step configuration-style T-RES refutation π of F together with the clause space, total space, and variable space of each step:

Step	Configuration	Clause Space	Total Space	Variable Space
1	$\{\}$	0	0	0
2	$\{\ \{x_1, x_2\}\ \}$	1	2	2
3	$\{\ \{x_1, x_2\}, \{\neg x_2, x_1\}\ \}$	2	4	2
4	$\{\ \{x_1\}\ \}$	1	1	1
5	$\{\ \{x_1\}, \{\neg x_1, x_3\}\ \}$	2	3	2
6	$\{\ \{x_1\}, \{\neg x_1, x_3\}, \{\neg x_3, \neg x_1\}\ \}$	3	5	2
7	$\{\ \{x_1\}, \{\neg x_1\}\ \}$	2	2	1
8	$\{\ \emptyset\ \}$	1	0	0

It is not hard to see that $CS(\pi) = 3$, $TS(\pi) = 5$, and $VS(\pi) = 2$. It is similarly easy to see that for all F, $CS(F \vdash_{\mathsf{RES}} C) \leq TS(F \vdash_{\mathsf{RES}} C)$, and $w(F \vdash_{\mathsf{RES}} C) \leq VS(F \vdash_{\mathsf{RES}} C) \leq TS(F \vdash_{\mathsf{RES}} C)$. The same holds for any Resolution refinement such as T-RES or I-RES, and clearly $CS(F \vdash_{\mathsf{RES}} C) \leq CS(F \vdash_\alpha C)$, $TS(F \vdash_{\mathsf{RES}} C) \leq TS(F \vdash_\alpha C)$, and $VS(F \vdash_{\mathsf{RES}} C) \leq VS(F \vdash_\alpha C)$ for any formula and any Resolution refinement α.

The measures $VS(F \vdash_{\mathsf{RES}} C)$ and $CS(F \vdash_{\mathsf{RES}} C)$ are incomparable, since each of these quantities can be made larger than the other by choosing an appropriate formula for F. For example, let $F_1 = \{\{x_1\}, \{\neg x_1\}\}$, and let $F_2 = \{\{x_1, x_2, x_3\}, \{\neg x_1\}, \{\neg x_2\}, \{\neg x_3\}\}$. $VS(F_1 \vdash_{\mathsf{RES}} \emptyset) = 1$, and $CS(F_1 \vdash_{\mathsf{RES}} \emptyset) = 2$, whereas $VS(F_2 \vdash_{\mathsf{RES}} \emptyset) = 3$ and $CS(F_2 \vdash_{\mathsf{RES}} \emptyset) = 2$.

These definitions of space immediately raise interesting questions about popular proof systems. For example, given a formula F and integer k, does F have a RES refutation of clause space at most k? There is nothing special about space; in fact, we can ask similar questions for any interesting resource such as size, space, width, or depth. We refer to this type of question as a proof complexity resource problem, and define it formally as follows:

Definition 3.2.7 (Proof Complexity Resource Problem). *A proof complexity resource problem asks questions of the following form: Given two formulas F and G and an integer k, does there exist a [Type of Proof System] derivation of G from F with [Type of Resource] at most k?*

In Part II of this thesis we settle the complexity of several proof complexity resource problems for Resolution-based proof systems. We can of course add more constraints to the definition, but usually consider cases in which F is unsatisfiable, and G is the empty clause $\emptyset$.

3.2.3 Pebbling Circuits & Games

The investigation of RES space is closely associated with the well-known pebbling game and pebbling number of a DAG, originally explored in [Coo73, CS76] as a means of investigating bounds on storage requirements. There are several different versions of pebbling games, but the most popular ones are captured by the following generalized description:

Definition 3.2.8 (Pebbling Games on Monotone Circuits). *The generalized black-white pebbling game is a single-player game where the goal is to 'pebble' a monotone circuit C in which each node is an AND gate, an OR gate, or a source. The leaves of C have in-degree 0 and are referred to as 'source' nodes. A single vertex in C is referred to as a 'target' or 'output' node, and has out-degree 0. All non-target nodes can have arbitrary out-degree, except of course when C is a tree, in which case the maximum out-degree is 1. In all cases, non-source nodes can have arbitrary in-degree. A circuit in which all non-source nodes have in-degree 2 is called a 'binary circuit'. The game involves two different types of pebbles (black and white), and has the following rules:*

1. *The game starts with no pebbles on the circuit.*

2. *A gate may have at most one pebble on it at any time.*
3. *At any point, the player may place any pebble onto any vacant source node or remove any pebble from any source.*
4. *At any point, the player may remove any black pebble from any gate or place a white pebble on any empty gate.*
5. *For any unpebbled AND gate v, if all of v's immediate predecessors have pebbles on them, then the player may place a black pebble on v. Alternatively, if the sliding rule is present, the player may slide a black pebble (if present) from u to v, where u is a predecessor of v.*
6. *For any unpebbled OR gate v, if at least one of v's immediate predecessors has a pebble on it, then the player may place a black pebble on v. Alternatively, if the sliding rule is present, the player may slide a black pebble (if present) from u to v, where u is a predecessor of v.*
7. *For any AND gate v with a white pebble on it, if all of v's immediate predecessors have pebbles on them, then the player may remove the white pebble from v. Alternatively, if the sliding rule is present and all but one of v's predecessors are pebbled, then the player my slide the white pebble from v to the remaining unpebbled predecessor of v.*
8. *For any OR gate v with a white pebble on it, if at least one of v's immediate predecessors has a pebble on it, then the player may remove the white pebble from v. Alternatively, if the sliding rule is present, then the player may slide the white pebble from v to any unpebbled predecessor of v.*
9. *The game ends once the circuit has a black pebble on the target node and there are no white pebbles present on any nodes.*

Most of the pebble games found in the literature can be viewed as restricted versions of this generalized game on monotone circuits. For example, if all nodes in C are AND gates, then we usually refer to it using the letter 'G' and call it a 'DAG'. The game resulting from this restriction is the standard 'black-white pebbling game'. Similarly, we can further restrict the game by disallowing the use of white pebbles, resulting in the standard 'black pebbling game'. The black pebbling game and the black-white pebbling game played on DAGs are the most common forms of the game found in the literature, although pebbling on monotone circuits has also been investigated. This leads us to the definitions of the 'black-white pebbling number' and the 'black pebbling number' of a monotone circuit:

Definition 3.2.9 (Pebbling Numbers of Monotone Circuits). *Given a monotone circuit C, the 'black-white pebbling number' of C, denoted $BW\text{-}Peb(C)$ is the minimum number of total pebbles that the player needs in order to complete the black-white pebbling game on C without violating any of its rules.*

The 'black pebbling number' of C, denoted $B\text{-}Peb(C)$ is the minimum number of black pebbles that the player must be given in order to complete the black pebbling game on C without violating any of its rules.

Note that each version of the pebbling game can either be defined with or without sliding; this is an important point because allowing sliding has a slight effect on the pebbling number (see Lemma 3.3.1 for details). Another issue is whether sliding is compulsory or optional, i.e. if one is allowed to place a pebble even when sliding is possible. For the purposes of this thesis, we will allow sliding unless otherwise stated.

A final concept related to pebbling games is a method of documenting the moves made during a pebbling game. We refer to such a record as a 'pebbling history':

Definition 3.2.10 (Pebbling Histories & Strategies). *A pebbling history on a monotone circuit G with j moves is a sequence of pairs $H = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$ in which each (B_i, W_i) pair completely describes which nodes of G have pebbles on them at step i of the pebbling game; B_i is the set of nodes having black pebbles on them, and W_i is the set of nodes having white pebbles on them at time i. By the definition of the game, $B_i \cap W_i = \emptyset$ for all i. A step in the pebbling game is defined as being one move by the* *player consisting of the removal of a pebble, the placement of a pebble, or the sliding of a pebble (if sliding is allowed).*

When dealing with pure black or pure white pebbling histories, we allow $H = S_0, S_1, ..., S_j$ to be a sequence of sets in which each S_i is the set of nodes at time i having pebbles on them. This simplifies our notation, since pairs are not necessary in this case.

A 'pebbling strategy' is a history which has met the termination condition of the game.

3.2.4 Pebbling Contradictions

A number of important families of unsatisfiable formulas called the 'pebbling contradictions' are based on the various different forms of the pebbling game. A brief history of how these formulas have been used in the literature is given in [BS02].

Definition 3.2.11 (One-Colour Pebbling Contradictions for Monotone Circuits). *The one-colour pebbling contradiction of a monotone circuit C, denoted $Peb_1(C)$, is an unsatisfiable formula constructed by taking C, and creating the following clauses in which each variable x is interpreted as meaning that vertex x has a pebble on it:*

1. *For each source node s, create the singleton clause $\{s\}$.*
2. *For each AND node y_0 of degree d with immediate predecessors $y_1, y_2, ..., y_d$, create the propagation clause $\{\neg y_1, \neg y_2, ..., \neg y_d, y_0\}$.*
3. *For each OR node y_0 of degree d with immediate predecessors $y_1, y_2, ..., y_d$, create the propagation clauses $\{\neg y_1, y_0\}, \{\neg y_2, y_0\}, ..., \{\neg y_d, y_0\}$.*
4. *Finally, for the target node t, create the singleton clause $\{\neg t\}$.*

We say that a formula is minimally unsatisfiable if it is unsatisfiable, but the same cannot be said for any proper subset of its clauses. It is easy to see that $Peb_1(G)$ is minimally unsatisfiable for almost any DAG G which we might encounter in practice:

Theorem 3.2.12. *Let G be any arbitrary DAG. $Peb_1(G)$ is minimally unsatisfiable if and only if for every vertex x in G, there exists a path from x to G's target node t.*

Proof: $\Rightarrow$ Suppose that $Peb_1(G)$ is minimally unsatisfiable, and assume that there exists a vertex y in G such that there is no path from y to t. Therefore, neither y nor any of its ancestors are predecessors to t or any of t's predecessors. Delete y and all of its ancestors from G to produce G'. $Peb_1(G')$ remains unsatisfiable, but $Peb_1(G') \subset Peb_1(G)$, contradicting our assumption that $Peb_1(G)$ is minimally unsatisfiable. Therefore for every vertex x in G, there exists a path from x to G's target node.

$\Leftarrow$ Suppose that for every vertex x in G, there exists a path from x to G's target node t. We will show how to satisfy $Peb_1(G)$ if we leave out one arbitrary clause C. There are three cases to consider:

Case 1: C is the clause associated with the target node and is of the form $\{\neg t\}$. In this case we set all variables in $Peb_1(G)$ to $True$, thereby satisfying all clauses except for C.

Case 2: C is the clause associated with a source node s and is of the form $\{s\}$. We know from our assumption that there is a path P from s to t. Set every variable associated with a node on P (including s and t) to $False$, and set all remaining variables to $True$. This truth assignment falsifies C but satisfies all other clauses.

Case 3: C is the clause associated with an AND gate y_0 and is of the form $\{\neg y_1, \neg y_2, ..., \neg y_d, y_0\}$. This case is similar to Case 2: Set every variable associated with a node on the path from y_0 to t (including s and t) to $False$, and set all remaining variables to $True$. This truth assignment falsifies C but satisfies all other clauses.

Therefore, in all cases there exists a truth assignment which satisfies all but one clause, showing that $Peb_1(G)$ is minimally unsatisfiable, as required. □

It is interesting to note that this fact does not hold for circuits; that is, even if a circuit C has a path from every vertex to the target, $Peb_1(C)$ may not be minimally unsatisfiable. For example, consider an OR gate t with two source nodes s_1 and s_2 as predecessors.

Nevertheless, the above theorem does have a useful corollary:

Corollary 3.2.13. *For any DAG G in which there exists a path from every vertex to the target, every possible pebbling of G must use every vertex in G at some point during the pebbling.*

We shall also make use of a more complicated type of contradiction, the two-colour pebbling contradictions:

Definition 3.2.14 (Two-Colour Pebbling Contradictions for Binary Monotone Circuits). *The two-colour pebbling contradiction of a binary monotone circuit C, denoted $Peb_2(C)$, is an unsatisfiable formula constructed by taking C, and creating the following clauses in which each variable x_i is interpreted as meaning that vertex x has a pebble on it:*

1. *For each source node s, create the clause $\{s_0, s_1\}$.*
2. *For each AND node c with immediate predecessors a and b, create four propagation clauses $\{\neg a_0, \neg b_0, c_0, c_1\}$, $\{\neg a_0, \neg b_1, c_0, c_1\}$, $\{\neg a_1, \neg b_0, c_0, c_1\}$, and $\{\neg a_1, \neg b_1, c_0, c_1\}$.*
3. *For each OR node c with immediate predecessors a and b, create four propagation clauses $\{\neg a_0, c_0, c_1\}$, $\{\neg b_0, c_0, c_1\}$, $\{\neg a_1, c_0, c_1\}$, and $\{\neg b_1, c_0, c_1\}$.*
4. *Finally, for the target node t, create the two singleton clauses $\{\neg t_0\}$ and $\{\neg t_1\}$.*

These formulas are important for the investigation of clause space because for any monotone circuit C, $CS(Peb_1(C) \vdash_{\textsf{T-RES}} \emptyset) \leq 2$.

It should be easy to see that for any monotone circuit C, both $Peb_1(C)$ and $Peb_2(C)$ are unsatisfiable; this is because the source node variables must all be true, and all of the propagation clauses send this truth towards the target. The target variable must therefore be true, but the negative singleton clause(s) for the target forces it to be false, ultimately creating a contradiction.

It is also worth noting that the clauses of $Peb_1(C)$ are all Horn clauses, which means that each one contains at most one positive literal.

3.2.5 The Prover/Delayer Game

The Prover/Delayer game, described in [PI00, BSIW04], is a combinatorial game between two players, the 'Prover', and the 'Delayer', and is played on an unsatisfiable CNF formula F. The point of the game is for the Prover to falsify some initial clause of F, thereby falsifying the formula. Since the formula is unsatisfiable, this is inevitable. Roughly speaking, the Delayer's goal is to delay the falsification of the formula for as long as possible.

The game proceeds in rounds. Each round starts with the Prover choosing a variable, and asking the Delayer what the value of that variable is. The Delayer can give one of three answers: 'True', 'False', or 'You Choose'.

If the Delayer says 'You Choose', then the Prover gets to decide the value of that variable. In addition, every time 'You Choose' is said, the Delayer wins one point. This is the only way in which points can be scored.

The game finishes when any clause has been falsified. The real goal of the game is not actually to prove or delay; rather, the Delayer's aim is to win as many points as possible, while the Prover's aim is to make sure that the Delayer wins as few as possible. This leads us to the following definition:

Definition 3.2.15 (Prover/Delayer Number). *Let F be an unsatisfiable CNF formula. The Prover/Delayer number of F, denoted $PD(F)$ is the greatest number of points the Delayer can score on F with the Prover playing optimally.*

3.2.6 Automatizability

First formalized in [BPR97], automatizability is an important concept in proof complexity:

Definition 3.2.16 (Automatizability). *A propositional proof system S is automatizable if there exists a deterministic algorithm A such that for every formula $F \in UNSAT$ (or $TAUT$), A returns an S-proof of F in time polynomial in $|F|$ and $|\pi|$, where $|F|$ is the number of clauses in F, and $|\pi|$ is the size of the smallest S-proof of F. If A is as above except that it returns a proof of F that is not an S-proof, then S is said to be weakly automatizable.*

Intuitively, automatizability tells us if a proof system can be automated to always find proofs which are close to optimal size. If a proof system is not automatizable, then it is impossible to implement a proof search strategy that is guaranteed to find proofs that are in any way close in length to the shortest possible proof. In other words, even if we have a powerful proof system that has very short proofs for certain formulas, if it is not automatizable, then its strengths are inaccessible from the point of view of automated theorem proving.

Note that if a proof system α p-simulates an automatizable proof system β but β does not p-simulate α, then that does not let us conclude anything about the automatizability of α. Only when α and β are p-equivalent does that tell us anything about automatizability; i.e. if α and β are p-equivalent then α is automatizable if and only if β is automatizable.

By contrast, if α p-simulates β and α is automatizable, then β is weakly automatizable by simply using α's algorithm.

Positive Results

The only complete proof systems known to be automatizable are NS and PC [BPR97], but to say that they are automatizable is not quite correct; there is a caveat. For example, PC is automatizable using the Groebner basis algorithm [CEI96], which is an $O(n^{3d})$ algorithm, where d is the largest degree of all equations in the input set. Since d could be as large as n, PC is only automatizable for certain inputs.

Negative Results

As for negative results, under various cryptographic assumptions, a number of proof systems are known to be non-automatizable:

Theorem 3.2.17 ([AR02]). *If $W[P]$ is not tractable, then neither* T-RES *nor* RES *are automatizable.*

Theorem 3.2.18 ([CK01, p.393]). *If $\forall \epsilon > 0$ the Diffie-Hellman cryptographic function cannot be computed by circuits of size 2^{n^ϵ}, then* AC^0-Frege *systems are not automatizable.*

Theorem 3.2.19 ([BPR00]). *If factoring Blum integers is hard, then* Frege *systems are not automatizable.*

Theorem 3.2.20 ([CK01, p.393]). *If RSA is secure, then* E-Frege *systems are not automatizable.*

Each of these is considered to be good evidence of true non-automatizability since these complexity-theoretic and cryptographic assumptions are widely believed to be true. It is worth noting that most of these results are proved by first showing that under their respective cryptographic assumptions, the various proof systems have no feasible interpolation. Since automatizability implies feasible interpolation [CK01, p.298], the non-automatizability results follow. This is important because CP in fact does have feasible interpolation [CK01, p.362], which means that we cannot rule out its automatizability. A non-automatizability result for CP, if possible, would have to be proved using a different strategy.

3.3 Previous Results Related to Resolution Space

Although Resolution space has not been studied nearly as extensively as Resolution size, a number of space results are known. In addition, several peripheral results concerning pebbling and games have been proved. In this section we list some previous results from the literature which are most important to this part of the thesis and extend them where necessary.

3.3.1 Pebbling

An important survey on pebbling by Pippenger [Pip80] is a very good overview of the area. The literature also contains some useful results for manipulating pebbling strategies which we shall generalize upon. The first is a connection between normal pebbling and pebbling with sliding. In [GLT80], Gilbert, Lengauer, and Tarjan prove that any DAG G can be black pebbled using k pebbles with sliding if and only if it can be black-pebbled using $k+1$ pebbles without sliding. We generalize this result to the black-white pebbling of monotone circuits:

Lemma 3.3.1. *Any monotone circuit C can be black-white pebbled using k pebbles with sliding if and only if it can be black-white pebbled using $k+1$ pebbles without sliding.*

Proof: $\Rightarrow$ Suppose that C can be black-white pebbled using k pebbles with sliding. Replace each instance of the sliding rule from gate x to gate y with two moves; first place a pebble of the same colour as the pebble on x onto y, and then remove the pebble from x. This substitution increases the number of pebbles used by 1 momentarily during the intermediate step and works regardless of whether the pebble is white or black, regardless of what type of gate x is, and leaves us with a pebbling strategy that involves no sliding and requires at most $k+1$ pebbles, as required.

$\Leftarrow$ Suppose that C can be black-white pebbled using $k+1$ pebbles without sliding, and let H be the pebbling strategy associated with this pebbling.

Let (B_i, w_i) be any time in H containing $k+1$ pebbles. Therefore, the move transitioning from step (B_{i-1}, w_{i-1}) to (B_i, w_i) was the placement of either a black or white pebble on a vertex x. Either this is the final move in the pebbling or it is not. If this is the final move, then it must be the placement of a black pebble. Then all of x's predecessors must be pebbled at step (B_{i-1}, w_{i-1}), and there are no white pebbles on C, so simply slide one of the black pebbles from one of x's predecessors to x instead of placing a new black pebble on x, thereby saving a pebble.

If this is not the final move in the pebbling, then the pebble being placed on x is either black or white, and the move transitioning from step (B_i, w_i) to step (B_{i+1}, w_{i+1}) must be the removal of a pebble from a vertex y. If the pebble being placed on x is white, then it does not require any predecessors, so simply remove the pebble from y before placing the white pebble on x, thereby saving a pebble.

If the pebble being placed on x is black, then y either is or is not an immediate predecessor of x. If y is not an immediate predecessor, then it is not needed for the placement of x, so simply remove the pebble from y first, and then place the black pebble on x, thereby saving a pebble.

If y is an immediate predecessor of x, then the pebble being removed from y is either black or white. If it is black, then instead of placing a black pebble on x and then removing the black pebble from y, we slide the black pebble from y to x, thereby saving a pebble and still ending up in the same pebbling configuration (B_{i+1}, w_{i+1}).

If the pebble being removed from y is white, then instead of placing a black pebble on x and removing a white pebble from y, we first remove the white pebble from y, then place a black pebble on y, and then slide it from y to x, once again saving a pebble and ending up in the same pebbling configuration (B_{i+1}, w_{i+1}).

We perform this local transformation to save a pebble at every stage of H where $k+1$ pebbles are used. This shows that C can be pebbled using k pebbles with sliding, as required. □

Another useful result for manipulating pebblings is found in [Hei81]. In this paper, Meyer Auf Der Heide proves that for any DAG G, any black-white pebbling strategy (without sliding) using k pebbles can be turned into its 'dual' white-black pebbling strategy using k pebbles by reversing the strategy, converting all black pebbles to white and all white pebbles to black.

We generalize this result to black-white pebbling monotone circuits with the sliding rule present:

Lemma 3.3.2. *Given a black-white pebbling strategy (with sliding) of a monotone circuit C using at most k pebbles, if one reverses the strategy, converts all black pebbles to white, and all white pebbles to black, then the resulting strategy is also a valid black-white pebbling strategy of C which uses at most k pebbles.*

Proof: From the description of the black-white pebbling game in Definition 3.2.8, it is easy to see that every possible move in the game has a corresponding countermove which is exactly equivalent to making the move in reverse with all pebbles having exactly the opposite colour. For example, consider a node v in C with $d \geq 0$ predecessors $u_1, ..., u_d$. The following are pebbling moves together with their duals:

1. Let v be an unpebbled AND gate with all predecessors pebbled, and suppose we place a black pebble on v. In this case the dual move is to start with a white pebble on v, have all other corresponding nodes contain opposite pebbles, and then remove the white pebble from v.
2. Let v be an unpebbled OR gate with at least one predecessor pebbled, and suppose we place a black pebble on v. In this case the dual move is to start with a white pebble on v, have all other corresponding nodes contain opposite pebbles, and then remove the white pebble from v.
3. Let v be an unpebbled AND gate with all predecessors pebbled and suppose that we slide a black pebble from some predecessor u_i to v. In this case the dual move is to start with a white pebble on v but none on u_i, have all other corresponding nodes contain opposite pebbles, and then slide the white pebble from v to u_i.
4. Let v be an unpebbled OR gate with at least one predecessor pebbled and suppose that we slide a black pebble from some predecessor u_i to v. In this case the dual move is to start with a white pebble on v but none on u_i, have all other corresponding nodes contain opposite pebbles, and then slide the white pebble from v to u_i.
5. Suppose that a black pebble is removed from v, where v is either type of gate. In this case the dual move is to have all other corresponding nodes pebbled with opposite pebbles, and then place a white pebble on v.

Of course, all of these dual relationships hold in the opposite direction as well, and each of these cases the number of total pebbles used is identical. It is therefore possible to take any black-white pebbling strategy (with sliding) and convert it step-by-step into a dual strategy which uses exactly the same number of pebbles, as required. □

3.3.2 Space & Games

In [ET01], Esteban and Torán show that the pebbling game is related closely to RES clause space:

Lemma 3.3.3 ([ET01]). *Let F be any arbitrary unsatisfiable formula, let π be a configuration-style* RES *refutation of F, and let G be the DAG underlying the structure of π. The clause space required in order to compute π is exactly equal to B-Peb(G).*

Intuitively, this is because the pebbles represent the clauses which must be kept in memory during the computation of π. This lemma has an immediate and important corollary:

Corollary 3.3.4. *For any unsatisfiable formula F, $CS(F \vdash_{\mathsf{RES}} \emptyset) = $ B-Peb(G), where G is the DAG with the smallest pebbling number of all DAGs underlying valid* RES *refutations of F.*

These results show that both RES clause space as well as T-RES clause space are very closely related to the pebbling game. In fact, this shows how configuration-style refutations differ from the more standard way of viewing refutations as sequences of clauses: A sequence-style refutation implicitly describes a proof DAG, whereas a configuration-style refutation not only has an underlying DAG, but also describes a pebbling on that DAG.

A particularly relevant result relating black-white pebbling number to space was proved by Ben-Sasson:

Theorem 3.3.5 ([BS02]). *For any monotone circuit C, BW-$Peb(C) \leq VS(Peb_1(C) \vdash_{\textsf{RES}} \emptyset)$, where BW-$Peb(C)$ is defined without sliding.*

In fact, the above theorem was not stated explicitly in [BS02], but rather forms the basis of the following theorem, which follows from it immediately since variable space is always bounded above by total space:

Theorem 3.3.6 ([BS02]). *For any monotone circuit C, BW-$Peb(C) \leq TS(Peb_1(C) \vdash_{\textsf{RES}} \emptyset)$, where BW-$Peb(C)$ is defined without sliding.*

T-RES clause space is also related closely to the Prover/Delayer game, described in Section 3.2.5. The results in [ET03] are the definitive work relating the Prover/Delayer number $PD(F)$ of an unsatisfiable CNF formula to its T-RES clause space $CS(F \vdash_{\textsf{T-RES}} \emptyset)$:

Theorem 3.3.7 ([ET03]). *For any unsatisfiable CNF formula F, $CS(F \vdash_{\textsf{T-RES}} \emptyset) = PD(F) + 1$.*

In other words, the Prover/Delayer game perfectly captures the notion of clause space for T-RES.

In addition to doing some of the pioneering research in the area of Resolution space, Esteban and Torán also showed that an upper bound on tree clause space yields an upper bound on the size of T-RES proofs:

Theorem 3.3.8 ([ET01]). *Let F be an unsatisfiable formula on n distinct variables. If F has a* T-RES *refutation with tree clause space $s = CS(F \vdash_{\textsf{T-RES}} \emptyset)$, then it has a* T-RES *refutation of size $\binom{n+s}{s}$.*

The Prover/Delayer game can also be used to give a lower bound on T-RES size; in [PI00], Pudlák and Impagliazzo show that lower bounds on $PD(F)$ can be used to prove lower bounds on the size of T-RES proofs:

Corollary 3.3.9 ([PI00, BSIW04]). *For any CNF formula F, $Size(F \vdash_{\textsf{T-RES}} \emptyset) \geq 2^{PD(F)}$.*

When we combine this with Theorem 3.3.7, we also get the following useful result:

Corollary 3.3.10. *For any CNF formula F, $Size(F \vdash_{\textsf{T-RES}} \emptyset) \geq 2^{CS(F \vdash_{\textsf{T-RES}} \emptyset)-1}$.*

These results motivate why it would be so helpful for researchers working with SAT-solvers to have algorithms which, given a formula F, are capable of efficiently calculating $PD(F)$ or $CS(F \vdash_{\textsf{T-RES}} \emptyset)$: These algorithms could be used as preprocessing steps for SAT-solvers; if $PD(F)$ or $CS(F \vdash_{\textsf{T-RES}} \emptyset)$ are sufficiently large, then there is no sense in even attempting to run a Resolution-based SAT-solver on F, since it will take far too long to terminate.

In [BSIW04], the authors also use the Prover/Delayer game to show that for every pebbling graph G, the size of the smallest T-RES refutation of the formula $Peb_2(G)$ is $2^{B-Peb(G)}$. Combined with the result from [CPT77] that there exist pebbling graphs on n nodes requiring $\Omega(n/log\ n)$ pebbles, this yields a T-RES lower bound of $2^{\Omega(n/log\ n)}$, where n is the number of pebbles. Since T-RES has an $O(\frac{n\ log\ log\ n}{log\ n})$ upper bound for the pebbling formulas, and RES has an $O(n)$ upper bound, this proves a near-optimal separation between the two systems.

3.3.3 Space & Width

In an important paper relating space and width [AD03], Atserias and Dalmau prove that for unsatisfiable k-CNF formulas, space is an upper bound on width:

Theorem 3.3.11 ([AD03]). *If F is an unsatisfiable CNF formula, then*

$$CS(F \vdash_{\mathsf{RES}} \emptyset) \geq w(F \vdash_{\mathsf{RES}} \emptyset) - w(F).$$

This is a very powerful result, because it can be used to derive space lower bounds for all formulas for which width lower bounds are known. Of course, since $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) \geq CS(F \vdash_{\mathsf{RES}} \emptyset)$, another corollary of this result is that RES width is also a lower bound on T-RES clause space.

3.3.4 Width & Size

The relationship between width and size is very important because it shows that lower bounds for width imply lower bounds for size. This simplifies proofs for size lower bounds by allowing us to focus on lower bounds for width. The main result linking width and size was proved by Ben-Sasson and Wigderson [BSW01], and a somewhat more general version is due to Urquhart [Urq06].

Theorem 3.3.12 ([BSW01]). *Let F be a contradictory set of clauses with an underlying set of variables V, and let $Size(F \vdash_{\mathsf{RES}} \emptyset)$ be the minimum size of any* RES *refutation of F. Then*

$$Size(F \vdash_{\mathsf{RES}} \emptyset) = exp\left(\Omega\left(\frac{(w(F \vdash_{\mathsf{RES}} \emptyset) - w(F))^2}{|V|}\right)\right)$$

In [BSW01], Ben-Sasson and Wigderson also prove a more exact relationship between T-RES proof size and width. In fact, a slightly weaker form of the following result can be proved independently by combining Corollary 3.3.10 and Theorem 3.3.11 above.

Theorem 3.3.13 ([BSW01]). *Let F be a contradictory set of clauses and let $Size(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$ be the minimum size of any* T-RES *refutation of F. Then*

$$Size(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) \geq 2^{w(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) - w(F)}$$

Much like the results above which show that T-RES refutation size is exponential in both $PD(F)$ and $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$, these result shows that RES refutation size is exponential in $w(F \vdash_{\mathsf{RES}} \emptyset)$ and T-RES refutation size is exponential in $w(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$, giving a similar motivation for why researchers working with SAT-solvers might want to have preprocessing algorithms which are capable of calculating the width of a formula: Such an algorithm could be used a a preprocessing steps for SAT-solvers; if the width of a formula F is sufficiently large, then there is no sense in even attempting to run a Resolution-based SAT-solver on F, since it will take far too long to terminate.

3.3.5 Tradeoff Results

Another important paper is [BS02], in which Eli Ben-Sasson proves a number of interesting tradeoff results. For example, he provides families of formulas for which there are:

- Linear-sized T-RES proofs and constant width T-RES proofs, but no T-RES proof that has both small width and small size.
- RES proofs with constant clause space, and RES proofs with constant width, but no RES proof that has both small width and small clause space.
- Linear-sized RES proofs that also have constant width, but no RES proof that has both small clause space and small size.

These results rely on the pebbling contradiction formulas from Definition 3.2.11 above.

In a more recent paper [Nor06], Jakob Nordström proves the first non-trivial separation between RES width and clause space. More specifically, he shows the existence of a family of unsatisfiable formulas with constant width but clause space which is bounded by $\Theta(log(n))$, where n is the number of variables in the formulas.

3.3.6 The Complexity of Pebbling

A number of complexity results involving pebbling are known. In [Lin78], Andrzej Lingas uses an ingenious reduction in order to prove the following interesting result about the black pebbling game on monotone circuits:

Theorem 3.3.14 ([Lin78]). *Given a binary monotone circuit C and an integer k, the problem of determining if C can be black-pebbled using at most k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

This result was later extended to DAGs by Gilbert, Lengauer, and Tarjan in [GLT80]:

Theorem 3.3.15 ([GLT80]). *Given a binary DAG G and an integer k, the problem of determining if G can be black-pebbled using at most k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

More recently, the black-white pebbling game on monotone circuits with arbitrary fan-in was also shown to be $\mathcal{PSPACE}$-Complete by Philipp Hertel and Toniann Pitassi [HP07]:

Theorem 3.3.16 ([HP07]). *Given a monotone circuit C and an integer k, the problem of determining if C can be black-white-pebbled using at most k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

They also extend this result to DAGs in [HP07], thereby settling the long-standing open problem of determining its complexity:

Theorem 3.3.17 ([HP07]). *Given a DAG G and an integer k, the problem of determining if G can be black-white-pebbled using at most k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

By Lemma 3.3.1, we get the following Corollary:

Corollary 3.3.18. *The $\mathcal{PSPACE}$-Completeness of all four versions of the pebbling game above holds even when sliding is not allowed.*

All of these results are particularly interesting because they are $\mathcal{PSPACE}$-Completeness results for a game which has only one player, whereas most $\mathcal{PSPACE}$-Complete games have two.

Another important result from [HP07] very closely related to the work in this thesis is that computing RES total space requirements is $\mathcal{PSPACE}$-Complete. More formally, the RES derivation total space problem ($RTSDP$) is defined as follows:

Definition 3.3.19 ($RTSDP$). $RTSDP = \{(F, D, k) \mid$ *F is a formula from which there exists a* RES *derivation with total space at most k of the clause D.*$\}$

Theorem 3.3.20 ([HP07]). *$RTSDP$ is $\mathcal{PSPACE}$-Complete.*

This close relationship between pebbling and Resolution space is similarly emphasized in this thesis: In Chapter 4 we reduce from Lingas's black pebbling result on monotone circuits to show that calculating T-RES clause space is $\mathcal{PSPACE}$-Complete. Next, in Chapter 5 we reduce from Gilbert, Lengauer, and Tarjan's black pebbling result on DAGs to show that calculating the total space of I-RES derivations is $\mathcal{PSPACE}$-Complete. Finally, in Chapter 6 we reduce from Hertel and Pitassi's black-white pebbling result on monotone circuits to show that calculating RES variable space is $\mathcal{PSPACE}$-Hard.

3.4 Summary of Part II

In the next four chapters we will explore Resolution space, which is essentially the amount of Turing Machine memory required to compute a Resolution proof. We shall study various space measures for various forms of Resolution. The following is a brief summary of each of these chapters:

3.4.1 Chapter Summary

In Chapter 4 we prove our first substantial result by settling the complexity of the T-RES clause space problem as well as that of the Prover/Delayer game. Given a formula F and integer k, the T-RES clause space problem asks, 'Does F have a T-RES proof with clause space at most k?'. We reduce from the $\mathcal{PSPACE}$-Complete black pebbling game on monotone circuits due to Lingas [Lin78] to show that the Prover/Delayer game is $\mathcal{PSPACE}$-Complete, which immediately yields the $\mathcal{PSPACE}$-Completeness of the T-RES clause space problem as a corollary, since a previous result by Esteban and Torán [ET03] shows that for any unsatisfiable formula F, the score from the Prover/Delayer game played on F is identical to the T-RES clause space of refuting F. As a corollary, this yields the $\mathcal{PSPACE}$-Completeness of a particular (but unfortunately not very applicable) definition of clause learning clause space.

Since most modern SAT-solvers use variations of DPLL and other Resolution-based algorithms such as clause learning, the results in this chapter have practical implications for the area of automated theorem proving, because we show that predicting the memory requirements of DPLL SAT-solvers on a given formula F is at least as hard as simply solving it. This is important because T-RES proof size is exponential in T-RES clause space, so researchers developing SAT-solvers will probably not be able to take direct advantage of the lower bound relationships between space and size for T-RES proofs in order to design preprocessing algorithms for SAT-solvers which can predict if a formula is even worth attempting to solve. The possibility of approximation algorithms for T-RES clause space still exists, but since size is exponential in clause space, anything but an algorithm which can calculate the correct answer to within a small constant would be useless in practice.

Another practical motivation behind this research has to do with combined resources for SAT-solvers: The limiting factor on most modern SAT-solving algorithm tends to be memory space, but on the other hand, if one is too frugal with memory, then proofs can become intractably large (for example, see Section 5.7.2 or [HP07]). Algorithms must therefore carefully balance attempts to save memory with a potentially massive increase in minimum proof size and running time if they save too much. We hope that future researchers will be able to build on these present results in order to better understand the tension between size and space.

The final section of this chapter contains a number of corollaries to the main result. These include $\mathcal{PSPACE}$ algorithms for several resource problems, and, as already mentioned, the $\mathcal{PSPACE}$-Completeness of clause learning clause space, which hopefully once again is an important first step since clause learning algorithms are such important SAT-solvers.

Our next substantial result concerns the complexity of I-RES, and is presented in Chapter 5. I-RES is a very restrictive Resolution refinement which requires that at least one premise for each resolution step be an input clause. In a sense, this chapter is paradoxical because we show that some aspects of I-RES are computationally very simple, and others are extremely difficult. We start by exploring a number of its tractable aspects. For example, given a minimally unsatisfiable formula F and integer k, determining if F has an I-RES refutation of size at most k is in $\mathcal{P}$. We also give the ultimate testament to the tractability of I-RES by showing that it is automatizable, and optimally automatizable for minimally unsatisfiable formulas.

However, even though I-RES is considered to be trivial and so many of its aspects are computationally easy, some of its resource problems are fiendishly complex. For example, we prove that the I-RES size problem is $\mathcal{NP}$-Complete, and our main result shows that for any binary DAG G, the black pebbling number of G is equal within a constant to the I-RES total space of refuting a slight modification of the formula $Peb_1(G)$. This has two immediate corollaries. The first is the $\mathcal{PSPACE}$-Completeness of various forms of the I-RES derivation total space problem which, given a formula F, integer k and clauses C and D, asks if F has an I-RES derivation of D with top clause C and total space at most k. This result falls in line with our main theme of applying games, since the reduction is from the $\mathcal{PSPACE}$-Complete black pebbling game on DAGs due to Gilbert, Lengauer, and Tarjan [GLT80].

The second corollary is an extreme (and optimal) size / total space tradeoff for I-RES. More specifically, we show that there exists an infinite family of formulas whose I-RES proofs with minimum required total space have size $2^{\Omega(n)}$, where n is the number of variables. However, if just one single additional unit of total space is permitted, then the size drops to only $O(n)$. Apart from being a massive size / space tradeoff, this is also a tradeoff similar to those explored by Ben-Sasson in [BS02] because it shows that for certain formulas, it is not possible to optimize both I-RES size and space at the same time.

All of these results together illustrate the subtle complexities of this interesting Resolution refinement.

Chapter 6 contains results regarding yet another type of RES space measure called variable space. Although variable space is not as natural a measure as clause space or total space, in this chapter we prove results for the full-strength RES proof system rather than the T-RES or I-RES refinements which we saw in the previous two chapters. More specifically, we show that for any monotone circuit C, the black-white pebbling number of C is exactly equal to the RES variable space of refuting the formula $Peb_1(C)$.

This equivalence allows us to prove the $\mathcal{PSPACE}$-Hardness of the RES variable space problem, which, when given a formula F and integer k asks whether F has a RES refutation with variable space at most k. The reduction is from the black-white pebbling game on monotone circuits, recently proven to be $\mathcal{PSPACE}$-Complete by Philipp Hertel and Toniann Pitassi [HP07].

Another interesting aspect of this equivalence comes to light when it is juxtaposed with the equivalence from the previous chapter; for any binary DAG G, its black pebbling number is almost exactly equal to the I-RES total space of $Peb_1(G)$, and for any monotone circuit C, its black-white pebbling number is equal to the RES variable space of $Peb_1(G)$. This shows how intimately pebbling games and Resolution space measures are related.

In Chapter 7 we thought that we had proved one of our most important results by settling the complexity of the RES width problem, but unfortunately discovered a subtle (yet fatal) flaw in one of the main proofs. Without enough time to repair the proof, and because of the relationship of width to the other results in this thesis, we decided to retain this chapter in vestigial form rather than deleting it altogether. However, as it stands, this chapter does not contain any major results.

To briefly describe and motivate the RES width problem, it gives as input a formula F and integer k, and asks whether F has a RES refutation of width at most k. Because of the important relationship between RES width and size showed by Ben-Sasson and Wigderson [BSW01], proving RES size lower bounds can be reduced to proving width lower bounds, which motivates the RES width problem because a polytime width solver would be of immense practical interest in the areas of automated theorem proving and propositional reasoning. Researchers developing SAT-solvers often find that there are inputs on which their solvers do not halt in any reasonable amount of time. A polytime width solver could be used as a preprocessing step to predict ahead of time which formulas require large width, and therefore large size, allowing SAT-solving algorithms to avoid high-width formulas rather than trying in vain to solve them. This motivation of course is very similar to that behind the research in Chapter 4.

The original purpose of this chapter was to confirm a conjecture made by Moshe Vardi that the RES width problem is $\mathcal{EXPTIME}$-Complete, thereby showing that RES width is provably intractable and that it is not possible to build a polytime RES width solver.

There is a particularly strong parallel between the results in Chapters 4, 5, and 6, especially with respect to proof technique. In each of these cases we reduce from a $\mathcal{PSPACE}$-Complete pebbling game problem to a Resolution resource problem using some version of the pebbling contradiction formulas in order to show that the resource problem is $\mathcal{PSPACE}$-Hard. The following table summarizes this parallel:

Chapter	PSPACE-Hardness Result	Reduction From	Formula Used
4	T-RES Clause Space	Black Pebbling Lingas Circuits [Lin78]	$Peb_2(C)$
5	I-RES Derivation Total Space	Black Pebbling DAGs [GLT80]	$Peb_1(G)$
6	RES Variable Space	Black-White Pebbling Monotone Circuits [HP07]	$Peb_1(C)$

In the case of T-RES clause space, we use the $Peb_2(C)$ formulas rather than $Peb_1(C)$ because $Peb_1(C)$ can be refuted in constant space for any circuit C, and therefore cannot possibly be used as a reduction to at $\mathcal{PSPACE}$-Complete set.

3.4.2 Summary of Results

The following table summarizes many of the results from Part II by listing the complexities of several proof complexity resource problems for RES and two of its refinements, T-RES and I-RES. In some cases we are able to prove completeness results; for example, we show that T-RES clause space is $\mathcal{PSPACE}$-Complete and that I-RES size is $\mathcal{NP}$-Complete. In other cases, we give upper bounds for where the problems lie by providing algorithms, and lower bounds by proving hardness results; for example, the T-RES size problem is in $\mathcal{PSPACE}$, and is at least co$\mathcal{NP}$-Hard. By 'no parameter', we refer to the basic version of the appropriate resource problem without a k parameter. The main results by chapter are Corollaries 4.7.7, 5.6.14, and 6.3.2.

Resource	RES	T-RES	RT-RES	I-RES
Clause Space	$\in \mathcal{PSPACE}$ co$\mathcal{NP}$-Hard Cor. 4.8.4	$\mathcal{PSPACE}$-Complete Cor. 4.7.7	$\mathcal{PSPACE}$-Complete Cor. 4.8.14	$\mathcal{P}$-Complete Cor. 5.3.6
Total Space	$\mathcal{PSPACE}$-Complete [HP07], Thm. 3.3.20	$\in \mathcal{PSPACE}$ co$\mathcal{NP}$-Hard Cor. 4.8.5	$\in \mathcal{PSPACE}$ co$\mathcal{NP}$-Hard Cor. 4.8.6	$\mathcal{PSPACE}$-Complete Cor. 5.6.14
Variable Space	$\in \mathcal{EXPSPACE}$ $\mathcal{PSPACE}$-Hard Cor. 6.3.2	$\in \mathcal{NEXPTIME}$ co$\mathcal{NP}$-Hard Cor. 6.4.5	$\in \mathcal{NEXPTIME}$ co$\mathcal{NP}$-Hard Thm. 6.4.2	$\in \mathcal{PSPACE}$ $\mathcal{P}$-Hard Cor. 6.4.7
Width	$\in \mathcal{EXPTIME}$ co$\mathcal{NP}$-Hard Cor. 7.4.2	$\in \mathcal{EXPTIME}$ co$\mathcal{NP}$-Hard Cor. 7.4.5	$\in \mathcal{PSPACE}$ co$\mathcal{NP}$-Hard Cor.7.4.10	$\mathcal{PSPACE}$-Complete Cor. 5.7.1
Size	$\in \mathcal{NEXPTIME}$ co$\mathcal{NP}$-Hard Lem. 4.9.2	$\in \mathcal{PSPACE}$ co$\mathcal{NP}$-Hard Cor. 4.8.10	$\in \mathcal{PSPACE}$ co$\mathcal{NP}$-Hard Cor. 4.8.8	$\mathcal{NP}$-Complete Cor. 5.5.4
No k Parameter	co$\mathcal{NP}$-Complete [Coo71]	co$\mathcal{NP}$-Complete [Coo71]	co$\mathcal{NP}$-Complete [Coo71]	$\mathcal{P}$-Complete Cor. 5.3.4

Chapter 4

The PSPACE-Completeness of Tree Resolution Clause Space

4.1 Introduction & Motivation

The Tree Resolution (T-RES) proof system has been studied extensively, and is understood quite well. Its algorithmic incarnation, DPLL, forms the basis of many SAT-solvers including clause learning algorithms. Any lower bounds proved for T-RES immediately imply lower bounds for real-world DPLL algorithms. However, compared with Resolution proof size, the amount of space needed to compute a proof has not been studied nearly as well.

From a practical point of view, DPLL and clause learning algorithms have been very successful at solving SAT and SAT-related problems. The main limiting factor on these algorithms is space, namely the size of the cache used for memoization. This has inspired much research into methods for pruning space in a Resolution search. Thus there are both practical and theoretical motivations for understanding Resolution space as a resource.

In this chapter we prove that the problems of calculating T-RES clause space requirements and calculating the number of points that can be scored in the Prover/Delayer game are both $\mathcal{PSPACE}$-complete. This unfortunately implies that computing the T-RES clause space requirements for a formula is at least as hard as actually refuting it, and shows that it is therefore probably not feasible for researchers working with DPLL-based SAT-solvers to predict a formula's memory requirements. In addition, there are many formulas on which these SAT-solvers fail because the smallest refutation size is exponential. It would be very useful to be able to tell ahead of time if this will be the case. From Corollary 3.3.10, we know that for any CNF formula, the size of its minimum T-RES refutation is exponential in its minimum T-RES clause space, and in fact we understand this relationship exactly rather than just asymptotically. In other words, if we had a practical algorithm for calculating T-RES clause space, then we would have a way to predict size (and therefore running time) lower bounds for DPLL algorithms. If this number is too big, then we would know ahead of time that there is no sense in even trying to run a SAT-solving algorithm on the input. Unfortunately, the present result casts serious doubts on this approach. The possibility of approximation algorithms for T-RES clause space still exists, but since size is exponential in clause space, anything but an algorithm which can calculate the correct answer to within a small constant would not be very useful in practice.

In practice, researchers are more interested in clause learning space than T-RES space, but T-RES nevertheless forms the underpinnings of the clause learning algorithm, and in fact we also show that predicting clause learning clause space requirements is $\mathcal{PSPACE}$-Complete, albeit for a definition of clause learning clause space which is not very practical. We therefore view this work as a first step which future researchers will hopefully build upon to better understand these important problems.

The key to this chapter's main theorem is the intimate connection between Resolution space and games, and we combine three previous results and ideas in order to prove both of our $\mathcal{PSPACE}$-Completeness results. The high-level idea behind this chapter is as follows: From Theorem 3.3.7 ([ET03]), Prover/Delayer number is known to equal T-RES clause space, and from Theorem 3.3.14 ([Lin78]), pebbling Lingas circuits is known to be $\mathcal{PSPACE}$-Complete. But the former deals with formulas, and the latter uses circuits, so we use pebbling contradiction formulas to bridge this gap. Specifically, we give tight bounds relating the black pebbling number of any Lingas circuit C to the Prover/Delayer number of the formula $Peb_2(C)$, thereby proving the $\mathcal{PSPACE}$-Completeness of the Prover/Delayer game (which implies the $\mathcal{PSPACE}$-Completeness of T-RES clause space by the equivalence from Theorem 3.3.7).

This chapter is organized as follows: In Section 4.2 we define a set of binary DAGs called the 'increasing binary DAGs' whose pebbling numbers can be computed in polynomial time. In Section 4.3 we follow up on this result by giving a Prover strategy for these pebbling graphs. Specifically, we show that for any increasing binary DAG G, when playing on the formula $Peb_2(G)$, the Prover has a strategy which limits the Delayer to at most $B\text{-}Peb(G)$ points.

Next, in Section 4.4 we define a set of monotone circuits called the 'Lingas circuits' for which determining the pebbling number is $\mathcal{PSPACE}$-Complete, and show that much like the increasing binary DAGs, for any Lingas circuit C, when playing on the formula $Peb_2(C)$, the Prover has a strategy limiting the Delayer to at most $B\text{-}Peb(C)$ points. Section 4.5 contains a proof showing that for any monotone circuit C, the Delayer has a strategy which is guaranteed to score at least $B\text{-}Peb(C)$ points when playing on the formula $Peb_2(C)$. We are forced to use the $Peb_2(C)$ rather than the $Peb_1(C)$ formulas because for any monotone circuit C, $CS(Peb_1(C) \vdash_{\textsf{T-RES}} \emptyset) \leq 2$.

In Section 4.7 we use these results to prove our main results, namely the $\mathcal{PSPACE}$-Completeness of the T-RES clause space problem and Prover/Delayer game.

Next, in Section 4.8 we give $\mathcal{PSPACE}$ algorithms for several problems related to T-RES clause space, including the RES clause space, T-RES total space, and T-RES size problems. In addition, we show that the clause space problems for RT-RES and clause learning are both $\mathcal{PSPACE}$-Complete.

Finally, in Section 4.9 we discuss some interesting open problems related to this research.

4.2 An Easy Case of the Pebbling Game

Although pebbling circuits is $\mathcal{PSPACE}$-Complete in general (see Section 3.3.6), in this section we define an interesting set of binary DAGs whose pebbling numbers can be computed in polynomial time. To this end, we first need to define the concept of the pebbling number of an internal vertex in a graph.

Definition 4.2.1 (Pebbling Number of a Vertex). *In a DAG, the pebbling number of a vertex x, denoted $B\text{-}Peb(x)$, is the pebbling number of the subgraph rooted at x, with x set to be the target node.*

Given a binary DAG G, each node can be labelled with its black pebbling number according to the definition above. Clearly, each source node has a pebbling number of 1. It is impossible for a vertex to have a pebbling number which is greater than that of both of its predecessors by 2 or more, and it is also impossible for the pebbling number of a vertex to be less than that of its predecessors.

Consider a vertex c with predecessors a and b where the pebbling number of a is less than or equal to that of b. There are only three possibilities for their pebbling numbers:

1. If vertex a has a pebbling number of at most k - 1 and vertex b has pebbling number k, then vertex c also has pebbling number k. To see this, simply pebble b with k pebbles, leave one pebble on b and remove the rest (if any). Then pebble a with the remaining k - 1 pebbles. Since a has a pebbling number strictly less than k, even when the extra pebble on b is taken into account, this cannot take more than k pebbles. Finally, slide one of the pebbles from a or b up to c.

2. If vertices a and b both have pebbling number k, then it is possible for c to have a pebbling number of $k + 1$.

3. If vertices a and b both have pebbling number k, then it is possible for c to also have pebbling number k. (For example, see Figure 4.1 below.)

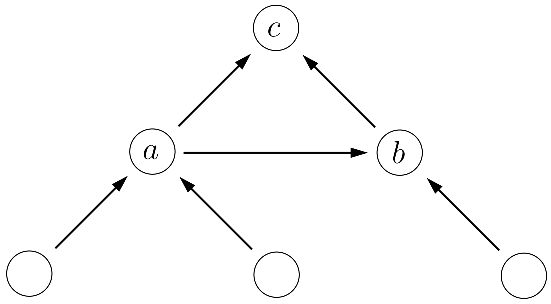

Figure 4.1: An Example of a DAG in Which c and Both of its Predecessors Have the Same Pebbling Number

The three possibilities above give rise to an interesting class of binary DAGs called the 'Increasing Binary DAGs'. This set consists of all binary DAGs in which the pebbling number of each node is strictly greater than that of at least one of its predecessors. The pyramid graphs from [CS76] shown in Figure 4.4 are good examples of these graphs.

More formally, the increasing binary DAGs are defined as follows:

Definition 4.2.2 (Increasing Binary DAGs). $\mathcal{G}_\mathcal{I} = \{(G,k) \mid$ *G is a binary DAG with pebbling number k such that there is no* $c \in V(G)$ *with predecessors a and b with a, b, and c all having the same pebbling number.*$\}$

Although in general the pebbling game on binary DAGs is $\mathcal{PSPACE}$-Complete (see Theorem 3.3.15 above), when we restrict ourselves to inputs from $\mathcal{G}_\mathcal{I}$, the problem becomes much easier:

Lemma 4.2.3. *Given an increasing binary DAG G, the problem of determining if G can be pebbled using at most k pebbles (with sliding) is in* $\mathcal{P}$.

Proof: Start at the source nodes and set each of their pebbling numbers to 1. Then find a vertex c for which the pebbling numbers of both predecessors a and b have been determined (since G is a DAG, such a node always exists), and set c's pebbling number to $max(B\text{-}Peb(a), B\text{-}Peb(b))+1$. Repeat until the pebbling number of the target node has been determined. If it is at most k, then accept, and otherwise reject. The correctness of this algorithm follows from the fact that the pebbling number of each vertex is uniquely determined, and it has a running time of $O(n^2)$, where n is the number of nodes in G. $\square$

This lemma gives some insight into why the pebbling game is so difficult. A close inspection of the constructions from both Theorem 3.3.14 ([Lin78]) and Theorem 3.3.15 ([GLT80]) shows that the circuits resulting from the reductions contain a large number of vertices which have the same pebbling numbers as both predecessors, so the pebbling number of each vertex is not uniquely determined by that of its children, and the obvious algorithm above fails.

4.3 Prover Strategy for the $\mathcal{G}_\mathcal{I}$ DAGs

In this section we show that for any increasing binary DAG G, when playing the Prover/Delayer game on $Peb_2(G)$, the Prover has a strategy limiting the Delayer to scoring at most $k = B\text{-}Peb(G)$ points (with sliding).

For the purposes of this proof, we will use the DAG G as a 'scaffold' to keep track of the Prover's progress. As such, we imagine G with the target node t at the top and the sources at the bottom. The Prover's strategy is to label each node in G with its pebbling number, and work down from the target towards the sources. The Prover must only yield a point when moving to a child with a lower pebbling number, and will therefore be able to limit the Delayer to at most $B\text{-}Peb(G)$ points before $Peb_2(G)$ is falsified, since that is the pebbling number of t.

A contradiction can be obtained (thereby ending the game) in one of four ways:

1. By falsifying a target clause; for target node t, either t_0 is set to True or t_1 is set to True.

2. By falsifying a propagation clause associated with an AND gate; for some AND node c with predecessors a and b, both of c's variables are set to False, and one of a's variables as well as one of b's variables is set to True.

3. By falsifying a propagation clause associated with an OR gate; for some OR node c with predecessors a and b, both of c's variables are set to False, and one of a's variables or one of b's variables is set to True.

4. By falsifying a source clause; for some source node s, both of s's variables are set to False.

Lemma 4.3.1. *For any increasing binary DAG G, when playing on the formula $Peb_2(G)$, the Prover has a strategy limiting the Delayer to at most $B\text{-}Peb(G)$ (with sliding) points.*

Proof: After the nodes have been labelled with their pebbling numbers, the Prover's strategy proceeds as follows: Start at the target t and query t_0. Since $(\neg t_0)$ is an initial clause, the Delayer must reply 'False', or else the game would immediately be over with at most one point scored. The Prover then queries t_1, which the Delayer must set to False for the same reason. Now the Prover tries to 'move down' the graph by having both of the variables associated with one of the predecessors of t set to False. In general, suppose that we are currently on node c which has pebbling number k and predecessors a and b, and both c_0 and c_1 have been set to False. These nodes are shown below in Figure 4.2. There are three cases to consider:

1. a has pebbling number $< k$, and b has pebbling number k

2. a and b both have pebbling number $k - 1$

3. (Base Case) c has pebbling number 2 and both a and b have pebbling number 1

Although case 3 is just a special case of 2, it is the base case and therefore warrants some extra discussion and should be considered separately.

In all cases, the Prover follows the same strategy, which is shown as a decision tree below in Figure 4.2. Nodes in the decision tree represent the variables queried by the Prover, and edges are labelled with the Delayer's possible responses. Whenever the Delayer says 'You Choose', the Prover always responds 'True'. This possibility is represented by the label 'YC, T'. Similarly, edges labelled 'F' correspond with the Delayer's responding, 'False'. Note that there is no need to explore the third option where the Delayer says 'True', because it could only be worse for the Delayer. Please also note that duplicate subtrees are shown by having two parents.

In all cases where the game terminates, the Delayer wins at most two points, which means that even if our game is currently rooted in the base case, the Delayer has not scored more than $B\text{-}Peb(c)$ points. Furthermore, whenever the game moves into the subgraph rooted at a by setting both a_0 and a_1 to false, the pebbling number has decreased, but the Prover has given up only one point. Similarly, whenever the game moves into the subgraph rooted at b by setting both b_0 and b_1 to false, the pebbling number may or may not have decreased, but no points have been scored.

It is therefore possible to take any increasing binary DAG G, and combine multiple copies of the Prover's local strategy for each node from Figure 4.2 in order to give an overall Prover strategy for G which gives up at most one point whenever the game moves into a subgraph with a lower pebbling number. Since the Delayer can also only score at most two points in the base case, the Prover can limit the total number of points scored to $B\text{-}Peb(G)$, as required. □

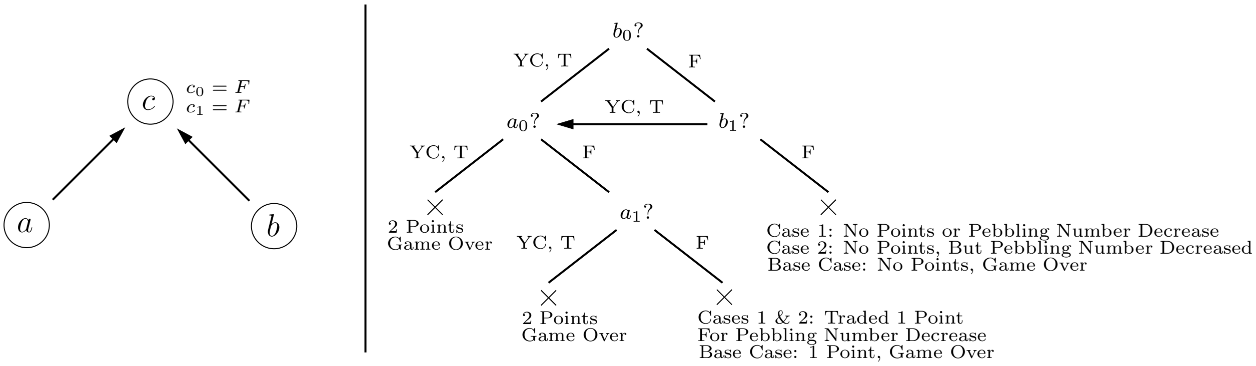

Figure 4.2: A Decision Tree Showing the Prover's Strategy for Traversing the Nodes of an Increasing Binary DAG

4.4 Prover Strategy for the Lingas Circuits

In his 1978 paper [Lin78], Lingas shows that the black pebbling game on monotone circuits is $\mathcal{PSPACE}$-Complete by reducing from the 3-QBF problem with alternating quantifiers. He does this by providing a reduction which takes as input a 3-QBF formula F with alternating quantifiers and outputs a binary circuit C and integer k such that F is True if and only if C has a pebbling number of at most k.

The reduction proceeds by taking F and building a number of 'widgets'. The example from [Lin78] can be seen below in Figure 4.3. The construction includes one widget for each quantifier, one for each literal, one for each clause, and one pyramid graph which acts as a conjunction between the clauses. Each clause widget is incident on the widgets corresponding to the literals contained in the clause. The literal widgets are shown using shorthand notation; they are the pyramid graphs from [CS76], and a more detailed example can be found below in Figure 4.4. The numbers on each shorthand version indicates the pebbling number of the apex of the pyramid.

The pebbling number output by the reduction is $k = 2U + E + M$, where U is the number of universal quantifiers, E is the number of existential quantifiers, and M is the number of clauses in F. In the case of our example, $k = 2 \times 2 + 3 + 4 = 11$.

Intuitively, the reduction works by forcing the pebbling to go through each truth assignment which sets F to True. Specifically, for each of the universal quantifiers, the pebbling must travel up both of its sides, requiring that the entire circuit below that point be re-pebbled, thereby ensuring that F is True. If F is false, then the circuit requires $k + 1$ pebbles and cannot be pebbled using only k.

We shall refer to the circuits corresponding to true QBF formulas as the 'Lingas circuits'. Although the main result in [Lin78] is that pebbling monotone circuits in general is $\mathcal{PSPACE}$-Complete, the reduction specifically shows that the problem of pebbling the Lingas circuits is $\mathcal{PSPACE}$-Complete. Formally, we define the Lingas circuits as follows:

Definition 4.4.1 (Lingas Circuits). $\mathcal{C}_\mathcal{L} = \{(C, k) \mid \exists$ *a true* 3-QBF *formula* F *with alternating quantifiers such that applying Lingas's reduction to* F *yields* (C, k).$\}$

We shall make use of the $\mathcal{PSPACE}$-Completeness of the Lingas circuits to prove the $\mathcal{PSPACE}$-Completeness of both the Prover/Delayer game as well as the T-RES clause space problem. To this end we must first prove that when playing on the $Peb_2(C)$ formula of any Lingas circuit C, the Prover has a strategy limiting the Delayer to at most B-$Peb(C)$ points (for more details on the Peb_2 formulas, please refer to Definition 3.2.14). The strategy is quite simple: The Prover starts at the target node, and forces the Delayer to admit that both variables associated with it must be False. The Prover then proceeds to propagate this 'Double False' setting downwards. We shall show that it is possible to traverse each existential widget while only giving up at most one point, and that it is possible to traverse each universal widget while only giving up at most

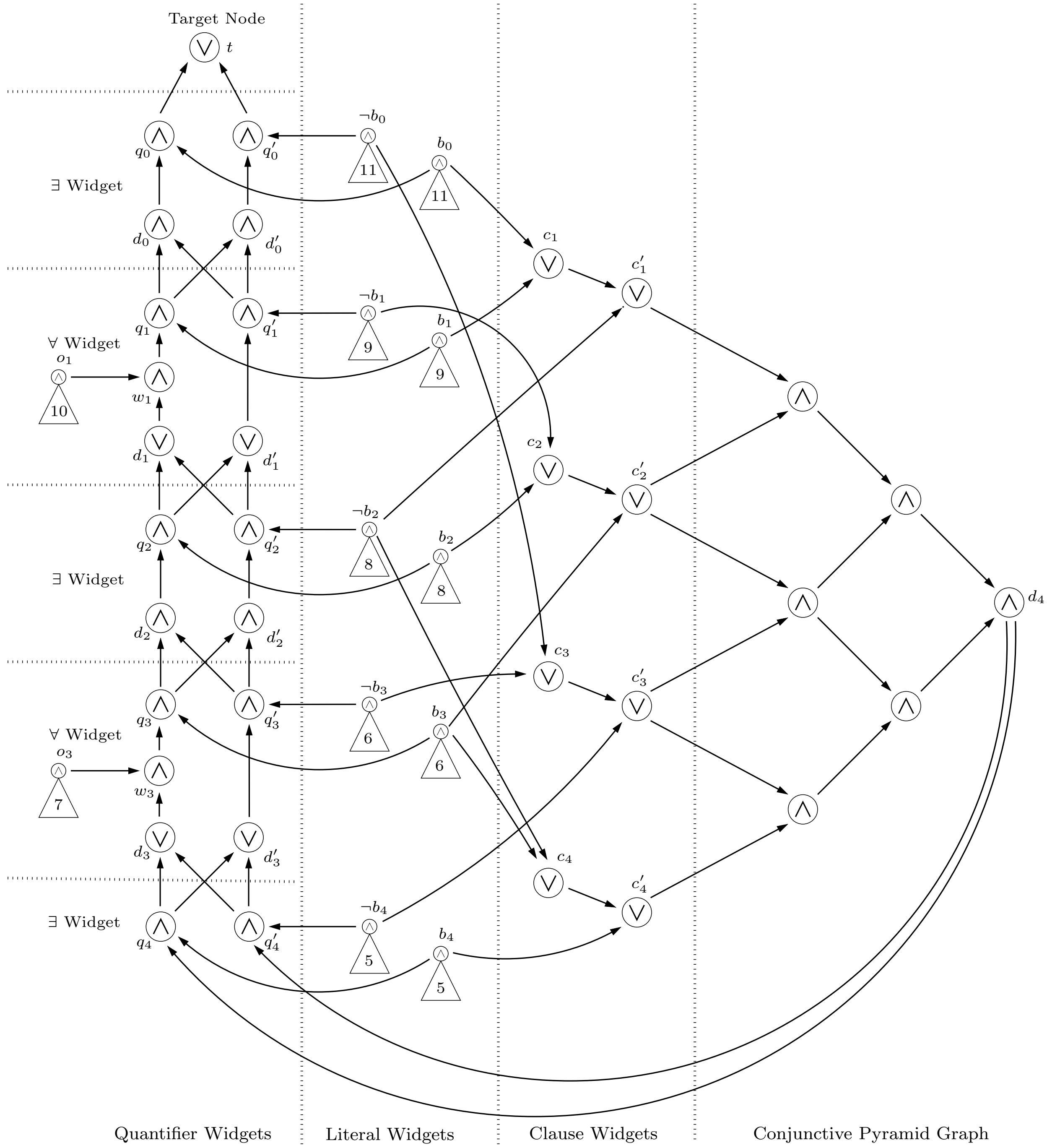

Figure 4.3: The Monotone Circuit Resulting from Applying Lingas's Reduction to the True 3-QBF Formula $F = \exists x_0 \forall x_1 \exists x_2 \forall x_3 \exists x_4\ (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_0 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$. In This Example $k = 2U + E + M = 2 \times 2 + 3 + 4 = 11$.

two points. Finally, we shall show that it is possible to traverse the conjunctive pyramid to derive the final contradictions while giving up at most M points, where M is the number of clauses in the formula underlying the circuit. This amounts to a total of $2U + E + M$ points, which is exactly the pebbling number of the

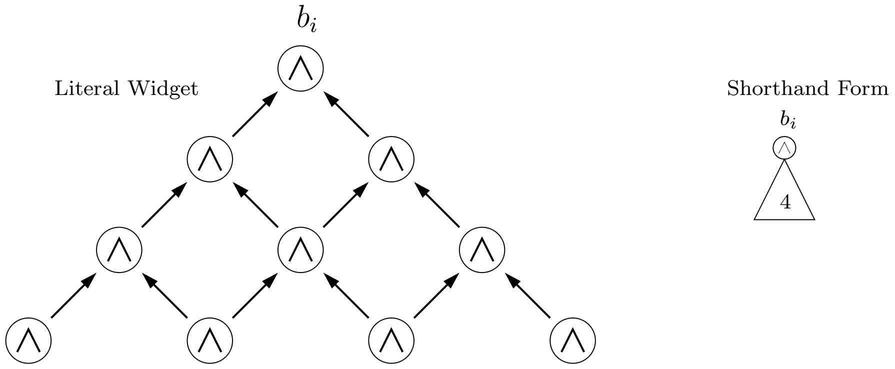

Figure 4.4: An Example of a Literal Widget from Lingas's Construction

Lingas circuits.

Each of these steps shall be proven by giving a decision tree showing the order in which the Prover queries variables. Each branch in this tree shall terminate with the Delayer scoring no more than $B\text{-}Peb(C)$ points.

As in Section 4.3, whenever the Delayer responds 'You Choose' to a query, the Prover shall always respond with 'True', and never with 'False'. Once again, although the Delayer has the choice of three different responses after each variable is queried, our decision tree only needs to be binary. This is because if the Prover is willing to give the Delayer one point by responding to the Delayer's 'You Choose' answer, then there is no sense in exploring the path in which the Delayer gives the same answer without winning a point, since that path can only be better for the Prover.

Theorem 4.4.2. *For any binary circuit and integer pair $(C, k) \in \mathcal{C_L}$, when playing on the formula $Peb_2(C)$, the Prover has a strategy limiting the Delayer to at most $k = B\text{-}Peb(C)$ (with sliding) points.*

Proof: The Prover's strategy proceeds as follows: The first query is on t_0, one of the variables associated with the target node t. If the Delayer says 'You Choose', then the Prover sets it to True, and the game is over with just one point scored. If the Delayer says 'True', then the game is over with no points scored, so the Delayer must set it to False. Next the Prover queries t_1, which is set to False for the same reasons. The Delayer has therefore scored no points, so we enter the first quantifier widget with k points remaining.

Now the Prover inductively traverses the quantifier widgets in order, propagating the 'Double False' setting downwards towards the conjunctive pyramid. We will show that the Prover has a strategy which gives up at most one point while traversing an existential widget, and at most two points when traversing a universal widget.

We shall first give the Prover's strategy for traversing the existential widget, shown below in Figure 4.5. We assume that we have j points remaining before the Delayer reaches k points, and that both variables associated with the node d_{i-1} have been set to False or that both variables associated with the node d'_{i-1} have been set to False. The decision tree showing the Prover's strategy is shown below, beside the widget.

Note that edges are labelled with the different possibilities at each step during the Prover/Delayer game; 'YC, T' represents the case when the Delayer says 'You Choose', followed by the Prover setting the variable to True, and 'F' represents the case when the Delayer sets the variable to False. Duplicate subtrees are indicated by having multiple parents. In addition, leaves are labelled in the Prover/Delayer game outcomes which they lead to. Leaves labelled with numbers represent situations in which the game is over since an initial clause has been falsified, and the number indicates how many points were scored. Although we said that the Prover may give up at most 1 point while traversing this widget, some leaves end with the Delayer scoring 2 points. This is not a problem because even if this is the final quantifier widget, the formula has at least one clause, so we have at least one extra point's leeway.

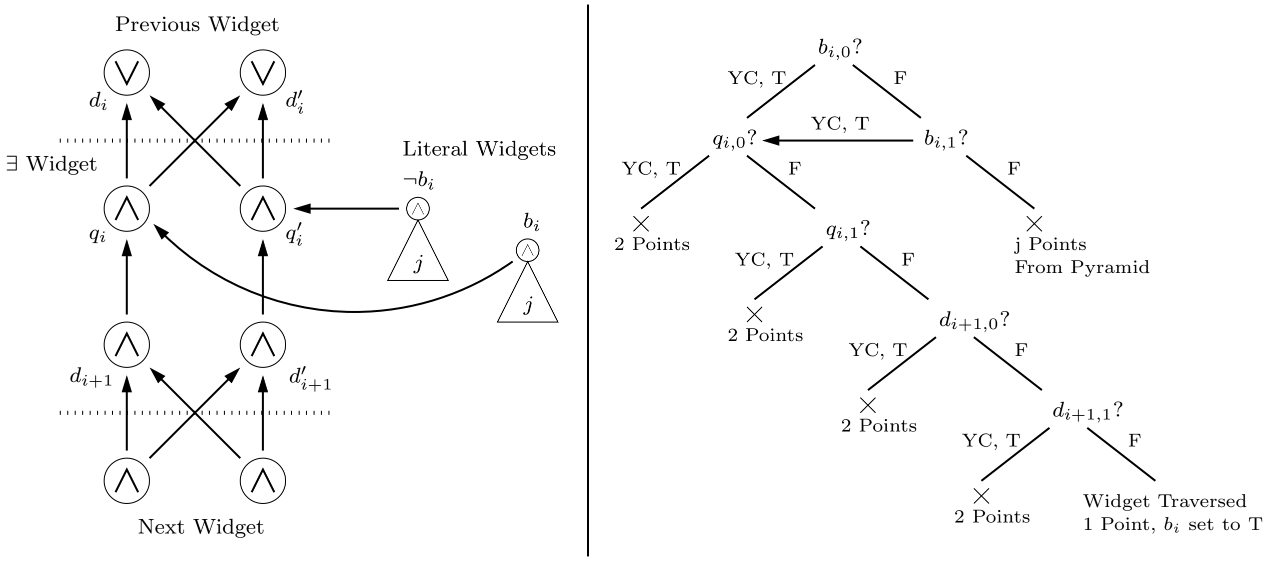

Figure 4.5: The Existential Widget From Lingas's Construction Together with the Decision Tree Showing the Prover's Strategy for Traversing it

The leaf corresponding to both of b_i's variables being set to False corresponds to the game entering the pyramid graph associated with variable b_i which has pebbling number j. Since pyramid graphs belong to the $\mathcal{G}_\mathcal{I}$ family, by Lemma 4.3.1 the Delayer can score at most j points on them.

The remaining leaf is the one in which the widget has been traversed with only 1 point scored. Note that this decision tree corresponds to the strategy in which the Prover traversed the widget while setting one of the variables associated with b_i to True and both of the variables associated with d_{i+1} to False, thereby leading to the next widget. The case in which the Prover traverses the widget while setting one of the variables associated with $\neg b_i$ to True and both of the variables associated with d'_{i+1} to False is completely symmetrical. In other words, the Prover has complete control over which of the literal widgets is set to True during the traversal. This will be important because it allows the Prover to set the literal widgets in such a way so as to make the formula underlying the circuit true.

We now give the Prover's strategy for traversing the universal widget, shown below in Figure 4.6. We assume that we have j points remaining before the Delayer reaches k points, and that either both variables associated with the node d_i have been set to False or that both variables associated with the node d'_i have been set to False. The decision tree showing the Prover's strategy for the universal widget is shown below in Figure 4.7.

Just as with the existential widget, leaves in the decision tree are labelled with the number of points scored in falsifying an initial clause corresponding to that path. Once again, some paths lead to the Delayer scoring 3 points, but this is not a problem even if this is the final quantifier widget, since the formula underlying this circuit has at least one clause, thereby giving us leeway of at least one point. Also as before, the pyramid graphs corresponding to the literal widgets as well as the o_i widget belong to the $\mathcal{G}_\mathcal{I}$ family, so by Lemma 4.3.1, the Delayer can score at most $j-1$, $j-1$, and j points on them, respectively.

The remaining two leaves are ones in which the widget has been traversed with only 2 points scored. In one case, one of the variables associated with b_i is set to True and both variables associated with d_{i+1} are set to False, and in the other case one of the variables associated with $\neg b_i$ is set to True, and both of the variables associated with d'_{i+1} are set to False, thereby leading to the next widget.

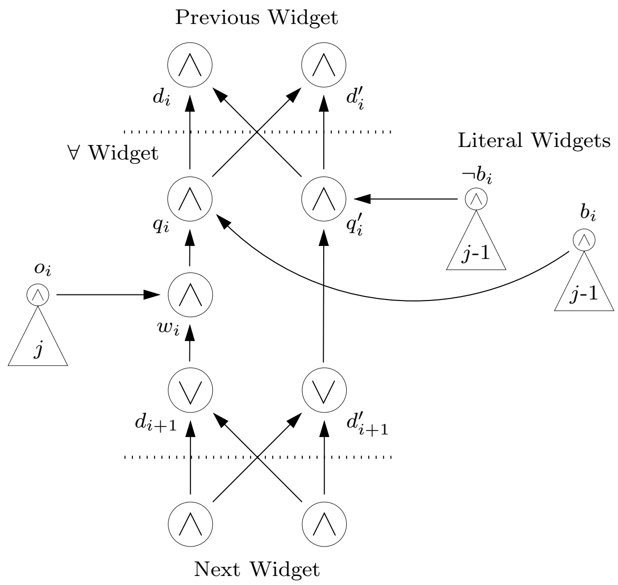

Figure 4.6: The Universal Widget From Lingas's Construction

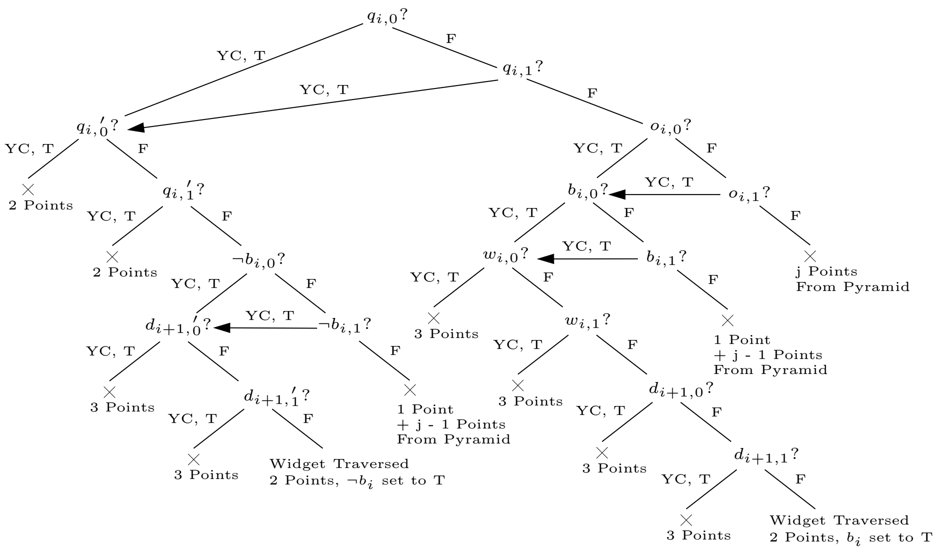

Figure 4.7: The Decision Tree Showing the Prover's Strategy for Traversing the Universal Widget

We now give the Prover's strategy for traversing the conjunctive pyramid graph. The apex of the conjunctive pyramid is part of the final quantifier widget, so both of its variables have been set to False. Since the formula underlying the Lingas circuit has M clauses and we have successfully traversed all of the clause widgets while giving up only the minimum number of points, we still have M points remaining. The conjunctive pyramid required to join M clauses has $M-1$ AND gates on its base, and therefore has pebbling number $M-1$.

Let us assume for a moment that the leaves of the conjunctive pyramid have no children. Since the pyramid belongs to the $\mathcal{G}_{\mathcal{I}}$ family, by Lemma 4.3.1 the Prover has a strategy limiting the Delayer to at most $M-1$ points. However, since each of the conjunctive pyramid's leaves does have two children, we must explore the decision tree rooted at the possibility that the two variables associated with a leaf l are both set to False. Since the Delayer did not say 'You Choose' on either of l's variables, the number of points scored in setting them both to False is at most $M-2$. This is the worst-case scenario branching down the conjunctive pyramid, so the number of points scored on all other paths to leaves is strictly less than $M-2$.

We therefore have at least 2 points left with which to derive a contradiction within the clause widgets, and shall show how to force a contradiction when given a conjunction pyramid leaf l which has had both of its associated variables set to False. The Prover's strategy for doing this is shown below in Figure 4.8.

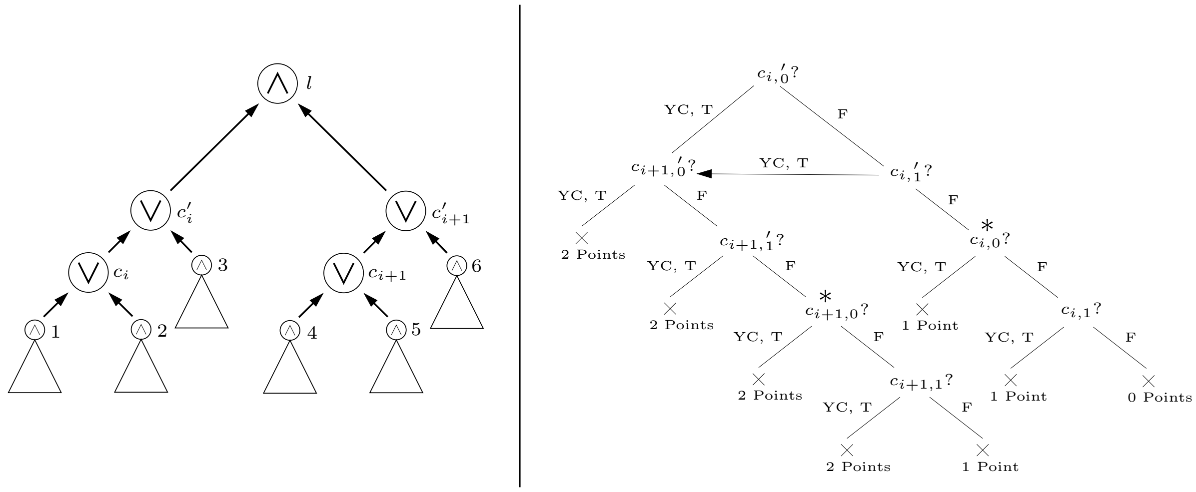

Figure 4.8: Two Clause Widgets Attached to a Leaf from the Conjunctive Pyramid in Lingas's Construction, Together with the Decision Tree Showing the Prover's Strategy for Finishing the Game

A key observation to make is that it is possible for the Prover to traverse the quantifier widgets such that the literal widgets attached to the existential widgets are set to True in any arbitrary way. Since the formula on which the overall circuit is based is a true QBF formula F, it is therefore possible to set them so that each clause widget is incident on at least one literal widget which has been set to True. This ensures that the Prover's strategy given by the decision tree in Figure 4.8 will always work. For example, it is possible for the Prover to traverse the existential widgets and set the truth values on the literal widgets in such a way that at least one of the variables associated with the literal widgets labelled 1, 2, and 3 is True, and at least one of the variables associated with the literal widgets labelled 4, 5, and 6 is True, thereby guaranteeing that the final contradictions needed can be obtained without exceeding 2 points. Note that the subtrees in the Prover's strategy labelled with $*$ may or may not be necessary, depending on which literal widgets were set to True.

Since each existential widget can be traversed while only giving up 1 point, each universal widget can be traversed while only giving up 2 points, the conjunctive pyramid can be traversed while only giving up $M-2$ points, and then a final contradiction can always be derived within the clause widgets while giving up at most 2 points, it follows that all of the above decision trees can be combined to show that the Prover has a strategy limiting the Delayer to at most $B\text{-}Peb(C)$ points (defined with sliding), as required. $\square$

The Prover/Delayer tree constructed above encodes a brute-force proof that F is a true QBF formula. This is particularly apparent when looking at the decision tree associated with the universal widget; it has two leaves at which we attach the decision tree associated with the next (existential) widget. In other words, the trivial decision tree proof system for the Prover/Delayer game which we have been using simulates a trivial brute force QBF proof system. This can be contrasted to the other brute force QBF proof system described in Section 12.3.3.

4.5 Delayer Strategy for All Monotone Circuits

We now show that for any monotone circuit C, when playing the Prover/Delayer game on $Peb_2(C)$, the Delayer has a strategy which will always win at least B-$Peb(C)$ points. In [BSIW04], Ben-Sasson, Impagliazzo, and Wigderson give a Delayer strategy which wins at least B-$Peb(G) - 3$ points, where B-$Peb(G)$ is defined without sliding. We improve this argument to show that the Delayer has a strategy which is guaranteed to win at least B-$Peb(G)$ points, defined with sliding.

In the Delayer's strategy in [BSIW04], the Delayer maintains a DAG G, with certain nodes marked as source nodes, and other nodes marked as target nodes. In the improved strategy, which applies to the more general case of monotone circuits, the Delayer also maintains such a monotone circuit C marked with source and target nodes, but in addition, certain source nodes have pebbles on them. Thus at each stage of the game, the Delayer maintains a structure of the form $\langle C, S, T, P\rangle$, where C is a monotone circuit, S and T are disjoint subsets of C, and $P \subseteq S$. By B-$Peb(C, S, T, P)$, we mean the pebbling number of the marked, pebbled circuit $\langle C, S, T, P\rangle$, where the pebbles on the nodes in P can be used as 'free pebbles', and in addition we allow sliding, which seems to simplify the proof.

The Delayer's strategy proceeds as follows: Initially, at the start of the game, P is empty (there are no free pebbles on the circuit), S is the set of source nodes of the circuit C, and T contains the unique target node of C. At the start of each round in the game, the Prover queries a variable v_i, associated with a vertex v in the circuit C. The Delayer's strategy consists of two stages. In the first stage, the Delayer updates the sets S and T; in the second stage, the Delayer answers the Prover's query and updates P.

For the first stage, assume that the marked, pebbled circuit at the start of the round is $\langle C, S, T, P\rangle$; in the first stage, the Delayer updates S and T to S' and T' as follows:

Case 1a: If $v \in S \cup T$, then set $S' := S$, and $T' := T$.

Case 1b: If $v \notin S \cup T$, and B-$Peb(C, S, T, P) = B$-$Peb(C, S, T \cup \{v\}, P)$, then set $S' := S$ and $T' := T \cup \{v\}$.

Case 1c: If $v \notin S \cup T$, and B-$Peb(C, S, T, P) > B$-$Peb(C, S, T \cup \{v\}, P)$, then set $S' := S \cup \{v\}$ and $T' := T$.

The second stage of the Delayer's strategy proceeds by updating P to P', and responding to the Prover's query as follows:

Case 2a: If $v \in S'$, and v has no pebble on it, then respond 'You Choose', and place a pebble on v (that is to say, $P' := P \cup \{v\}$). If $v \in S'$ and v already has a pebble on it, then set the queried variable the value $True$.

Case 2b: If $v \in T'$, then set the queried variable the value False, and set $P' := P$.

Lemma 4.5.1. *When the game terminates, B-$Peb(C, S, T, P) = 0$.*

Proof: When the game terminates, an initial clause is falsified. Because of the strategy followed by the Delayer in the second stage of each round, this clause cannot be a clause associated with one of the initial source nodes of C, nor can it be the clause associated with the target node of C. Consequently, it must be a pebbling propagation axiom associated with a vertex v and its immediate predecessors $u_1, \ldots, u_k$ if v is an $\wedge$ gate, or just one of its immediate predecessors if v is an $\vee$ gate.

Let us assume that v is an $\wedge$ gate. Since the clause containing v was falsified, both variables associated with v must have been set to False, and for each u_i, at least one of its two associated variables must have been set to True. By Case 2a of the Delayer's strategy, it is impossible for v to be in S and have both of its variables set to False, so v must be in T. However, each u_i had at least one of its associated variables set to True. Therefore by Case 2b of the Delayer's strategy, u_i cannot be in T, so $u_i \in S$ and has a pebble on it.

Therefore we can pebble $\langle C, S, T, P\rangle$ by sliding the pebble on u_i to v, showing that $B\text{-}Peb(C, S, T, P) = 0$. The second case, where v is an $\vee$ gate, proceeds by essentially the same argument. □

Lemma 4.5.2. *If $\langle C, S, T, P\rangle$ is a marked, pebbled circuit, and v a vertex in C, then*

$$B\text{-}Peb(C, S, T, P) \leq \max\{B\text{-}Peb(C, S, T \cup \{v\}, P), B\text{-}Peb(C, S \cup \{v\}, T, P \cup \{v\}) + 1\}.$$

Proof: We employ the following strategy to pebble T from S, using only the free pebbles in P. First, pebble $T \cup \{v\}$ from S, using $B\text{-}Peb(G, S, T \cup \{v\}, P)$ pebbles. If the result has a pebble on a node in T, then we are finished, and otherwise the result has a pebble on v. Keeping this pebble on v, remove the other pebbles, and then pebble T from $S \cup \{v\}$; this final step uses a total of $B\text{-}Peb(C, S \cup \{v\}, T, P \cup \{v\}) + 1$ pebbles. □

Lemma 4.5.3. *If at the beginning of a round in the game, the marked, pebbled circuit is $\langle C, S, T, P\rangle$, then the Delayer has a strategy which is guaranteed to score $B\text{-}Peb(C, S, T, P)$ more points in the game.*

Proof: We argue by induction on the depth of the game tree. Lemma 4.5.1 settles the base case. Assume that our statement holds for a subtree in which the marked, pebbled circuit is $\langle C, S', T', P'\rangle$; we wish to show that it also holds for its immediate supertree, associated with the marked, pebbled circuit $\langle C, S, T, P\rangle$. If $B\text{-}Peb(C, S, T, P) = B\text{-}Peb(C, S', T', P')$, then there is nothing to prove, so we can assume that $B\text{-}Peb(C, S, T, P) > B\text{-}Peb(C, S', T', P')$. By Lemma 4.5.2, $B\text{-}Peb(C, S, T, P) = B\text{-}Peb(C, S', T', P') + 1$.

Now consider the round of the game in which the Delayer's initial circuit is $\langle C, S, T, P\rangle$, and the final circuit is $\langle C, S', T', P'\rangle$, and the variable queried was v_i. If v were in T', or if v had a pebble on it at the start of the round, then $B\text{-}Peb(C, S, T, P) = B\text{-}Peb(C, S', T', P')$. It follows from this that v is in S', but does not have a pebble on it at the start of the round, so the Delayer scores a point during this round. This proves that the condition of the Theorem holds for the supertree as well. □

Theorem 4.5.4. *For any binary monotone circuit C, when playing on the formula $Peb_2(C)$, the Delayer has a strategy which wins at least $B\text{-}Peb(C)$ points (with sliding).*

Proof: By Lemma 4.5.3, at the beginning of the first round of the game the Delayer can win $B\text{-}Peb(C, S, T, P)$ points, but $P = \emptyset$, so this is exactly $B\text{-}Peb(C)$ points. □

4.6 Black Pebbling, Prover/Delayer Game, & Tree Clause Space Equivalence

Since Theorem 4.5.4 applies to all binary monotone circuits, we can combine it with the upper bound on points from Lemma 4.3.1 to get an exact equivalence between black pebbling number and Prover/Delayer number for the $\mathcal{G}_\mathcal{I}$ DAGs. However, when we further include the equivalence between $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) - 1$ and $PD(F)$ from Theorem 3.3.7, we get the following three-way equivalence for the polytime solvable pebbling graphs:

Corollary 4.6.1. *For any increasing binary DAG G, if the pebbling game is defined using sliding, then $PD(Peb_2(G)) = B\text{-}Peb(G) = CS(Peb_2(G) \vdash_{\mathsf{T\text{-}RES}} \emptyset) - 1$.*

More importantly, the Delayer's lower bound on points for binary monotone circuits also includes the Lingas circuits, for which we already know that the Prover can limit the Delayer to at most $B\text{-}Peb(C)$ points because of the result in Theorem 4.4.2, thereby showing that for these circuits, black pebbling number and Prover/Delayer number are equivalent. When we further combine this with Theorem 3.3.7, we get the following three-way equivalence:

Corollary 4.6.2. *For any binary circuit $C \in \mathcal{C}_\mathcal{L}$, if the pebbling game is defined using sliding, then $PD(Peb_2(C)) = B\text{-}Peb(C) = CS(Peb_2(C) \vdash_{\mathsf{T\text{-}RES}} \emptyset) - 1$.*

4.7 The PSPACE-Completeness of The Prover/Delayer Game & Tree Clause Space

This section contains an immediate corollary to the results in the previous sections, namely the $\mathcal{PSPACE}$-Completeness of both the T-RES clause space problem as well as the Prover/Delayer game. The T-RES clause space problem ($TCSP$) is defined as follows: Given a formula F and integer k, does F have a T-RES refutation with clause space at most k? The Prover/Delayer game problem ($PDGAME$) is defined as follows: Given a formula F and integer k, is the Prover/Delayer number of F at most k?

More formally, the languages associated with these problems are defined as follows:

Definition 4.7.1 ($TCSP$). $TCSP = \{(F, k) \mid F$ *is a formula for which there exists a* T-RES *refutation with clause space at most* k.$\}$

Definition 4.7.2 ($PDGAME$). $PDGAME = \{(F, k) \mid F$ *is a formula for which the Prover/Delayer number is at most* k.$\}$

Given Theorem 4.4.2 and Theorem 4.5.4 from the previous sections, proving the $\mathcal{PSPACE}$-Completeness of $TCSP$ and $PDGAME$ follows via a fairly straightforward reduction from the $\mathcal{PSPACE}$-Complete black pebbling problem of Lingas circuits with sliding. However, before proving this result we must first give $\mathcal{PSPACE}$ algorithms for $TCSP$ and $PDGAME$.

4.7.1 PSPACE Algorithms for TCSP and PDGAME

We first show that $TCSP \in \mathcal{PSPACE}$, which is not immediately obvious. In order to do this, we shall make use of the following Lemma:

Lemma 4.7.3. *Every unsatisfiable CNF formula F has an* RT-RES *(and therefore* T-RES *and also* RES*) refutation with clause space at most $n + 1$, where n is the number of distinct variables in F.*

Proof: Choose some ordering for the variables in F. Build a complete DPLL tree for F, branching on this ordering. This tree can be viewed as a T-RES proof which can be pebbled with at most $n + 1$ pebbles, since any binary tree can be pebbled with $h + 1$ pebbles, where h is the height of the tree. These pebbles correspond to which clauses need to be kept in memory in order to verify the proof, showing that for any unsatisfiable F, $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) \leq n + 1$, as required. □

Lemma 4.7.4. $TCSP \in \mathcal{PSPACE}$

Proof: Given an input (F, k) we first determine if F is satisfiable. Since $SAT \in \mathcal{NP}$ and $\mathcal{NP} \subseteq \mathcal{PSPACE}$, this is not a problem. If F is satisfiable, then we reject. If it is unsatisfiable, then we look at k. If $k \geq n+1$, where n is the number of distinct variables in F, then by Lemma 4.7.3 we simply accept.

Otherwise F is unsatisfiable and $k \leq n$, so if F has a (configuration style) T-RES refutation π with $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) \leq k$, then each configuration contains at most k clauses. Since $k \leq n$ and since each clause contains at most n variables, each configuration in π only requires space which is polynomial in n. We nondeterministically guess π as follows: Start with a configuration $\mathbb{C}_0 = \{\}$. Guess configuration $\mathbb{C}_1$, check to ensure that it follows from $\mathbb{C}_0$ by a legal T-RES step, and erase configuration $\mathbb{C}_0$. Next, guess configuration $\mathbb{C}_2$, check to make sure that it follows from $\mathbb{C}_1$, and erase configuration $\mathbb{C}_1$. Continue this way until a configuration containing the empty clause has been derived. Note that at any time, there are only two configurations in memory, but since each configuration takes only a polynomial amount of space by Lemma 4.7.3, our computation is in $\mathcal{NPSPACE}$. Finally, we appeal to Savitch's Theorem [Sav70] to show that determining whether or not F has T-RES refutation π with $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) \leq k$ is in $\mathcal{PSPACE}$, thereby completing our $\mathcal{PSPACE}$ algorithm for TCS. □

Since $TCSP \in \mathcal{PSPACE}$ and it is virtually identical to $PDGAME$, it is trivial to design a $\mathcal{PSPACE}$ algorithm for $PDGAME$:

Lemma 4.7.5. $PDGAME \in \mathcal{PSPACE}$

Proof: To show that $PDGAME \in \mathcal{PSPACE}$, we take the input (F, k) and check if $CS(F \vdash_{\textsf{T-RES}} \emptyset) - 1 \leq k$ using the $\mathcal{PSPACE}$ $TCSP$ algorithm from Lemma 4.7.4 as a subroutine and give the same answer. By Theorem 3.3.7, $CS(F \vdash_{\textsf{T-RES}} \emptyset) = PD(F) + 1$ for any formula F, which immediately proves the correctness of this algorithm. □

4.7.2 The PSPACE-Completeness of TCSP and PDGAME

We are now ready to prove this section's main result:

Theorem 4.7.6. *$PDGAME$ is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

Proof: By Lemma 4.7.5, we know that $PDGAME \in \mathcal{PSPACE}$. Next we show that $PDGAME$ is $\mathcal{PSPACE}$-Hard by reducing from the $\mathcal{PSPACE}$-Complete problem of black pebbling Lingas circuits $\mathcal{C_L}$. Our proof proceeds by taking the input (C, k), and outputting $(Peb_2(C), k)$. Clearly this is a logspace reduction, and it is easy to see that it is correct by showing that $(C, k) \in \mathcal{C_L}$ if and only if $(Peb_2(C), k) \in PDGAME$: If $(C, k) \in \mathcal{C_L}$, then $B\text{-}Peb(C) \leq k$, so by Theorem 4.4.2, $PD(Peb_2(C)) \leq k$, and $(Peb_2(C), k) \in PDGAME$. On the other hand, if $(C, k) \notin \mathcal{C_L}$, then $B\text{-}Peb(C) > k$, so by Theorem 4.5.4, $PD(Peb_2(C)) > k$, and $(Peb_2(C), k) \notin PDGAME$.

Therefore $PDGAME$ is $\mathcal{PSPACE}$-Complete. □

Since by Theorem 3.3.7 we know that for any formula F, $PD(F) = CS(F \vdash_{\textsf{T-RES}} \emptyset) - 1$, the $\mathcal{PSPACE}$-Completeness of $PDGAME$ immediately yields the $\mathcal{PSPACE}$-Completeness of $TCSP$ as a corollary:

Corollary 4.7.7 (Main Result). *$TCSP$ is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

Proof: The proof is very similar to Theorem 4.7.6. By Lemma 4.7.4, we know that $TCSP \in \mathcal{PSPACE}$. To show that $TCSP$ is $\mathcal{PSPACE}$-Hard we reduce from $PDGAME$: Given an input (F, k) for $PDGAME$, output $(F, k+1)$. Clearly this is a logspace reduction, and its correctness is immediate from Theorem 3.3.7. □

4.8 Related Complexity Results

In this section we discuss a number of corollaries to the T-RES clause space results in the previous section, which is worthwhile, since T-RES is such an important proof system in practice, giving a strong motivation for better understanding as many of its aspects as possible. We shall begin by proving that RES Clause Space, T-RES Total space, and T-RES size all have $\mathcal{PSPACE}$ algorithms, and then prove that the clause space problem for Regular Tree Resolution (RT-RES) and the form of Resolution corresponding to clause learning (CL-RES) are both $\mathcal{PSPACE}$-Complete.

4.8.1 The Complexities of Resolution Clause Space & Tree Total Space

Although we are unable to prove completeness results, it is easy to see that the RES clause space problem (CSP), T-RES total space problem ($TTSP$), and RT-RES total space problem ($RTTSP$) are all co$\mathcal{NP}$-Hard, which gives at least some kind of lower bound, and they are both in $\mathcal{PSPACE}$, giving an upper bound. These languages are formally defined as follows:

Definition 4.8.1 (CSP). $CSP = \{(F, k) \mid F$ *is a formula for which there exists a* RES *refutation with clause space at most* $k.\}$

Definition 4.8.2 ($TTSP$). $TTSP = \{(F, k) \mid F$ *is a formula for which there exists a* T-RES *refutation with total space at most* $k.\}$

Definition 4.8.3 ($RTTSP$). *$RTTSP = \{(F, k) \mid F$ is a formula for which there exists a* RT-RES *refutation with total space at most k.$\}$*

Although a gap still exists, we are able to prove simple hardness results and give upper bound algorithms for these problems:

Corollary 4.8.4. *CSP is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

Proof: To show that CSP is co$\mathcal{NP}$-Hard, we reduce from $UNSAT$. It is easy to see that there are 3^n possible clauses on n variables, so any a configuration can contain at most 3^n distinct clauses. Given a formula F, we therefore output $(F, k = 3^n)$, and $F \in UNSAT$ if and only if $(F, k) \in CSP$.

Showing that $CSP \in \mathcal{PSPACE}$ is also very easy: Since RES subsumes T-RES in the sense that every T-RES proof is a RES proof, the $\mathcal{PSPACE}$ $TCSP$ algorithm from Section 4.7.1 immediately implies a $\mathcal{PSPACE}$ algorithm for CSP. This is because any formula which is refutable within $TCS = k$ is also refutable within $CS \leq k$, so by Lemma 4.7.3 we can repeat the proof of Lemma 4.7.4 for CS rather than TCS. □

Corollary 4.8.5. *$TTSP$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

Proof: The co$\mathcal{NP}$-Hardness of $TTSP$ is very similar to that of CSP in Corollary 4.8.4 above. Once again, we reduce from $UNSAT$. We saw that there are 3^n different clauses on n variables, but each clause contains at most n literals, so any unsatisfiable formula is refutable in total space at most $n \cdot 3^n$. For our reduction we therefore take F and output $(F, k = n \cdot 3^n)$, and it is easy to see that $F \in UNSAT$ if and only if $(F, k) \in TTSP$.

The proof that $TTSP \in \mathcal{PSPACE}$ is almost identical to Lemma 4.7.4. We first check to see if F is satisfiable, and if so we reject. Otherwise F is unsatisfiable, so by Lemma 4.7.3, every unsatisfiable formula has a T-RES refutation with clause space at most $n + 1$, and furthermore each clause can only contain at most n variables. Therefore if $k \geq n^2 + n$, then we simply accept. Otherwise F is unsatisfiable, and $k < n^2 + n$, so we nondeterministically guess the configurations of π as in Lemma 4.7.4, keeping only two configurations in memory at any time. Since $k < n^2 + n$, each configuration requires only polynomial space, giving us an $\mathcal{NPSPACE}$ algorithm for $TTSP$. Finally, we appeal to Savitch's Theorem [Sav70] to show that $TTSP \in \mathcal{PSPACE}$. □

An almost identical argument gives corresponding lower and upper bounds for $RTTSP$:

Corollary 4.8.6. *$RTTSP$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

Proof: The reduction showing that $RTTSP$ is co$\mathcal{NP}$-Hard is the same as in Corollary 4.8.5, as is the $\mathcal{PSPACE}$ algorithm, with one exception: Our nondeterministic algorithm for $TTSP$ had no provision in it for guaranteeing that the proof it found would be regular, and our nondeterministic algorithm for $RTTSP$ must include such a provision. Luckily, this is not difficult: The key observation to make is that in a tree-like configuration-style proof, each clause in a configuration implicitly represents a node in the proof's underlying tree. We therefore augment each clause C in the configuration with a set S containing all of the variables which were resolved in the entire subtree which was used to derive C. With each additional resolution step, we simply take the union of these extra sets. This allows us to detect if a variable has been reintroduced after being eliminated along any path. Since we keep only two configurations in memory at any time, this requires at most $n^2 + n$ additional space, so our nondeterministic algorithm still only requires polynomial space, and finally we appeal to Savitch's Theorem as before to show that $RTTSP \in \mathcal{PSPACE}$. □

4.8.2 The Complexity of Tree Resolution Size

Another interesting corollary to the $\mathcal{PSPACE}$-Completeness of T-RES clause space from Section 4.7 concerns the size of T-RES proofs.

Given a formula F and integer k, a natural question to ask is whether F has a RT-RES refutation of size at most k. We shall refer to this as the 'RT-RES size problem', and to its associated language as $RTRSP$, which is defined formally as follows:

Definition 4.8.7 ($RTRSP$)**.** $RTRSP = \{(F, k) \mid F$ *is a formula for which there exists a* RT-RES *refutation with size at most* $k.\}$

Although we do not prove a completeness result for $RTRSP$, we make partial progress by showing that $RTRSP$ is in $\mathcal{PSPACE}$ using techniques very similar to those in Lemma 4.7.4.

Corollary 4.8.8. *$RTRSP$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

Proof: To prove the co$\mathcal{NP}$-Hardness of $RTRSP$, we once again reduce from $UNSAT$ as in Corollary 4.8.4. Since a RT-RES refutation π can be at most as large as the complete binary decision tree on n variables, π can contain at most $2^{n+1} - 1$ clauses. For our reduction we therefore take F and output $(F, k = 2^{n+1} - 1)$, and it is obvious that $F \in UNSAT$ if and only if $(F, k) \in RTRSP$.

It is also easy to see that every RT-RES proof tree can be pebbled using at most $n + 1$ pebbles (with sliding), where n is the height of π. This is because DPLL is equivalent to RT-RES and every DPLL tree is a subtree of the complete binary tree of height n, which itself can be pebbled with at most $n + 1$ pebbles, since we need to keep at most one pebble at each depth in the tree at any time during the pebbling.

This of course includes the smallest such tree π, making it easy to design an $\mathcal{NPSPACE}$ algorithm for $RTRSP$: Given a formula F and integer k, first check to see if it is satisfiable, which is not a problem since $\mathcal{NP} \subseteq \mathcal{PSPACE}$. If so, then reject, and if not, then nondeterministically guess configurations containing clauses corresponding to the pebbling steps in the minimum pebbling of the smallest RT-RES proof tree π of F, keeping at most two configurations in memory at any time, and all the while keeping track of s, the total number of clauses in the refutation. Since the pebbling number of π is at most $n+1$, we need only ever keep at most $n+1$ clauses in memory during the refutation of F, which requires only polynomial space, since each clause can contain at most n variables. In addition, we must ensure that our nondeterministic algorithm produces only regular proofs, which is accomplished by augmenting each clause C in each configuration with a set containing all of the variables used in deriving C, as described in Corollary 4.8.6; these extra sets require at most $n^2 + n$ additional space.

Once π is complete, compare s with k. If $s \leq k$, then accept, and otherwise reject. We therefore have an $\mathcal{NPSPACE}$ algorithm for $RTRSP$, which by Savitch's Theorem [Sav70] gives us a $\mathcal{PSPACE}$ algorithm, as required. □

This in turn yields a $\mathcal{PSPACE}$ algorithm and co$\mathcal{NP}$-Hardness result for the corresponding size problem for T-RES without the regularity requirement. The language associated with the T-RES size problem is formally defined as follows:

Definition 4.8.9 ($TRSP$)**.** $TRSP = \{(F, k) \mid F$ *is a formula for which there exists a* T-RES *refutation with size at most* $k.\}$

Since T-RES refutations of minimal size are regular [Tse70, Urq95], $TRSP = RTRSP$, so Corollary 4.8.8 immediately shows that $TRSP$ is co$\mathcal{NP}$-Hard and in $\mathcal{PSPACE}$ as well:

Corollary 4.8.10. *$TRSP$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

4.8.3 The PSPACE-Completeness of Regular Tree Resolution Clause Space

Yet another interesting corollary to the $\mathcal{PSPACE}$-Completeness of $TCSP$ from Corollary 4.7.7 is that the corresponding problem for RT-RES is also $\mathcal{PSPACE}$-Complete. Recall from Section 2.4.3 that RT-RES is simply T-RES in which the underlying proof tree may not contain any irregularities, so a configuration-style RT-RES proof is a configuration-style T-RES proof in which the underlying proof tree does not contain any irregularities. Our notion of a configuration-style RT-RES proof is given in Definitions 3.2.3, and RT-RES clause space is the same as in Definition 3.2.4, only applied to RT-RES.

The language associated with the RT-RES clause space problem ($RTCSP$) is formally defined as follows:

Definition 4.8.11 ($RTCSP$)**.** $RTCSP = \{(F, k) \mid F$ *is a formula for which there exists a* RT-RES *refutation with clause space at most* $k.\}$

In order to show that calculating RT-RES clause space is $\mathcal{PSPACE}$-Complete, we will reduce from $TCSP$. We shall make use of the following lemma:

Lemma 4.8.12. *For any CNF formula F, $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) = CS(F \vdash_{\mathsf{RT\text{-}RES}} \emptyset)$.*

Proof: $\Rightarrow$ To show that $CS(F \vdash_{\mathsf{RT\text{-}RES}} \emptyset) \leq CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$, we use the fact that any T-RES proof π containing an irregularity can be pruned so as to remove the irregularity, producing a proof π' which is no larger than π [Tse70, Urq95]. Editing π in this way simplifies its underlying proof by removing nodes in its underlying proof tree T to produce T'. This means that $B\text{-}Peb(T') \leq B\text{-}Peb(T)$, since deleting nodes in a pebbling graph removes constraints, which cannot possibly increase its depth or pebbling number. Therefore T' can be pebbled using at most $B\text{-}Peb(T)$ pebbles, showing that $CS(F \vdash_{\mathsf{RT\text{-}RES}} \emptyset) \leq CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$, as required.

$\Leftarrow$ Showing that $CS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) \leq CS(F \vdash_{\mathsf{RT\text{-}RES}} \emptyset)$ is trivial, since any RT-RES refutation is a T-RES refutation. $\square$

This lemma shows that the T-RES and RT-RES clause space problems are identical:

Corollary 4.8.13. *$TCSP = RTCSP$*

Combining this with Corollary 4.7.7, which showed that $TCSP$ is $\mathcal{PSPACE}$-Complete immediately implies the $\mathcal{PSPACE}$-Completeness of $RTCSP$ as well:

Corollary 4.8.14. *$RTCSP$ is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

4.8.4 The PSPACE-Completeness of Clause Learning Clause Space

An interesting corollary to the $\mathcal{PSPACE}$-Completeness of the RT-RES clause space problem in the previous section is that predicting the space requirements for clause learning algorithms is also $\mathcal{PSPACE}$-Complete. From a practical point of view, this is probably of more interest than the corresponding result for T-RES or RT-RES, since the state of the art in automated theorem proving consists of clause learning rather than strict DPLL algorithms.

In addition to being of practical interest, clause learning algorithms have also been studied from a formal theoretical point of view. For example, in [BKS03] Beame, Kautz, and Sabharwal formally define clause learning as a proof system (CL-RES), and compare it to RES. Defining CL-RES formally is somewhat problematic because the literature contains so many different clause learning implementations, involving dozens of different characteristics including learning heuristics, restarting conditions, branching orders, and more. These different variations strongly effect proof size.

However, exploring clause learning space is a much simpler proposition, and it is easy to give a definition of CL-RES from a space point of view which is robust enough to capture all reasonable variants of the proof system; from the point of view of space, issues such as random restarts, which learning heuristics are being used, how the cache is being updated, etc. are all irrelevant. Intuitively, CL-RES is just DPLL (or RT-RES) with an extra cache in which clauses can be stored or 'learned' so that they need not be re-derived. Adding an extra cache to the configuration-style RT-RES proof system therefore allows us to define a configuration-style clause learning proof. As before, we first define the concept of a configuration-style clause learning proof:

Definition 4.8.15 (Configuration-Style Clause Learning Proofs). *A configuration $\mathbb{C}$ is a set of clauses. Each step in a clause learning proof is a pair of configurations ($\mathbb{C}_a$,$\mathbb{C}_b$), where $\mathbb{C}_a$ is the set of clauses in memory as in any* T-RES *proof, and $\mathbb{C}_b$ is the cache of learned clauses. If F is a formula (set of clauses), then the sequence of configuration pairs $\pi = (\mathbb{C}_{0,a}, \mathbb{C}_{0,b}), (\mathbb{C}_{1,a}, \mathbb{C}_{1,b}), ..., (\mathbb{C}_{k,a}, \mathbb{C}_{k,b})$ is a* CL-RES *proof of C from F if ($\mathbb{C}_{0,a} = \emptyset$, $\mathbb{C}_{0,b} = \emptyset$), $C \in \mathbb{C}_{k,a}$, and for each $i < k$, $(\mathbb{C}_{i+1,a}, \mathbb{C}_{i+1,b})$ is obtained from $(\mathbb{C}_{i,a}, \mathbb{C}_{i,b})$ by one of the following rules:*

1. *Adding one or more of the clauses of F (initial clauses) to $\mathbb{C}_{i,a}$,*

2. *Deleting one or more of the clauses in* $\mathbb{C}_{i,a}$ *or* $\mathbb{C}_{i,b}$,

3. *Adding any clause from* $\mathbb{C}_{i,a}$ *to* $\mathbb{C}_{i,b}$,

4. *Adding any clause from* $\mathbb{C}_{i,b}$ *to* $\mathbb{C}_{i,a}$, *or*

5. *Adding the resolvent of two clauses of* $\mathbb{C}_{i,a}$ *to* $\mathbb{C}_{i,a}$ *and deleting both parent clauses, provided that this does not introduce an irregularity in the underlying proof tree.*

Having established what configurations look like for CL-RES proofs, we can now define our notion of CL-RES clause space, which is similar to our previous notion of clause space in Definition 3.2.4. This definition is robust in the sense that it is compatible with any formalization of clause learning, including those in [BKS03], and is also consistent with our intuitive notion that clause space is a measure of the amount of memory being used during the execution of a clause learning algorithm.

Definition 4.8.16 (Clause Learning Clause Space). *Let F be a set of clauses and π be a configuration-style* CL-RES *proof of clause C from F. The* CL-RES *clause space of a configuration $(\mathbb{C}_a, \mathbb{C}_b)$ in π, denoted $CLCS(\mathbb{C}_a, \mathbb{C}_b)$, is $|\mathbb{C}_a| + |\mathbb{C}_b|$. The* CL-RES *clause space of π, denoted $CLCS(\pi)$, is the maximum $CLCS(\mathbb{C}_a, \mathbb{C}_b)$ over all $(\mathbb{C}_a, \mathbb{C}_b) \in \pi$. Finally, the clause learning clause space of resolving C from F with a cache of size c, denoted $CLCS(F \vdash_{\text{CL-RES}} C, c)$, is the minimum $CLCS(\pi)$ over all clause learning proofs π of C from F such that for all configurations $(\mathbb{C}_a, \mathbb{C}_b)$ in π, $|\mathbb{C}_b| \leq c$.*

Apart from the extra cache, the important difference between CL-RES clause space and our previous definition of clause space is the c parameter in $CLCS(F \vdash_{\text{CL-RES}} C, c)$. We include this parameter in the definition because it is important to be able to limit the size of the cache being used, as is the case in real-world implementations of clause learning algorithms.

This brings us to the CL-RES clause space problem ($CLCSP$), which is given a triple (F, k, c), and asks if there exists a CL-RES refutation of F with clause space bounded by k and cache size bounded by c. In other words, it asks if $CLCS(F \vdash_{\text{CL-RES}} C, c) \leq k$.

Definition 4.8.17 ($CLCSP$). $CLCSP = \{(F, k, c) \mid$ *F is a formula for which there exists a* CL-RES *refutation with clause space at most k and a cache size of at most c.*$\}$

We shall show that $CLCSP$ is $\mathcal{PSPACE}$-Complete by reducing from the corresponding RT-RES clause space problem from Corollary 4.8.14:

Corollary 4.8.18. *$CLCSP$ is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

Proof: We first show that $CLCSP \in \mathcal{PSPACE}$ by giving an algorithm which is almost identical to the TCS algorithm from Lemma 4.7.4. Given an input (F, k, c) we first determine if F is satisfiable. Since $SAT \in \mathcal{NP}$ and $\mathcal{NP} \subseteq \mathcal{PSPACE}$, this is not a problem. If F is satisfiable, then we reject. If it is unsatisfiable, then we look at k. If $k \geq n + 1$, where n is the number of distinct variables in F, then by Lemma 4.7.3 we simply accept, since CL-RES contains RT-RES as a sub-proof system since we can simply refuse to use the cache. Otherwise F is unsatisfiable and $k \leq n$, so if F has a (configuration style) CL-RES refutation π with $CLCS(F \vdash_{\text{CL-RES}} \emptyset, c) \leq k$, then each configuration contains at most k clauses. Since $k \leq n$ and since each clause contains at most n variables, each configuration in π requires only polynomial space. This allows us to use the same $\mathcal{NPSPACE}$ algorithm from Lemma 4.7.4, and then simply appeal to Savitch's Theorem [Sav70] to show that $CLCSP \in \mathcal{PSPACE}$.

To show $\mathcal{PSPACE}$-Hardness, we give a trivial reduction from $RTCSP$: Given (F, k), we output $(F, k, 0)$, which hamstrings our CL-RES proof system by giving it a cache size of zero, effectively turning it into exactly RT-RES. Since this is clearly a log space reduction and since it is obvious that $(F, k) \in TCSP$ if and only if $(F, k, 0) \in CLCSP$, our hardness result follows, showing that $CLCSP$ is $\mathcal{PSPACE}$-Complete. ☐

4.9 Open Problems & Conjectures Related to Tree Resolution Clause Space

The research in this chapter has highlighted several interesting open problems:

4.9.1 The Complexity of Resolution Clause Space

Although proving the $\mathcal{PSPACE}$-Completeness of T-RES clause space was an important step, determining the complexity for the RES clause space problem remains an interesting open proof complexity resource problem, and is one of the most important unsettled problems from the summary table from Section 3.4. This problem was given formally by Definition 4.8.1.

One of the issues which makes this problem so difficult is that unlike T-RES clause space, which is known to be equivalent to the Prover/Delayer game, nobody has yet been able to show that RES clause space has a natural game characterization which could be used to help prove a $\mathcal{PSPACE}$-Completeness result.

From Corollary 3.3.10 we saw that the minimum size of T-RES refutations is exponential in T-RES clause space. However, since $CS(F \vdash_{\mathsf{RES}} C) \leq CS(F \vdash_{\mathsf{T\text{-}RES}} C)$ for all formulas, it follows that the minimum size of T-RES refutations are exponential in RES clause space as well. This means that although researchers working with SAT-solvers will probably not be able to build a T-RES clause space algorithm to use as a pre-processor, the possibility still exists of designing a RES clause space algorithm.

Nevertheless, we consider this to be unlikely, and conjecture that the RES clause space problem is also $\mathcal{PSPACE}$-Complete. We already saw from Corollary 4.8.4 that it is co$\mathcal{NP}$-Hard, but in $\mathcal{PSPACE}$, and with the RES total space and T-RES clause space problems both being $\mathcal{PSPACE}$-Complete, it seems safe to conjecture that RES clause space is $\mathcal{PSPACE}$-Complete as well.

Another related problem worth mentioning which does not fit the paradigm of a proof complexity resource problem concerns the relationship between RES proof size and clause space. We know that RES proof size is exponential in width and that T-RES proof size is exponential in T-RES clause space. However, is RES size exponential in RES clause space, or do there exist formulas with linear RES clause space lower bounds which have poly-sized proofs?

4.9.2 The Complexities of Tree Resolution Total Space & Tree Resolution Size

Corollaries 4.8.5, 4.8.6, 4.8.8, and 4.8.10 respectively showed us that T-RES total space, RT-RES total space, RT-RES size, and T-RES size are co$\mathcal{NP}$-Hard, but have $\mathcal{PSPACE}$ algorithms. Since RES total space is $\mathcal{PSPACE}$-Complete [HP07] and I-RES total space is $\mathcal{PSPACE}$-Complete (see Chapter 5), it seems reasonable to conjecture that the corresponding problems for T-RES and RT-RES are also $\mathcal{PSPACE}$-Complete and within reach. It is more difficult to gauge whether the T-RES size problem is $\mathcal{PSPACE}$-Complete, but this is also an interesting open problem.

4.9.3 The Complexity of Resolution Size

Although Corollaries 4.8.8 and 4.8.10 showed that $RTRSP$ and $TRSP$ are both in $\mathcal{PSPACE}$, it is far from clear that the corresponding size problem for RES has a $\mathcal{PSPACE}$ algorithm. This problem is defined formally as follows:

Definition 4.9.1 (RSP). $RSP = \{(F, k) \mid$ *F is a formula for which there exists a* RES *refutation with size at most k.*$\}$

In fact, we are unable to even show that RSP is in $\mathcal{EXPTIME}$, let alone that it is complete for one of these complexity classes. However, it is easy to show that this problem has a lower bound of being co$\mathcal{NP}$-Hard, and an upper bound of being in $\mathcal{NEXPTIME}$:

Lemma 4.9.2. *RSP is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{NEXPTIME}$.*

Proof: The co$\mathcal{NP}$-Hardness of RSP is very similar to that of $TRSP$ from Corollary 4.8.8, and once again we reduce from $UNSAT$. Since any unsatisfiable formula has an RT-RES refutation of size at most $2^{n+1}-1$, and since every RT-RES refutation is a RES refutation, we take F and output $(F, k = 2^{n+1}-1)$. It is easy to see that $F \in UNSAT$ if and only if $(F,k) \in RSP$.

The following algorithm shows that $RSP \in \mathcal{NEXPTIME}$: Given a formula F and integer k, if $k > 2^{n+1}-1$, then run an $\mathcal{NP}$ SAT-solver, and return the same answer. Otherwise nondeterministically guess a RES refutation π of size $\leq k$. Since RES has a size of at most $2^{n+1}-1$, π contains at most an exponential number of clauses and can therefore be computed in nondeterministic exponential time, as required. □

This is one of the most interesting open Resolution resource problems, and its large complexity gap leaves for a great deal of improvement. We tentatively conjecture that RSP is $\mathcal{EXPTIME}$-Hard, and maybe even $\mathcal{NEXPTIME}$-Complete.

4.9.4 The Complexity of Clause Learning Space

Corollary 4.8.18 showed us that calculating clause learning clause space is $\mathcal{PSPACE}$-Complete. However, the proof relied on the presence of the c parameter, which we set to zero in order to eliminate the presence of the cache. Although this was an important first step, the complexity of the CL-RES clause space problem without the c parameter would be of great interest to the SAT-solving community, because in practice the cache uses up so much memory, so predicting its memory requirements is very important. This remains an important open problem, and we conjecture that it is also $\mathcal{PSPACE}$-Complete.

4.9.5 The Complexity of Resolution Depth

Another Resolution resource problem which has not been addressed in this thesis nor in the literature is the depth of proofs, which is sometimes referred to as rank. Resolution size, width, and space measures have all been studied, but they are not the only interesting resources. Depth can also be considered a resource, and we can easily ask if a formula F has a RES, T-RES, or I-RES refutation of depth at most k. To date, the complexity of this problem has not been addressed for Resolution. This is somewhat surprising given the obvious relationship between circuit complexity and bounded-depth proof systems.

In fact, depth is similar to T-RES clause space in that it is not difficult to formulate a new Prover/Delayer game characterization which captures it. We have done some preliminary work in this area, but have yet to prove a hardness result for one of the standard complexity classes.

4.9.6 Approximation Algorithms

This area of research also has some natural approximation problems associated with it. In this thesis we show that several space problems are $\mathcal{PSPACE}$- or $\mathcal{EXPTIME}$-Complete, but do approximation algorithms exist? The size of T-RES and RES refutations are respectively exponential in T-RES clause space and RES width, so it is unlikely that approximation algorithms would be very useful in practice, but it would nevertheless be interesting to develop such algorithms or prove that these problems cannot be approximated to within a certain factor.

Of course, the same question can be asked of the several different pebbling problems which we used in our reductions: Does there exist an algorithm for approximating the pebbling number of a graph or circuit to within a certain factor?

4.9.7 Tension Between Size & Space

Another interesting open problem related to this research has to do with combined resources for SAT-solvers: The limiting factor on most modern SAT-solving algorithm tends to be memory space, but on the other hand, if one is too frugal with memory, then proofs can become intractably large (for example, see

Section 5.7.2 or [HP07]). Algorithms must therefore carefully balance attempts to save memory with a potentially massive increase in minimum proof size and running time if they save too much. We hope that future researchers will be able to build on these present results in order to better understand the tension between size and space and perhaps be able to use this to optimize SAT-solvers so that they are able to strike the right balance between using too much memory and requiring too much time.

4.9.8 The Space Complexity of Other Proof Systems

In this thesis we address many resource problems for Resolution and its refinements, which is appropriate given the close relationship between Resolution-based proof systems and SAT-solvers, but we could just as easily ask the same questions about any other proof system. For example, for some appropriate resource R (e.g. space, width, depth, size, etc.), given a formula F and integer k, does there exist a Frege proof of F with R bounded above by k? Similarly, we know that all Frege systems are p-equivalent with respect to proof size, but are they also space-equivalent? Is LK with cut space-equivalent to Frege systems?

Chapter 5

The PSPACE-Completeness of Input Resolution Total Space

5.1 Introduction & Motivation

When devising SAT-solving algorithms, there is always a tension between proof size and proof search; more powerful proof systems have shorter proofs, but they are much harder to find, making it difficult to design proof search algorithms. For this reason, virtually all automated theorem provers implement weak proof systems such as RES rather than more powerful systems such as Frege. However, automated proof search is even too complicated in RES, which is why researchers have further developed 'Resolution refinements', weakened forms of Resolution aimed at simplifying proof search at the possible expense of increasing minimum proof size. Common refinements include T-RES / DPLL and clause learning, which are both very successful SAT-solving algorithms. The ultimate goal of automated theorem proving is to develop algorithms capable of finding proofs which are only polynomially larger than the optimal. This property is referred to as 'automatizability' (see Definition 3.2.16).

One particularly extreme refinement is Input Resolution (I-RES), in which we require that at least one of the inputs to each application of the resolution rule be an initial, or input clause. This clearly restricts the search space that any I-RES algorithm would have to deal with, but the cost is very high, since I-RES is not even complete. However, despite its simplicity and obvious limitations, we shall show that the I-RES proof system is much more interesting than it may first appear. It is of interest theoretically as well as practically because of a theorem by Chang [Cha70] which states that a formula has an I-RES refutation if and only if it has a Unit Resolution (U-RES) refutation. Since U-RES is such an important and widely-used subroutine in SAT-solving as well as other areas of computer science, this gives a strong motivation for better understanding I-RES. One interesting way of looking at I-RES is through its relationship to Linear Resolution L-RES. Much like Clause Learning can be viewed as RT-RES with caching, so can L-RES be viewed as I-RES with caching. Similarly, I-RES can be viewed as tree-like L-RES. For a more formal description of I-RES, U-RES, and L-RES, please refer to Definition 3.2.3.

In this chapter we prove a number of complexity results for I-RES including $\mathcal{P}$-Completeness, $\mathcal{NP}$-Completeness, $\mathcal{PSPACE}$-Completeness, and exponential tradeoff results. In addition, we show that I-RES is automatizable, and optimally automatizable on minimally unsatisfiable formulas. The first part of this chapter contains straightforward results dedicated to investigating the more tractable aspects of I-RES, whereas the second half investigates its more complex characteristics.

Our results start in Section 5.2, where we show how formulas with I-RES refutations ($IRES\text{-}UNSAT$) are related to Horn formulas, and minimally unsatisfiable formulas which have I-RES refutations ($MU\text{-}IRES\text{-}UNSAT$) are related to the minimally unsatisfiable formulas in $MU(1)$. In addition, we prove a separation between I-RES and U-RES and give a matrix characterization of $MU\text{-}IRES\text{-}UNSAT$.

Next, in Section 5.3, we prove a number of tractability results for I-RES. For example, we combine some previous results to show that $IRES\text{-}UNSAT$ is $\mathcal{P}$-Complete, and that the problem of determining if a formula is minimally unsatisfiable and has an I-RES refutation, as well as the problem of determining if a formula F is minimally unsatisfiable and has an I-RES refutation of size at most k are both in $\mathcal{P}$. These results lead us to Section 5.4, in which we develop algorithms for automatizing I-RES and automatizing I-RES optimally on minimally unsatisfiable formulas.

The second half of this chapter is devoted to the more complicated aspects of I-RES: In Section 5.5, we note that the problem of approximating optimal I-RES and U-RES refutation size for $IRES\text{-}UNSAT$ to within a linear factor is $\mathcal{NP}$-Hard and that given a formula F and integer k, the problem of determining whether F has an I-RES refutation of size at most k is $\mathcal{NP}$-Complete. This stands in contrast with the tractability of the same problem for $MU\text{-}IRES\text{-}UNSAT$ from the previous section.

The rest of this chapter contains our main results, and is dedicated to the study of I-RES space. These results are more complicated than the earlier ones.

In Section 5.6 we prove an equivalence between I-RES total space and pebbling, followed immediately by its implications for complexity theory. More specifically, we show that for any binary DAG G, its black pebbling number (defined with sliding) is off by exactly a constant from the total space required by any I-RES proof of a slight modification to the formula $Peb_1(G)$. This equivalence allows us to prove this chapter's main results, namely the $\mathcal{PSPACE}$-Completeness of various forms of the I-RES total space problem, which is proved by reducing from the black pebbling game on DAGs, itself shown to be $\mathcal{PSPACE}$-Complete by Gilbert, Lengauer, and Tarjan [GLT80]. These results demonstrate that although I-RES is very simple in some ways, with respect to others it is extremely complex.

Next, in Section 5.7 we prove several interesting corollaries to our main results, including the $\mathcal{PSPACE}$-Completeness of I-RES derivation width and a $\mathcal{PSPACE}$ algorithm for the I-RES variable space problem. However, the most interesting corollary that we prove is an extreme (and optimal) size / total space tradeoff for I-RES; we show that there exists an infinite family of formulas whose I-RES proofs with the minimum required total space have size $2^{\Omega(n)}$, where n is the number of distinct variables. However, if only one single additional unit of total space is permitted, then the size drops to only $O(n)$, where n is the number of variables. Apart from being a massive size / space tradeoff, this is also a tradeoff similar to those explored by Ben-Sasson in [BS02] because it shows that for certain formulas, it is not possible to optimize both I-RES size and space at the same time.

Finally, in Section 5.8 we discuss some interesting open problems related to the research in this chapter.

5.2 Input Resolution, Horn Formulas, and MU Formulas

In this section we define $IRES\text{-}UNSAT$, the set of formulas which have I-RES refutations, as well as $MU\text{-}IRES\text{-}UNSAT$, its minimally unsatisfiable subset, and relate these languages to unsatisfiable Horn formulas and minimally unsatisfiable (MU) formulas. More specifically, we show that $HORN\text{-}UNSAT$ is a proper subset of $IRES\text{-}UNSAT$, and $MU\text{-}IRES\text{-}UNSAT$ is a proper subset of $MU(1)$. Before proceeding, we formally define these languages:

Our first languages are the formulas for which I-RES and U-RES are complete:

Definition 5.2.1 (IRES-UNSAT & URES-UNSAT). *$IRES\text{-}UNSAT$ is the set of all unsatisfiable formulas which have* I-RES *refutations, and $URES\text{-}UNSAT$ is the set of all unsatisfiable formulas which have* U-RES *refutations.*

Our next set of languages are related to Horn formulas and are defined as follows:

Definition 5.2.2 (Horn Formulas, HORN-SAT, & HORN-UNSAT). *A formula F is said to be Horn if each of its clauses contains at most one positive literal. We will refer to the set of all satisfiable Horn formulas as $HORN\text{-}SAT$, and the set of all unsatisfiable Horn formulas as $HORN\text{-}UNSAT$.*

Our final set of languages are based on minimally unsatisfiable formulas:

Definition 5.2.3 (MU(k), MU-IRES-UNSAT, & MU-HORN-UNSAT). *A formula F is said to be minimally unsatisfiable if it is unsatisfiable, but the same cannot be said of any proper subset of its clauses. These minimally unsatisfiable formulas comprise the set MU. We define $MU(k)$ as the set of all minimally unsatisfiable formulas on n variables which contain exactly $n+k$ clauses. In addition, MU-$IRES$-$UNSAT$ contains all of the minimally unsatisfiable formulas in $IRES$-$UNSAT$, and MU-$HORN$-$UNSAT$ contains all of the minimally unsatisfiable formulas in $HORN$-$UNSAT$.*

It is worth noting that $Peb_1(G) \in HORN$-$UNSAT$, and that by Corollary 3.2.13, $Peb_1(G) \in MU$-$HORN$-$UNSAT$ for any DAG G in which every node is essential (i.e. every pebbling of G uses every node in G at some point during the pebbling).

5.2.1 Separation Between Input Resolution & Unit Resolution

In this section we review a previous result relating $IRES$-$UNSAT$ and $URES$-$UNSAT$, give upper bounds on the size of I-RES refutations, and prove a separation between the I-RES and U-RES proof systems.

A very important and useful previous result is the relationship between I-RES and U-RES proved by Chang:

Theorem 5.2.4 ([Cha70]). *A formula F has an* I-RES *refutation if and only if it has a* U-RES *refutation and therefore $IRES$-$UNSAT = URES$-$UNSAT$.*

This equivalence makes it easy to see that I-RES is an incomplete proof system, since there are unsatisfiable formulas which have no unit clauses, and therefore have no U-RES or I-RES refutations.

The following theorem proves upper bounds on the size of I-RES refutations:

Theorem 5.2.5. *For any formula $F \in IRES$-$UNSAT$ there exists an* I-RES *refutation of F with size at most $2n+1$, where n is the number of distinct variables in F.*

Proof: The proof is by induction on n:

Basis: For $n = 0$, $F = \{\emptyset\}$, which has an I-RES refutation containing $2n + 1 = 1$ clauses.

Induction Hypothesis: Suppose that our statement is true for $n - 1$.

Induction Step: We now show that our statement holds for n. Let F be any arbitrary formula in $IRES$-$UNSAT$ with n distinct variables. Since F has a U-RES refutation, it either contains a unit clause (x_i) or $(\neg x_i)$. Suppose that it contains the unit clause (x_i). Restrict F by setting x_i to True to produce the formula $F\lceil_{x_i=True}$, which has $n-1$ variables and also has an I-RES refutation. Our induction hypothesis therefore applies, so $F\lceil_{x_i=True}$ has an I-RES refutation π' of size $\leq 2(n-1)+1 = 2n-1$. Lifting the restriction on F and all corresponding clauses in π' yields an I-RES derivation π of the empty clause or the clause $(\neg x_i)$, where π has size exactly $2n-1$. If π is a derivation of the empty clause, then we are done. In the case where π proves $(\neg x_i)$, recall that F contains the input clause (x_i), so simply resolve with this in order to produce the empty clause. In either case we have therefore produced an I-RES refutation containing $\leq 2n-1+2 = 2n+1$ clauses, as required. The case in which F contains the unit clause $(\neg x_i)$ is completely symmetrical: Simply restrict by $x_i = False$ instead, and then resolve with $(\neg x_i)$ at the end. □

However, although I-RES and U-RES are in some sense equivalent by Theorem 5.2.4 and I-RES has linear size upper bounds by Theorem 5.2.5, the same does not hold for U-RES proof size, and there exists a separation between the two proof systems. In fact, U-RES has $\Omega(n^2)$ size lower bounds on the following family of formulas:

Definition 5.2.6 (**F_U Formulas**). *There is one formula F_i in F_U for each value of $n \geq 1$, where n is the number of distinct variables. F_i is defined inductively as follows: $F_1 = \{\{x_1\}, \{\neg x_1\}\}$, and each subsequent F_i is created by taking F_{i-1}, adding the literal $\neg x_i$ to each clause, and finally adding the singleton clause $\{x_i\}$.*

A simple analysis of these formulas yields a separation between the I-RES and U-RES proof systems:

Theorem 5.2.7. *Each $F_n \in F_U$ requires* U-RES *refutations of size at least $\frac{n^2 + 3n + 2}{2}$, where n is the number of distinct variables in F_n.*

Proof: By construction, it is not hard to see that F_n contains exactly $n + 1$ clauses and that each variable x_i has exactly one positive occurrence, i negative occurrences, and deriving the empty clause requires all variables in the formula to be eliminated. Eliminating each x_i requires exactly i resolutions, although these resolution steps are only possible after the positive unit clause $\{x_i\}$ has been derived, since we are dealing with U-RES. F_n therefore requires $n + (n - 1) + (n - 2) + ... + 1$ applications of the resolution rule, which sums to exactly $\frac{n(n+1)}{2}$ clauses derived. In addition, each of the $n + 1$ initial clauses must be present in the proof. Any refutation of F_n therefore contains at least $\frac{n^2 + 3n + 2}{2}$ clauses, as required. □

5.2.2 The Relationship Between Input Resolution and Horn Formulas

Horn formulas are an important class of formulas which come up rather frequently in computer science. For example, the algorithm employed by the Prolog programming language is called SLD Resolution, which is a special case of Horn Resolution. In this section we shall show that $HORN\text{-}UNSAT$ is a proper subset of $IRES\text{-}UNSAT$:

Lemma 5.2.8. $HORN\text{-}UNSAT \subsetneq IRES\text{-}UNSAT$

Proof: We first show that $HORN\text{-}UNSAT \subset IRES\text{-}UNSAT$ by proving that any unsatisfiable Horn formula F has an I-RES refutation. The proof is by induction on n:

Basis: For $n = 1$, F contains the clauses $\{x_1\}$ and $\{\neg x_1\}$, and therefore clearly has an I-RES refutation.

Induction Hypothesis: Assume that every Horn formula on n variables has an I-RES refutation.

Induction Step: Let F be any Horn formula on $n + 1$ variables, and F' be any minimally unsatisfiable subset of its clauses. If F' contains fewer than $n + 1$ variables, then we appeal to our Induction Hypothesis and are done, so we may assume that F' contains $n + 1$ variables. Any unsatisfiable formula contains at least one clause which has no positive literals in it, or else it could be satisfied by simply setting all variables to True. Similarly, it has at least one clause which has no negative literals in it, or else it could be satisfied by setting all variables to False. However, since F' is a Horn formula and it contains an all-positive clause, it must be a unit clause, so every unsatisfiable Horn formula contains a positive unit clause.

Let x_1 be the variable in a positive unit clause of F'. We restrict F' to create $F' \restriction_{x_1 = True}$, which has only n variables, so our Induction Hypothesis applies, showing that it has an I-RES refutation π'. Since F' is minimally unsatisfiable, lifting the restriction on π' gives us π, which is an I-RES derivation of the clause $\{\neg x_1\}$. We resolve this with our positive unit clause $\{x_1\}$ to complete the refutation of F'. This is also a refutation of F, since we said that $F' \subset F$. Therefore by induction every Horn formula has an I-RES refutation, so $HORN\text{-}UNSAT \subsetneq IRES\text{-}UNSAT$.

It is easy to see that the set inclusion is proper because there are many non-Horn formulas which have I-RES refutations. For instance, the unsatisfiable formula $F = (x \vee y) \wedge (\neg x) \wedge (\neg y)$ is not Horn but has an I-RES refutation. $\square$

This same argument immediately applies to the minimally unsatisfiable versions of these sets as well:

Corollary 5.2.9. *MU-$HORN$-$UNSAT \subsetneq MU$-$IRES$-$UNSAT$*

5.2.3 The Relationship Between Input Resolution and MU Formulas

Having established the relationship between $IRES$-$UNSAT$ and Horn formulas in the previous section, we now investigate the relationship between $IRES$-$UNSAT$ and $MU(k)$. In this section we will prove exact bounds on the size of I-RES proofs, and relate Horn Resolution, I-RES, and $MU(1)$. Finally, we will prove some facts about $MU(1)$, MU-$IRES$-$UNSAT$, and unit propagation which will be useful in later sections.

Exact Bounds on the Size of Input Resolution Refutations

The smallest k for which $MU(k)$ is interesting is $k = 1$. This is because a lower bound on the number of clauses in minimally unsatisfiable formulas is known, showing that the set $MU(0)$ is empty:

Lemma 5.2.10 ([AL86]). *Any minimally unsatisfiable CNF formula on n variables contains at least $n+1$ clauses.*

This lower bound also has a useful corollary:

Corollary 5.2.11. *Any CNF formula with fewer than n clauses is not minimally unsatisfiable, and any unsatisfiable formula F containing exactly $n+1$ clauses is minimally unsatisfiable (i.e. $F \in MU(1)$).*

In addition, Lemma 5.2.10 helps us to prove bounds on the size of refutations for minimally unsatisfiable formulas which hold for all forms of Resolution:

Corollary 5.2.12. *For any minimally unsatisfiable formula F, any refutation of F in any form of Resolution contains at least $2n+1$ clauses.*

Proof: The lower bound is easy to see: Since F is minimally unsatisfiable, each of its clauses must be used in any refutation of F. Therefore by Lemma 5.2.10, any refutation of F contains at least $n+1$ clauses. However, since every variable in F is introduced into the proof at some point and must be eliminated, we need to derive at least n more clauses (one for each variable elimination), bringing our total to at least $2n+1$. $\square$

Together with the upper bound from Theorem 5.2.5, this gives us tight bounds on the size of I-RES refutations for MU-$IRES$-$UNSAT$:

Corollary 5.2.13. *For any formula $F \in MU$-$IRES$-$UNSAT$ the size of every* I-RES *refutation of F is at least $2n+1$, where n is the number of distinct variables in F. In addition, there exists an* I-RES *refutation of F with size at most $2n+1$.*

Proof: The lower bound follows from Corollary 5.2.12, and the upper bound from Theorem 5.2.5. $\square$

Relating Horn Resolution, Input Resolution, and MU(1)

In [DDB98], Davydov, Davydova, and Kleine Büning prove a result corresponding to Lemma 5.2.10, except that theirs is specific to U-RES and also includes an exact count of the number of clauses for formulas in MU-$IRES$-$UNSAT$:

Theorem 5.2.14 ([DDB98]). *Any minimally unsatisfiable formula on n variables which has a* U-RES *refutation contains exactly $n+1$ clauses.*

Together, Theorems 5.2.4 and 5.2.14 allow us to relate $MU\text{-}IRES\text{-}UNSAT$ to $MU(1)$:

Corollary 5.2.15. $MU\text{-}IRES\text{-}UNSAT \subsetneq MU(1)$

Proof: By Theorems 5.2.4 and 5.2.14, $MU\text{-}IRES\text{-}UNSAT \subset MU(1)$. It is easy to see that the set inclusion is proper because there exist formulas such as $F = (x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee z) \wedge (\neg x \vee \neg z)$ in $MU(1)$ which do not have I-RES refutations. $\square$

Combining Lemma 5.2.8, Corollary 5.2.9, and Corollary 5.2.15 allows us to produce a simple set diagram which shows the exact relationship between $IRES\text{-}UNSAT$, $MU(1)$, $MU\text{-}IRES\text{-}UNSAT$, $HORN\text{-}UNSAT$, and $MU\text{-}HORN\text{-}UNSAT$. This diagram is shown below in Figure 5.1.

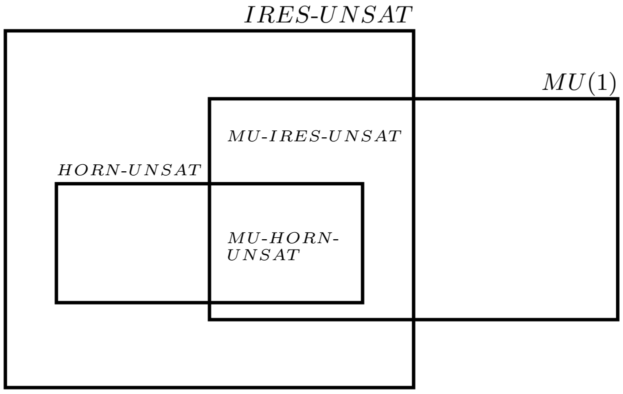

Figure 5.1: The Relationship Between I-RES, Horn Formulas, and MU(1) Formulas

MU(1), MU-IRES-UNSAT, and Unit Propagation

We now prove that both $MU(1)$ and $MU\text{-}IRES\text{-}UNSAT$ are closed under unit propagation, a fact which will be useful in subsequent sections. In order to do this, we first prove the following lemma:

Lemma 5.2.16. *For any CNF formula F, if F contains two clauses C_1 and C_2 such that $C_1 \subset C_2$, then F is not minimally unsatisfiable.*

Proof: Let C_1 and C_2 be clauses of F such that $C_1 \subset C_2$. If F is satisfiable, then it is not minimally unsatisfiable, and we are done, so assume that it is unsatisfiable. Let $F' = F - \{C_2\}$. Because $C_1 \subset C_2$, if F' is satisfiable, then the same truth assignment which satisfies it also satisfies F, so F' is not satisfiable. But F' contains a proper subset of the clauses of F, therefore showing that F is not minimally unsatisfiable. $\square$

We can use this lemma to prove that $MU(1)$ is closed under unit propagation:

Lemma 5.2.17. *For any formula $F \in MU(1)$, if F contains a unit clause $\{l\}$ for some literal l, then $F\restriction_{l=True} \in MU(1)$.*

Proof: If $F \in MU(1)$ contains a unit clause $\{l\}$ for some literal l, then l does not occur anywhere else in F by Lemma 5.2.16. Therefore $F\restriction_{l=True}$ contains $n-1$ variables and exactly n clauses, which means that it is minimally unsatisfiable by Corollary 5.2.11, so $F\restriction_{l=True} \in MU(1)$, as required. $\square$

Combining the previous lemma with Corollary 5.2.15 yields the following corollary which shows that $MU\text{-}IRES\text{-}UNSAT$ is closed under unit propagation:

Corollary 5.2.18. *For any formula $F \in MU\text{-}IRES\text{-}UNSAT$, if F contains a unit clause $\{l\}$ for some literal l, then $F\restriction_{l=True} \in MU\text{-}IRES\text{-}UNSAT$.*

5.2.4 A Matrix Characterization of MU-IRES-UNSAT

Although matrix algebra and propositional formulas may at first glance have very little to do with each other, matrices can be used to encode formulas. This can be done by giving each cell in the matrix one of three values: $+, -$, and 0 such that each row in the matrix represents a distinct variable, and each column represents a distinct clause. If column j of row i is $+$, then the literal x_i occurs in clause j. Similarly, if column j in row i is $-$, then the literal $\neg x_i$ occurs in clause j. Finally, if position (i, j) in the matrix is 0, then the variable i does not occur in clause j.

For example, the formula $F = (x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ has the following matrix representation:

$$\begin{pmatrix} + & - & 0 \\ 0 & + & - \\ 0 & 0 & + \end{pmatrix}$$

In [DDB98] the authors show that the 'basic matrices', which are of interest in the area of linear algebra exactly capture the formulas in $MU(1)$. From Corollary 5.2.15, we know that $MU\text{-}IRES\text{-}UNSAT$ is a proper subset of $MU(1)$, so it stands to reason that $MU\text{-}IRES\text{-}UNSAT$ comprises a proper subset of the basic matrices. In this section we shall show that this is in fact the case. We begin by defining this subset of matrices, which we call the 'sub-basic matrices':

Definition 5.2.19 (Sub-Basic Matrices). *The sub-basic matrices are defined inductively:*

1. $\begin{pmatrix} + & - \end{pmatrix}$ *is a sub-basic matrix.*
2. *If B is a sub-basic matrix and b is a vector in which each $b_j \in \{-, 0\}$ and b contains at least one $-$-sign, then the following matrix is also sub-basic:*

$$\begin{pmatrix} + & b \\ 0 & B \end{pmatrix}$$

3. *If B is a sub-basic matrix and b is a vector in which each $b_j \in \{+, 0\}$ and b contains at least one $+$-sign, then the following matrix is also sub-basic:*

$$\begin{pmatrix} - & b \\ 0 & B \end{pmatrix}$$

Having defined the sub-basic matrices, we now show that a formula is in $MU\text{-}IRES\text{-}UNSAT$ if and only if it can be encoded by a sub-basic matrix. We begin by proving the forward direction:

Lemma 5.2.20. *Any formula $F \in MU\text{-}IRES\text{-}UNSAT$ can be represented as a sub-basic matrix.*

Proof: The proof is by induction on n, the number of distinct variables in F:

Basis: For $n = 1$, $F = (x_1) \wedge (\neg x_1)$, and $\begin{pmatrix} + & - \end{pmatrix}$ is sub-basic, so our statement holds for the base case.

Induction Hypothesis: Suppose that any formula $F \in MU\text{-}IRES\text{-}UNSAT$ containing n variables can be represented as a sub-basic matrix.

Induction Step: Let F be any arbitrary formula in $MU\text{-}IRES\text{-}UNSAT$ containing $n+1$ variables. Since $F \in MU\text{-}IRES\text{-}UNSAT$, it contains a unit clause. Without loss of generality, let us call this unit clause $\{l_1\}$, where $l_1 = x_1$ or $\neg x_1$. Let F' be $F\lceil_{l_1=True}$, which has n variables and by Corollary 5.2.18 is in $MU\text{-}IRES\text{-}UNSAT$. Therefore our induction hypothesis applies, and F' has the sub-basic matrix B. By Lemma 5.2.16, l_1 does not appear in any other clause of F, so if $l_1 = x_1$, then F has the sub-basic matrix

$\begin{pmatrix} + & b \\ 0 & B \end{pmatrix}$, where b is a vector in which each $b_j \in \{-, 0\}$ and b contains at least one $-$-sign (because F is minimally unsatisfiable). The case where $l_1 = \neg x_1$ is completely symmetrical, so in either case F has a sub-basic matrix, and by induction any formula $F \in$ *MU-IRES-UNSAT* can be represented as a sub-basic matrix. □

We now prove the reverse direction:

Lemma 5.2.21. *Every sub-basic matrix encodes a formula $F \in$ MU-IRES-UNSAT.*

Proof: The proof is by induction on n, the number of rows of a sub-basic matrix:

Basis: For $n = 1$, the only sub-basic matrix is $\begin{pmatrix} + & - \end{pmatrix}$, corresponding to $F = (x_1) \wedge (\neg x_1)$, which is in *MU-IRES-UNSAT*.

Induction Hypothesis: Suppose that every sub-basic matrix containing n rows encodes a formula $F \in$ *MU-IRES-UNSAT*.

Induction Step: Let M by any arbitrary sub-basic matrix containing $n + 1$ rows. By definition, M is $\begin{pmatrix} + & b \\ 0 & B \end{pmatrix}$, where b is a vector in which each $b_j \in \{-, 0\}$ and b contains at least one $-$-sign, or M is $\begin{pmatrix} - & b \\ 0 & B \end{pmatrix}$, where b is a vector in which each $b_j \in \{+, 0\}$ and b contains at least one $+$-sign. In either case, B is sub-basic and contains n variables, so our induction hypothesis applies, and it encodes a formula $F_B \in$ *MU-IRES-UNSAT*. In our first case, we add the singleton clause $\{x_1\}$ to F_B and add the literal $\neg x_1$ to at least one clause of F_B to produce F, which is clearly still unsatisfiable and has a U-RES refutation, so therefore also has an I-RES refutation by Theorem 5.2.4. However, F contains $n + 1$ variables and $n + 2$ clauses, so by Corollary 5.2.11 it is minimally unsatisfiable.

Our second case where we add the singleton clause $\{\neg x_1\}$ to F_B and add the literal x_1 to at least one clause of F_B to produce F is completely symmetrical. Therefore, in either case M encodes a formula $F \in$ *MU-IRES-UNSAT*, so by induction every sub-basic matrix encodes a formula $F \in$ *MU-IRES-UNSAT*. □

Putting these two lemmas together shows that the sub-basic matrices are a perfect characterization of *MU-IRES-UNSAT*:

Theorem 5.2.22. *A formula F is in MU-IRES-UNSAT if and only if it can be encoded as a sub-basic matrix.*

This result of course has implications for *IRES-UNSAT* as well, because it shows that any minimally unsatisfiable subset of clauses from formula which has an I-RES refutation can be encoded as a sub-basic matrix. In other words, any formula $F \in$ *IRES-UNSAT* can be encoded as a sub basic matrix which has been augmented with any arbitrary clauses, giving us a matrix characterization of *IRES-UNSAT* as well.

5.3 Tractable Aspects of Input Resolution

In the previous section we related various languages such as *HORN-UNSAT*, $MU(1)$, and *MU-IRES-UNSAT* to *IRES-UNSAT*. We shall use these results to explore the complexities of some of the tractable aspects of the I-RES proof system. First we shall review some previous results showing that *HORN-UNSAT* and *IRES-UNSAT* are $\mathcal{P}$-Complete, and that $MU(1) \in \mathcal{P}$. Following this we show that *MU-IRES-UNSAT* $\in \mathcal{P}$ and that the size problem for *MU-IRES-UNSAT* is in $\mathcal{P}$.

5.3.1 The Complexities of HORN-UNSAT, IRES-UNSAT, and MU(1)

We now review some previous results about the tractability of $HORN$-$UNSAT$, $IRES$-$UNSAT$, and $MU(1)$. For example, the complexity of $HORN$-$UNSAT$ is well-understood:

Theorem 5.3.1 ([Pla84], [Pap94, p.176]). *The problem of determining whether or not a given Horn formula is satisfiable (i.e. of deciding the language $HORN$-SAT) is $\mathcal{P}$-Complete.*

This shows that determining whether or not Horn formulas are unsatisfiable is co$\mathcal{P}$-Complete, so since $\mathcal{P}$ is closed under complement, this immediately gives us the complexity of $HORN$-$UNSAT$:

Corollary 5.3.2. *$HORN$-$UNSAT$ is $\mathcal{P}$-Complete.*

Similar $\mathcal{P}$-Completeness results were proved for $URES$-$UNSAT$ by Jones and Laaser:

Theorem 5.3.3 ([JL77]). *The problem of determining whether or not a given formula has a* U-RES *refutation (i.e. of deciding the language $URES$-$UNSAT$) is $\mathcal{P}$-Complete.*

Since we know from Theorem 5.2.4 that $IRES$-$UNSAT = URES$-$UNSAT$, this immediately implies the $\mathcal{P}$-Completeness of $IRES$-$UNSAT$:

Corollary 5.3.4. *$IRES$-$UNSAT$ is $\mathcal{P}$-Complete, and furthermore there exists an $O(n \cdot m)$ algorithm which takes as input a formula F and determines whether or not it has an* U-RES *(and therefore* I-RES*) refutation, where n is the number of distinct variables in F, and m is the number of clauses.*

Proof: Showing that $IRES$-$UNSAT$ is $\mathcal{P}$-Complete is trivial, since $IRES$-$UNSAT = URES$-$UNSAT$, which is $\mathcal{P}$-Complete by Theorem 5.3.3. It can be solved by the following unit propagation algorithm: Repeatedly pick a unit clause and resolve it with every other clause possible. Since U-RES is closed under restriction, this algorithm is clearly correct, will take at most $O(n \cdot m)$ time in total, and will either yield the empty clause, in which case it has a U-RES refutation (and therefore I-RES refutation by Theorem 5.2.4), or a formula where no more unit resolutions are possible, in which case it does not.

This also solves the complexity of the clause space problem for $IRES$-$UNSAT$, which is formally defined as follows:

Definition 5.3.5 ($ICSP$). $ICSP = \{(F, k) \mid$ *F is a formula for which there exists a* I-RES *refutation with clause space at most k.*$\}$

Corollary 5.3.6. *$ICSP$ is $\mathcal{P}$-Complete.*

Proof: By the very nature of I-RES, a formula has an I-RES refutation if and only if it has an I-RES refutation of clause space at most 2. Therefore we use the same algorithm as in Corollary 5.3.4, but first check to see that $k \geq 2$. Our reduction is from $IRES$-$UNSAT$, and similarly proceeds by outputting $(F, k = 2)$. □

In addition, the complexity of $MU(k)$ has been settled for $k = 1$:

Theorem 5.3.7 ([DDB98]). *$MU(1) \in \mathcal{P}$ and there exists an $O(n^2)$ algorithm which decides if any arbitrary formula F is in $MU(1)$, where n is the number of distinct variables in F.*

The complexity of $MU(k)$ for any constant $k \geq 2$ remains open.

5.3.2 The Tractability of MU-IRES-UNSAT

Having proved that *IRES-UNSAT* is $\mathcal{P}$-Complete, we now show that *MU-IRES-UNSAT* $\in \mathcal{P}$ by counting clauses:

Corollary 5.3.8. *MU-IRES-UNSAT* $\in \mathcal{P}$ *and has an* $O(n{\cdot}m)$ *algorithm, where* n *is the number of distinct variables and* m *is the number of clauses.*

Proof: Given a formula F, we determine if $F \in$ *MU-IRES-UNSAT*. By Corollary 5.2.11 we know that any formula containing fewer than n variables is not minimally unsatisfiable, so if this is the case, then reject. Similarly, by Corollary 5.2.15, any formula containing more than $n+1$ variables cannot be in *MU-IRES-UNSAT*, so if this is the case, then reject. Therefore we are left with the case where F contains exactly $n+1$ clauses, so it is either has an I-RES refutation (in which case it is in *MU-IRES-UNSAT*) or it does not (in which case it is not). This can be determined using the $O(n \cdot m)$ algorithm from Corollary 5.3.4. □

5.3.3 The Tractability of the MU-IRES-UNSAT Size Problem

The size problem for *MU-IRES-UNSAT* takes as input a formula / integer pair (F, k) and asks if F is minimally unsatisfiable and has an I-RES refutation of size at most k. We now show that this problem is in $\mathcal{P}$:

Corollary 5.3.9. *Given a formula* F *and integer* k*, the problem of determining whether or not* $F \in$ *MU-IRES-UNSAT and has an* I-RES *refutation of size at most* k *is in* $\mathcal{P}$ *and has an* $O(n \cdot m)$ *algorithm, where* n *is the number of distinct variables in* F*, and* m *is the number of clauses.*

Proof: Simply use the algorithm from Corollary 5.3.8 to determine if $F \in$ *MU-IRES-UNSAT*. If not, then reject, and otherwise compare k with $2n+1$. If $k \geq 2n+1$, then by Corollary 5.2.13 we can accept, and otherwise we reject. □

5.3.4 The Tractability of the MU-IRES-UNSAT Problem with Top Clause

In previous sections we have seen that various versions of the *MU-IRES-UNSAT* problem are in $\mathcal{P}$. We now prove that another generalized form of the problem is also tractable. Instead of asking whether a minimally unsatisfiable formula F has an I-RES refutation, we can ask if it has an I-RES refutation in which one of the top clauses is C. We shall refer to this as the *MU-IRES-UNSAT* problem with top clause. In order to prove that it is in $\mathcal{P}$, we will first prove a 'top clause' lemma:

Lemma 5.3.10 (Top Clause Lemma). *If a formula* $F \in MU(1)$ *has an* I-RES *refutation, then it has an* I-RES *refutation with any arbitrary clause* $C \in F$ *as one of the top clauses.*

Proof: By induction on n, the number of distinct variables in F:

Basis: In order to simplify later arguments, we will cover the cases of $n=0$ and $n=1$ as our base cases. For $n=0$, $F=\emptyset$, which trivially unsatisfiable, and its only proof contains only the empty clause. For $n=1$, the only minimally unsatisfiable formula is $F=(x)\wedge(\neg x)$, in which the only refutation has both of its clauses as top clauses, so our statement holds for both $n=0$ and $n=1$.

Induction Hypothesis: Suppose that our statement is true for $n-1$.

Induction Step: We now show that our statement holds for n. Let F be any arbitrary formula in $MU(1)$ on n variables which has an I-RES refutation. By Theorem 5.2.4, F must contain a unit clause, call it $\{l\}$ for some literal l. We want to show that F has an I-RES refutation with top clause C, where C is any arbitrary clause in F. In order to apply our induction hypothesis, we restrict F to produce $F' = F\restriction_{l=True}$. By Lemma 5.2.17, $F\restriction_{l=True} \in MU(1)$. In addition, I-RES is closed under restriction, so our induction hypothesis applies to F', which therefore has an I-RES refutation with any arbitrary $C\lceil_{x=True} \in F'$ as top clause; let us call this proof π'.

There are two cases to consider:

Case 1: Suppose that $C \neq \{l\}$. Since $F \in MU(1)$ and $\{l\} \in F$ is a unit clause, Lemma 5.2.16 applies, so the literal l does not occur anywhere else in F, including C. Similarly, $C \neq \{\neg l\}$, since this is not the base case. Therefore the restricted clause $C\lceil_{l=True}$ does not disappear from F'. Furthermore, since F' is minimally unsatisfiable, every clause is used in π', so lifting the restriction of $l = True$ yields π which is an I-RES derivation from F of $\{\neg l\}$ with top clause C. We resolve the final clause $\{\neg l\}$ of π with our unit input clause $\{l\}$ to complete the refutation.

Case 2: Suppose that $C = \{l\}$. We know that F is minimally unsatisfiable, so it must contain a clause of the form $D = E \cup \{\neg l\}$, and $n \geq 2$, so $E \in F'$. By our induction hypothesis, there exists an I-RES refutation π' from F' with top clause E. Lift the restriction of $l = True$ from all clauses of π', but do not add $\{\neg l\}$ back to the top clause E. Now resolve $\{l\}$ with $D = E \cup \{\neg x\}$ at the top of π' to produce E, and if necessary, also resolve $\{l\}$ with the singleton clause $\{\neg l\}$ (if it exists) at the bottom of the proof to complete the refutation. This yields an I-RES refutation from F with top clause $C = \{l\}$.

Therefore, in either case we have produced an I-RES refutation from F with top clause C, as required, and our result follows by induction. □

This lemma allows us to prove that the *MU-IRES-UNSAT* problem with top clause is in $\mathcal{P}$:

Theorem 5.3.11. *Given a minimally unsatisfiable formula F and a clause $C \in F$, determining if F has an* I-RES *refutation with top clause C is in $\mathcal{P}$.*

Proof: To show that this problem is in $\mathcal{P}$, we first determine if $F \in$ *MU-IRES-UNSAT*, which can be done in polynomial time by Corollary 5.3.8. If not, then reject. Otherwise, by Lemma 5.3.10 F has an I-RES refutation with C as top clause, so accept. □

5.4 The Automatizability of Input Resolution

In this section we show that I-RES is automatizable and that *MU-IRES-UNSAT* is optimally automatizable. These results form an interesting contrast with the results in the next section which show that the problem of computing the optimum size of I-RES refutations is $\mathcal{NP}$-Complete.

Informally, a proof system S is automatizable if there exists an algorithm which can always find proofs which are only polynomially larger than the optimal. Such algorithms are obviously of great practical interest because they allow us to efficiently automate theorem proving. For a more formal description, please refer to Definition 3.2.16.

Because they form the basis of so many practical SAT-solving algorithms, RES and its refinement T-RES are some of the best candidate proof systems which researchers would like to automatize. However, under widely-believed cryptographic assumptions, they are not automatizable. Please refer to section 3.2.6 for more information on the relationship between cryptographic assumptions and automatizability.

It is therefore likely that the only possible automatizable Resolution refinements are incomplete. In other words, results such as the ones in this section are in a sense the best we can hope for.

The most obvious candidate for automatizability is Horn Resolution. Our results from previous sections make it easy to see that Horn formulas are an automatizable example of restricted SAT inputs:

Corollary 5.4.1. I-RES *on HORN-UNSAT is automatizable and there exists an $O(n \cdot m)$ algorithm which takes as input any Horn formula F (where n is the number of distinct variables in F and m is the number of clauses), and either produces an* I-RES *refutation containing at most $2n + 1$ clauses, or returns that no refutation exists.*

Proof: We simply use the $O(n \cdot m)$ algorithm from Corollary 5.3.4 to determine if F is unsatisfiable. If not, then return that no refutation exists. Otherwise use the $O(n \cdot m)$ algorithm implicit in Lemma 5.2.8 to produce the desired I-RES refutation containing at most $2n + 1$ clauses. □

Another restriction which has received much attention is 2-SAT, in which we require the input formula to have exactly two literals per clause. Unlike Horn-SAT, which is $\mathcal{P}$-Complete, 2-SAT is $\mathcal{NL}$-Complete [Pap94, p.398]. In addition, 2-SAT is also known to be automatizable [Coo71], and in fact optimal polytime automatizability algorithms exist for finding the shortest T-RES refutation [Sub04] and the shortest RES refutation of any unsatisfiable 2-CNF formula [BOM07].

We now prove that for I-RES there exists a polytime algorithm for finding a refutation which is at worst linearly longer than the optimal, thereby showing its automatizability. In fact, I-RES is strongly automatizable in the sense that we can always find a refutation which is polynomial in the length of the input formula. Automatizability with respect to formula size is usually not even a reasonable thing to ask for because many proof systems have exponential size lower bounds, making this impossible. This further shows just how tractable the I-RES proof system is.

Theorem 5.4.2. *The* I-RES *proof system is automatizable. More specifically, given a formula F containing n distinct variables and m clauses, there exists an $O(n \cdot m)$ algorithm which finds an* I-RES *refutation containing at most $2n + 1$ clauses or reports that F has no* I-RES *refutation.*

Proof: First use the $O(n \cdot m)$ algorithm from Corollary 5.3.4 to determine if F has an I-RES refutation. If not, then we report that none exists. Next we apply the $O(n \cdot m)$ DPLL algorithm implicit in the induction step of Theorem 5.2.5 to produce an I-RES refutation of F containing at most $2n + 1$ clauses. □

It is worth noting that for any formula in $IRES\text{-}UNSAT$, this algorithm is guaranteed to produce a proof which is larger than the smallest possible by at most a linear factor. However, by Corollary 5.2.12 minimally unsatisfiable formulas require refutations containing at least $2n + 1$ clauses, so this algorithm produces the shortest possible proofs for any formula in $MU\text{-}IRES\text{-}UNSAT$:

Corollary 5.4.3. *The* I-RES *proof system is optimally automatizable on minimally unsatisfiable formulas.*

5.5 The NP-Completeness of Input Resolution Size

In previous sections we saw that I-RES is automatizable, $IRES\text{-}UNSAT$ is $\mathcal{P}$-Complete, that I-RES is optimally automatizable on $MU\text{-}IRES\text{-}UNSAT$, and the size problem for $MU\text{-}IRES\text{-}UNSAT$ is in $\mathcal{P}$. However, in this section we show an interesting contrast with these tractability results by noting that the size problem for $IRES\text{-}UNSAT$ is $\mathcal{NP}$-Complete. This result follows immediately from an interesting result proved by Alekhnovich, Buss, Moran and Pitassi [ABMP01]:

Theorem 5.5.1 ([ABMP01]). *The problem of approximating the size (regardless of whether size is measured in total symbols or number of clauses) of the smallest Resolution refutations of formulas from $HORN$-$UNSAT$ to within a factor of $2^{log^{1-o(1)} n}$ is $\mathcal{NP}$-Hard.*

Corollary 5.5.2. *The problem of approximating the size (regardless of whether size is measured in total symbols or number of clauses) of the smallest* I-RES *or* U-RES *proofs of formulas from $IRES\text{-}UNSAT$ to within a factor of $2^{log^{1-o(1)} n}$ is $\mathcal{NP}$-Hard.*

Proof: There is no need to change the proof in [ABMP01]; it is easy to see that the reduction from Circuit MMSA produces Horn formulas which have both I-RES and U-RES refutations, so the result holds for these proof systems as well. □

This corollary in turn yields another one, namely that given a formula F and integer k, the problem of determining if F has an I-RES refutation of size at most k is $\mathcal{NP}$-Complete. The language associated with this problem is defined as follows:

Definition 5.5.3 (*IRES-SIZE*). *$IRES\text{-}SIZE = \{(F,k) \mid F$ is a formula for which there exists a* I-RES *refutation with size at most k.$\}$*

Corollary 5.5.4. *IRES-SIZE is $\mathcal{NP}$-Complete under Turing reducibility.*

Proof: We first show that *IRES-SIZE* is in $\mathcal{NP}$: *IRES-UNSAT* is in $\mathcal{P}$ by Corollary 5.3.4 and is therefore also in $\mathcal{NP}$, so we first check to see if F has an I-RES refutation. If it does not, then we reject. Otherwise it has an I-RES refutation of size at most $2n+1$ by Theorem 5.4.2. We therefore compare k with $2n+1$ and if k is greater, then we accept immediately. Otherwise we nondeterministically guess the shortest I-RES refutation of F, which we know must have a size of at most $2n+1$ and accept if and only if its size is at most k, thereby completing our $\mathcal{NP}$ algorithm.

Next we show that *IRES-SIZE* is $\mathcal{NP}$-Hard via a Turing reduction from the problem of approximating minimum I-RES refutation size which was proved $\mathcal{NP}$-hard in Corollary 5.5.2. Assuming that we have an algorithm for *IRES-SIZE*, we can use it as a subroutine to create an algorithm for determining the size of the smallest refutation of F as follows: By Theorem 5.4.2, F has an I-RES refutation containing at most $2n+1$ clauses. We therefore perform a binary search between the range 1 and $2n+1$ using our *IRES-SIZE* algorithm to determine the size of the smallest refutation. Since this is only a logarithmic number of subroutine calls, it follows that *IRES-SIZE* is $\mathcal{NP}$-Complete, as required. □

5.6 The PSPACE-Completeness of Input Resolution Derivation Total Space

This section contains a number of results pertaining to black pebbling and the total space of I-RES. In Section 5.6.1 we prove that for any DAG G, $B\text{-}Peb(G)$ is equivalent to the total space of any I-RES-W$^-$ derivation of a certain variation of $Peb_1(G)$ formulas. Following that we show how to dispense with weakening and prove the equivalent result for I-RES. In Section 5.6.2 we put these equivalence results to use in order to prove the $\mathcal{PSPACE}$-Completeness of various forms of the I-RES total space problem. Finally, in Section 5.7.2 we prove another corollary to our equivalence results, namely an optimal I-RES size / total space tradeoff.

5.6.1 Equivalence of Black Pebbling & Input Resolution Total Space

We shall prove that for any DAG G, $B\text{-}Peb(G)$ (with sliding) is almost exactly equivalent to the minimum total space of any I-RES-W$^-$ derivation from $Peb_1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$.

In order to proceed, we must first define the I-RES-W$^-$ proof system, which is simply I-RES with an added weakening rule, as well as the $Peb_1(G)^*$ formulas, which are a slight modification of the $Peb_1(G)$ formulas:

Definition 5.6.1 (I-RES-W$^-$). I-RES-W$^-$ *is identical to the* I-RES *proof system from Definition 3.2.3 except that it has the following additional rule:*

4. replacing one of the clauses $C \in \mathbb{C}_i$ with the clause $C \cup \{\neg x\}$, which was obtained by weakening C with an arbitrary negative literal $\neg x$.

The $Peb_1(G)^*$ formulas, closely related to the $Peb_1(G)$ formulas from Definition 3.2.11, are defined as follows:

Definition 5.6.2 ($Peb_1(G)^*$). *In the case of binary DAGs without OR gates, in order to ensure that our resulting formula is a 3-CNF formula, we define $Peb_1(G)^*$ to be just like $Peb_1(G)$ except that we include dummy literals $\neg\alpha$ and $\neg\beta$ in each singleton clause so that each source clause $\{s\}$ becomes $\{s, \neg\alpha, \neg\beta\}$ and* *the target clause becomes* $\{\neg t, \neg\alpha, \neg\beta\}$. *Note that there are no positive instances of* α *or* β, *so a proof of* $Peb_1(G) \vdash \emptyset$ *will correspond exactly with a proof of* $Peb_1(G)^* \vdash \{\neg\alpha, \neg\beta\}$.

We make use of the $Peb_1(G)^*$ because unlike the $Peb_1(G)$ formulas, they are in 3-CNF.

We shall first show that the equivalence between pebbling number and I-RES space holds for I-RES-W$^-$, and then we will show that it also holds for standard I-RES by proving that the weakening rule is dispensable. In order to prove these results, we first need to define the concept of an 'I-RES Spine', which is very similar to the concept of a one-colour pebbling history from Definition 3.2.10:

Definition 5.6.3 (I-RES Spine). *Let* T *be the tree underlying an* I-RES *refutation* π *of a formula* F. *The spine* B *of* π *is the linear portion of* T *starting at* π*'s top clause* C_0 *and ending at the goal clause* C_k, *with all of the remaining clauses (which are all input clauses) removed from* T. B *can therefore be written as a sequence of clauses* $B = B_0, B_1, ..., B_k$. *Depending on our application we may consider* B_0 *to be the top clause and* B_k *to be the goal clause, or the other way around.*

The Equivalence of Black Pebbling & Input Total Space With Weakening

Our first lemma proves the forward direction of our equivalence and states that it is possible to take a pure black pebbling strategy of an arbitrary DAG G and from it build an I-RES-W$^-$ derivation from $Peb_1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which the spine perfectly encodes the strategy.

The intuition behind this translation is illustrated by the following example: Consider the pyramid graph G shown below in Figure 5.2. We will use the labels on its nodes to correspond to the variable names in its pebbling formula, so $Peb_1(G)^* = (4 \vee \neg\alpha \vee \neg\beta) \wedge (5 \vee \neg\alpha \vee \neg\beta) \wedge (6 \vee \neg\alpha \vee \neg\beta) \wedge (\neg 4 \vee \neg 5 \vee 2) \wedge (\neg 5 \vee \neg 6 \vee 3) \wedge (\neg 2 \vee \neg 3 \vee 1) \wedge (\neg 1 \vee \neg\alpha \vee \neg\beta)$.

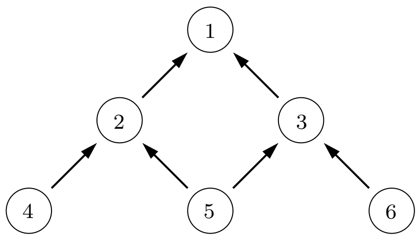

Figure 5.2: An Example of a Pyramid Graph; The Target Node is Vertex 1.

Figure 5.3 below shows how to translate the pure black pebbling strategy $S_0, S_1, ..., S_8$ for G into an I-RES-W$^-$ derivation of from $Peb_1(G)^*$ with top clause $\{\neg 1, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which the spine perfectly encodes the strategy:

More formally, translating a pebbling strategy into an I-RES-W$^-$ proof can be carried out according to the following lemma:

Lemma 5.6.4. *For any DAG* G *with target node* t, *if* G *has a pure black* k*-pebbling strategy (using sliding)* $S = S_0, S_1, S_2..., S_{j-1}, S_j$ *where* $t \in S_j$, *then* $Peb_1(G)^*$ *has an* I-RES-W$^-$ *derivation* π *with top clause* $\{\neg t, \neg\alpha, \neg\beta\}$ *and goal clause* $\{\neg\alpha, \neg\beta\}$ *in which* π *has a spine* $B = B_0, B_1, ..., B_{j-1}, B_j$ *(where* B_j *is the top clause and* B_0 *is the goal clause) such that for all* $0 \leq i \leq j$, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$.

Proof: Let G be any arbitrary DAG with target node t and let $S = S_0, S_1, S_2..., S_{j-1}, S_j$ be any pure black k-pebbling strategy (using sliding) of G. Note that $S_0 = \emptyset$ and $t \in S_j$. We will show that there is a corresponding I-RES-W$^-$ derivation with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ by induction on the number of steps in the pebbling strategy S. It is useful to picture the pebbling strategy as a linear sequence of sets drawn with the first step at the bottom, and the last step at the top. We will construct

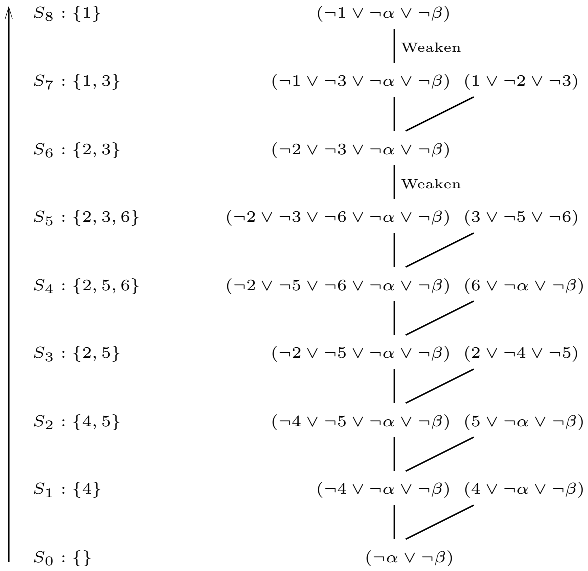

Figure 5.3: A Pure-Black Pebbling Strategy for G (Left) and its Corresponding $\textsf{I-RES-W}^-$ Refutation (Right)

the $\textsf{I-RES-W}^-$ proof and spine from the bottom (goal clause) to the top (top clause). Note that because the dummy literals $\neg\alpha$ and $\neg\beta$ are present in every clause of the proof spine, each spine clause has a width that is two greater than the corresponding step in the pebbling strategy.

Basis: Step 0 of the pebbling strategy contains no pebbles. Therefore $S_0 = \emptyset$. The corresponding clause in our spine is the clause that we are trying to derive. Therefore $B_0 = \{\neg\alpha, \neg\beta\}$. Since $\bigcup_{v \in S_0} \{\neg v\} = \emptyset$, it is clear that $B_0 = \bigcup_{v \in S_0} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$.

Induction Hypothesis: Suppose that we have been able to translate our pebbling strategy up to and including step i to $\textsf{I-RES}$ proof steps in which the spine clauses encode the pebbling strategy. More specifically, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, holds for step S_i.

Induction Step: We now show how to translate step $i+1$ of our pebbling strategy into a corresponding spine clause. More specifically, we need to show that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$ holds for S_{i+1}, step $i+1$ in our pebbling strategy. Pebbling step S_{i+1} could have come from step S_i in one of exactly three ways:

1. By pebbling a source node s,
2. By removing a pebble from vertex u, or
3. If nodes a and b are pebbled predecessors of c, by sliding one of the pebbles from a or b to c.

Case 1: In this case, $S_{i+1} = S_i \cup \{s\}$. Let $B_{i+1} = B_i \cup \{\neg s\}$. Then we can derive B_i from B_{i+1} by resolving B_{i+1} with the input clause $\{s, \neg\alpha, \neg\beta\}$, so this is a valid resolution step resolving on the variable s. By our induction hypothesis, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, but $B_{i+1} = B_i \cup \{\neg s\}$, so $B_{i+1} = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\} \cup \{\neg s\}$. Since $S_{i+1} = S_i \cup \{s\}$, we know that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, as required.

Case 2: In this case, $S_{i+1} = S_i - \{u\}$. Let $B_{i+1} = B_i - \{\neg u\}$. Then we can derive B_i from B_{i+1} by weakening B_{i+1} to introduce $\{\neg u\}$, so this is a valid resolution step. By our induction hypothesis, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, but $B_{i+1} = B_i - \{\neg u\}$, so $B_{i+1} = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\} - \{\neg u\}$. Since

$S_{i+1} = S_i - \{u\}$, we know that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, as required.

Case 3: Without loss of generality, assume that we are sliding the pebble from a to c. In this case, $S_{i+1} = S_i - \{a\} \cup \{c\}$. Let $B_{i+1} = B_i - \{\neg a\} \cup \{\neg c\}$. Then we can derive B_i from B_{i+1} by resolving B_{i+1} on variable a with the input clause $\{\neg a, \neg b, c\}$, so this is a valid I-RES-W$^-$ step resolving on the variable c. By our induction hypothesis, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, but $B_{i+1} = B_i - \{\neg a\} \cup \{\neg c\}$, so $B_{i+1} = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\} - \{\neg a\} \cup \{\neg c\}$. Since $S_{i+1} = S_i - \{a\} \cup \{c\}$, we know that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, as required.

Therefore, in all cases, there exists a valid next step in the spine such that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, so by induction we can translate a pure black pebbling into an I-RES-W$^-$ proof in which the spine perfectly encodes the pebbling strategy. In order to ensure that the I-RES-W$^-$ proof has top clause $\{\neg t, \neg\alpha, \neg\beta\}$, we may have to apply several weakenings in reverse. This is because the pebbling strategy may end with more than just the target node pebbled, in which case we would have to remove the literals corresponding to these superfluous pebbles in order to ensure that the top clause is indeed at the top of our I-RES-W$^-$ proof. □

Our next lemma shows the opposite direction, namely that it is possible to take any arbitrary I-RES-W$^-$ derivation π and translate it into a pebbling strategy which uses at most two fewer pebbles than π's spine width.

Once again, we use the pyramid graph G from Figure 5.2 above as our example. Figure 5.4 below shows how to translate the I-RES-W$^-$ derivation of from $Peb_1(G)^*$ with top clause $\{\neg 1, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ into a pure white pebbling strategy $S_0, S_1, ..., S_{10}$ in which the strategy perfectly encodes the spine of the proof. In fact, weakening is never required in this example, so it actually translates an I-RES derivation to a pebbling strategy. Note how each resolution step may require up to two corresponding pebbling moves.

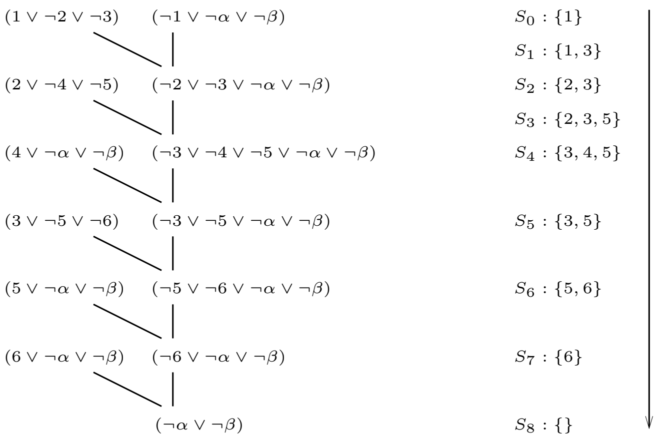

Figure 5.4: An I-RES-W$^-$ Derivation of $\{\neg\alpha, \neg\beta\}$ from $Peb_1(G)^*$ (Left) and its Corresponding Pure White Pebbling Strategy (Right)

More formally, translating an I-RES-W$^-$ proof into a pebbling strategy can be carried out according to the following lemma:

Lemma 5.6.5. *For any DAG G with target node t, if $Peb_1(G)^*$ has an* I-RES-W$^-$ *derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has a spine $B = B_0, B_1, ..., B_j$ (where B_0 is the top clause and B_j is the goal clause), then there exists a pure white pebbling strategy (using sliding) $S = S_0, S_1, S_2 ..., S_l$ where l is $\leq 2j$ with $S_0 = \{t\}$ and $S_l = \emptyset$ such that it is possible to translate each B_i into either one pebbling* *step S_q such that $S_q = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$, or to translate B_i into two consecutive pebbling steps S_{q_1}, S_{q_2} such that $|S_{q_1}| \leq |B_i| - 2$ and $S_{q_2} = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$.*

Proof: Let G be any arbitrary DAG with target node t and let π be an I-RES-W$^-$ derivation with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has a spine of width $k+2$. We will show how to take this spine and use it to create a pure white pebbling strategy which uses at most k pebbles. It is useful to picture π with the top clause at the top and the goal clause at the bottom. We shall move down the spine of π and show how to build a corresponding pure white pebbling strategy. Note that every clause in the spine contains the dummy literals $\neg\alpha$ and $\neg\beta$ and that this accounts for the '+ 2' in $k+2$.

Basis: Consider the top clause of π, $B_0 = \{\neg t, \neg\alpha, \neg\beta\}$; we create the equivalent first step of a pebbling by placing a white pebble on node t to create pebbling step $S_0 = \{t\}$. Therefore $S_0 = \bigcup_{\neg v \in B_0} \{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Induction Hypothesis: Suppose that we have been able to translate our spine into a pebbling strategy up to and including spine clause B_i and that B_i was translated to S_p such that $S_p = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$.

Induction Step: We will now show how to translate spine clause B_{i+1} to create either one or two corresponding pebbling steps. Spine clause B_{i+1} could have come from B_i in one of exactly three ways:

1. By resolving B_i with a source clause $\{u, \neg\alpha, \neg\beta\}$ (resolving on variable u),
2. By resolving B_i with a propagation clause $\{\neg a, \neg b, c\}$ (resolving on variable c), or
3. By weakening B_i to introduce a negative literal $\neg u$.

Case 1: By our induction hypothesis, we have translated spine clause B_i into pebbling step S_p such that $S_p = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$. We now resolve B_i with the input clause $\{u, \neg\alpha, \neg\beta\}$ on variable u to create $B_{i+1} = B_i - \{\neg u\}$. Our corresponding pebbling move is to take S_p and create S_q by removing the white pebble on node u. Therefore $S_q = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\} - \{\neg u\}$, but $B_{i+1} = B_i - \{\neg u\}$, so $S_q = \bigcup_{\neg v \in B_{i+1}} \{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Case 2: By our induction hypothesis, we have translated spine clause B_i into pebbling step S_p such that $S_p = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$. We now resolve B_i with a propagation clause $\{\neg a, \neg b, c\}$ on variable c to create $B_{i+1} = B_i \cup \{\neg a, \neg b\} - \{\neg c\}$. We make one or two corresponding pebbling moves, depending on whether or not node a already has a pebble on it. If a already has a pebble on it (which means that $\neg a \in B_i$), then we slide the white pebble on c to b. Therefore $S_q = S_p \cup \{b\} - \{c\}$. But $S_p = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$, so $S_q = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\} \cup \{b\} - \{c\}$, but $B_{i+1} = B_i \cup \{\neg b\} - \{\neg c\}$, so $S_q = \bigcup_{\neg v \in B_{i+1}} \{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Otherwise, if a does not already have a pebble on it, then we make two corresponding pebbling moves. The first move is to place a white pebble on a to create an intermediate pebbling step $S_{q'}$ which does not correspond directly to a spine clause. In other words, $S_{q'} = S_p \cup \{a\}$, so the size of our pebbling state has increased by 1. Since $B_{i+1} = B_i \cup \{\neg a, \neg b\} - \{\neg c\}$, and neither a nor b were pebbled in S_p, this means that the variables a and b do not occur in B_i but we resolved on c, which means that our spine lost one literal and gained two. Therefore $|B_{i+1}| = |B_i| + 1$, so both the spine and the number of pebbles have increased by exactly 1, and our intermediate step has not violated the maximum number of allowed pebbles.

The second pebbling move is to slide the white pebble on c to b, thereby bringing the pebbling state in line with the spine clause. This creates state S_q from $S_{q'}$, where $S_q = S_p \cup \{a, b\} - \{c\}$. But $S_p = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$ by our induction hypothesis, so $S_q = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\} \cup \{a, b\} - \{c\}$, but $B_{i+1} = B_i \cup \{\neg a, \neg b\} - \{\neg c\}$, so $S_q = \bigcup_{\neg v \in B_{i+1}} \{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Case 3: By our induction hypothesis, we have translated spine clause B_i into pebbling step S_p such that $S_p = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$. We now weaken B_i to introduce the negative literal $\neg u$, so $B_{i+1} = B_i \cup \{\neg u\}$. Therefore our corresponding pebbling move is to take S_p and create S_q by placing a white pebble on u. Therefore $S_q = S_p \cup \{u\} = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\} \cup \{u\} = \bigcup_{\neg v \in B_{i+1}} \{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Therefore, in all three cases, there exists a valid next step in the pebbling such that $S_q = \bigcup_{\neg v \in B_{i+1}} \{v\} - \{\neg\alpha, \neg\beta\}$, so by induction we can translate an I-RES-W$^-$ spine B into a pure white pebbling strategy S such that S perfectly encodes B. □

The following corollary combines the previous two Lemmas in order to prove an equivalence between the black pebbling number of DAG G and the I-RES-W$^-$ total space of $Peb_1(G)^*$.

Corollary 5.6.6. *For any DAG G with target node t, G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_j$ with j steps where $t \in S_j$ and j is $O(l)$ if and only if $Peb_1(G)^*$ has an* I-RES-W$^-$ *derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and the spine of π contains l clauses where l is $O(j)$.*

Proof: Let G be any arbitrary DAG with target node t.

⇒ Suppose that G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_j$ with j steps where $t \in S_j$ and j is $O(q)$. By Lemma 5.6.4, it is possible to translate this pebbling strategy into an I-RES-W$^-$ derivation π of $Peb_1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ such that the spine of π contains l clauses, each of which has width at most $k+2$. However, since $Peb_1(G)^*$ is a 3-CNF formula, each of the input clauses used to resolve with clauses in the spine has width 3. Since input refutations only require there to be two clauses in memory at any time, π therefore has total space $k+5$ and its spine contains $O(j)$ clauses.

⇐ Suppose that $Peb_1(G)^*$ has an I-RES-W$^-$ derivation π of $\{\neg\alpha, \neg\beta\}$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and π's spine contains l clauses where l is $O(j)$. Since $Peb_1(G)^*$ is a 3-CNF formula, π has a spine with width $k+2$. By Lemma 5.6.5, it is possible to translate this spine into a pure white pebbling strategy containing l steps which uses at most k pebbles. But by Lemma 3.3.2, it is possible to convert this pure white pebbling strategy into a pure black pebbling strategy with $O(l)$ steps which uses exactly the same number of pebbles. Therefore G has a pure black k-pebbling strategy with $O(l)$ steps. □

The Equivalence of Black Pebbling & Input Derivation Total Space

We now show that the weakenings in any I-RES-W$^-$ proof π can be removed to turn it into a weakening-free proof π' with the same spine width.

Lemma 5.6.7. *For any unsatisfiable set of Horn clauses F, any $C \in F$, and any goal clause D, there exists an* I-RES-W$^-$ *proof π from F with top clause C and goal clause D such that the spine width of π is at most k and the size of π is s if and only if there exists an* I-RES *proof π' from F with top clause C and goal clause $D^{'} \subseteq D$ such that the spine width of π' is at most k and the size of π' is $\leq s$.*

Proof: Let F be any unsatisfiable set of Horn clauses, let C be any arbitrary initial clause in F, and let D be any arbitrary goal clause.

⇒ Suppose that there exists an I-RES-W$^-$ proof π from F with top clause C and goal clause D such that the spine width of $\pi \leq k$ and the size of π is $O(s)$. In order to show that there exists an I-RES proof π' from F with top clause C and goal clause $D^{'} \subseteq D$ such that the spine width of $\pi' \leq k$ and the size of π' is $O(s)$, we will show how to translate π into π' without losing any of these properties. We shall first translate π into π^* by removing all of the weakenings in π as well as all applications of the resolution rule which no longer apply due to the fact that weakened variables (and all other variables derived by Resolving on them) are now missing. It is important to note that π^* is not necessarily a legitimate I-RES proof, because our translation method may cause two successive clauses in the spine of π to be translated into successive duplicate clauses in the spine of π^*.

We translate π into π^* via induction on the depth (the top clause having the smallest depth) of π according to the following inductive procedure / proof. We shall translate each clause C_i in the spine of π into a clause C_i^* in the spine of π^* such that $\forall i\ C_i \subseteq C_i^*$.

Basis: The top clause in π is C. Set the top clause in π^* to be C as well. Therefore $C_0 \subseteq C_0^*$, as required.

Induction Hypothesis: Suppose that $C_i \subseteq C_i^*$.

Induction Step: We now show how to translate π into π^* such that $C_{i+1} \subseteq C_{i+1}^*$:

Case 1: Suppose that C_{i+1} came from C_i in π via weakening. In this case, simply omit this weakening step in π^*. Since $C_i \subseteq C_i^*$ by our induction hypothesis, and C_{i+1} has grown, whereas C_{i+1}^* has not, we can conclude that $C_{i+1} \subseteq C_{i+1}^*$.

Case 2a: Suppose that C_{i+1} came from C_i in π by resolving C_i with the input clause I on variable x, but this resolution step is not possible in π^* because $x \notin C_i^*$. In this case, simply omit this resolution step in π^*. We know by the induction hypothesis that $C_i \subseteq C_i^*$, and since $x \notin C_i^*$, we can conclude that $C_{i+1} \subseteq C_{i+1}^*$.

Case 2b: Suppose that C_{i+1} came from C_i in π by resolving C_i with the input clause I on variable x, and $x \in C_i^*$. In this case, simply perform the same resolution step in π^*; i.e. resolve C_i^* with I to get C_{i+1}^*. By the induction hypothesis that $C_i \subseteq C_i^*$, so $C_{i+1} \subseteq C_{i+1}^*$.

Therefore, in all cases, $C_{i+1} \subseteq C_{i+1}^*$, showing that by induction, $D \subseteq D^*$, where D is the goal clause of π and D^* is the goal clause of π^*. Finally, since π^* might contain duplicate clauses along its spine, we simply remove all but one out of each group of duplicates to create π', which is now a legitimate I-RES proof. Therefore there exists an I-RES proof π' from F with top clause C and goal clause $D' \subseteq D$ such that the spine width of $\pi' \leq k$ and the size of π' is $\leq s$, as required.

$\Leftarrow$ Suppose there exists an I-RES proof π' from F with top clause C and goal clause $D' \subseteq D$ such that the spine width of $\pi' \leq k$ and the size of π' is $\leq s$. Since every I-RES proof is an I-RES-W$^-$ proof, π' is also an I-RES-W$^-$ proof with goal clause $D' \subseteq D$. We convert π' to π by weakening D' to get D. Therefore there exists an I-RES-W$^-$ proof π from F with top clause C and goal clause D such that the spine width of $\pi \leq k$ and the size of π is $O(s)$, as required. □

The following corollary is identical to Corollary 5.6.6 except that instead of proving an equivalence between the black pebbling number of a DAG G and the I-RES-W$^-$ total space of $Peb_1(G)^*$, here we use our weakening removal Lemma from above to show that this equivalence also holds for I-RES without weakening:

Corollary 5.6.8. *For any DAG G with target node t, G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_j$ with j steps where $t \in S_j$ and j is $O(l)$ if and only if $Peb_1(G)^*$ has an* I-RES *derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π' has total space $k+5$ and size l where l is $O(j)$.*

Proof: Let G be any arbitrary DAG with target node t.

$\Rightarrow$ Suppose that G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_j$ with j steps where $t \in S_j$ and j is $O(l)$. By Corollary 5.6.6, $Peb_1(G)^*$ has an I-RES-W$^-$ derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and size l where l is $O(j)$. Because $Peb_1(G)^*$ is a 3-CNF formula, π's spine has width $\leq k+2$. By Lemma 5.6.7 there exists an I-RES proof π' from F with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $D' \subseteq \{\neg\alpha, \neg\beta\}$ such that the spine width of $\pi' \leq k+2$ and the size of π' is $O(j)$. Since F does not include any positive instances of the variables α or β, we can conclude that $D' = \{\neg\alpha, \neg\beta\}$. Therefore, since $Peb_1(G)^*$ is a 3-CNF formula, π' has total space $k+5$ and size $O(j)$, with the same top and goal clauses as π.

$\Leftarrow$ Suppose that $Peb_1(G)^*$ has an I-RES derivation π' with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π' has total space $k+5$ and size l where l is $O(j)$. But every I-RES proof is an I-RES-W$^-$ proof, so there exists an I-RES-W$^-$ derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and size l, which is $O(l)$. Therefore, by Corollary 5.6.6 G has a pure black k-pebbling strategy. $\square$

5.6.2 The PSPACE-Completeness of Input Derivation Total Space

This section contains some immediate corollaries to the previous results. Here we prove the $\mathcal{PSPACE}$-Completeness of three slightly different versions of the I-RES derivation total space problem, which is surprising since I-RES seems like such a trivial Resolution refinement.

The I-RES derivation total space problem (ITS) is defined as follows: The input instance is (F, C, D, k), where F is an unsatisfiable set of clauses, C is any clause in F, D is a goal clause, and k is an integer. The problem is to determine if there exists an I-RES derivation from F with top clause C, goal clause D, and total space at most k.

We shall prove the $\mathcal{PSPACE}$-Completeness of three different versions of this problem:

1. The set F is restricted to containing only Horn clauses, and instead of asking whether there exists an I-RES derivation from F with top clause C, goal clause D, and total space at most k, we ask whether there exists an I-RES-W$^-$ derivation with negative weakening from F with top clause C, goal clause D, and total space at most k. This is the input derivation with negative weakening total space problem, and we shall refer to the language associated with it as $HWITS$, where the 'HW' stands for '**H**orn with Negative **W**eakening'.
2. This version of the problem is almost identical to the previous one; the set F is still restricted to containing only Horn clauses, but this time we are dealing with I-RES rather than I-RES-W$^-$. We shall refer to the language associated with this version of the problem as $HITS$.
3. Again, this version of the problem is almost identical to the previous one, only this time there is no restriction on F. We shall refer to the language associated with this version of the problem as ITS.

The PSPACE-Completeness of Input Total Space with Weakening

The language $HWITS$ is formally defined as follows:

Definition 5.6.9 ($HWITS$). $HWITS = \{(F, C, D, k) \mid F$ *is a Horn formula for which there exists an* I-RES-W$^-$ *refutation with top clause C, goal clause D, and total space at most k.*$\}$

The $\mathcal{PSPACE}$-Completeness of $HWITS$ follows from the equivalence between the black pebbling number of DAG G and the I-RES-W$^-$ total space of $Peb_1(G)^*$ given in Corollary 5.6.6 from the previous section. This language is formally defined as follows:

Theorem 5.6.10. *$HWITS$ is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

Proof: We first show that $HWITS \in \mathcal{PSPACE}$. We are given an input instance (F, $C \in F$, D, k) and asked to determine if F has an I-RES-W$^-$ proof π with top clause C and goal clause D such that π's total space is bounded above by k. Since we are dealing with input refutations, we know that if such a π exists, then each of its configurations contains no more than two clauses. Since each clause contains at most n literals, it is clear that every configuration in π takes only polynomial space.

Our algorithm proceeds as follows: Start with a configuration $\mathbb{C}_0 = \{C\}$. Guess configuration $\mathbb{C}_1$, check to ensure that it follows from $\mathbb{C}_0$ by a legal I-RES-W$^-$ step, and erase configuration $\mathbb{C}_0$. Next, guess configuration $\mathbb{C}_2$, check to make sure that it follows from $\mathbb{C}_1$, and that $TS(\mathbb{C}_2) \le k$. Erase configuration $\mathbb{C}_1$, and continue in this way until the goal clause has been derived. Note that at any time, there are only two configurations in memory. We are dealing with input refutations, so we know that each configuration contains no more than two clauses. Since each clause contains at most n literals, it is clear that this non-deterministic algorithm requires polynomial space. This shows that $HWITS \in \mathcal{NPSPACE}$. Finally, we appeal to Savitch's Theorem [Sav70] to show that $HWITS \in \mathcal{PSPACE}$.

Next we show that $HWITS$ is $\mathcal{PSPACE}$-Hard by giving a reduction from the black pebbling number problem with sliding ($BLACK$-PEB) from [GLT80] which was shown to be $\mathcal{PSPACE}$-Complete. We are given an instance (G, k) where t is the target node in G and we wish to convert it to an instance (F, $C \in F$, D, k) such that $(G, k) \in BLACK$-PEB if and only if (F, $C \in F$, D, k) $\in HWITS$. Our reduction proceeds as follows: Take (G, k) and output ($Peb_1(G)^*$, $\{\neg t, \neg\alpha, \neg\beta\}$, $\{\neg\alpha, \neg\beta\}$, $k + 5$), which is clearly a logspace reduction. The proof of correctness for this reduction is given by Corollary 5.6.6. □

The PSPACE-Completeness of Horn Input Total Space

The language $HITS$ is formally defined as follows:

Definition 5.6.11 ($HITS$). $HITS = \{(F, C, D, k) \mid F$ *is a Horn formula for which there exists an* I-RES *refutation with top clause C, goal clause D, and total space at most k.*$\}$

The $\mathcal{PSPACE}$-Completeness of $HITS$ follows immediately from the equivalence between the black pebbling number of DAG G and the I-RES total space of $Peb_1(G)^*$ given in Corollary 5.6.8 from the previous section.

Corollary 5.6.12. *$HITS$ is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

Proof: To show that $HITS \in \mathcal{PSPACE}$, we simply use the same algorithm given in Theorem 5.6.10.

To show that $HITS$ is $\mathcal{PSPACE}$-Hard, we give a reduction from $BLACK$-PEB. As in Theorem 5.6.10, take (G, k) and output ($Peb_1(G)^*$, $\{\neg t, \neg\alpha, \neg\beta\}$, $\{\neg\alpha, \neg\beta\}$, $k+5$), which is clearly a logspace reduction, and its proof of correctness is given by Corollary 5.6.8. □

The PSPACE-Completeness of Input Total Space

The language ITS is formally defined as follows:

Definition 5.6.13 (ITS). $ITS = \{(F, C, D, k) \mid F$ *is a formula for which there exists an* I-RES *refutation with top clause C, goal clause D, and total space at most k.*$\}$

The $\mathcal{PSPACE}$-Completeness of ITS follows trivially from the $\mathcal{PSPACE}$-Completeness of ITS:

Corollary 5.6.14 (Main Result). *ITS is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

Proof: To show that $ITS \in \mathcal{PSPACE}$, we simply use the same algorithm given in Theorem 5.6.10.

To show that ITS is $\mathcal{PSPACE}$-Hard, we reduce from $HITS$. Take the input F for $HITS$ and check to see if it is a set of Horn clauses. If not, then output a pre-chosen formula which is not in ITS. Otherwise, simply output F. Clearly this is a logspace reduction, and its correctness is immediate since Horn formulas are just a special case of what an ITS solver can decide. □

5.7 Related Complexity Results

The $\mathcal{PSPACE}$-Completeness of I-RES total space has some interesting corollaries:

5.7.1 The PSPACE-Completeness of the Input Derivation Width Problem

One interesting point of note is that I-RES total space and width are virtually identical. More specifically, for every k-CNF formula, $w(F \vdash_{\text{I-RES}} D) = TS(F \vdash_{\text{I-RES}} D) - k$.

The $Peb_1(G)^*$ formulas are 3-CNF formulas, so by Corollary 5.6.8, for any DAG G with pebbling number $k = B$-$Peb(G)$, the width of any I-RES derivation of goal clause $\{\neg\alpha, \neg\beta\}$ from $Peb_1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ is $k + 2$. This implies that the I-RES derivation width problem is therefore also $\mathcal{PSPACE}$-Complete:

Corollary 5.7.1. *Given a formula F, a top clause C, goal clause D, and integer k, the problem of determining if there exists an* I-RES *derivation of D from F with top clause C with width at most k is $\mathcal{PSPACE}$-Complete under logspace reducibility.*

5.7.2 An Optimal Size / Space Tradeoff For Input Resolution

In this section we describe another corollary to the results in Section 5.6.1. This result also relies closely on one of the most surprising facts concerning pebbling, namely that there exist infinite families of DAGs which take an exponential amount of time to be pebbled with the minimum number of pebbles, but if one is willing to use just one or two more pebbles, then they can be pebbled in linear time. The earliest known example of this phenomenon can be found in [Lin78], where Lingas gives an infinite family of monotone circuits such that pebbling any of them with the minimum number of pebbles requires $2^{\Omega(n^{1/3})}$ time, where n is the number of nodes in the circuit. However, if given only two more pebbles, the amount of time required drops exponentially to only $O(n)$, giving a massive pebble / pebbling time tradeoff.

This result was later improved by Gilbert, Lengauer, and Tarjan:

Lemma 5.7.2 ([GLT80]). *There exists an infinite family of DAGs $\mathcal{G}$, such that pebbling any $G \in \mathcal{G}$ with the minimum number of pebbles takes $\Omega(2^n)$ time, but if only one more pebble is used, then the amount of time required to pebble G drops exponentially to only $O(n)$, where n is the number of vertices in G.*

Such an exponential separation at the cost of only one pebble is extraordinary, but since Corollary 5.6.8 gives an exact relationship between black pebbling and total space for I-RES derivations, it is possible to translate this amazing result from the world of pebbling to the realm of Resolution, thereby giving a massive size / total space tradeoff for I-RES:

Corollary 5.7.3. *There exists an infinite family of formulas $\mathcal{F} = \{Peb_1(G)^* \mid G \in \mathcal{G}\}$ such that for every $F \in \mathcal{F}$, every* I-RES *derivation π of F with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ where π has the minimum required total space, has size $\Omega(2^n)$, where n is the number of variables in F. However, if only one more unit of total space is permitted, then the size of π drops to only $O(n)$.*

Proof: Let $\mathcal{G}$ be the family of DAGs from Lemma 5.7.2, and let F be any arbitrary formula from $\mathcal{F}$. Therefore $F = Peb_1(G)^*$ for some $G \in \mathcal{G}$. G can be pebbled with k but not with fewer than k pebbles. By Corollary 5.6.8, F has an I-RES derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ such that π requires total space $k+5$, but does not have an I-RES derivation with total space less than $k+5$. We know that to pebble G with k pebbles requires $\Omega(2^n)$ time, so by Corollary 5.6.8, π has a size of at least $\Omega(2^n)$. Similarly, we know that pebbling G with $k+1$ pebbles requires $O(n)$ time, so again by Corollary 5.6.8, there exists a proof π' with total space $k+5+1 = k+6$ and size $O(n)$, as required. $\square$

Therefore, using just one extra unit of total space yields an exponential decrease in size, giving an optimal size / total space tradeoff. As with our $\mathcal{PSPACE}$-Completeness results from the previous section, this is more evidence that the computational complexity of I-RES is underestimated.

5.8 Open Problems & Conjectures Related to Input Resolution Total Space

Although we have determined the complexities of several problems related to I-RES, this area of research still has a few open problems. For example, in Section 5.3.2 we showed that several problems such as deciding *MU-IRES-UNSAT* and determining if a minimally unsatisfiable formula has an I-RES refutation of size at most k are in $\mathcal{P}$. However, Corollary 5.3.4 showed us that *IRES-UNSAT* is $\mathcal{P}$-Complete. It therefore may be possible to show that many of the problems pertaining to minimally unsatisfiable formulas are also $\mathcal{P}$-Complete or complete for some other complexity class. One problem in particular which stands out is the complexity of determining if a formula F has an I-RES refutation with top clause C and goal clause D. In Corollary 5.6.14 we showed that determining the total space for the corresponding problem is $\mathcal{PSPACE}$-Complete, and being able to compare the complexities of the same problem, both with and without the k parameter would be interesting.

This brings up another interesting open problem. We were able to show the $\mathcal{PSPACE}$-Completeness of the I-RES total space derivation problem with top clause and goal clause, but have not been able to prove any results about the corresponding refutation problem without these extra clauses as part of the input instance. We conjecture that this problem is also $\mathcal{PSPACE}$-Complete, and this result would help to tie up some loose ends.

Another open problem worth mentioning is related to tradeoffs. In Section 5.7.2 we showed that I-RES derivations have massive size / total space tradeoffs. Given the nature of the results in Part II of this thesis, it is reasonable to conjecture that many exponential tradeoffs of this nature exist for various forms of Resolution and various resources such as space, width, and depth. One of the most interesting examples of a massive tradeoff would be for RES width, and it is somewhat surprising that this problem has not already been studied.

One final open problem not pertaining to complexity is that of counting the number of formulas in MU-$IRES$-$UNSAT$. Since we have given an exact matrix characterization of these formulas, it may be possible to give an exact characterization for the number of MU-$IRES$-$UNSAT$ formulas on n variables as a function of n.

Chapter 6

The PSPACE-Hardness of Resolution Variable Space

6.1 Introduction & Motivation

In this chapter we prove the $\mathcal{PSPACE}$-Hardness of another Resolution resource problem, that of RES variable space, and also show a strong relationship between it and the black-white pebbling game on arbitrary monotone circuits of unbounded fan-in. These results rely directly on previous work done in [BS02, HP07] and are proved by once again employing the pebbling formulas as we did in Chapters 4 and 5.

Although variable space is not as practical or natural a measure as clause space or total space, this result differs from those in the previous two chapters in that it pertains to the full-strength RES proof system, whereas they contain results for the Resolution refinements T-RES and I-RES. One final point worth mentioning is that even though we prove that RES variable space is $\mathcal{PSPACE}$-Hard, we are unable to show that it is in $\mathcal{PSPACE}$, suggesting the possibility that this problem may even be $\mathcal{EXPTIME}$-Hard or worse.

This chapter is organized as follows: In Section 6.2 we show that for any monotone circuit C, the minimum RES variable space of refuting $Peb_1(C)$ is exactly equal to $BW\text{-}Peb(C)$. Next, in Section 6.3 we put this equivalence result to use in order to prove the $\mathcal{PSPACE}$-Hardness of the RES Variable Space problem. Finally, in Sections 6.4 and 6.5 we respectively discuss related complexity results and open problems.

6.2 The Equivalence of Black-White Pebbling & Resolution Variable Space

In this section we prove that for any monotone circuit C, the minimum RES variable space of refuting $Peb_1(C)$ is exactly equal to $BW\text{-}Peb(C)$, where $BW\text{-}Peb(C)$ is defined without sliding. Theorem 3.3.5 (proved by Ben-Sasson) gives half of this equivalence, and we prove the other half by showing that for any monotone circuit C with unbounded fan-in, $VS(Peb_1(C) \vdash_{\textsf{RES-EM}} \emptyset) \leq BW\text{-}Peb(C)$, where $BW\text{-}Peb(C)$ is defined without sliding and RES-EM is a slight variation of RES which allows the use of some trivial axioms. We then show how to dispense with these axioms to prove that the result also holds for RES. The RES-EM proof system is defined as follows:

Definition 6.2.1 (RES-EM)**.** *The proof system* RES-EM *is just like* RES *except that all n clauses representing the law of excluded middle of the form $(a \vee \neg a)$ are present as axioms and may be used at any time as if they were initial clauses.*

Before proving our equivalence theorem, we first need to define the notion of a 'dependency set'. Intuitively, the dependency set of a vertex i is the set of all white pebbles that were needed in order to pebble i. More formally dependency sets are defined as follows:

Definition 6.2.2 (Dependency Set). *Given a pebbling strategy $S = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$ on a monotone circuit C, for each integer $0 \leq k \leq j$ and each vertex $i \in B_k \cup W_k$, we define the Dependency Set of i, denoted $\Delta(i)$, as follows:*

1. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a black pebble was placed on AND gate i of degree d, with immediate predecessors $p_1, ..., p_d$, then set $\Delta(i) = \bigcup_{1 \leq w \leq d} \Delta(p_w)$.*
2. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a black pebble was placed on OR gate i, then set $\Delta(i) = \Delta(p_w)$ for some pebbled predecessor p_w of i.*
3. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a white pebble was placed on node i where i is either an AND gate or an OR gate, then set $\Delta(i) = \{i\}$.*
4. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a black pebble was removed from node i where i is either an AND gate or an OR gate, then $\Delta(i)$ becomes undefined, and there is no change in the other Dependency Sets.*
5. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a white pebble was removed from AND gate i of degree d, with immediate predecessors $p_1, ..., p_d$, then for all v such that $i \in \Delta(v)$, $\Delta(v) = (\Delta(v) - \{i\}) \bigcup_{1 \leq w \leq d} \Delta(p_w)$.*
6. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a white pebble was removed from OR gate i then for all v such that $i \in \Delta(v)$, $\Delta(v) = (\Delta(v) - \{i\}) \cup \Delta(p_w)$ for some pebbled predecessor p_w of i.*
7. *Finally, if a vertex i has no pebble on it, then $\Delta(i)$ is undefined.*

We now prove a lemma giving the first half of our variable space / black-white pebbling number equivalence. The high-level idea of this proof is as follows: For a monotone circuit C, we take a black-white pebbling strategy $S = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$, and for each (B_k, W_k), we create a special RES-EM configuration M_k. We then build a RES refutation of $Peb_1(C)$ using S by induction on k to create an M_k for each (B_k, W_k) such that $VS(M_k) \leq |B_k \cup W_k|$. This is done by showing how to derive each M_{k+1} from M_k using variable space bounded by $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. Since $(B_0, W_0) = (\emptyset, \emptyset)$, and since $(B_j, W_j) = (\{t\}, \emptyset)$, the resulting proof is a RES-EM refutation of $Peb_1(C)$ bounded by variable space $BW\text{-}Peb(C)$.

This brings us to the final definition that we shall need before proceeding, that of the special configuration called M_k:

Definition 6.2.3 (M_k). $M_k = \{(\Delta(i) \supset i) \mid i$ *is a pebbled vertex in pebbling step* $(B_k, W_k)\}$

Lemma 6.2.4. *For any monotone circuit C with unbounded fan-in and target t, $VS(Peb_1(C) \vdash_{\mathsf{RES\text{-}EM}} \emptyset) \leq BW\text{-}Peb(C)$, where $BW\text{-}Peb(C)$ is defined without sliding.*

Proof: Let C be any arbitrary monotone circuit with unbounded fan-in and target t, and let $S = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$ be any pebbling strategy for C with $(B_0, W_0) = (\emptyset, \emptyset)$, and $(B_j, W_j) = (\{t\}, \emptyset)$.

We show by induction on k how to take each pebbling step (B_k, W_k) and build a configuration-style RES refutation from $Peb_1(C)$ which includes the configurations $M_0, M_1, ..., M_j$ such that for all k, we can derive M_{k+1} from M_k within variable space $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$.

Of course, we write $(\Delta(i) \supset i)$ for notational purposes only; all clauses are in CNF. The set $\Delta(i)$ should be interpreted as a conjunction and $(p_1 \wedge ... \wedge\ p_{m-1} \supset p_m)$ is equivalent to the clause $\{\neg p_1, ..., \neg p_{m-1},\ p_m\}$.

Basis: $(B_0, W_0) = (\emptyset, \emptyset)$, so $M_0 = \emptyset$. Clearly the empty configuration can be derived from $Peb_1(C)$ via RES-EM within variable space $0 = |B_0 \cup W_0|$.

Induction Hypothesis: Suppose that for (B_k, W_k), $VS(M_k) \leq |B_k \cup W_k|$.

Induction Step: We now show that for (B_{k+1}, W_{k+1}), we can derive M_{k+1} from M_k all within variable space $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. Since we are not considering sliding, (B_{k+1}, W_{k+1}) could have come from (B_k, W_k) by placing or removing a single pebble according to one of the following 6 cases:

1. By placing a black pebble on AND gate i,
2. By placing a black pebble on OR gate i,
3. By placing a white pebble on node i, where i is either an AND or an OR gate,
4. By removing a black pebble from node i, where i is either an AND or an OR gate,
5. By removing a white pebble from AND gate i (This is the most difficult case.), or
6. By removing a white pebble from OR gate i.

Case 1: Since we placed a black pebble on AND gate i, $(B_{k+1}, W_{k+1}) = (B_k \cup \{i\}, W_k)$, which means that $M_{k+1} = M_k \cup \{(\Delta(i) \supset i)\}$, where $\Delta(i) = \bigcup_{1 \leq w \leq d} \Delta(p_w)$. By our induction hypothesis, we know that for (B_k, W_k), $VS(M_k) \leq |B_k \cup W_k|$, so we need to show how to derive M_{k+1} from M_k within variable space $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. This is done as follows: Node i has predecessors $p_1, ..., p_d$, all of which are pebbled. Therefore, $(\Delta(p_w) \supset p_w) \in M_k$ for every predecessor p_w of i. We must show how to derive $(\Delta(i) \supset i)$. Since i has $p_1, ..., p_d$ as predecessors, $Peb_1(C)$ contains the clause $(p_1 \wedge ... \wedge p_d \supset i)$ as an initial clause. Download this clause into memory. Since the variables $p_1, ..., p_d$ are already all present in M_k, this adds at most one to our total variable space. Next, resolve $(p_1 \wedge ... \wedge p_d \supset i)$ with $(\Delta(p_1) \supset p_1)$ on variable p_1 to get $(\Delta(p_1) \wedge p_2 \wedge ... \wedge p_d \supset i)$. Repeat this for every $(\Delta(p_w) \supset p_w)$ to get $\{\bigcup_{1 \leq w \leq d} \Delta(p_w) \supset i\}$. But since $\Delta(i) = \bigcup_{1 \leq w \leq d} \Delta(p_w)$, we have derived $(\Delta(i) \supset i)$, all within variable space $|B_{k+1} \cup W_{k+1}|$.

Case 2: This is a simpler form of the previous case. Since we placed a black pebble on OR gate i, $(B_{k+1}, W_{k+1}) = (B_k \cup \{i\}, W_k)$, which means that $M_{k+1} = M_k \cup \{(\Delta(i) \supset i)\}$, where $\Delta(i) = \Delta(p_w)$ for some pebbled predecessor p_w of i. Therefore, $(\Delta(p_w) \supset p_w) \in M_k$, and $Peb_1(C)$ contains the initial clause $(p_w \supset i)$. Download this clause into memory, thereby adding at most one to the total variable space. Now we resolve $(\Delta(p_w) \supset p_w)$ with $(p_w \supset i)$ to get $(\Delta(p_w) \supset i)$. But $\Delta(i) = \Delta(p_w)$, so we have derived $(\Delta(i) \supset i)$, all within variable space $|B_{k+1} \cup W_{k+1}|$.

Case 3: Since we placed a white pebble on node i which is either an AND gate or an OR gate, $(B_{k+1}, W_{k+1}) = (B_k, W_k \cup \{i\})$, which means that $M_{k+1} = M_k \cup \{i \supset i\}$. By our induction hypothesis, we know that for (B_k, W_k), $VS(M_k) \leq |B_k \cup W_k|$, so we need to show how to derive $(i \supset i)$. Since we are using the system RES-EM, this is an axiom, so simply download it. Clearly this does not exceed variable space $|B_{k+1} \cup W_{k+1}|$, since W_{k+1} has a pebble on i.

Case 4: Since we removed a black pebble from node i which is either an AND or an OR gate, $(B_{k+1}, W_{k+1}) = (B_k - \{i\}, W_k)$, which means that $M_{k+1} = M_k - \{(\Delta(i) \supset i)\}$. Therefore, in order to derive M_{k+1} from M_k, simply drop the clause $(\Delta(i) \supset i)$, which can clearly be done without increasing the variable space.

Case 5: This case is more complicated than the others, because i may be in the dependency sets of many other vertices. Since we removed a white pebble from vertex i, $(B_{k+1}, W_{k+1}) = (B_k, W_k - \{i\})$. In order to derive M_{k+1} from M_k, we must therefore drop the clause $(\Delta(i) \supset i)$ from M_k, and also show how to remove the literal $\neg i$ from all other clauses that contain it. Because removing a white pebble requires all predecessors to be pebbled, there are two sub-cases to consider:

Case 5a: Suppose that i has no predecessors; i.e. i is a source node. In this case we simply drop the clause $(i \supset i)$ from M_k, and resolve every remaining clause in M_k which contains $\neg i$ as part of its dependency set with the initial clause (i). Clearly this removes the literal $\neg i$ from all clauses containing it, and does not increase the variable space, since i is already present in M_k.

Case 5b: Suppose that i has in-degree d and has predecessors $p_1, ..., p_d$, of which 0 or more have black pebbles on them in (B_k, W_k). Therefore M_k contains the clause $(\Delta(i) \supset i)$, and for each predecessor p_w

which has a black pebble on it, M_k contains the clause $(\Delta(p_w) \supset p_w)$. Furthermore, $Peb_1(C)$ contains the clause $(p_1 \wedge ... \wedge\ p_d \supset i)$ as an initial clause. In order to derive M_{k+1} from M_k, first drop the clause $(\Delta(i) \supset i)$. Next, we need to show how to turn each clause $(\Delta(v) \supset v)$ where $i \in \Delta(v)$ and turn it into the clause $((\Delta(v) - \{i\}) \bigcup_{1 \leq w \leq d} \Delta(p_w) \supset v)$.

To do this, first download initial clause $(p_1 \wedge ... \wedge\ p_d \supset i)$ into memory. All of its variables are already present in M_k, so this does not change the variable space. Then, for each clause $(\Delta(v) \supset v)$ where $i \in \Delta(v)$, do the following: Resolve $(\Delta(v) \supset v)$, with the initial clause $(p_1 \wedge ... \wedge\ p_d \supset i)$ to get $((\Delta(v) - \{i\}) \cup \{p_1, ..., p_d\} \supset v)$. Finally, for each p_w which is a predecessor of i, if p_w has a black pebble on it in (B_k, W_k), then resolve with $(\Delta(p_w) \supset p_w)$. For each p_w with a white pebble on it, $\Delta(p_w) = \{p_w\}$, so this process results in the derivation of the clause $((\Delta(v) - \{i\}) \bigcup_{1 \leq w \leq d} \Delta(p_w) \supset v)$, all without increasing the variable space.

Case 6: This case is a slightly simpler form of the previous case. Once again, there are two sub-cases to consider:

Case 6a: This case is identical to 5a.

Case 6b: Suppose that i has in-degree d with predecessors $p_1, ..., p_d$. Let p_w be one of i's pebbled predecessors. Therefore M_k contains the clauses $(\Delta(i) \supset i)$ and $(\Delta(p_w) \supset p_w)$. To derive M_{k+1} from M_k, first drop the clause $(\Delta(i) \supset i)$. Next, download the initial clause $(p_w \supset i)$ into memory. All of its variables are already present in M_k, so this does not change the variable space. Then, for each clause $(\Delta(v) \supset v)$ where $i \in \Delta(v)$, turn $(\Delta(v) \supset v)$ into $((\Delta(v) - \{i\})\ \cup\ \{p_w\}) \supset v)$ by taking $(\Delta(v) \supset v)$ and resolving on variable i with $(p_w \supset i)$. None of these steps increase the variable space.

Therefore, in all cases, each M_{k+1} can be derived in RES-EM from M_k by induction such that the whole proof uses a variable space of at most $BW\text{-}Peb(C)$. This yields the configuration $M_j = \{(t)\}$ which corresponds to the final pebbling step $(B_j, W_j) = (\{t\}, \emptyset)$. In order to derive the empty clause, simply resolve with the initial clause $(\neg t)$, which does not change the variable space. Hence we can conclude that for any circuit C with target t, $VS(Peb_1(C) \vdash_{\mathsf{RES\text{-}EM}} \emptyset) \leq BW\text{-}Peb(C)$ without sliding, as required. $\square$

The following Lemma proves that RES and RES-EM are equivalent with respect to variable space:

Lemma 6.2.5. *For any CNF formula F and clause C, $VS(F \vdash_{\mathsf{RES\text{-}EM}} C) = VS(F \vdash_{\mathsf{RES}} C)$.*

Proof: $\Rightarrow$ Let π be a RES-EM derivation of C from F. Since resolving with law of excluded middle axioms $(a \vee \neg a)$ results in exactly the same clause that we started with, we may remove all such resolution steps to give us a RES derivation π' of C. Furthermore, since a must already be present for us to be able to resolve with $(a \vee \neg a)$, removing these steps does not change the variable space of π. Therefore $VS(F \vdash_{\mathsf{RES\text{-}EM}} C) \leq VS(F \vdash_{\mathsf{RES}} C)$.
$\Leftarrow$ Let π be a RES derivation of C from F. Since every RES derivation is a RES-EM derivation, π is also a RES-EM derivation of C from F, so $VS(F \vdash_{\mathsf{RES}} C) \leq VS(F \vdash_{\mathsf{RES\text{-}EM}} C)$. $\square$

This allows us to prove our main result in this section, namely that the black-white pebbling number of monotone circuits and the RES variable space of their pebbling contradictions are equivalent:

Theorem 6.2.6. *For any monotone circuit C, $BW\text{-}Peb(C) = VS(Peb_1(C) \vdash_{\mathsf{RES}} \emptyset)$, where $BW\text{-}Peb(C)$ is defined without sliding.*

Proof: $\Rightarrow$ From Theorem 3.3.5, $BW\text{-}Peb(C) \leq VS(Peb_1(C) \vdash_{\mathsf{RES}} \emptyset)$.
$\Leftarrow$ Conversely, by Lemmas 6.2.4 and 6.2.5, $VS(F \vdash_{\mathsf{RES}} C) \leq BW\text{-}Peb(C)$. $\square$

This result together with Lemma 3.3.1 immediately gives us the corresponding result with sliding as a corollary:

Corollary 6.2.7. *For any monotone circuit C, $BW\text{-}Peb(C) = VS(Peb_1(C) \vdash_{\mathsf{RES}} \emptyset)+1$, where $BW\text{-}Peb(C)$ is defined with sliding.*

6.3 The PSPACE-Hardness of Resolution Variable Space

We now prove this chapter's main result, namely the $\mathcal{PSPACE}$-Hardness of the RES variable space problem, which follows as an immediate corollary to the results in the previous section. Given a formula F and integer k, the RES variable space problem asks if F has a RES refutation with variable space at most k. We shall refer to the language associated with this problem as RVS, and it is formally defined as follows:

Definition 6.3.1 (RVS). *$RVS = \{(F, k) \mid F$ is a formula for which there exists a* RES *refutation with variable space at most k.$\}$*

The proof is a reduction from the $\mathcal{PSPACE}$-Complete black-white pebbling problem $BW\text{-}PEB$ on monotone circuits with sliding from [HP07]. In addition to the lower bound, we also give an upper bound, albeit a weak one, by showing that $RVS \in \mathcal{EXPSPACE}$:

Corollary 6.3.2 (Main Result). *RVS is $\mathcal{PSPACE}$-Hard under logspace reducibility, and is also in $\mathcal{EXPSPACE}$.*

Proof: To show that RVS is $\mathcal{PSPACE}$-Hard, our reduction proceeds by taking the input (C, k), where C is a circuit and k an integer, and outputting $(Peb_1(C), k-1)$. Clearly this is a logspace reduction, and the proof of correctness is immediate from Corollary 6.2.7: If $(C, k) \in BW\text{-}PEB$, then $BW\text{-}Peb(C) \leq k$, so by Corollary 6.2.7, $VS(Peb_1(C) \vdash_{\mathsf{RES}} \emptyset) \leq k-1$, which means that $(Peb_1(C), k-1) \in RVS$. Otherwise, if $(C, k) \notin BW\text{-}PEB$, then $BW\text{-}Peb(C) > k$, so by Corollary 6.2.7, $VS(Peb_1(C) \vdash_{\mathsf{RES}} \emptyset) > k-1$, which means that $(Peb_1(C), k-1) \notin RVS$, thereby showing that RVS is $\mathcal{PSPACE}$-Hard.

To show that $RVS \in \mathcal{EXPSPACE}$, we give a simple nondeterministic algorithm: Let F be an arbitrary unsatisfiable formula on n variables. Since there are exactly 3^n different clauses on n variables, each configuration in any RES refutation π of F can contain at most 3^n clauses. Since each clause can have at most n literals, each configuration of π requires a total space of at most $n \cdot 3^n$. However, $3^n = 2^{log_3(n)}$, so $n \cdot 3^n = n \cdot 2^{log_3(n)} \leq 2^n \cdot 2^{log_3(n)} \leq 2^{log_3(n)+1}$. Each configuration in any RES refutation therefore requires at most exponential space. This fact allows us to create a very simple $\mathcal{NEXPSPACE}$ algorithm: Given a formula F and integer k, simply guess configurations one at a time as in Theorem 5.6.10, all the while making sure that no configuration has a variables space of more than k, and never keeping more than two of these at most exponentially-sized configurations in memory at any instant. Finally, we appeal to Savitch's Theorem [Sav70] to show that this problem is in $\mathcal{EXPSPACE}$ as well. $\square$

6.4 Related Complexity Results

There are a few minor problems related to Resolution variable space:

6.4.1 The Complexity of Regular Tree Resolution Variable Space

The RT-RES variable space problem is formally defined as follows:

Definition 6.4.1 ($RTRVS$). *$RTRVS = \{(F, k) \mid F$ is a formula for which there exists a* RT-RES *refutation with variable space at most k.$\}$*

Although we are unable to give an $\mathcal{EXPTIME}$ algorithm for $RTRVS$, it is not hard to see that this problem is both co$\mathcal{NP}$-Hard and in $\mathcal{NEXPTIME}$:

Theorem 6.4.2. *$RTRVS$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{NEXPTIME}$.*

Proof: It is easy to see that $RTRVS$ is co$\mathcal{NP}$-Hard by reducing from $UNSAT$. Given a formula F, output $(F, k = n)$. Since any unsatisfiable formula has variable space at most n, $F \in UNSAT$ if and only if $(F, k) \in RTRVS$.

To show that $RTRVS \in \mathcal{NEXPTIME}$, we use the following nondeterministic algorithm: Given a formula F and integer k, we guess a configuration-style RT-RES refutation π of F. A RT-RES refutation of a formula F on n variables can contain at most $2^{n+1} - 1$ clauses, since this is the number of nodes in the complete binary decision tree of height n. Each configuration of π therefore contains at most $2^{n+1} - 1$ clauses, and there are at most $2^{n+1} - 1$ such configurations. This is a gross upper bound, but it still shows that π can be output in nondeterministic exponential time. This shows that $RTRVS \in \mathcal{NEXPTIME}$, so $TRVS \in \mathcal{NEXPTIME}$ as well. □

An interested reader may recall that RT-RES clause space, total space, and size all have $\mathcal{PSPACE}$ algorithms (see Corollaries 4.8.14, 4.8.6, and 4.8.8), and wonder why the same techniques cannot be used to show that $RTRVS \in \mathcal{PSPACE}$. The reason is that although the tree T underlying any RT-RES refutation is a subtree of the complete binary decision tree on n variables, and therefore can be pebbled using at most $n + 1$ pebbles, we have no guarantee that one of the pebblings of T which uses $n + 1$ pebbles induces a configuration-style proof with minimal variable space.

6.4.2 The Complexity of Tree Resolution Variable Space

The T-RES variable space problem is very similar to $RTRVS$, and is formally defined as follows:

Definition 6.4.3 ($TRVS$). $TRVS = \{(F, k) \mid$ *F is a formula for which there exists a* T-RES *refutation with variable space at most k.*$\}$

At first glance, T-RES variable space appears to be identical to RES variable space, since expanding the DAG underlying a RES refutation into a tree by copying the necessary subtrees may increase size exponentially, but appears to have no effect on variable space, since two copies of the same clause use exactly the same variable space. However, a problem arises when we closely examine the definition of a configuration-style T-RES proof, because it requires that any two parent clauses being resolved must be deleted immediately. Therefore, if we want to create multiple parallel copies of clauses in order to expand a RES proof into a T-RES proof, the extra copies of the parent clauses may violate the variable space of the corresponding step from the RES proof.

Note that this problem does not arise when expanding a RES refutation into a T-RES refutation in order to show that RES and T-RES width are identical.

With the following lemma, we are able to prove that the result in Theorem 6.4.2 also holds for $TRVS$:

Lemma 6.4.4. $RTRVS = TRVS$

Proof: Given a configuration-style T-RES refutation π with underlying proof tree T (where the nodes are labeled with clauses) and associated pebbling strategy S containing irregularities, it is possible to prune those irregularities in the standard way [Tse70, Urq95] to produce an RT-RES refutation π' with underlying proof tree T' and associated pebbling strategy S' such that $VS(\pi') \leq VS(\pi)$. This is because every node v' in T' has a corresponding node v in T such that $v' \subseteq v$, and some nodes in T do not have any corresponding node in T' because they were removed by the pruning procedure. This means that every configuration in π' contains clauses which are a subset of the clauses in the corresponding configuration of π, and some configurations in π do not have corresponding configurations in π' because they were removed by the pruning procedure. Therefore $VS(\pi') \leq VS(\pi)$, so for any formula F, $VS(F \vdash_{\textsf{RT-RES}} \emptyset) \leq VS(F \vdash_{\textsf{T-RES}} \emptyset)$, and $TRVS \subseteq RTRVS$.

The other direction is trivial, since every RT-RES refutation is a T-RES refutation: Clearly $VS(F \vdash_{\textsf{T-RES}} \emptyset) \leq VS(F \vdash_{\textsf{RT-RES}} \emptyset)$, so $RTRVS \subseteq TRVS$, and $RTRVS = TRVS$, as required. □

Corollary 6.4.5. *$TRVS$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{NEXPTIME}$.*

6.4.3 The Complexity of Input Resolution Variable Space

Another interesting related problem is that of I-RES variable space. Its corresponding language is formally defined as follows:

Definition 6.4.6 ($IRVS$)**.** $IRVS = \{(F, k) \mid F$ *is a formula for which there exists an* I-RES *refutation with variable space at most* $k.\}$

Corollary 6.4.7. *$IRVS$ is $\mathcal{P}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

Proof: To see that $IRVS$ is $\mathcal{P}$-Hard, we reduce from the $\mathcal{P}$-Complete $IRES$-$UNSAT$ problem from Corollary 5.3.4. Given a formula F, output $(F, k = n)$. Since any unsatisfiable formula has variable space at most n, $F \in UNSAT$ if and only if $(F, k) \in IRVS$.

To show that $IRVS \in \mathcal{PSPACE}$, we use essentially the same algorithm as in Corollary 5.6.10 by nondeterministically guessing configurations one at a time, keeping at most two configurations in memory at any time, where each contains at most two clauses, and finally appeal to Savitch's Theorem [Sav70]. □

6.5 Open Problems & Conjectures Related to Resolution Variable Space

Although we were able to prove the $\mathcal{PSPACE}$-Hardness of the RVS, we have been unable to show that it has a $\mathcal{PSPACE}$, $\mathcal{EXPTIME}$, or even $\mathcal{NEXPTIME}$ algorithm. Finding a tighter upper bound on RVS therefore remains an interesting open problem, which is somewhat surprising given that we have very powerful tools such as Savitch's Theorem [Sav70] to help us develop such algorithms. This suggests that perhaps RVS does not have a $\mathcal{PSPACE}$ algorithm, and it may be $\mathcal{EXPTIME}$- or even $\mathcal{NEXPTIME}$-Hard.

In addition, significant complexity gaps still exist for $RTRVS$, $TRVS$, and $IRVS$. We conjecture that $RTRVS$ and $TRVS$ both have $\mathcal{PSPACE}$ algorithms, but are hesitant to conjecture what the tightest hardness lower bounds might be. In any case, it would be interesting to get tighter results for all of these problems.

Chapter 7

The Complexity of Resolution Width

7.1 Introduction & Motivation

In its original incarnation, this chapter's main result showed that the problem of calculating RES width requirements is $\mathcal{EXPTIME}$-Complete. In keeping with the theme of this thesis, the proof proceeded by reducing from the $(\exists, k)$-pebble game of Kolaitis and Vardi [KV95] which was proved $\mathcal{EXPTIME}$-Complete by Kolaitis and Panttaja [KP03], itself proved $\mathcal{EXPTIME}$-Complete via a reduction from the KAI game [KAI79]. Our reduction was from the $(\exists, k)$-pebble game to the k-width game of Atserias and Dalmau [AD03], which they proved to be equivalent to RES width. In other words, it reduced a game for which the complexity is known to a game characterization of a proof complexity resource problem, just like the reduction from the black pebbling game to the Prover/Delayer game characterization of T-RES clause space in Chapter 4.

Unfortunately, we found a subtle but fatal flaw in our proof which invalidated this main result, leaving us with a decision to be made: Either delete this chapter completely, or remove the main result but retain its exposition and minor results. We settled for the latter option, since we believe that what remains ultimately still adds to this thesis. Even in its vestigial form, this chapter still contains some interesting results, including a restatement and modification of the width game due to Atserias and Dalmau [AD03] which we use to show that RES and R-RES width are subtly different, and also to help prove both upper and lower bounds for the RES and R-RES width problems. In addition, we show that there is no difference between width for DAG-like and tree-like forms of Resolution, which gives upper and lower bounds for the T-RES and RT-RES width problems respectively corresponding to those for RES and R-RES width. These results show that width is blind to the normally important difference between tree-like and DAG-like proofs, but is sensitive to the normally less-important difference between regular and irregular proofs.

However, before giving these results, we first review the importance of width for Resolution-based proof systems and explain how the width problem is related to the rest of this thesis:

The width of proofs has played a key part in investigations of the complexity of RES, starting with Tseitin [Tse70] and Galil [Gal77]. In these early papers, and again in the seminal paper by Haken [Hak85], lower bounds on the size of RES proofs rest on width lower bounds.

This connection between size and width was systematized and explained in a remarkable paper by Ben-Sasson and Wigderson [BSW01]. In that article the authors prove a general theorem, of which one consequence is that if Σ is a contradictory set of 3-CNF clauses, then the minimal size of a RES (T-RES) refutation of Σ is exponential in the minimal RES (T-RES) width of such a refutation. These results are restated in this thesis as Theorems 3.3.12 and 3.3.13 and provide the major theoretical motivations for understanding the problem of deciding whether or not a set of clauses has a proof of a given width.

However, there are also strong practical motivations from the areas of automated theorem proving and propositional reasoning for understanding this problem. As well as their important theoretical results, Ben-Sasson and Wigderson in [BSW01] propose a simple dynamic programming procedure for automated theorem proving, one which simply searches for proofs with small width. In addition, the practical motivations for studying RES width are very similar to those behind studying T-RES clause space in Chapter 4: Although SAT-solvers have been remarkably successful, there are some inputs on which they fail. Researchers working with SAT-solvers may therefore be tempted to design preprocessing algorithms which determine the width required to refute a given unsatisfiable formula F. Since we know from [BSW01] that short proofs are narrow, if the minimum width of any refutation of F is sufficiently large, then any Resolution-based SAT-solver will fail by not halting in a feasible amount of time, so researchers would be able to tell ahead of time whether or not to bother trying to solve F, potentially saving a great deal of time and computer resources. Before the discovery of the flaw in our main result, any hopes for such a preprocessing algorithm are dashed by this chapter, but the question of whether or not a width solver exists is once again open.

The width problem for RES is as follows: Given a set of clauses Σ, and integer k as input, determine whether or not there is a RES refutation of Σ of width at most k. This problem was conjectured to be $\mathcal{EXPTIME}$-complete by Moshe Vardi and conveyed to us by Stephen Cook.

This chapter is organized as follows: In Section 7.2 we describe the k-width game, which was shown by Atserias and Dalmau in [AD03] to characterize RES width. Next, in Section 7.3, we modify the RES width game in order to characterize R-RES width. This is followed by Section 7.4, in which we give both upper and lower bounds for the RES, T-RES, R-RES, and RT-RES width problems. Finally, in Section 7.5 we review a number of open problems related to this area of research.

7.2 A Game Characterization of Resolution Width

In this section, we give a slightly modified proof of a result by Atserias and Dalmau which characterizes the width of RES refutations [AD03]. The characterization is in terms of a two-player game which we shall call the 'k-width game', and it is played between two opponents called the Prover and the Adversary.[1] The game is played as follows:

The players are given a set of clauses Σ, on a set V of variables, and an integer parameter $k > 0$. They then together construct a succession of assignments to the variables V. Initially, the assignment is empty; at every round of the game, all assignments involve at most k variables. Each round of the game proceeds as follows: First, the Prover can delete zero or more of the variable assignments from a previous round. Second, the Prover queries an unassigned variable, and the Adversary (re)assigns a value to it.

The Prover wins if the current assignment falsifies an initial clause in Σ. The Adversary wins if an earlier assignment is repeated during the play of the game. In this we depart slightly from the game as defined by Atserias and Dalmau; in their version it is possible for the game to continue indefinitely, in which case the Adversary wins.

Since there are only finitely many possible truth assignments, every play of the game must eventually terminate in a win for the Prover or for the Adversary. It follows that either the Prover or the Adversary must have a winning strategy.

Definition 7.2.1 (Extendible k-Family of Assignments). *If Σ is a set of clauses on a set V of variables, then a non-empty family $\mathcal{F}$ of V-assignments is an extendible k-family for Σ if it satisfies the following conditions:*

1. *No assignment in $\mathcal{F}$ falsifies a clause in Σ,*
2. *Each assignment α in $\mathcal{A}$ satisfies the condition $|\alpha| \leq k$;*
3. *If $\alpha \in \mathcal{F}$, and $\beta \subseteq \alpha$, then $\beta \in \mathcal{F}$, and*
4. *If $\alpha \in \mathcal{F}$, $|\alpha| < k$, and $x \in V$, then there is a $\beta \in \mathcal{F}$, so that $\alpha \subseteq \beta$, and $\beta(x)$ is defined.*

[1] Atserias and Dalmau, following the tradition of finite model theory, call their players the Spoiler and the Duplicator, but our terminology seems clearer in the present context.

The next theorem was proved by Atserias and Dalmau [AD03], and shows that a RES refutation of width k constitutes a winning strategy for the Prover, while an extendible $k+1$-family provides a winning strategy for the Adversary in the $k+1$-width Resolution game:

Theorem 7.2.2 ([AD03]). *Let Σ be a contradictory set of clauses, and $k \geq w(\Sigma)$. Then the following are equivalent:*

1. *There is no* RES *refutation of Σ of width k,*
2. *There exists an extendible $k+1$-family for Σ, and*
3. *The Adversary wins the $k+1$-width game based on Σ.*

Proof: ($1 \Rightarrow 2$): First, let us suppose that there is no RES refutation of Σ of width k. Define $\mathcal{C}$ to be the set of all clauses which have a RES proof from Σ of width at most k; by assumption, $\Sigma \subseteq \mathcal{C}$. Let $\mathcal{F}$ be the set of all assignments of size at most $k+1$ that do not falsify any clause in $\mathcal{C}$. We claim that $\mathcal{F}$ is an extendible $k+1$-family for Σ. First, $\mathcal{F}$ is non-empty, because it contains the empty assignment (since $\mathcal{C}$ does not contain the empty clause). Second, $\mathcal{F}$ satisfies the first three conditions of Definition 7.2.1, by construction. To prove the fourth condition, let $\alpha \in \mathcal{F}$, and $|\alpha| \leq k$, $x \in V$, but assume that there is no extension β of α in $\mathcal{H}$ with $\beta(x)$ defined. It follows that there is a clause $D \in \mathcal{C}$ that is falsified if we extend α by setting x to $False$. Then $D = E \vee x$ for some E, since otherwise α would falsify D. Similarly, there is a clause $F \vee \overline{x}$ in $\mathcal{C}$ that is falsified by the extension of α that sets x to $True$. Then α must falsify $E \vee F$; but $E \vee F$ is in $\mathcal{C}$, contradicting our assumption.

($2 \Rightarrow 3$): Secondly, let us suppose that there exists an extendible $k+1$-family $\mathcal{F}$ for Σ. Then the Adversary can play the $k+1$-width game on Σ by responding to the Prover's queries with the appropriate assignment from the family, starting with the empty assignment. Since no assignment in the family falsifies an initial clause, this strategy must eventually end in a win for the Adversary, no matter how the Prover plays.

($3 \Rightarrow 1$, Contrapositive): Finally, let us suppose that there is a RES refutation of Σ of width k. Then the refutation provides the Prover with a winning strategy in the $k+1$-width game based on Σ. Starting from the empty clause at the root, the Prover follows a path in the refutation to one of the leaves in the refutation. At each round, the current assignment (after appropriate deletions), is a minimal assignment falsifying a clause in the path. The variable queried is the variable resolved upon to derive the current clause, and the next clause in the path is one of the premisses of the clause from the previous round. Since the refutation has width k, every assignment has size bounded by $k+1$, so this strategy must result in a win for the Prover. $\square$

7.3 A Game Characterization of Regular Resolution Width

Another interesting Resolution resource problem related to this research is that of determining the complexity of R-RES width. It is not hard to modify Atserias and Dalmau's width characterization described in Section 7.2 to give a game characterization of R-RES width. The characterization is again in terms of a two player game which we shall call the 'regular k-width game'. The game is exactly the same as the k-width game, but with the added condition that the Prover can never query a previously queried variable.

Once again, the Adversary can win in two ways: Firstly, if the current assignment assigns values to at least k variables; secondly, if the Prover has not won up to this point, but there are no unqueried variables, so the Prover has no legal move. The Prover wins if the current assignment falsifies an initial clause in Σ (if this clause contains k variables, then we count this as a win for the Adversary). As before, every play of the game must eventually terminate with a win for the Prover or for the Adversary.

As in the case of RES width, we can characterize R-RES width in terms of extendible families of assignments. However, we need to redefine the notion of an assignment. In the earlier notion of assignment, a variable could be in three states: Positive ($True$), negative ($False$), and unassigned ($*$). For the case of R-RES, we define an extended assignment as follows:

Definition 7.3.1 (Extended Assignment, Live & Dead Variables). *An extended assignment is an assignment of values in which each variable can be in four states:*

1. *Positive (True),*
2. *Negative (False),*
3. *Unassigned ($*$), and*
4. *Dead (☠)*

The empty extended assignment to a set V of variables consists of the assignment in which all variables in V are unassigned ($*$), which should be distinguished from assignments where all of the variables are unassigned or dead (☠)).

If α is an extended assignment, then those variables that are assigned the values $True$ or $False$ are the live variables in α, and we write $|\alpha|$ for the number of live variables in α. If α and β are extended assignments to a set of variables V, then we write $\alpha \subseteq \beta$ if α results from β by replacing some unassigned variables by live variables. We also write $\alpha \sqsubseteq \beta$ if β results from α by replacing some dead variables by live variables.

This brings us to our concept of a regular extendible k-family of assignments; note the similarity to Definition 7.2.1:

Definition 7.3.2 (Regular Extendible k-Family of Assignments). *If Σ is a set of clauses on a set V of variables, then a family $\mathcal{A}$ of extended V-assignments is a* regular extendible k-family for Σ *if it satisfies the following conditions:*

1. *The empty assignment belongs to $\mathcal{A}$,*
2. *No assignment in $\mathcal{A}$ falsifies a clause in Σ,*
3. *Each assignment α in $\mathcal{A}$ satisfies the condition $|\alpha| \leq k$,*
4. *If $\alpha \in \mathcal{A}$, and $\beta \sqsubseteq \alpha$, then $\beta \in \mathcal{A}$, and*
5. *If $\alpha \in \mathcal{A}$, $|\alpha| < k$, $x \in V$, and $\alpha(x) = *$, then there is a $\beta \in \mathcal{A}$, so that $\alpha \subseteq \beta$, and $\alpha(x)$ is defined.*

The next theorem is the analogue of Theorem 7.2.2, but for R-RES. Note that the extended assignment associated with each round of the game acts as follows: At the beginning of a play of the game, the current extended assignment is empty. Let us suppose that the current extended assignment is α. Then if the Prover deletes an assignment to a live variable x in α, then x is dead in the next assignment, that is to say, we change $\alpha(x)$ from $True$ or $False$ to ☠.

Theorem 7.3.3. *Let Σ be a contradictory set of clauses, and $k \geq w(\Sigma)$. Then the following are equivalent:*

1. *There is no* R-RES *refutation of Σ of width k,*
2. *There is a regular extendible $k+1$-family for Σ, and*
3. *The Adversary wins the regular $k+1$-width game played on Σ.*

Proof: ($1 \Rightarrow 2$): Let us suppose that there is no R-RES refutation of Σ of width k, and that α is an extended assignment to the variables in Σ. We define such an assignment to be *bad* if it satisfies the following condition: There is a R-RES derivation π of a clause C from Σ so that π has width at most k, $\alpha(C) = 0$, and if $\alpha(x) =$ ☠, then x is never resolved on in π. An assignment is *good* if it is not bad. Define $\mathcal{A}$ to be the family of all good assignments α, with $|\alpha| \leq k+1$. We claim that $\mathcal{A}$ is a regular extendible $k+1$-family for Σ.

The fact that $\mathcal{A}$ satisfies the first and second conditions of Definition 7.3.2 follows from our assumption that that there is no R-RES refutation of Σ of width k. The third condition holds by definition. For the fourth condition, let us assume that $\alpha \in \mathcal{A}$, and $\beta \sqsubseteq \alpha$, but $\beta \notin \mathcal{A}$. Since $|\beta| \leq |\alpha|$, it follows that β is bad. However, in that case, α must be bad as well, because all of the variables that are dead in α are also dead in β.

Finally, we need to show that the family $\mathcal{A}$ satisfies the fifth and final condition in Definition 7.3.2. Assume that $\alpha \in \mathcal{A}$, $|\alpha| \leq k$, and $\alpha(x) = *$, but there is no $\beta \in \mathcal{A}$, so that $\alpha \subseteq \beta$, and $\alpha(x)$ is defined. Let α^0 and α^1 be the extended assignments that result from α by changing $\alpha(x)$ from $*$ to 0 and 1 respectively. Then both α^0 and α^1 must be bad. Thus there are R-RES derivations π^0 and π^1 of clauses C_0 and C_1, each having width at most k, so that for $i = 0, 1$, $\alpha^{i(C_i)} = 0$, and no variable that is dead in α is resolved upon in either of the two derivations.

Since α is good by assumption, it follows that $C_0 = D \vee x$, and $C_1 = E \vee \overline{x}$, for some clauses D and E. However, if we extend the R-RES derivations π^0 and π^1 by resolving on x, so that the final clause is $D \vee E$, the result is a R-RES derivation of $D \vee E$, where $\alpha(D \vee E) = 0$. However, since $|\alpha| \leq k$, it follows that α is bad, contrary to assumption. This completes the proof of the fifth condition in Definition 7.3.2, showing that $\mathcal{A}$ is a regular extendible $k+1$-family for Σ.

($2 \Rightarrow 3$): Secondly, let us suppose that there is an extendible $k+1$-family for Σ. Then the Adversary can play the k-width game on Σ by responding to the Prover's queries with the appropriate assignment from the family, starting with the empty assignment. When the Prover removes a live variable from the current assignment, then the Adversary sets it to ☠. Because of the regularity restriction, a variable, once dead, can never be queried again. Since no assignment in the family falsifies an initial clause, this strategy must eventually end in a win for the Adversary, no matter how the Prover plays.

($3 \Rightarrow 1$, Contrapositive): Finally, let us suppose that there is a R-RES refutation of Σ of width k. Then the refutation provides the Prover with a winning strategy in the regular $k+1$-width game based on Σ: Starting from the empty clause at the root, the Prover follows a path in the refutation so that at each round, the assignment (after appropriate deletions) is a minimal assignment falsifying the current clause. The variable queried is the variable resolved upon to derive the current clause. This strategy must result in a win for the Prover. □

7.4 The Complexities of Several Resolution Width Problems

In this section we give upper and lower bounds for the RES, T-RES, R-RES, and RT-RES width problems.

7.4.1 The Complexities of Resolution and Tree Resolution Width

The language associated with the RES width problem is formally defined as follows:

Definition 7.4.1 (RES-$WIDTH$). *RES-$WIDTH = \{(F, k) \mid F$ is a formula for which there exists a* RES *refutation with width at most k.$\}$*

Although a gap still exists, we are able to use Theorem 7.2.2 due to Atserias and Dalmau to show both upper and lower bounds for RES-$WIDTH$:

Corollary 7.4.2. *RES-$WIDTH$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{EXPTIME}$.*

Proof: To prove the co$\mathcal{NP}$-Hardness of RES-$WIDTH$, we reduce from $UNSAT$ as in Corollary 4.8.4. For our reduction, we are given a formula F and output $(F, k = n)$, where n is the number of distinct variables in F. Since n is the maximum width of any clause, it follows immediately that $F \in UNSAT$ if and only if $(F, k) \in RES$-$WIDTH$.

To show that RES-$WIDTH \in \mathcal{EXPTIME}$, we show that the k-width game above is in $\mathcal{EXPTIME}$ and appeal to Theorem 7.2.2: On a given play of the game, it is possible to keep track of the number of current assignments that have appeared up to a given round, and so determine if a repetition has occurred. Since there are at most $N = \binom{n}{k}3^k$ possible assignments, where n is the number of variables in the clause set Σ employed in the game, when the count reaches $N + 1$, a repetition must have occurred.

Consequently, the description of the game given above shows that there is an alternating Turing Machine operating in polynomial space that determines whether the Prover or the Adversary wins a given instance of the game. Hence, the problem is in $\mathcal{EXPTIME}$. □

Despite the great difference between RES and T-RES with respect to proof size, it turns out that the T-RES width problem is identical to the RES width problem. We formally define the language associated with T-RES width as follows:

Definition 7.4.3 ($TRES$-$WIDTH$)**.** $TRES$-$WIDTH = \{(F,k) \mid F$ *is a formula for which there exists a* T-RES *refutation with width at most* $k.\}$

It is not hard to see that RES-$WIDTH$ and $TRES$-$WIDTH$ are in fact the same language:

Lemma 7.4.4. RES-$WIDTH = TRES$-$WIDTH$

Proof: It suffices to show that for any formula F, $w(F \vdash_{\mathsf{RES}} \emptyset) = w(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$. The forward direction is trivial, since any T-RES refutation is a RES refutation. In the reverse direction, if we are given a RES refutation π of F with width at most k, then we can easily construct a T-RES refutation of F with width k by simply turning the DAG underlying the proof tree of π into a tree by duplicating subtrees as necessary in the obvious way. This may lead to an exponential increase in proof size, but does not introduce any new clauses, and therefore does not affect the proof width at all. This means that RES-$WIDTH = TRES$-$WIDTH$, as required. □

An immediate corollary of this lemma is that the lower and upper bounds for RES-$WIDTH$ from Corollary 7.4.2 also hold for $TRES$-$WIDTH$:

Corollary 7.4.5. *$TRES$-$WIDTH$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{EXPTIME}$.*

In addition, Lemma 7.4.4 also shows that Theorems 3.3.12 and 3.3.13 by Ben-Sasson and Wigderson [BSW01] apply equally to RES and T-RES width.

7.4.2 The Complexities of Regular and Regular Tree Resolution Width

If asked whether regularity has any impact on the width of refuting a formula, Lemma 7.4.4 and the fact that irregularities can often be pruned with no ill effects (see [Tse70, Urq95]) might give one the initial impression that it does not. After all, the p-simulation hierarchy in Figure 2.2 shows us that RES is strictly stronger than R-RES, which in turn is strictly stronger than T-RES and RT-RES, so if RES and T-RES width are identical, then why should R-RES, which is sandwiched between them in terms of p-simulation be any different? Unfortunately, this intuition is wrong, and it is easy to confuse regular and general width. It turns out that the width of R-RES and RT-RES is subtly different from that of RES and T-RES, and in this section we give upper and lower bounds on the complexities of width problems, only this time for the regular versions.

The width problem for R-RES is formally defined as follows:

Definition 7.4.6 ($RRES$-$WIDTH$)**.** $RRES$-$WIDTH = \{(F,k) \mid F$ *is a formula for which there exists a* R-RES *refutation with width at most* $k.\}$

Just as we used the width game from Section 7.2 and its equivalence to RES width to give bounds on the computational complexity of RES-$WIDTH$, so do we now use the regular width game from Section 7.3 and its equivalence to R-RES width to give bounds on $RRES$-$WIDTH$:

Corollary 7.4.7. *$RRES$-$WIDTH$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

Proof: The reduction showing the co$\mathcal{NP}$-Hardness of $RRES$-$WIDTH$ is identical to that of RES-$WIDTH$ in Corollary 7.4.2: Simply reduce from $UNSAT$, taking a formula F, and outputting $(F, k = n)$, where n is the number of distinct variables in F. Since n is the maximum width of any clause, it follows immediately that $F \in UNSAT$ if and only if $(F, k) \in RRES$-$WIDTH$.

In order to prove that $RRES$-$WIDTH \in \mathcal{PSPACE}$, we appeal to Theorem 7.3.3, which shows that this question can be answered by an alternating Turing Machine operating in polynomial time. Alternating polynomial time is equivalent to $\mathcal{PSPACE}$, so our claim that $RRES$-$WIDTH \in \mathcal{PSPACE}$ follows. $\square$

With respect to this proof, an important subtlety in the case of RES-$WIDTH$ is that it is not clear whether it is in $\mathcal{PSPACE}$, because there is no polynomial upper bound on how long the k-width game might last.

In Section 7.4.1 we saw that RES and T-RES width are identical, and the same holds true for R-RES and RT-RES; the language associated with the latter is formally defined as follows:

Definition 7.4.8 ($RTRES$-$WIDTH$). $RTRES$-$WIDTH = \{(F, k) \mid F$ *is a formula for which there exists a* RT-RES *refutation with width at most* k.$\}$

The proof that R-RES and RT-RES width are equal is identical to that of Lemma 7.4.4:

Corollary 7.4.9. $RRES$-$WIDTH = RTRES$-$WIDTH$

Since $RRES$-$WIDTH = RTRES$-$WIDTH$, the upper and lower bounds of Corollary 7.4.7 immediately apply to $RTRES$-$WIDTH$ as well:

Corollary 7.4.10. *$RTRES$-$WIDTH$ is co$\mathcal{NP}$-Hard under logspace reducibility, but is also in $\mathcal{PSPACE}$.*

7.5 Open Problems & Conjectures Related to Resolution Width

With the flaw that we found invalidating our original proof that RES-$WIDTH$ and $TRES$-$WIDTH$ are $\mathcal{EXPTIME}$-Complete, this has once again become an interesting open problem. We believe that the result is still true, and have an idea for how to prove it, but this will have to be a post-thesis project.

The other main open problem related to this research is the complexity of $RRES$-$WIDTH$. Dr. Barnaby Martin at the University of Durham in England has claimed that $RRES$-$WIDTH$ is $\mathcal{PSPACE}$-Complete and gave a proof sketch at the 2007 London Algorithmic Workshop (LAW 2007) [Mar07]. We believe that this statement is correct, and if this is true, then of course $RTRES$-$WIDTH$ is also $\mathcal{PSPACE}$-Complete by lemma 7.4.9.

Since T-RES and RT-RES are p-equivalent and R-RES is strictly stronger that T-RES as a proof system, showing RES / T-RES width to be $\mathcal{EXPTIME}$-Complete, and R-RES / RT-RES width to be $\mathcal{PSPACE}$-Complete would be an interesting deviation from the standard proof complexity hierarchy.

Another interesting corollary of the potential $\mathcal{EXPTIME}$-Completeness of RES-$WIDTH$ concerns a special case of the $(\exists, k)$-pebble game of Kolaitis and Vardi [KV95] which was proved $\mathcal{EXPTIME}$-complete by Kolaitis and Panttaja [KP03]. Albert Atserias has remarked that a reduction from the $(\exists, k)$-pebble game to RES-$WIDTH$ would also settle the complexity of the fixed template version of the $(\exists, k)$-pebble game. This remark is based on an observation by Feder and Vardi [FV98] (see also [Ats04]) showing that the satisfiability problem for formulas in r-CNF can be encoded as a constraint satisfaction problem in the form where the target structure $\mathcal{B}$, or 'template' is fixed, while only the source structure, or 'instance' varies.

Yet another interesting corollary of the potential $\mathcal{EXPTIME}$-Completeness of $RES\text{-}WIDTH$ concerns tradeoffs between different resources. If we can prove that $RES\text{-}WIDTH$ is $\mathcal{EXPTIME}$-Complete, and make the assumption that every formula F having a refutation with width w also has a refutation with width w and clause space $|F|^{O(1)}$, then we can prove that $\mathcal{PSPACE} = \mathcal{EXPTIME}$, which is probably false, thereby showing that there are formulas with minimal width which do not have polynomially-bounded clause space. The proof is easy: If formulas with minimal width also had polynomially-bounded clause space, then we could nondeterministically guess the refutations using only polynomial space. Appealing to Savitch's Theorem [Sav70] gives us a $\mathcal{PSPACE}$ algorithm, thereby showing that there exists an $\mathcal{EXPTIME}$-Complete problem with a $\mathcal{PSPACE}$ algorithm, so $\mathcal{PSPACE} = \mathcal{EXPTIME}$.

Of course, this proof also holds for width and total space, and for any resource problem which is at least $\mathcal{EXPTIME}$-Hard. For example, if RES variable space or size are $\mathcal{EXPTIME}$-Hard, then there is a similar tradeoff between those resources and clause/total space. This proof therefore has the potential to yield numerous tradeoff results conditioned on the commonly-believed separations in the standard complexity hierarchy.

Part III

Proof Complexity Size Results & Dangerous Reductions

Chapter 8

Introduction to Part III

This chapter serves as an introduction to Part III of this thesis, which is focused on the potential danger that reductions represent from the point of view of proof complexity and SAT-solvers. We start in Section 8.1 below by describing what 'dangerous reductions' are and explaining why they are important. This is followed by Section 8.2 in which we give definitions specific to this part of the thesis. Finally, in Section 8.3 we shall give a summary of the next four chapters in which we prove this part's main results.

8.1 Description of Dangerous Reductions

Because there are so many $\mathcal{NP}$-Hard problems which come up in practice, it is not feasible for computer scientists to develop and implement algorithms for them all. One established major approach which researchers have employed for these problems is to focus their efforts on designing and implementing highly-optimized SAT-solvers, and then rely on the robust expressiveness of the SAT language so that they only need to implement reductions from the various different domains to SAT. Since reductions are so easy to develop, and since SAT-solvers have become so sophisticated and powerful, this strategy has proved to be quite successful.

Nevertheless, this approach is not without its pitfalls. Researchers have noted empirically that not all reductions from an input domain to SAT are equal, and that it is possible for a poor choice of encoding to map easy instances from the input language to hard SAT instances. For example, Kautz, McAllester and Selman investigate numerous approaches for translating planning problems to SAT in [KMS96]. Some of these translations are found to result in formulas which are much harder to solve than others, and the problem of better understanding these 'dangerous reductions' is listed as one of the ten important and challenging open problems in the area [KMS97, KS03].

In this part of the thesis we will combine results from several seemingly unrelated areas of research to give the first formal example of a 'dangerous reduction'. Of course, this phenomenon also has a flip side, and we also give a formal example proving that it is possible for reductions to map hard instances from the input domain to easy SAT instances. This shows that reductions can also be beneficial.

8.2 Definitions Specific To Part III

There are two important definitions which come up repeatedly in this part of the thesis, particularly in Chapters 9, 10, and 11. The first is an important reduction, and the second is the definition of two important families of graphs.

8.2.1 A SAT Encoding For The Hamiltonian Cycle Problem

The following reduction is from the $\mathcal{NP}$-Complete Hamiltonian Cycle problem to SAT, and proceeds as follows: Take a graph $G = (V, E)$ and create a formula F which enforces a mapping from V to the positions

p_1, p_2, ..., $p_{|V|}$ of a Hamiltonian Cycle H. Intuitively, H can be thought of as a cyclic ordering on the $|V|$ vertices of G, where vertex i can be mapped to the j^{th} position in H if i is adjacent in G to the vertices mapped to positions $j-1$ and $j+1$. F contains variables of the form $m_{i,j}$, each of which is interpreted as meaning that element i from G (the domain) is mapped to position j in the Hamiltonian Cycle H (the range). F partially consists of clauses which enforce a bijection between V and H. A conjunction of the following groups of clauses ensures such a bijection:

Total: $\bigwedge_{i=1}^{|V|}(\bigvee_{j=1}^{|V|} m_{i,j})$ *i.e. Every vertex in V maps to at least one position in H.*

Onto: $\bigwedge_{j=1}^{|V|}(\bigvee_{i=1}^{|V|} m_{i,j})$ *i.e. Every position in H has at least one vertex mapped to it.*

1-1: $\bigwedge_{j=1}^{|V|}\bigwedge_{i_1=1}^{|V|}\bigwedge_{\substack{i_2=1\\ i_1\neq i_2}}^{|V|}(\neg m_{i_1,j}\vee\neg m_{i_2,j})$ *i.e. At most one vertex maps to each position.*

Fn.: $\bigwedge_{i=1}^{|V|}\bigwedge_{j_1=1}^{|V|}\bigwedge_{\substack{j_2=1\\ j_1\neq j_2}}^{|V|}(\neg m_{i,j_1}\vee\neg m_{i,j_2})$ *i.e. Every vertex maps to at most one position.*

To ensure that F is satisfiable if and only if G is Hamiltonian, we also need to add the following clauses which place constraints corresponding to the structure of G's edges on the bijection:

Edge: $\bigwedge_{j=1}^{|V|}\bigwedge_{i=1}^{|V|}\bigwedge_{\substack{k:(i,k)\notin E\\ i\neq k}}(\neg m_{i,j}\vee\neg m_{k,(j+1)\bmod |V|})$

Informally, the edge constraint clauses ensure that for every non-edge (i,k), if vertex i has been mapped to the j^{th} position in the cycle H, then vertex k cannot be mapped to position $j+1$ (mod $|V|$). It is not hard to see that the reduction is correct: If G is Hamiltonian, then these edge constraints will not cause a contradiction with the clauses enforcing the bijection, so F will be satisfiable. Likewise, if F is satisfiable, then it means that there is a bijection from V to H which respects the constraints enforced by the edge clauses, so G must be Hamiltonian.

Of course, the total, onto, 1-1, and function clauses are more than enough to ensure a bijection. In fact, the total and 1-1 clauses by themselves are sufficient, as are the onto and function clauses by themselves. This leads us to define some notation:

Definition 8.2.1 ($H(G)$ Formulas). *For any graph G, the formula $H(G)$ is the result of applying the above reduction. To this we add a subscript showing which clause groups were used in its construction. We abbreviate total as T, onto as O, 1-1 as 1, and function as F. For example, if we used clauses from the total and 1-1 groups, then the formula is labelled as $H(G)_{T,1}$. Since every variation of this encoding requires the edge clauses, there is no need to specify them among the subscripts.*

8.2.2 Important Families of Non-Hamiltonian Graphs

In addition to the reduction described in the previous section, our results in Chapters 9, 10, and 11 repeatedly make reference to certain families of non-Hamiltonian graphs:

The K_n^* Graphs

Definition 8.2.2 (K_n^* Graphs). *The graph K_n^* consists of the complete graph on n vertices, K_n, with the addition of a single degree-0 vertex.*

Note that since each K_n^* is disconnected, it is trivially non-Hamiltonian. Examples of K_n^* graphs are shown below in Figure 8.1.

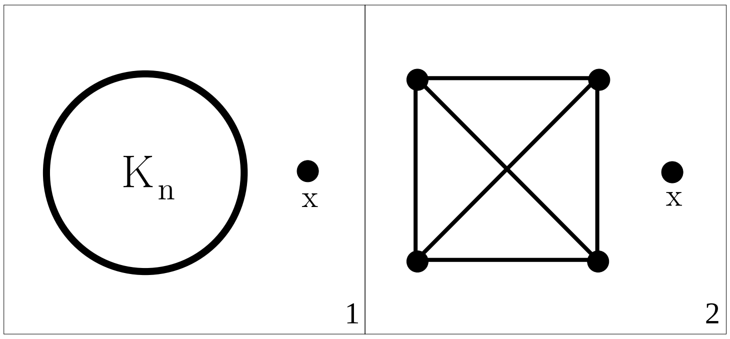

Figure 8.1: K_n^* (Left) and K_4^* (Right)

The $G_{\frac{n}{2},\frac{n}{2}}$ Graphs

Some other important families of non-Hamiltonian graphs are shown below in Figure 8.2. Although these graphs are very dense, they are nevertheless non-Hamiltonian. This is obvious for the first two families, but not quite as obvious for the third. In Chapter 9 we show that our Non-Hamiltonicity Proof System (NHPS) has exponential size lower bounds for each of these families of graphs.

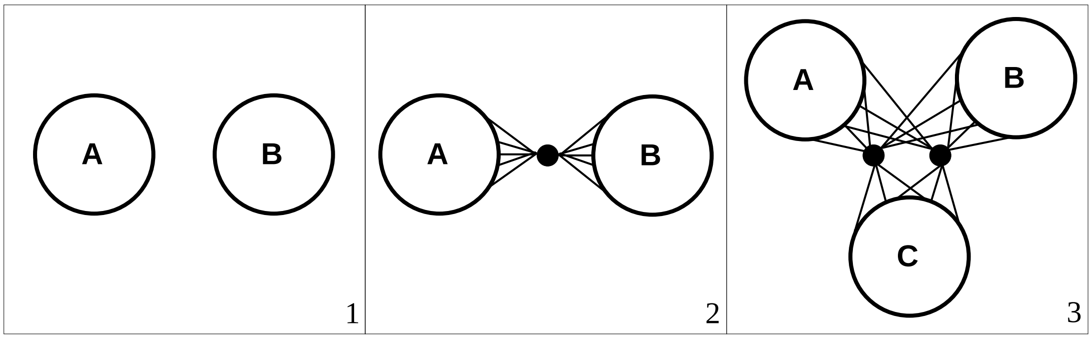

Figure 8.2: Graph Families Requiring Exponentially Long NHPS Proofs

The sets labelled 'A', 'B', and 'C' represent complete subgraphs, each containing $O(n)$ vertices. The family represented by Figure 8.2.1 contains all disconnected graphs containing a constant number of complete components such that each component contains $O(n)$ vertices. The family represented by Figure 8.2.2 contains all graphs consisting of a constant number of size $O(n)$ complete components separated by a single cut vertex which is adjacent to all other vertices. Finally, the family represented by Figure 8.2.3 contains all graphs containing a constant number $q \geq k+1$ of size $O(n)$ complete components which are held together by k vertices that are incident on all other vertices. A graph is said to be 1-tough if it contains no subset of k vertices, that, if removed, would disconnect the graph into $k+1$ or more components. All of the graphs in these families are non-Hamiltonian because none of them are 1-tough, which is a necessary condition for Hamiltonicity [Her04]. In fact, family 3 subsumes families 1 and 2, but the first two families also happen to violate biconnectivity, making it worthwhile to study them separately.

We are especially interested in a special case of the first family above:

Definition 8.2.3 (**$G_{\frac{n}{2},\frac{n}{2}}$ Graphs**). *The $G_{\frac{n}{2},\frac{n}{2}}$ graphs are all graphs containing an even number of vertices and consisting of two disjoint cliques of size $\frac{n}{2}$.*

As with the K_n^* graphs, the $G_{\frac{n}{2},\frac{n}{2}}$ graphs are disconnected and therefore are trivially non-Hamiltonian.

8.3 Summary of Part III

Much like the space results from Part II, the results in this part of the thesis once again apply combinatorial games to propositional proof complexity, and are strongly-related to and motivated by the area of automated theorem proving and propositional reasoning. This part of the thesis can be viewed in two ways. On one hand, the results in its four chapters are quite diverse. For example, it is not immediately obvious how our Non-Hamiltonicity Proof System, Prover/Delayer game upper bounds, and results concerning intuitionistic logic are in any way related to dangerous reductions. They are motivated individually, and it is easy to view them in isolation from each other. However, when combined they contribute to the dangerous reductions research in interesting ways, and it is useful to look at these results in this context as well.

In Section 8.3.1 below we will give a summary of the results individually, and in Section 8.3.2 we describe them in the broader context of dangerous reductions.

8.3.1 Summary of the Individual Results in Part III

Our first results in this part of the thesis are found in Chapter 9, which contains a formalization of a graphical proof system for non-Hamiltonicity, or NHPS for short, together with proofs of soundness, completeness, exponential lower bounds, and results relating it to other proof systems. The NHPS came from the analysis of an algorithm for determining non-Hamiltonicity, and the overall motivation behind this research was to better understand graphical proof systems, as well as general limitations of algorithms of this type. Specifically, since the NHPS deals with local configurations in the graph, the goal was to prove lower bounds for graph algorithms which involved ideas such as local search. While we did not achieve this goal, our investigations motivated the search for a simulation result with T-RES, which necessarily required the use of a reduction. The development of such a reduction from the Hamiltonian Cycle problem to SAT led to the result concerning 'dangerous reductions' in Chapter 11.

The NHPS is of interest primarily because of its relationship with other parts of this thesis. In particular, it is important to the results in Chapter 11. That being said, it is also a novelty and of interest in its own right because most proof systems are for propositional logic, and to date no proof systems for non-Hamiltonicity have been developed. A version of the NHPS has been implemented, and it performs very well in practice.

Chapter 10 concerns the relationship between T-RES proof size and the Prover/Delayer game from Chapter 4, which has been used in the literature to prove T-RES size lower bounds [BSIW04]. We prove that the Prover/Delayer game can also be used to prove upper bounds on T-RES size. This is useful several times in this part of the thesis, and is also interesting because it adds another dimension to the relationship between T-RES and the Prover/Delayer game, since it is also known that the Prover/Delayer game can be used to prove both upper and lower bounds on T-RES clause space [ET03]. This chapter shows that it can also be used to prove upper bounds on T-RES proof size, providing the final result needed to show that it perfectly captures both the size and clause space characteristics of T-RES. In addition, we use this upper bound to show that this game can also be used to simplify T-RES proofs.

In Chapter 11 we formalize the concept of dangerous reductions. Much like our space results from Part II, this research is particularly relevant to the area of automated theorem proving and SAT-solving. As already mentioned in Section 8.1, SAT's highly expressive nature has allowed for many $\mathcal{NP}$-Hard problems to be encoded as SAT instances, enabling research into solving these varied problems to be concentrated on SAT-solving. In fact, many problems have numerous different SAT encodings to choose from, and some of them are considerably better than others. Unfortunately, some encodings take easy instances from one domain and turn them into intractable SAT instances. Similarly, some encodings take very hard instances from one domain and turn them into easy SAT instances. We respectively refer to these as explosive and implosive reductions.

Although explosive and implosive reductions have been encountered many times empirically [KMS96, KMS97, KS03, BB03], in the first part of this chapter we provide the first formally proven non-trivial example of an explosive reduction by showing that slightly different versions of the reduction from the Hamiltonian Cycle problem to SAT from Section 8.2.1 give results with drastically different proof complexities. Following that, we also provide the first formal non-trivial example of an implosive reduction, ultimately leading to the second part of this chapter, in which we develop a framework for comparing different encodings with respect to how beneficial or harmful they are.

This research is of interest because it gives the first formal examples of dangerous as well as beneficial reductions. An exponential separation between two similar and natural reductions is of clear theoretical interest, but of course has immense practical implications because SAT-solvers are heavily dependent on translations from other domains, and to date there has been little progress in dealing with the issue of comparing competing reductions.

Our final result is contained in Chapter 12, where we explore the proof complexity of intuitionistic logic, probably the best-studied non-classical logic. Intuitionistic logic has long been associated with the notion of constructivism in mathematics, since it disallows proofs by contradiction, and therefore requires proofs to be constructive. Recent interest in intuitionistic logic has been sparked by Pavel Hrubeš's remarkable lower bounds for intuitionistic Frege systems [Hru07]. Another motivating factor behind this type of logic is well-known paper by Statman [Sta79], which shows via a natural reduction from QBF (itself shown to be $\mathcal{PSPACE}$-Complete in [Sto76]) that the problem of determining whether a formula is intuitionistically valid is $\mathcal{PSPACE}$-Complete.

Consistent with the theme of expanding the arsenal of tools used to prove results in the area of proof complexity, this chapter uses fairly powerful techniques to show that unless a variant of Gentzen's famous proof system LK is a super proof system (and therefore $\mathcal{NP} = co\mathcal{NP}$), Statman's reduction from QBF to IPL cannot even translate trivial classical instances of the law of excluded middle into intuitionistic formulas with polynomially-bounded proofs, thereby showing that Statman's translation is probably dangerous.

8.3.2 Relationship To Dangerous Reductions

The results in this part of the thesis support the dangerous reductions research in the following way: In Chapter 9 we show that the K_n^* and $G_{\frac{n}{2},\frac{n}{2}}$ graphs from Section 8.2.2 respectively have trivially small NHPS proofs and exponential NHPS size lower bounds (see Section 9.8). Next, we apply the reduction from Section 8.2.1 to these graphs and use the Prover/Delayer game upper bound techniques from Chapter 10 to show that both the $H(K_n^*)_{T,O,F}$ and $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ formulas have polynomially-bounded $\mathsf{T\text{-}RES}$ refutations.

Chapter 11 unifies these results under the banner of dangerous reductions by showing that the $H(K_n^*)_{T,1,F}$ formulas have exponential $\mathsf{AC^0\text{-}Frege}$ lower bounds. The K_n^* graphs are easy NHPS instances, and if translated using the T, O, F version of the reduction, the resulting formulas are easy for $\mathsf{T\text{-}RES}$. However, if they are translated using the $T, 1, F$ version of the reduction (which is extremely similar to the T, O, F version, and just as natural) we get formulas which can not be solved by any Resolution-based SAT-solver. This is a formal example of a dangerous reduction, since it maps an easy instance from one domain, in this case Hamiltonian Cycles, to an intractable SAT instance.

However, reductions can have the opposite effect as well: The $G_{\frac{n}{2},\frac{n}{2}}$ graphs have exponential NHPS lower bounds, but the $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ formulas have polynomial $\mathsf{T\text{-}RES}$ upper bounds. This gives a formal example of a beneficial reduction, since it maps hard instances from the Hamiltonian Cycle domain to easy SAT instances.

Finally, Chapter 12 shows that unless the proof system $\mathsf{LK}[\vec{E_S}]$ is a super proof system (and therefore $\mathcal{NP} = co\mathcal{NP}$), Statman's reduction from QBF to IPL is dangerous because it cannot even translate trivial instances of the law of excluded middle into intuitionistic formulas with polynomially-bounded proofs.

Chapter 9

A Non-Hamiltonicity Proof System

9.1 Introduction & Motivation

In [PU95], Pitassi and Urquhart prove that the Hajós Calculus proof system for non-3-colourability is as powerful as any Extended Frege proof system. This surprising result remains the only major graph theoretic proof system. One goal of this present work is to further diversify the area by exploring a proof system for another graph theoretic co$\mathcal{NP}$-Complete problem. In this chapter we introduce the Non-Hamiltonicity Proof System (NHPS), which to our knowledge is the first proof system for this language.

A Hamiltonian Cycle is a simple cycle which passes through every vertex in a graph exactly once. Graphs containing Hamiltonian Cycles are termed to be 'Hamiltonian', and as its name suggests, the NHPS is a proof system for verifying any graph's non-Hamiltonicity. In this chapter we give a formal description of this proof system, provide proofs of soundness, completeness, as well as exponential lower bounds for the NHPS, and relate it to other proof systems. Our initial formalization is purposefully simplistic and minimal; if any of the formal characteristics of the NHPS are removed or weakened, then it is no longer complete. This results in the core proof system being relatively weak and for which it is easy to prove exponential lower bounds, but in Section 9.9.2 we discuss several ways in which it can be strengthened.

This chapter is organized as follows: In Section 9.2 we define a number of important concepts, and in Section 9.3 we describe the NHPS proof system. This is followed by Sections 9.4 and 9.4 in which we respectively prove the soundness and completeness of our proof system. Next, in Section 9.6 we describe a simplified version of the NHPS which does not require any marked edges.

This is followed in Section 9.7 by a proof that this proof system has exponential size lower bounds. The results in Section 9.8 are similar. It it we use related research from Chapters 10 and 11 to show that there exists an effective exponential separation between the NHPS and other proof systems; depending on which translation is used, it can be exponentially weaker than T-RES or exponentially stronger than AC^0-Frege systems.

Finally, in Section 9.9 we discuss open problems and potential future research in this area.

9.2 Terminology

The NHPS deals with standard undirected graphs that have no self loops nor any multiple edges, as well as 'marked' graphs, which are defined as follows:

Definition 9.2.1 (Marked Graph). *A marked graph $G = (V, E, M)$ is an undirected graph where $M \subseteq E$ is a (possibly empty) subset of special edges with markings on them.*

An example of a marked graph is shown below in Figure 9.1.1. The interpretation of marked edges in a graph G is that anyone searching for a Hamiltonian Cycle in G must include the marked edges in any Hamiltonian cycle found. In addition, we require the concept of 'forced edges', defined as follows:

Definition 9.2.2 (Forced Edge). *An edge is forced if it is necessarily part of every Hamiltonian cycle occurring in a graph, regardless of whether that edge is marked or not.*

Therefore, an otherwise Hamiltonian graph can be turned into a non-Hamiltonian marked graph by marking some set of edges which disqualifies all Hamiltonian Cycles, but marking forced edges does not disqualify any Hamiltonian cycles.

For example, the marked edges in Figure 9.1.1 are forced because they are incident on a degree-2 vertex. However, if three or more edges incident on a single vertex of a graph G are marked or forced, then that graph is not Hamiltonian:

Definition 9.2.3 (Marked Hubs & Forced Hubs). *If three or more marked edges are incident on a vertex, then that vertex a marked hub. Similarly, if three or more edges incident on a vertex are forced, then it is a forced hub.*

Similarly, if a cycle contains c edges where $c < n$ and all c edges are marked or forced, then that graph is also not Hamiltonian:

Definition 9.2.4 (Marked Subcycles & Forced Subcycles). *A cycle containing fewer than n edges in which all of them are marked is called a marked subcycle. If they are all forced, then it is called a forced subcycle.*

Marked hubs and subcycles are good examples of obstructions to Hamiltonicity, and are shown below in Figures 9.1.2 and 9.1.3, respectively. It is important to distinguish between marked hubs and forced hubs, and similarly important to distinguish between marked subcycles and forced subcycles. The NHPS deals exclusively with edges which have been marked to indicate that they are forced.

Note that every edge in a non-Hamiltonian graph is vacuously forced because it appears in all 0of the possible Hamiltonian Cycles; this point cannot be overemphasized, since it forms the basis of the NHPS proof system.

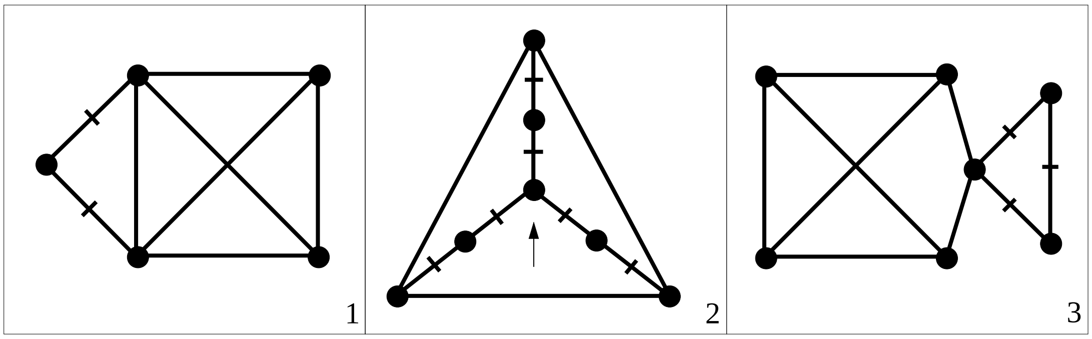

Figure 9.1: Examples of Marked Graphs Containing Forced Edges, a Forced Hub, and a Forced Subcycle

In addition, we make use of the following definition:

Definition 9.2.5 (Obivious Non-Hamiltonicity). *A marked graph $G = (V, E, M)$ is obviously non-Hamiltonian if any of the following conditions hold:*

1. *G contains a forced hub.*

2. *G contains a forced subcycle.*

9.3 Description of the Non-Hamiltonicity Proof System

The NHPS is quite simple and has the following characteristics:

1. Every NHPS proof is a tree.
2. Every tree node contains either a marked graph or an unmarked graph.
3. Tree edges are directed towards the root and represent applications of inference rules. It is possible for multiple edges to correspond with one application of a rule.
4. The root contains an unmarked graph that is to be proven as non-Hamiltonian.
5. Each leaf contains an axiom. The set of axioms consists of all unmarked graphs containing at least one vertex v such that $degree(v) \leq 1$.
6. All proof tree nodes at even depths contain unmarked graphs, whereas all proof tree nodes at odd depths contain marked graphs.
7. Every subtree rooted at an unmarked graph is an NHPS proof.
8. Rules of inference:
   (a) *Rule for marking edges:* Let $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, ..., $G_k = (V, E_k)$ be unmarked graphs with NHPS proofs of non-Hamiltonicity. If there exist two sets E and M subject to the following conditions:
      i. For each $G_i = (V, E_i)$, there exists exactly one edge e_i such that $E = E_i \cup \{e_i\}$.
      ii. $M = \bigcup_{i=1}^{k} \{e_i\}$

      then M's edges are all forced and can be safely marked. Therefore we may infer $G = (V, E, M)$.
   (b) *Rule for removing markings from all forced edges:* If a graph with marked forced edges is obviously non-Hamiltonian, then the corresponding unmarked graph is also non-Hamiltonian.

Example

Intuitively, the NHPS is used to show that any unmarked non-Hamiltonian graph that is not an axiom contains forced edges, that if marked, would constitute a forced hub or subcycle, thereby proving that it is indeed non-Hamiltonian. The first inference rule is based on the fact that any edge added to a non-Hamiltonian graph is forced [Her04]. The second inference rule states that any graph containing a forced hub or forced subcycle which has been marked has a corresponding unmarked graph which is also non-Hamiltonian. Of course, the second rule presupposes that all marked edges in hubs and subcycles are in fact forced, something that must be proven via NHPS proof sub-trees.

Figure 9.2 below illustrates an example of an NHPS proof of non-Hamiltonicity for the graph in the root. It is not biconnected, and is therefore not Hamiltonian. Note that all of the graphs in the leaves of the proof tree are axioms.

9.4 Soundness

In order to prove soundness for the NHPS proof system, we make use of the following lemma:

Lemma 9.4.1. *All marked edges in M introduced by Inference Rule a are forced.*

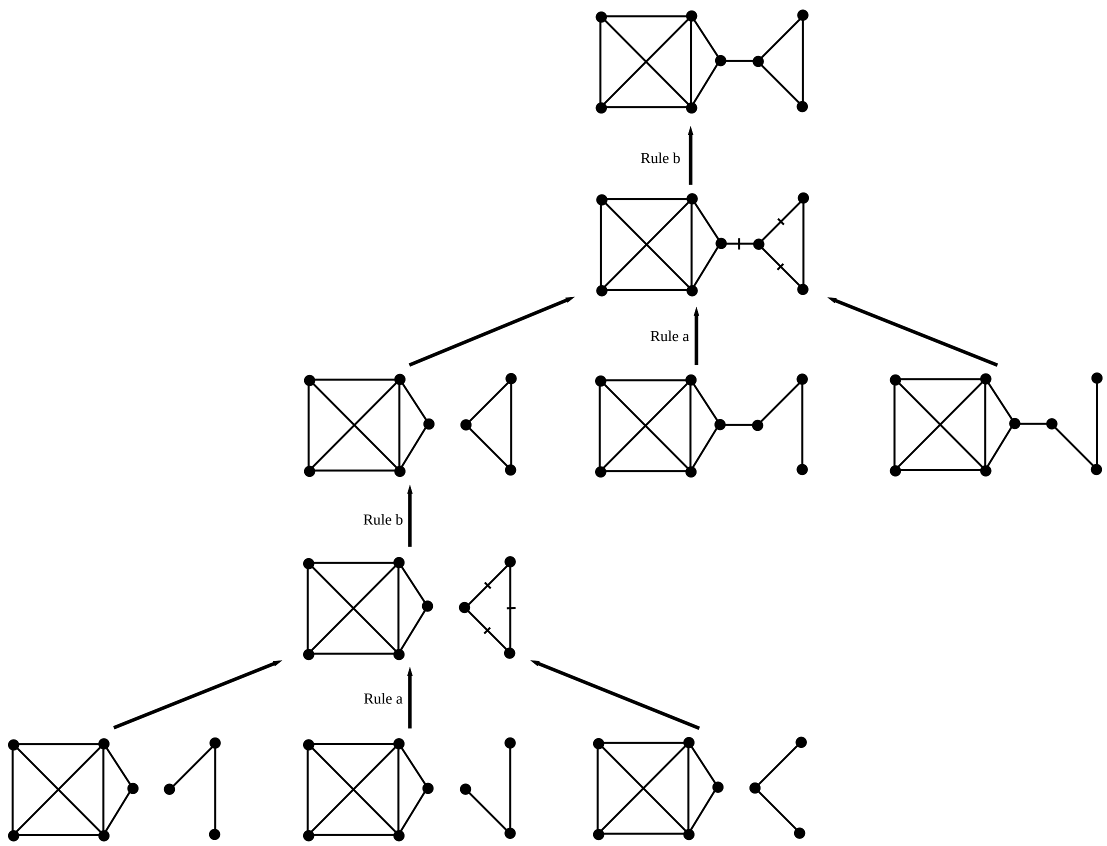

Figure 9.2: A NHPS Proof of Non-Hamiltonicity

Proof: From G_1, G_2, ... G_k we infer $G = (V, E, M)$. Suppose that G_1, G_2, ... G_k are all unmarked, non-Hamiltonian graphs and that for each $G_i = (V, E_i)$, there exists exactly one edge e_i such that $E = E_i \cup \{e_i\}$. Let $G_i{}'$ be G_i with the edge e_i added and marked. G_i is non-Hamiltonian, so any single edge added to it must be in every possible Hamiltonian Cycle of G_i, implying that each marked edge e_i in each $G_i{}'$ is forced. However, for each $G_i{}'$, $V_i{}' = V$, and $E_i{}' = E$. Therefore, each forced edge in each $G_i{}'$ must also be forced in G, as required. □

Theorem 9.4.2. *The* NHPS *is Sound.*

Proof: From Lemma 9.4.1, we know that Inference Rule a is sound; that is, all edges marked by the proof system are forced. It follows that Inference Rule b is therefore also sound, because all forced hubs and subcycles are clearly obstructions to Hamiltonicity, and cannot contain edges which are marked but not forced. Since its axioms are clearly non-Hamiltonian and all of its inference rules are sound, the NHPS is sound, as required. □

9.5 Completeness

In order to prove completeness for the NHPS proof system, we make use of the following lemma:

Lemma 9.5.1. *For any non-Hamiltonian graph $G = (V, E)$, at least one of the following holds:*

1. *G contains a vertex v such that $degree(v) \leq 1$*
2. *G contains a vertex u such that $degree(u) \geq 3$*

3. *G contains a cycle with c vertices, where $c < n$*

Proof: Suppose there exists a non-Hamiltonian graph $G = (V, E)$ which fails all three conditions. By failing conditions 1 and 2, every vertex of G must have a degree of 2. By also failing condition 3, we know that G must be a cycle containing n vertices. In other words, G is a Hamiltonian cycle, and is therefore Hamiltonian, a contradiction. □

Theorem 9.5.2. *The* NHPS *is Complete.*

Proof: The proof of completeness proceeds by induction on the number of edges in non-Hamiltonian graphs. We show that every non-Hamiltonian graph has an NHPS proof.

Basis: Consider any non-Hamiltonian graph that has no edges. It contains a vertex of degree 0, and is therefore a NHPS axiom, which is a NHPS proof, as required.

Induction Hypothesis: Suppose that each non-Hamiltonian graph with m edges has a NHPS proof.

Induction Step: Let $G = (V, E)$ be any arbitrary non-Hamiltonian graph with $m + 1$ edges. We know from Lemma 9.5.1 that at least one of the following cases holds:

1. G contains a vertex v such that $degree(v) \leq 1$
2. G contains a vertex u such that $degree(u) \geq 3$
3. G contains a cycle with c vertices, where $c < n$

If case 1 holds, then G is an axiom, so it has a NHPS proof. If case 2 holds, then G contains some vertex that would be a hub if its edges were forced. If case 3 holds, then G contains some proper subset of edges that if marked would constitute a forced subcycle. Every edge in a non-Hamiltonian graph is vacuously forced, so regardless of whether case 2 or case 3 is chosen, we need to show that the edges corresponding to the relevant obstruction are forced, thereby showing G to be non-Hamiltonian. Let us use G_M to refer to G with these edges marked. Let us use G_1, G_2, ..., G_k each to refer to G with exactly one of these edges removed. Since G is non-Hamiltonian, and removing edges preserves non-Hamiltonicity, each of G_1, G_2, ..., G_k is also non-Hamiltonian. Therefore, by the induction hypothesis, each of G_1, G_2, ..., G_k has a NHPS proof of non-Hamiltonicity. By placing G, G_M, and G_1, G_2, ..., G_k inside proof-tree nodes, and creating an edge corresponding with Inference Rule b from G_M to G, edges corresponding with Inference Rule a from G_1, G_2, ..., G_k to G_M, and nesting their NHPS proofs below, we can create an NHPS proof of non-Hamiltonicity for G. Therefore, in any case, G has a NHPS proof of non-Hamiltonicity.

Therefore, by induction, every non-Hamiltonian graph has a NHPS proof of non-Hamiltonicity, as required. □

9.6 NHPS Simplification

It is worth noting that the marked edges used by the NHPS are not actually necessary, and can be left out. They clarified the justification for why the proof system is correct, but with that understood, it is easy to see that they are dispensable. More specifically, we can combine rules of inference a and b into a single rule as follows:

Let $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, ... $G_k = (V, E_k)$ be unmarked graphs with NHPS proofs of non-Hamiltonicity. If the following hold:

1. For each $G_i = (V, E_i)$, there exists exactly one edge e_i such that $E = E_i \cup \{e_i\}$.
2. $\bigcup_{i=1}^{k} \{e_i\}$ constitutes either a hub or a forced subcycle.

Then we may infer $G = (V, E)$.

Eliminating marked edges from the system clearly simplifies it.

9.7 Exponential Lower Bounds

This proof system is weak in the sense that there exist infinitely large families of graphs which require exponentially long NHPS proofs of non-Hamiltonicity. Three such families are described in Section 8.2.2. Intuitively, these graphs are very dense and require NHPS proofs with large height, and therefore large size.

Theorem 9.7.1. *The lengths of* NHPS *proofs have* $\Omega(3^n)$ *size-lower bounds, where* n *is the number of nodes in the graph.*

Proof: Consider any of the graph families shown in Figure 8.2. Let G be any arbitrary graph chosen from any of these families. Since the minimum-degree vertex in G has $\Omega(n)$ edges incident on it, creating an axiom containing a degree 0 or degree 1 vertex requires a removal of $\Omega(n)$ edges. The number of edges in the nodes of any NHPS proof decreases linearly with the depth of those nodes, so the leaves in a NHPS proof of G's non-Hamiltonicity must all occur at a depth of $\Omega(n)$ or greater. Furthermore, every application of Inference Rule a must cause the proof tree to branch by at least 3, since forced hubs and subcycles both contain at least 3 edges. Every tree node branches into at least three more nodes for every branch. Since the depth of the tree is $\Omega(n)$, this branching behavior will be repeated recursively for every node at $\Omega(n)$ depths. This corresponds to a minimum proof tree which contains $3^{\Omega(n)} = \Omega(3^n)$ nodes. The growth rate for NHPS proof lengths for each of the families shown in Figure 8.2 therefore has exponential lower bounds, as required. □

9.8 Effective Separation From Other Proof Systems

As might be expected from the lower bounds above, NHPS is a fairly weak proof system, and as explained in the beginning of Section 2.4, because a reduction is always required between proof systems over different languages, and since a reduction can either introduce or reduce complexity, it does not make sense to talk about two proof systems over different languages p-simulating each other. Since the NHPS and T-RES are proof systems for entirely different languages, we cannot use p-simulation to compare them and instead need to make use of the more general notion of effective p-simulation given in Definition 2.4.2.

There exist reductions giving an effective exponential separation between the NHPS and T-RES proof systems. Recall that that the NHPS has exponential lower bounds for the disconnected graphs described in Section 8.2.2. In particular, let us consider the $G_{\frac{n}{2},\frac{n}{2}}$ graphs from that section which consist of two disjoint cliques of size $\frac{n}{2}$. Clearly these graphs are non-Hamiltonian, so applying the reduction from Section 8.2.1 yields an unsatisfiable formula.

As we will see from Corollary 10.4.4, if we apply the reduction which uses the Total, Onto, 1-1, and Function clauses, the resulting formulas have polynomial T-RES upper bounds. However, from Theorem 9.7.1, we know that the NHPS has exponential lower bounds for the $G_{\frac{n}{2},\frac{n}{2}}$ graphs. Since there exist polynomially-bounded T-RES proofs for the $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ formulas, we have an effective separation between the two proof systems with T-RES being stronger.

Similarly, the K_n^* graphs from Section 8.2.2 are NHPS axioms, and therefore have trivially small proofs, but as we shall see in Chapter 11, the $H(K_n^*)_{T,1,F}$ formulas have exponential AC^0-Frege lower bounds, giving another effective separation, this time with NHPS being stronger.

9.9 Open Problems Related to The NHPS

This research has several potential avenues of research:

9.9.1 Graph Algorithm Lower Bounds

One high-level motivation for better understanding graph theoretic proof systems is to help develop techniques for proving lower bounds on graph algorithms. For example, just as Resolution lower bounds give corresponding lower bounds for SAT-solving algorithms, so could lower bounds for graph theoretic proof systems help to prove lower bounds for a potentially wide range of graph algorithms. Non-3-colourability and non-Hamiltonicity are just some examples of proof systems which could be used in this way; given the number of co$\mathcal{NP}$-Complete graph theoretic problems and the importance of graph theory and algorithms, this is certainly an area of research worth pursuing.

9.9.2 Strengthening the NHPS

There are many ways in which the results in this chapter could be improved. The NHPS as described above is the most simplistic and weak version; it has only one simple axiom scheme, and both hubs and forced subcycles are required for completeness. The lower bounds are correspondingly simple. A natural course of research would be to strengthen the NHPS and develop more interesting and sophisticated lower-bounds arguments.

The NHPS could be strengthened in several ways. Note that these methods are all disjoint and orthogonal in the sense that they are very different and do not interfere with each other. In other words, any arbitrary number of them could be combined in order to maximize the strength of the system.

Add More Axioms

The most obvious way to defeat the lower bounds arguments so that none of the three families of graphs shown in Figure 8.2 require long proofs is by adding more axiom schemes. Checking for connectivity can be done in polynomial time, and more importantly, disconnected graphs have polynomially-sized certificates, so there is no reason why disconnectivity cannot be an axiom. The inclusion of such an axiom scheme would mean that graphs from the family shown in Figure 8.2.1 would no longer require exponentially large NHPS proofs, but would rather only require a single proof tree node in order to show that they are not Hamiltonian. Defeating the lower bounds for the family of graphs shown in Figure 8.2.2 would require a similar axiom scheme, this time for non-biconnectivity, which also has a polynomially-sized certificate. Finally, the family of graphs shown in Figure 8.2.3 would require an axiom scheme for non-1-toughness. Recognizing 1-tough graphs is co$\mathcal{NP}$-Hard [BHS90], so in practice applying such an axiom may be computationally intractable, but recognizing non-1-tough graphs is in $\mathcal{NP}$, and therefore has a short certificate, allowing us to include this axiom scheme as well. Since the third family of graphs subsumes the other two, including just the non-1-toughness axiom would suffice to defeat the lower bounds for all three families. A natural course of research would be to find lower bounds for the NHPS after it has been strengthened with the non-1-toughness axiom.

Add More Obstructions To Hamiltonicity

In its simplest form, the NHPS's definition of 'obviously non-Hamiltonian' contains hubs and forced subcycles as its only obstructions to Hamiltonicity. Another way to strengthen the proof system is by adding more obstructions to this list. Two more examples of obstructions to Hamiltonicity are 'barricades' and 'odd-forced-cuts' [Her04]. A barricade is a forced edge between the two vertices of any 2-vertex-cut, and is shown below in Figure 9.3.1. An odd-forced-cut is a cut across an odd number of edges, all of them forced. Figure 9.3.2 below illustrates a graph containing such cuts.

Adding these obstructions to the NHPS would not help to defeat the established lower bounds, but might be combined with some other way of strengthening the system to yield an interesting result.

Restrict The Input Class

Strengthening the NHPS does not necessarily involve editing its rules or structure. Instead, we could restrict its input to classes of graphs for which the Hamiltonian Cycle problem nonetheless remains $\mathcal{NP}$-

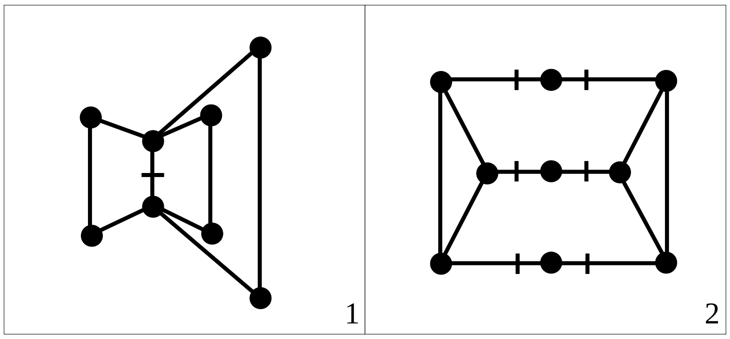

Figure 9.3: A Graph Containing a Barricade (left), and a Graph Containing 8 Odd-Forced-Cuts (right)

Complete. The Hamiltonian Cycle problem has been well-studied, and there are many such classes. For example, it remains $\mathcal{NP}$-Complete even when restricted to the following inputs:

1. Bipartite Graphs [Kri75],
2. Planar, Cubic, 3-Connected Graphs with no face having fewer than 5 sides [GJT76],
3. Planar, Cubic, 2-Connected, Bipartite Graphs [ANS80],
4. Cubic, 3-Connected, Bipartite Graphs [ANS80],
5. Maximal Planar Graphs (Triangulations) [Chv85, Wig82], and
6. 4-Connected, 4-Regular Graphs (M. Rosenfeld, Personal Communication, November 2003).

As might be expected, more restrictive classes strengthen the NHPS more. For example, if we limit our inputs to planar, cubic, 3-connected graphs with no face having fewer than 5 sides, then the graph families shown in Figure 8.2 are immediately excluded from consideration because they are neither planar nor cubic.

Restricting ourselves to cubic graphs is desirable when strengthening the NHPS because it ensures that each vertex has a constant degree. Since the NHPS depends on edge removals in order to expose degree-1 vertices (axioms), we are guaranteed that each degree-1 vertex can add a depth of at most 2 to the proof tree. Contrast this with the graph families in Figure 8.2, where each vertex requires a depth of $\Omega(n)$. Even the weakest form of the NHPS appears to be quite useful when constrained to this input class.

Restricting ourselves to planar graphs is further desirable because it allows us to include an axiom scheme for Grinberg's Theorem [BM76]. Grinberg's Theorem only applies to planar graphs, and its usefulness for proving graphs to be non-Hamiltonian should not be understated.

Theorem 9.9.1 (Grinberg's Theorem). *Every planar graph $G = (V, E)$ containing a Hamiltonian Cycle C satisfies the following equation:*

$$\sum_{i=3}^{n}(i-2)(f_i' - f_i'') = 0$$

where f_i' represents the number of faces of degree i on the interior of C, and f_i'' represents the number of faces of degree i on the exterior of C.

If the Grinberg equation associated with a planar graph is inconsistent, then the graph is non-Hamiltonian. Proofs showing that equations are inconsistent can be very short, thus allowing us to include Grinberg's condition as an axiom scheme if we desire.

One interesting research question is to ask what non-Hamiltonian planar, cubic, biconnected, 1-tough graphs which obey Grinberg's condition and have no odd-forced cuts look like. Would they translate to hard SAT instances?

Allow DAG-Like Proofs

In its simplest form, the NHPS is tree-like. A DAG-like NHPS proof is essentially a NHPS proof in which we allow proof nodes to be re-used an arbitrary number of times. This type of re-use requires us to use unlabelled graphs, or provide some kind of scheme for certifying isomorphism, neither of which are problems. For example, Figure 9.4 below shows a DAG-Like proof for the non-Hamiltonian graph consisting of two K_4 components.

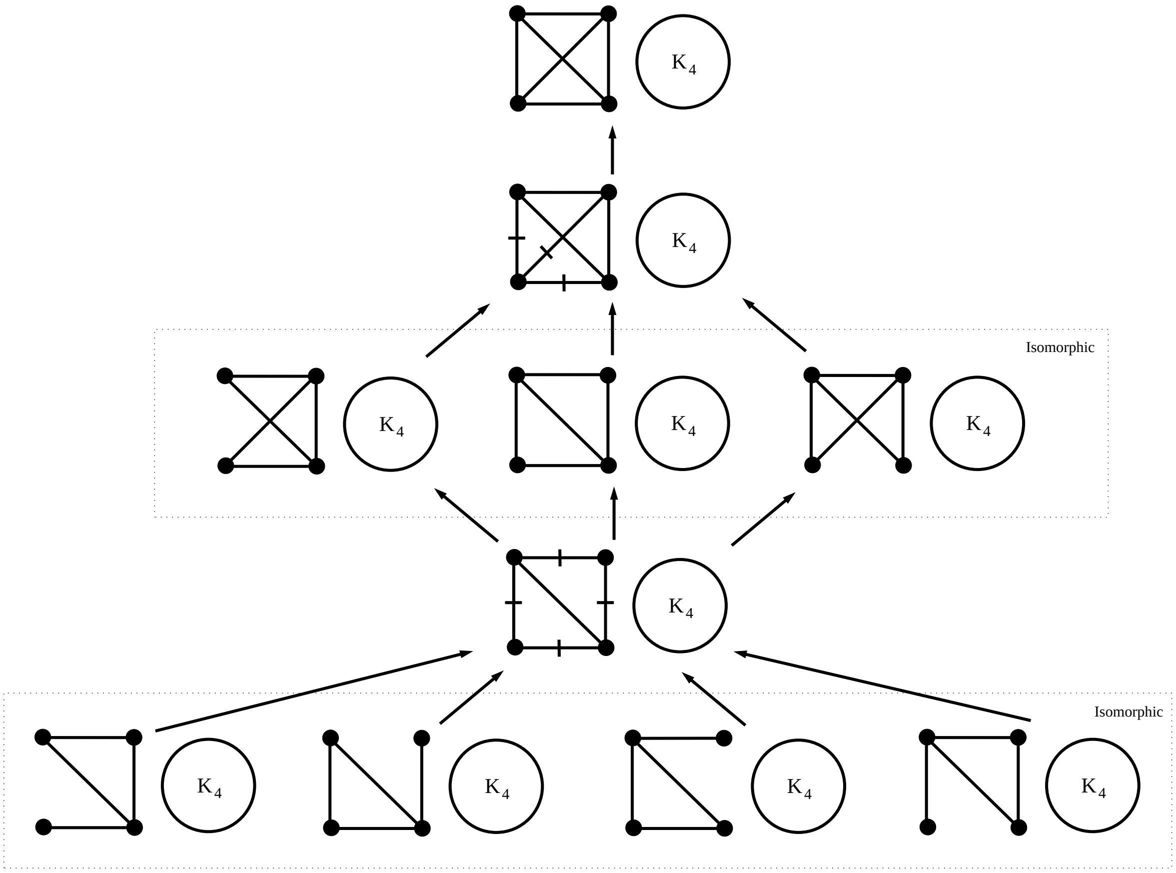

Figure 9.4: A DAG-Like NHPS Proof

The graph in the root is a member of one of the families that yielded our exponential lower bound. Unlike with tree-like proofs in which every successive level of the tree sees at least a tripling of the number of nodes from the level before, the ability to re-use nodes shortens our proof considerably. The argument used to establish our lower bounds therefore no longer holds.

Establishing lower bounds for the DAG-like NHPS raises some interesting questions. For example, does the DAG-like NHPS have polynomially-sized proofs for all non-biconnected graphs?

9.9.3 Relate the NHPS to Other Proof Systems

One final result worth pursuing would be to prove that the NHPS can effectively p-simulate (and be effectively p-simulated by) another proof system. Although we were able to show effective separations between the NHPS in which it is both stronger and weaker than other proof systems, we have yet to show that there exists a translation under which T-RES can p-simulate the NHPS. Our Hamiltonian Cycle to SAT reduction which uses the Total, Onto, 1-1, and Function clauses is a potential candidate of such a reduction.

It would also be interesting to ascertain the strength of the stronger NHPS incarnations. For example, can any of them effectively p-simulate Frege or Extended Frege?

Chapter 10

Prover/Delayer Game Upper Bounds

10.1 Introduction & Motivation

In this chapter we further investigate the Prover/Delayer game which we described in Section 3.2.5 and proved $\mathcal{PSPACE}$-Complete in Chapter 4. In addition, Corollary 3.3.9 showed that the Prover/Delayer game can be used to prove T-RES size lower bounds, and Theorem 3.3.7 showed that it also perfectly captures T-RES clause space. We now prove that this interesting game can also be used to give upper bounds on the size of T-RES proofs, thereby providing the final result needed to show that T-RES captures the upper and lower bounds of both T-RES clause space and T-RES size.

We will prove this in two separate ways: The first proof is non-constructive and follows very easily from previous results in the literature. The second proof is more constructive in the sense that it explicitly shows how to translate a playing of the Prover/Delayer game into a DPLL tree. It may seem odd to prove the same thing twice, but the upper bound conditions given by these two proofs are slightly different; the bounds given by the non-constructive proof are better under certain situations, and the bounds given by the constructive version are better under others.

Another motivation for this line of research is its practical applications for researchers publishing their work. We show that for publishing proofs of T-RES size upper bounds, proving bounds on the Prover/Delayer game is much simpler than having to draw out entire T-RES proofs or DPLL trees. The examples which we give illustrating these new upper bound techniques will be very important to the Dangerous Reductions research in Chapter 11.

In effect, the Prover/Delayer game can be used to simplify both upper and lower bounds for T-RES size as well as space, showing that this game truly captures the T-RES proof system.

This chapter is organized as follows: In Section 10.2 we review a result by Ben-Sasson, Impagliazzo, and Wigderson showing that the Prover/Delayer game can be used to prove T-RES size lower bounds. Next, in Section 10.3 we prove this chapter's main result, namely that the Prover/Delayer game can also be used to prove T-RES size upper bounds: In Section 10.3.1 we show how this can be proved non-constructively by combining previous results, and in Section 10.3.2 we give our corresponding constructive proof. Finally, in Section 10.4 we give examples of how the Prover/Delayer game can be used to show T-RES upper bounds for the formulas resulting from applying the Hamiltonian Cycle to SAT reduction from Section 8.2.1 to the graphs from Section 8.2.2.

10.2 Prover/Delayer Game & Tree Resolution Size Lower Bounds

The Prover/Delayer game was used in [BSIW04] to prove size lower bounds for T-RES proofs. More specifically, lower bounds on $PD(F)$ can be used to prove lower bounds on the size of T-RES proofs:

Theorem 10.2.1 ([PI00, BSIW04]). *If an unsatisfiable CNF formula F has a* DPLL *tree of size $\leq 2^k$, then the Prover has a strategy limiting the Delayer to $\leq k$ points.*

Proof: Let F be any arbitrary unsatisfiable formula with a DPLL tree of size $\leq 2^k$. The Prover uses this tree as a strategy to limit the Delayer to at most k points as follows: The prover starts by querying the variable corresponding to the root of the tree. If the Delayer says 'True' or 'False', then the Prover proceeds by querying the variable in the corresponding subtree of the root. If the Delayer says 'You Choose', then the Prover chooses the smaller subtree, and next queries on that variable. Therefore, even if the Delayer says 'You Choose' at every possible opportunity, 2^k can only be split in half at most k times, so the Delayer wins at most k points, as required. □

The contrapositive of this theorem yields the following corollary, giving a direct connection between the Prover/Delayer game and T-RES size lower bounds. Although this corollary was already stated in Section 3.3.2, it is worth repeating here because it allows researchers interested in proving exponential T-RES size lower bounds to focus instead on proving linear Prover/Delayer game lower bounds, and is therefore very similar to this chapter's main results in the next section.

Corollary 10.2.2 (Restated). *If the Delayer has a strategy guaranteed to win $> k$ points on F, then every* DPLL *tree for F has size $> 2^k$.*

10.3 Prover/Delayer Game & Tree Resolution Size Upper Bounds

In addition to lower bounds, the Prover/Delayer game can be used to prove upper bounds on the size of T-RES proofs. More specifically, upper bounds on $PD(F)$ yield T-RES size upper bounds. This result can be proved both constructively and non-constructively. These different proofs give slightly different bounds, each better than the other under certain conditions. The non-constructive proof follows closely from previous results, whereas the constructive one is new and gives some interesting insights into the relationship between the Prover/Delayer game and T-RES size upper bounds because it specifically describes how to translate the history of a completed game in which k points were scored into a DPLL tree of size $O(n^k)$, where n is the number of distinct variables.

This is interesting because it allows for researchers interested in proving polynomial T-RES size upper bounds to focus instead on proving constant Prover/Delayer game upper bounds, and is therefore very much the analog of Corollary 10.2.2.

10.3.1 Non-Constructive Proof

The non-constructive proof relating the Prover/Delayer game to T-RES size upper bounds follows quite easily as a corollary of results by Esteban and Torán from two separate papers. These results are described in Section 3.3.2, but we restate them here for convenience:

Theorem 10.3.1 ([ET01], Restated). *Let F be an unsatisfiable formula on n distinct variables. If F has a* T-RES *refutation with tree clause space $s = CS(F \vdash_{\text{T-RES}} \emptyset)$, then it has a* T-RES *refutation of size $\binom{n+s}{s}$.*

Theorem 10.3.2 ([ET03], Restated). *For any unsatisfiable CNF formula F, $CS(F \vdash_{\text{T-RES}} \emptyset) = PD(F) + 1$.*

From [Bol85, p.124], we know that $\binom{a}{b} \leq (\frac{e\ a}{b})^s$, so the expression $\binom{n+s}{s}$ is bounded above by $e^s(\frac{n}{s}+1)^s$. When we combine this fact with Theorems 10.3.1 and 10.3.2, we immediately prove that an upper bound on the number of points scored by the Delayer yields an upper bound on T-RES proof size:

Corollary 10.3.3. *If the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a* T-RES *refutation of size $\leq e^{k+1}(\frac{n}{k+1}+1)^{k+1}$.*

Proof: Suppose that the Prover has a strategy limiting the Delayer to at most k points playing on formula F. By Theorem 10.3.2, we know that F has a T-RES proof with $CS(F) = k+1$. By Theorem 10.3.1, F has a T-RES refutation of size $\binom{n+k+1}{k+1}$, which is bounded above by $e^{k+1}(\frac{n}{k+1}+1)^{k+1}$, as required. □

10.3.2 Constructive Proof

The following theorem constructively allows us to take the history of a Prover/Delayer game played on a formula F in which k points were scored, and build a DPLL tree of size $O(n^k)$, where n is the number of distinct variables in F. This bound is slightly different than the one from the non-constructive proof. The non-constructive proof gives better bounds when k is much larger than $log(n)$, whereas the constructive version gives better bounds when k is $O(log(n))$, and is particularly good when k is constant, which tends to be exactly when one would want to apply these techniques to prove T-RES size upper bounds.

Theorem 10.3.4. *If the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a* DPLL *tree of size $O(n^k)$.*

Proof: Proof by (strong) induction on k, the number of points scored by the Delayer.

Basis: For $k = 1$, let F be any arbitrary formula on which the Prover has a strategy limiting the Delayer to at most 1 point. We will use the series of queries and answers from the game played between the Prover and Delayer on F to construct a DPLL tree of size $O(n)$.

Since the Delayer scores at most 1 point, the very last query made by the Prover must result in the Delayer responding with 'You Choose' (YC), and the Prover's response must falsify the formula, or else the Delayer would be able to score ≥ 2 points. Therefore all $\leq n-1$ previous variable queries must have been answered with a 'True' or 'False'.

Rename the variables so that they are queried in the order $x_1, x_2, ..., x_j$, where x_j is the variable for which the Delayer responds, 'YC'. Without loss of generality, assume that the Prover sets x_j to True (the False case is analogous). Use this series of questions and answers to build a path from the root of the tree to a leaf, as shown below in Figure 10.1.1.

We now fill in the rest of the tree. For every 'True' or 'False' reply, the Delayer could have said 'YC', and we know that this change in strategy could not have increased the number of points scored. This means that every node in the DPLL tree corresponding to one of these $\leq n-1$ variables must be adjacent to a leaf, each marked with an 'x' in Figure 10.1.2.

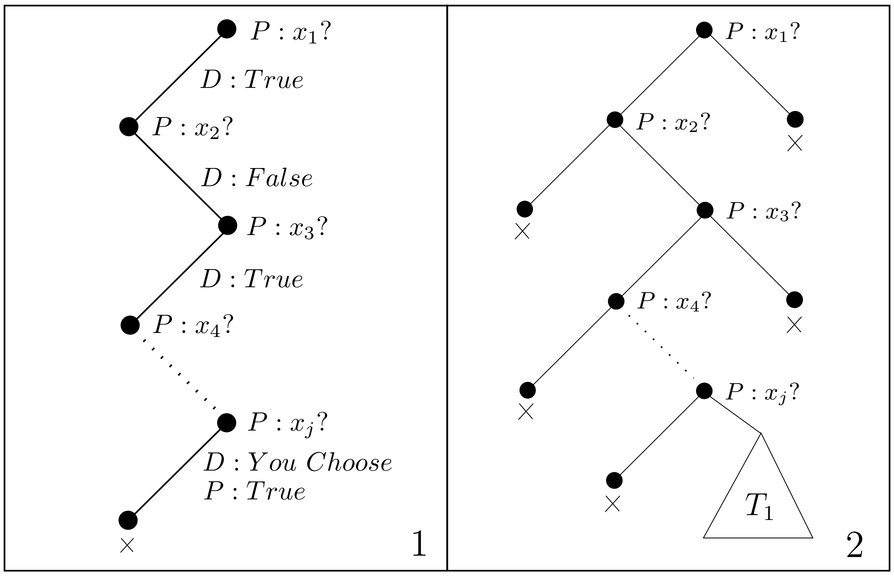

Figure 10.1: Example of a Game in Which 1 Point is Scored Together with its Corresponding DPLL Tree

Now we need only construct the subtree labelled T_1. If this subtree is a leaf, then we are done. However, it may be the case that the formula F_j at the root of T_1 is one in which the Delayer can win 1 point (the

Delayer cannot win ≥ 2 points on F_j, or else the reply to x_j would have been 'False' instead of 'YC', allowing for ≥ 2 points to be scored on F). In order to build T_1, we therefore recursively play a new game on F_j, and use exactly the same technique as above to turn this game into a tree. This is repeated as many times as necessary.

Note that because 'YC' was said, preceded by zero or more non-'YC' responses, F_j contains strictly fewer distinct variables than F, and the height of our entire DPLL tree can be at most n, so this process is guaranteed to terminate, leaving us with a DPLL tree in which each node either is a leaf or is adjacent to at least one leaf. Our tree therefore has height $\leq n$, and contains $\leq 2n+1$ nodes, which is $O(n)$.

Induction Hypothesis: Let F be any arbitrary unsatisfiable formula on $\leq n$ distinct variables. Assume that if the Prover has a strategy limiting the Delayer to at most $k-1$ points, then F has a DPLL tree of size $O(n^{k-1})$.

Induction Step: Let F be any arbitrary formula on which the Prover has a strategy limiting the Delayer to at most k points. As in the basis, we will again use the series of queries and answers from the game played between the Prover and Delayer on F to construct a 'scaffold' for a DPLL tree, which we will then fill in.

Consider the game played on F up to and including the first 'YC' reply to the query on variable x_j, as shown below in Figure 10.2.1. The path corresponding to the remainder of the game in which $k-1$ points are scored is labelled P_{k-1}.

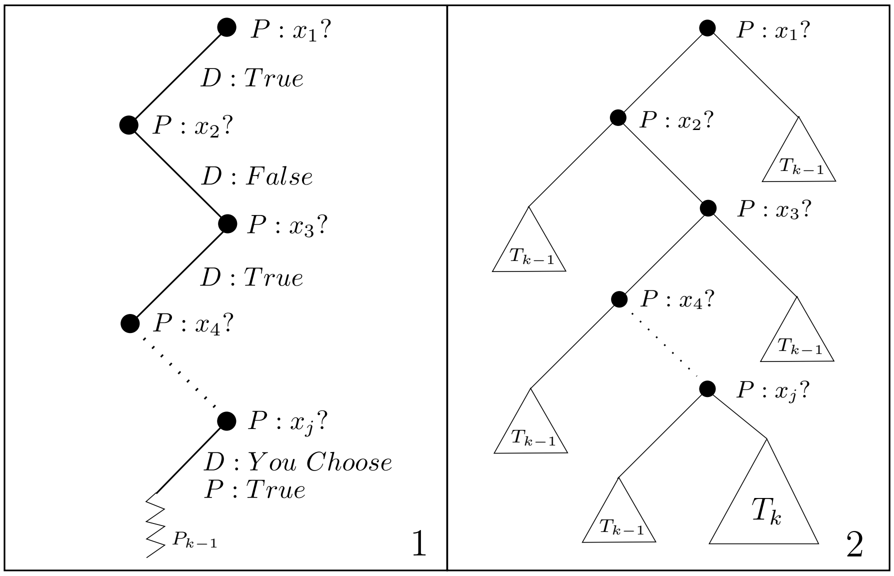

Figure 10.2: A Path Representing a Game in Which k Points are Scored Together With its Corresponding DPLL Tree

We will now fill in the rest of the tree. The first 'YC' is preceded by 0 or more non-'YC' responses. Consider any one of the variables for which a non-'YC' answer is given. Without loss of generality, let us examine variable x_2. The Delayer could have replied 'YC' to the query on x_2, and this change in strategy could not have increased the number of points won. But we have already seen that the false path leads to a situation where the Delayer can win k *more* points for a total of $k+1$ points, so in response, the Prover would have to have set x_2 to 'True'.

Let F_2 be the formula resulting from setting x_2 to true. The Delayer could not win k points here, because that would mean that saying 'YC' on x_2 is a better strategy than saying 'False'. Therefore the Prover has a strategy limiting the Delayer to at most $k-1$ points on F_2, and our Induction Hypothesis applies, so F_2 has a DPLL tree T_{k-1} of size $O(n^{k-1})$. The same argument applies to each of $x_1, ..., x_{j-1}$, as shown in Figure 10.2.2.

Now consider x_j, the first variable for which the Delayer says 'YC'. Without loss of generality, assume that the Prover decides to set x_j to true, resulting in the restricted formula $F_{j=True}$. Since one point has been scored, the Delayer can score a maximum of $k-1$ points on $F_{j=True}$, so the Induction Hypothesis applies and it has a DPLL tree of size $O(n^{k-1})$, as indicated.

The right subtree, however, must be treated similarly to its analogue in the basis. If $< k$ points can be won in it by the Delayer, then the induction hypothesis applies and we are done. We know that the Delayer cannot win $> k$ points in it or else the response on x_j would have been 'False' rather than 'YC', allowing for $> k$ points to be won overall, a contradiction.

Therefore assume that the right subtree is rooted at a restricted formula $F_{j=False}$ on which the Delayer can win exactly k points. In other words, it is as if we are recursively starting our tree-building process all over again. In order to build T_k, we therefore play a new game on $F_{j=False}$, and use exactly the same technique as above to turn this game into a tree. Again, it is important to note that $F_{j=False}$ contains strictly fewer distinct variables than F, and the height of our entire DPLL tree can be at most n, so in the worst case this process can continue until the remaining height is $\leq k$, at which point the Delayer must respond 'YC' to every query in order to get up to k points, and the induction hypothesis applies to both subtrees since each has height $\leq k-1$ and $\geq k$ points cannot be won in only $\leq k-1$ queries. When this process finishes, every subtree that we have filled in has size $O(n^{k-1})$, so we will end up with a tree consisting of a path of length $\leq n$ with each child tree hanging off of it having size $O(n^{k-1})$, therefore showing that our entire tree has size $O(n^k)$.

Therefore, by induction, if the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a DPLL tree of size $O(n^k)$, as required. $\square$

When we combine this result with Corollary 10.2.2, we get interesting T-RES size bounds: For any unsatisfiable formula F on n variables with $PD(F) = k$, the size of any T-RES refutation of F has size at least $\Omega(2^k)$ and at most $O(n^k)$.

10.4 Examples

Having T-RES size upper bounds based on the Prover/Delayer game is particularly useful because it allows us to simplify proofs of T-RES size upper bounds in terms of size and ease of writing. We now provide some examples illustrating how the Prover/Delayer game can be used to simplify T-RES proofs. These examples will be very important to the results in Chapter 11.

10.4.1 Example 1: Polynomial Upper Bounds for the $H(K_n^*)$ Formulas

In Section 8.2.1 we gave a reduction from the Hamiltonian Cycle problem to SAT. Intuitively, the reduction takes an input graph G and produces a formula that enforces a mapping from the vertices in G to the positions of a Hamiltonian Cycle. The mapping must be a bijection, and can include clauses that enforce the mapping to be Total, 1-1, a Function, and Onto. In addition, there are clauses which enforce the edge structure of G. The output formula has variables of the form $m_{i,j}$ which are interpreted as meaning that vertex i in G is mapped to position j in the Hamiltonian Cycle, and the resulting formulas are called $H(G)$. Additional subscripts are added to this notation to indicate which clauses were used to enforce the bijection. For example, if the reduction used clauses from the total and 1-1 groups, then the resulting formula is labelled as $H(G)_{T,1}$. For more information on this reduction, please refer to Section 8.2.1.

We can apply this reduction to the K_n^* graphs from Section 8.2.2. Each K_n^* graph consists of the complete graph K_n with the addition of a single degree-0 vertex called x. Since each K_n^* is disconnected, it is clearly non-Hamiltonian, which in turn means that every formula $H(K_n^*)$ is unsatisfiable.

If our reduction uses the Total, Onto, and Function clauses, then the resulting formula has polynomial upper bounds:

Theorem 10.4.1. T-RES *proofs for the unsatisfiability of $H(K_n^*)_{T,O,F}$ formulas have $O(n^2)$ size upper bounds, where n is the number of distinct variables contained in the formulas.*

Proof: For the following argument, please refer to the T-RES proof template shown in Figure 10.3. Note that some of the leaves are labelled with the specific clauses which are falsified at that position. In addition, to avoid diagrammatic clutter, the remaining leaves are labelled with the groups containing the clauses which are falsified.

We initially branch on $m_{x,1}$. Since x is an isolated vertex in K_n^*, setting $m_{x,1} = True$ ensures that assigning any future vertex to position 2 (i.e. setting $m_{i,2} = True$ for any $i \neq x$) will falsify an edge clause, and therefore falsify the formula. Setting $m_{x,2} = True$ also falsifies $H(K_n^*)_{T,O,F}$ by falsifying a function clause, because no vertex may be assigned to more than one position. Finally, setting $m_{i,2} = False$ for $i = 1, 2, ..., n, x$ will falsify the onto clause requiring that some vertex be mapped to position 2. This subtree requires $2n + 3$ nodes.

When we set $m_{x,1} = False$, we next branch on $m_{x,2}$. Clearly, the formula rooted by setting $m_{x,2} = True$, can be shown to be unsatisfiable with a tree of size $2n + 3$ for the same reasons as above.

For each i, after setting $m_{x,i-1} = False$, we branch on $m_{x,i}$. Each positive branching will result in a subtree of size $2n + 3$ as described above.

The all-negative assignment to every $m_{x,i}$ falsifies the total clause ensuring that vertex x is mapped to some position, after which all branching is complete. This T-RES proof therefore has size $(n+1)(2n+3)+1$, which is $O(n^2)$, as required. □

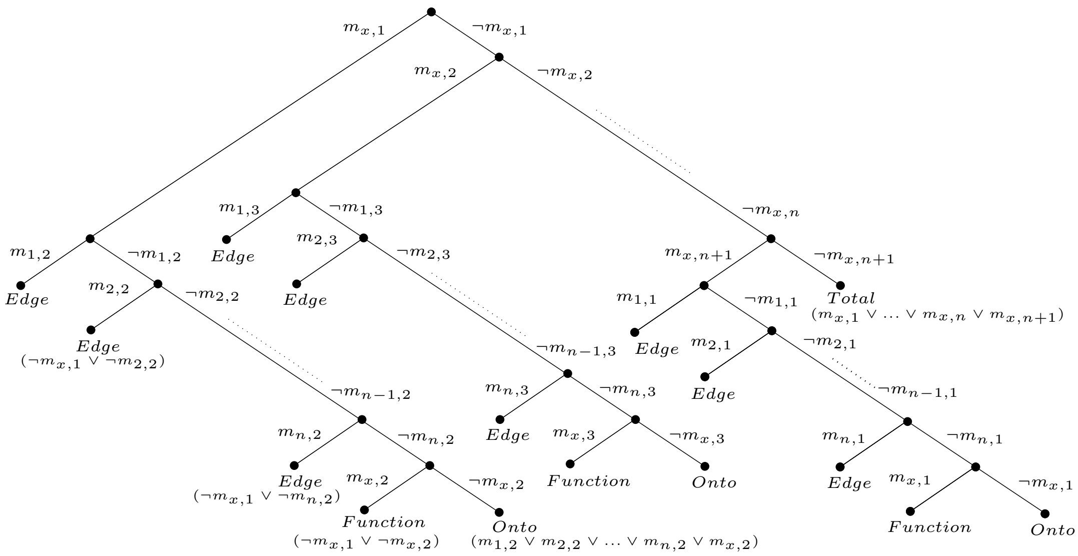

Figure 10.3: A Template for Polynomially-Sized T-RES Proofs for the Unsatisfiability of $H(K_n^*)_{T,O,F}$ Formulas

The proof of Theorem 10.4.1 shown in Figure 10.3 is a fairly standard argument which uses a DPLL tree template. Although clear enough, these types of proofs are difficult to write-up because drawing proof templates requires a lot of time and effort. The Prover/Delayer game can be used to significantly simplify T-RES upper bound arguments. For example, contrast the proof above with the proof of the following lemma:

Lemma 10.4.2. $PD(H(K_n^*)_{T,O,F}) \leq 2$

Proof: We describe a Prover strategy which limits the Delayer to at most 2 points: The Prover first tries to map vertex x to some position in the cycle. This is done by querying $m_{x,1}$. If the Delayer says 'False', then the Prover queries $m_{x,2}$, and so on until the Delayer finally says 'True' or 'You Choose'. If the Delayer says 'False' to every query, then a Total clause is falsified, and the game is over. Therefore the Delayer has no choice but to concede that x gets mapped somewhere, and can win at most one point during this phase by saying 'You Choose', at which point the Prover says 'True'.

Next the Prover tries to map every single vertex (including x) to the position adjacent to where x was mapped. Since x has degree zero, mapping any vertex other than x into an adjacent position would falsify an Edge clause. Mapping x to two places would falsify a Function clause. However, if no vertex is mapped to that position, then an Onto clause is falsified. The Delayer's best strategy is therefore to simply say 'You Choose', and accept the inevitable defeat while winning another point for a total of 2, the most possible given this Prover strategy, as required. □

When we combine this lemma with Theorem 10.3.4, we get Theorem 10.4.1 as an immediate corollary. It is easy to see that the proof using the Prover/Delayer game is simpler and easier to write than the size $O(n^2)$ diagrammatic DPLL tree given in Figure 10.3.

10.4.2 Example 2: Polynomial Upper Bounds for the $H(G_{\frac{n}{2},\frac{n}{2}})$ Formulas

In our next example we use the Prover/Delayer game to prove that the $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ formulas obtained by applying the reduction in Section 8.2.1 to the non-Hamiltonian graphs from Section 8.2.2. Recall that the $G_{\frac{n}{2},\frac{n}{2}}$ graphs each consist of two disjoint cliques of size $\frac{n}{2}$. If we take these graphs and apply the reduction from Section 8.2.1 which uses the Total, Onto, 1-1, and Function clauses, the resulting unsatisfiable formulas have a Prover/Delayer number of at most 3, which in turn shows that they have polynomial T-RES size upper bounds by Theorem 10.3.4.

Since it would be too large and complicated, we omit drawing a DPLL proof template for comparison and instead use the Prover/Delayer game to show polynomial T-RES upper bounds:

Theorem 10.4.3. $PD(H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}) \leq 3$

Proof: We describe a strategy for the Prover which limits the Delayer to at most 3 points: Assume that the vertices in one clique are red, labelled $r_1, ..., r_{\frac{n}{2}}$, and those in the other are blue, labelled $b_1, ..., b_{\frac{n}{2}}$.

First the Prover tries to map vertex r_1 to some position in the Hamiltonian Cycle. To avoid the Total clause for r_1 being falsified, the Delayer must let it be mapped somewhere, and can win at most 1 point in doing so by saying 'You Choose'. Without loss of generality, let us assume that r_1 was mapped to position 1.

Next, the Prover tries to map vertex b_1 to position 2. If the Delayer says 'False', then the Prover tries to map every remaining blue vertex to position 2. If the Delayer says 'True', or 'You Choose' (followed by the Prover saying 'True'), then a blue vertex and a red vertex are adjacent, which falsifies an Edge clause. In this case, the Delayer wins at most 2 points.

On the other hand, if the Delayer does not allow any blue vertices to be mapped to position 2, then the Prover attempts to map every blue vertex to position $i = 3, 4, ..., n$. This leads to two cases:

Case 1: The Delayer continuously says 'False'. In this case, the Prover attempts to map all possible blue vertices to position 1 (the position already occupied by r_1). If the Delayer says 'True' or 'You Choose' for one of them (followed by the Prover saying 'True'), then two vertices have been mapped to the same slot, causing the falsification of a 1-1 clause. If the Delayer again says 'False' and does not allow a blue vertex to be mapped to position 1, then there is a blue vertex which was mapped to nowhere, causing the falsification of an Onto clause. Within Case 1, the greatest possible number of points scored is 1, for a total of 2.

Case 2: The Delayer allows a blue vertex to be mapped to position i. We may assume that this came about by the Delayer saying 'You Choose', at which point the Prover would say 'True', thereby winning another 1 point for the Delayer. We now have a red vertex in position 1, a blue vertex in position i, and we know that positions $2, ..., i-1$ do not have blue vertices in them. At this point the Prover attempts to map all possible red vertices to position $i-1$. If the Delayer says 'False' to all of them, then an Onto clause is violated, and the game ends with 2 points. The alternative is that the Delayer allows a red vertex to be mapped to position $i-1$ (by saying 'You Choose' for a total of 3 points). Since position i already has a blue vertex in it, this falsifies an Edge clause. In this case, the Delayer wins 3 points, which is the maximum of all cases, as required. □

Together with Theorem 10.3.4, this result gives us the following Corollary:

Corollary 10.4.4. T-RES *proofs for the unsatisfiability of* $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ *formulas have* $O(n^3)$ *size upper bounds, where* n *is the number of distinct variables contained in the formulas.*

Once again, this proof is much simpler to understand and create than the alternative of drawing a size $O(n^3)$ DPLL proof tree.

Chapter 11

Formalizing Dangerous Reductions

11.1 Introduction & Motivation

Satisfiability, or SAT, is the archetypal $\mathcal{NP}$-Complete problem. It has long been known that every problem in $\mathcal{NP}$ can be reduced to SAT using Cook's Theorem [Coo71]. Since propositional formulas are very expressive, instances of many problems in $\mathcal{NP}$ can also be encoded as SAT instances in a much more direct and intuitive way than via Cook's Theorem. In fact, many problems have numerous different SAT encodings to choose from, and this has allowed the area of SAT-solving to become one of the most successful techniques for tackling $\mathcal{NP}$-Complete problems.

Although the strategy of translating problems from other domains to SAT has proved to be fruitful, this technique is not without its dangers. Empirical evidence suggests that natural encodings which seem to conserve much of the structure of the original problem can actually convert simple instances of the original problem to very difficult SAT formulas. For example, Kautz, McAllester and Selman investigate numerous approaches for translating planning problems to SAT in [KMS96]. Some of these translations are found to result in formulas which are much harder to solve than others, suggesting that a great deal of care must be taken in designing encodings since one cannot assume that they will conserve the simplicity of easy input instances.

This danger is so well-known to the propositional reasoning community that Kautz, McAllester, and Selman list understanding it as one of ten important and challenging open problems in the area [KMS97]. A more recent follow-up paper [KS03] reaffirms this problem's importance and notes that although some progress has been made, there is still much more work to be done.

We address this problem in two ways. Firstly, we show that the Hamiltonian Cycle to SAT reduction described in Section 8.2.1 can translate trivial instances of the Hamiltonian Cycle problem to an intractably difficult one for any Resolution-based SAT-solver. We also show that very minor modifications to this encoding can make it produce easy SAT instances, thereby giving the first formal example of an exponential separation between two very similar and natural encodings. We call this a 'dangerous reduction', because it injects unwanted complexity into the resulting formula, thereby defeating the entire purpose behind translating problems from other domains to SAT. However, this phenomenon has a brighter side as well, and we provide a formal example of a reduction which translates intractable instances for the NHPS proof system from Chapter 9 to easy SAT instances.

The second way in which we address the overall problem of choosing between reductions is by providing a domain-independent framework for comparing the effectiveness of competing encodings in terms of how easy it is to solve their outputs. This framework is based on the standard proof-complexity hierarchy shown in Figure 2.2 as well as the close relationship between SAT-solving and propositional proof complexity.

Ultimately, these results constitute an implicit suggestion to researchers working with SAT-solvers: When translating problems from other domains to SAT, ideally one should also provide a proof that the reduction is not dangerous.

This chapter is organized as follows: In Section 11.2 we give the first formal example of a dangerous reduction as well as examples of neutral, and beneficial ones as well. All of these examples depend on definitions and results from previous chapters of this thesis.

In Section 11.3 we define the formal notions of explosive, stable, and implosive reductions to correspond with our intuitive notions of dangerous, stable, and beneficial reductions. We use these definitions together with the standard proof complexity p-simulation hierarchy to design a framework for comparing encodings.

Next, in Section 11.4 we discuss the implications of this research for propositional proof complexity.

Finally, in Section 11.5 we discuss open problems and conjectures related to this area of research.

11.2 Formal Examples of Dangerous, Neutral, & Beneficial Reductions

In this section we apply three very similar versions of the reduction from Section 8.2.1 to the non-Hamiltonian graphs described in Section 8.2.2. We apply the first two reductions to the K_n^* graphs, and the remaining reduction to the $G_{\frac{n}{2},\frac{n}{2}}$ graphs. Since all of these graphs are disconnected, they are non-Hamiltonian, which in turn means that each resulting formula is unsatisfiable, regardless of which version of the reduction is used.

We show that the first reduction maps easy instances for the NHPS proof system from Chapter 9 to formulas with exponential lower bounds for all AC^0-Frege proof systems. This immediately implies exponential lower bounds for RES, and all Resolution-based SAT-solvers, including clause learning algorithms. This is an example of a dangerous reduction.

By contrast, the second reduction, which differs only slightly from the first, maps the same easy instances for the NHPS proof system to formulas with polynomial T-RES size upper bounds, giving an example of a neutral reduction.

These reductions are not pathologically designed to create problems, but rather are very intuitive and straightforward. This is relevant to the open problem in [KMS97, KS03] because as already mentioned, researchers working with SAT-solvers are aware of empirically tested instances where different reductions can have a significant impact on the complexity of a problem, but this is the first formal example.

Finally, our third reduction, which again is very similar to the other two, maps intractably hard instances for the NHPS proof system to formulas with polynomial T-RES size upper bounds, showing that reductions can be beneficial as well.

11.2.1 Formal Example of a Dangerous Reduction

In this section we show that the version of the Hamiltonian Cycle to SAT reduction from Section 8.2.1 which uses Total, 1-1, and Function clauses when applied to K_n^* graphs results in a formula which no Resolution-based SAT-solver can solve efficiently, even though the K_n^* graphs are trivially non-Hamiltonian. In other words, even though the encoding is very natural, it injects an exponential amount of unwanted complexity into our original problem instance. More specifically, we prove exponential AC^0-Frege size lower bounds for the $H(K_n^*)_{T,1,F}$ formulas:

Theorem 11.2.1. *Lengths of* AC^0-Frege *proofs for the unsatisfiability of* $H(K_n^*)_{T,1,F}$ *formulas have* $\Omega(2^{\sqrt[5^d]{n}})$ *lower bounds, where n is the number of distinct literals and d is the depth of the Frege proof, and if there exist size-N* AC^0-Frege *proofs of* $H(K_n^*)_{T,1,F}$*, then there exist size-*$N + O(n^3)$ *proofs of* $fPHP^n_{n-2}$.

Proof: The high-level overview of this proof is as follows: Assume that we have a size-N AC^0-Frege proof of $H(K_n^*)_{T,1,F}$. We show that this proof can be restricted with a specially-chosen truth assignment α to get a new, smaller proof of $H(K_n^*)_{T,1,F}\restriction_\alpha$. After unit propagation, this formula becomes the functional pigeonhole principle formula with n pigeons and $n-2$ holes, denoted $fPHP^n_{n-2}$. These formulas are known to have exponential AC^0-Frege lower bounds [BT88]. Since the lower bound is proved for AC^0-Frege proof systems, we show how to model the unit propagations using $O(n^3)$ steps in AC^0-Frege reasoning. Therefore, if there exists a sub-exponential AC^0-Frege (resp. RES) proof of $H(K_n^*)_{T,1,F}$, then there exists a sub-exponential

AC^0-Frege (resp. RES) proof of $fPHP^n_{n-2}$, which is a contradiction, since $fPHP^n_{n-2}$ has exponential lower bounds.

The details of the proof are as follows: The restriction that we apply to $H(K^*_n)_{T,1,F}$ is $m_{x,n} = True$. This will guarantee via the edge clauses that we cannot map any vertex to positions $n-1$ or $n+1$ because x has no edges incident on it. If we interpret the variables as mappings from pigeons to holes, we now have two more pigeons than holes. The restriction $m_{x,n} = True$ propagates as follows:

- For every function clause of the form $(\neg m_{x,n} \vee \neg m_{x,j})$, since we have set $m_{x,n}$ to True, we must set all of $m_{x,1}, m_{x,2}, ..., m_{x,n-1}$ as well as $m_{x,n+1}$ to False.
- For every 1-1 clause of the form $(\neg m_{x,n} \vee \neg m_{i,n})$, propagating $m_{x,n} = True$ causes us to set all of $m_{1,n}, m_{2,n}, ..., m_{n,n}$ to False.
- Finally, for every edge clause of the form $(\neg m_{x,n} \vee \neg m_{k,n+1})$ where (x, k) is a non-edge in G, propagating $m_{x,n} = True$ causes us to set all of $m_{1,n+1}, m_{2,n+1}, ..., m_{n,n+1}$ to False. Similarly, for each edge clause of the form $(\neg m_{i,n-1} \vee \neg m_{x,n})$, propagating causes us to set all of $m_{1,n-1}, m_{2,n-1}, ..., m_{n,n-1}$ to False.

The effect of these propagations on the various groups is as follows:

- Total Clauses: The restriction $m_{x,n} = True$ satisfies the clause $(m_{x,1} \vee m_{x,2} \vee ... \vee m_{x,n} \vee m_{x,n+1})$. Combined with this, the propagations $m_{i,n-1} = False$, $m_{i,n} = False$, and $m_{i,n+1} = False$ for all i causes the total clauses to become:
$$\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{n-2} m_{i,j})$$
- 1-1 Clauses: For each $1 \leq i \leq n+1, i \neq x$, there is a clause $(\neg m_{x,n} \vee \neg m_{i,n})$. Since every $m_{i,n}$ was set to False, every 1-1 clause involving any $m_{i,n}$ will be satisfied and eliminated. Due to the edge clause propagations, for every $i \neq x$, every clause involving $m_{i,n-1}$ or $m_{i,n+1}$ will also be eliminated. The 1-1 clauses therefore become:
$$\bigwedge_{j=1}^{n-2} \bigwedge_{i_1=1}^{n} \bigwedge_{\substack{i_2=1 \\ i_2 \neq i_1}}^{n} (\neg m_{i_1,j} \vee \neg m_{i_2,j})$$
- Function Clauses: For each $1 \leq j \leq n+1, j \neq n$ there is a clause $(\neg m_{x,n} \vee \neg m_{x,j})$. Since every $m_{x,j}$ was set to False, every function clause involving any $m_{x,j}$ will be satisfied and eliminated. Due to edge clause propagations, for every $i \neq x$, every clause involving $m_{i,n-1}$ or $m_{i,n+1}$ will also be eliminated. Due to the 1-1 clause propagations, for every $i \neq x$, every clause/ involving $m_{i,n}$ will also be eliminated. The function clauses therefore become:
$$\bigwedge_{i=1}^{n} \bigwedge_{j_1=1}^{n-2} \bigwedge_{\substack{j_2=1 \\ j_2 \neq j_1}}^{n-2} (\neg m_{i,j_1} \vee \neg m_{i,j_2})$$
- Edge Clauses: There are two types of edge clauses, those which contain the literal $\neg m_{x,n}$, and those which contain the literal $\neg m_{x,j}, j \neq n$. Note that this covers all edge clauses because vertex x is involved in every non-edge of K^*_n. Clauses of the first type are satisfied by unit propagation which forces the other literal in each such clause to be set to True. Those of the second type are satisfied by the $m_{x,j}$ propagations from the function clauses. All edge clauses are therefore eliminated.

These remaining clause groups when simplified by unit propagation are exactly the clauses from $fPHP^n_{n-2}$. In effect, the size $\leq N$ proof of $H(K^*_n)_{T,1,F}\restriction_{m_{x,n}=True}$ has been turned into a proof of $fPHP^n_{n-2}$. It is not hard to show that AC^0-Frege can perform unit propagations in polynomial size for some polynomial $p(n)$. This turns our size $\leq N$ proof of $H(K^*_n)_{T,1,F}\restriction_{m_{x,n}=True}$ to a size $N + p(n)$ proof of $fPHP^n_{n-2}$.

Let d be the depth bound imposed on a Frege system. Since AC^0-Frege proof systems are closed under restriction (i.e. restricting a proof yields a smaller proof), and since they have $\Omega(2^{\sqrt[5d]{n}})$ size lower bounds for $fPHP^n_{n-2}$ formulas [UF96], we may conclude that the $H(K^*_n)_{T,1,F}$ formulas also require proofs of size at least $\Omega(2^{\sqrt[5^d]{n}})$. Specifically, if there exist size $\leq N$ AC^0-Frege proofs of $H(K^*_n)_{T,1,F}\lceil_{m_{x,n}=True}$, then there exist size $\leq N + p(n)$ proofs of $fPHP^n_{n-2}$. $\square$

Clearly, this result also holds for all formulas such as $H(K^*_n)_{T,1}$ which are composed of proper subsets of the clauses from $H(K^*_n)_{T,1,F}$, because having fewer initial clauses certainly cannot help to find a shorter proof. The implications for SAT-solvers are immediate:

Corollary 11.2.2. *No SAT algorithm based on* AC^0-Frege *nor any weaker proof system can efficiently solve* $H(K^*_n)_{T,1,F}$ *formulas (or formulas containing a proper subset of those clauses). This includes* DPLL *as well as SAT-solving algorithms based on clause learning.*

We have therefore shown that the $H(G)_{T,1,F}$ reduction can convert trivial instances of the Hamiltonian Cycle problem to intractable SAT instances.

11.2.2 Formal Example of a Neutral Reduction

In Theorem 10.4.1 of Section 10.4.1 we showed that when applied to K^*_n graphs, the version of our reduction which uses Total, Onto, and Function clauses results in formulas which have polynomially-bounded T-RES refutations, providing an example of a neutral reduction. This is particularly interesting because both $H(K^*_n)_{T,O,F}$ and $H(K^*_n)_{T,1,F}$ are natural encodings of the Hamiltonian Cycle problem, and neither is a subset of the clauses of the other, but $H(K^*_n)_{T,O,F}$ is easy to solve, while $H(K^*_n)_{T,1,F}$ is intractably difficult. This formally proves that it is possible to have an exponential separation between two similar natural reductions applied to the same set of inputs.

Clearly, this result also holds for all formulas such as $H(K^*_n)_{T,O,1,F}$ which are composed of proper supersets of the clauses from $H(K^*_n)_{T,O,F}$, because having more initial clauses certainly cannot hurt minimum proof size:

Corollary 11.2.3. *The size of* T-RES *proofs for the unsatisfiability of* $H(K^*_n)_{T,O,1,F}$ *formulas have polynomial upper bounds.*

Our upper bound result of course has an immediate corollary for SAT-solvers:

Corollary 11.2.4. *For any* $H(K^*_n)_{T,O,F}$ *formula (or formula containing a superset of those clauses), there is a polynomially-bounded* DPLL *computation which solves it, assuming that the variables to branch on are chosen in the correct order.*

11.2.3 Formal Examples of Beneficial Reductions

As we saw from the previous two results, reductions can certainly present pitfalls for researchers working with SAT-solvers. However, they can also present opportunity, as was shown in earlier chapters. In this section we combine some earlier definitions and results to give a formal example of a beneficial reduction.

In Section 8.2.2 we introduced the $G_{\frac{n}{2},\frac{n}{2}}$ graphs. Each such graph consists of two cliques of size $\frac{n}{2}$, which are clearly non-Hamiltonian. Nevertheless, these graphs have exponential NHPS size lower bounds, which was shown in Theorem 9.7.1.

However, Corollary 10.4.4 proves that when we apply our reduction which uses the Total, Onto, 1-1, and Function clauses to the $G_{\frac{n}{2},\frac{n}{2}}$ graphs, we get the $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ formulas which have polynomial T-RES upper bounds.

In other words, this encoding maps an intractably hard input for one proof system to an easy input for another, and therefore constitutes a formal example of a beneficial reduction.

One interesting point of note is that the same reduction applied to different inputs can have different effects. For example, we saw from Corollary 11.2.3 that with the Total, Onto, 1-1, and Function clauses, our reduction is neutral when applied to the K_n^* graphs. However, as we just saw, the same reduction applied to the $G_{\frac{n}{2},\frac{n}{2}}$ graphs is beneficial.

In addition, researchers have empirically shown that adding redundant clauses to formulas can transform very difficult instances of SAT into very easy ones. We can further weaken the easy formulas from Theorem 10.4.1 with 1-1 clauses to obtain the formulas of Corollary 11.2.3 which contain the clauses of the hard formulas from Theorem 11.2.1 as a proper subset but now have T-RES refutations of polynomial size. This provides a formal example of hard T-RES/DPLL instances which can be converted to easy instances by the addition of redundant onto clauses, giving another example of a beneficial reduction.

11.3 Domain Independent Framework for Comparing Encodings

In this section we describe a formal domain-independent framework which captures our intuitive notions of what constitute dangerous and safe reductions, and show that not only can reductions increase the complexity of problems, but they can beneficially reduce their complexity by applying polytime reasoning which is unavailable to the target proof system. We use this framework to categorize encodings as being harmful, neutral, or beneficial.

Currently, no system exists to classify encodings according to whether they make problem instances harder or easier. Such a classification system might prove to be very helpful to researchers who are actively using SAT-solvers to tackle $\mathcal{NP}$-Complete problems. It might also prove to be beneficial for researchers who are interested in studying the phenomenon of dangerous encodings more abstractly with an eye to finding general principles for predicting which encodings will lead to complexity blow-ups on certain families of formulas. In this section, we provide a framework for such a system.

Although no such system has yet been devised, much work has gone into classifying the power of proof systems. These proof systems have been organized into a hierarchy based on p-simulations and exponential separations (see Definition 2.4.1) which is shown in Figure 2.2. A more detailed portion of this hierarchy dedicated to Resolution-based proof systems is shown below in Figure 11.1. This portion of the proof complexity hierarchy is particularly relevant to automated theorem proving and SAT-solving.

Each node in the diagram represents all of the families of formulas which have polynomial size refutations in the system labelling the node. Arrows represent p-simulation relationships between systems. An arrow from system α to system β means that α p-simulates β. A slash through an arrow from α to β represents an exponential separation between β and α. An arrow labelled with a question mark denotes an unknown relationship and open problem. Systems to the left in the diagram are generally stronger than systems to the right. Though short refutations exist for larger classes of formulas in stronger systems, finding them is generally more difficult than finding refutations in weaker systems. Hence SAT-solvers based on the resolution rule generally do not have the full power of RES, but instead implement some form of DPLL which is equivalent to T-RES and is very low in the hierarchy. It is therefore desirable for instances which we want to solve to exist in nodes that are low down in the hierarchy. This gives us a better chance of finding a refutation deterministically in a short amount of time.

We can use this hierarchy to judge the quality of SAT encodings. If some input to an encoding is at one level of the hierarchy and its corresponding output exists only in higher levels, then the encoding is dangerous with respect to that input since its result requires more power to solve. If the input and output of an encoding exist in all of the same levels of the hierarchy, then the encoding is neutral with respect to that input. If some input to an encoding exists nowhere below a certain level in the hierarchy, but its output does, then that encoding actually makes a potentially exponential contribution towards solving the instance. Since every encoding only takes polynomial time to compute, such beneficial encodings can be used as efficient preprocessing steps and identifying them is of great practical interest. We coin the terms *Explosive*, *Stable*, and *Implosive* to refer to encodings which are harmful, neutral, and beneficial with respect to certain families of formulas and certain proof systems. These are defined formally below.

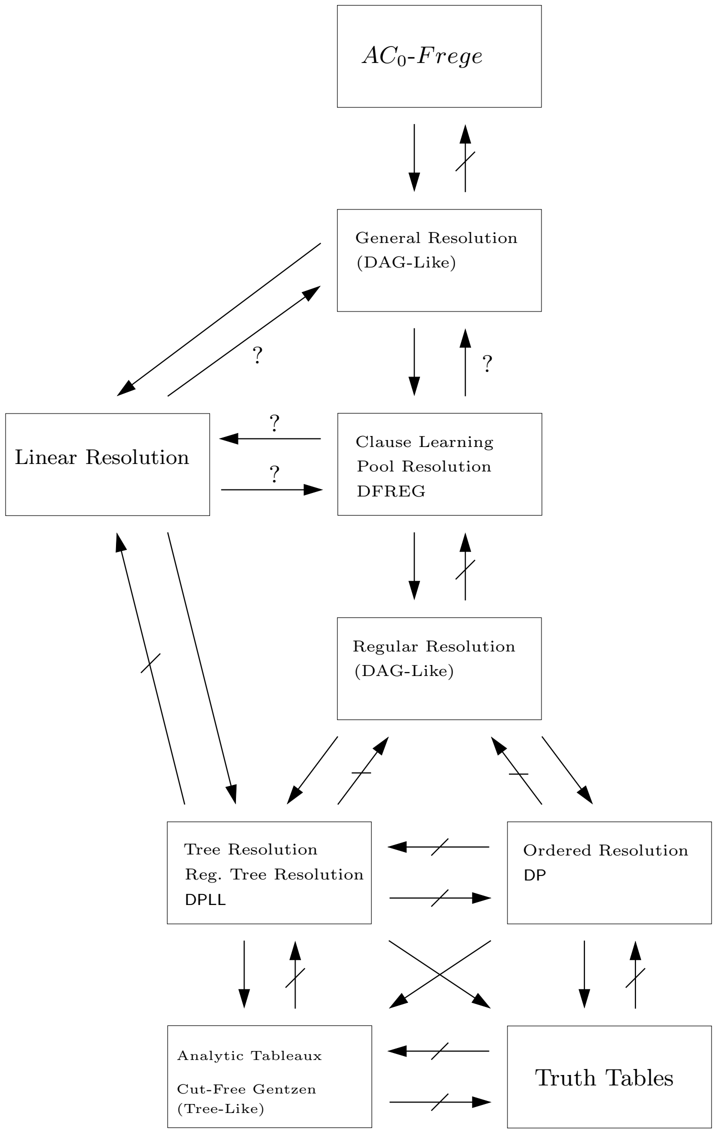

Figure 11.1: Part of the Proof Complexity Hierarchy which Relates Various Resolution Refinements

11.3.1 Explosivity

Definition 11.3.1 (Explosive Reduction). *Let α be a proof system for language L_1, let β be a proof system for language L_2, and let $R : L_1 \to L_2$ be a reduction from L_1 to L_2. If there exists some family of strings $X = \{x_1, x_2, ...\}$, $X \subseteq L_1$ such that for all k and for all $x_i \in X$ there exists an α-proof P_1 of x_i, but there exists no β-proof P_2 of $R(x_i)$ such that $|P_2| \leq |P_1|^k$, then we say that the reduction R is (α, β)-explosive on the set X.*

This definition corresponds to our intuitive notion of what constitutes a dangerous reduction, and we can immediately apply it to our main result:

Corollary 11.3.2. *The Hamiltonian Cycle to SAT reduction above which uses the Total, 1-1, and Function clauses is (α, AC^0-Frege)-explosive on the set containing the K_n^* graphs for any non-Hamiltonicity proof system α which has polynomially-bounded proofs of the K_n^* graphs.*

An example of such a non-Hamiltonicity proof system is the NHPS from Chapter 9.

Another formal example of explosivity comes from a corollary of the main result of Chapter 12 which proves that the reduction from QBF to Intuitionistic Propositional Logic (IPL) given by Statman in [Sta79] is probably explosive:

Corollary 11.3.3 (Restated From Chapter 12). *Unless the proof system $\mathsf{LK}[\vec{E_S}]$ is super, Statman's reduction is $(\alpha,\ \mathsf{LJ}[\vec{E_S}])$-explosive for any QBF proof system α which has polynomially-bounded proofs for any prenex instance of the law of excluded middle (i.e. formulas of the form $p \vee \neg p$).*

Explosivity is caused when an encoding increases the proof complexity of the input instance. In the case of RES, if a reduction fails to introduce clauses which are needed in order to provide a short RES proof, then the reduction is explosive, and there is no hope of solving the translation. The 'onto' clauses from our reduction are an example of ones which have no polynomially-bounded RES derivations themselves and can make an exponential difference to the proof complexity of the reduction's output.

In addition, explosivity is trivially associated with exponential separations between proof systems. Every example of p-simulation between two proof systems on the same language for which there is a superpolynomial separation implicitly gives an example of (α, β)-explosivity. If proof system α p-simulates proof system β, but β does not p-simulate α, then the trivial reduction of doing nothing is (α, β)-explosive on the set of formulas which provides the separation. For this reason, (α, β)-explosive reductions where α is a strictly stronger proof system than β (for example, $(\mathsf{AC^0}\text{-}\mathsf{Frege}, \mathsf{T}\text{-}\mathsf{RES})$-explosive reductions) are not nearly as interesting as (α, β)-explosive reductions in which α is a strictly weaker proof system than β.

11.3.2 Stability

Definition 11.3.4 (Stable Reduction). *Let α, β, L_1, L_2, and R be as in Definition 11.3.1. If there exist constants k_1 and k_2 and a family of strings $X = \{x_1, x_2, ...\}$, $X \subseteq L_1$ such that for any α-proof P_1 of an $x_i \in X$ there exists a β-proof P_2 of $R(x_i)$ where $|P_2| \leq |P_1|^{k_1}$ and $|P_1| \leq |P_2|^{k_2}$ then we say that the reduction R is (α, β)-stable on the set X.*

An example of stability comes from Theorem 10.4.1 which shows that the $H(K_n^*)_{T,O,F}$ formulas have polynomial T-RES size upper bounds:

Corollary 11.3.5. *The Hamiltonian Cycle to SAT reduction from Section 8.2.1 which uses the Total, Onto, and Function clauses is $(\alpha,\ \mathsf{T}\text{-}\mathsf{RES})$-stable on the set of K_n^* graphs for any non-Hamiltonicity proof system α which has polynomially-bounded proofs for the K_n^* graphs.*

As already mentioned, the NHPS proof system from Chapter 9 is such an α.

Another interesting example of stability is the relationship between RES and Linear Resolution (L-RES) shown by Buresh-Oppenheim and Pitassi in [BOP03]. The authors provide a very simple reduction R which consists of adding trivial clauses of the form $(x_i \vee \neg x_i \vee x_j)$ and $(x_i \vee \neg x_i \vee \neg x_j)$ for all pairs of variables x_i, x_j in the original formula, and show that for every RES refutation of a formula F, there exists an L-RES proof of $R(F)$ which is only polynomially larger. In other words, R is (RES, L-RES)-stable on all formulas which have polynomially-bounded RES proofs.

From a proof-complexity point of view, every example of p-equivalence implicitly gives a trivial example of (α, β)-stability. If α and β are two p-equivalent proof systems for the same language L, then the trivial reduction of doing nothing is both (α, β)-stable and (β, α)-stable for the entire language L. For this reason, (α, β)-stable reductions for p-equivalent proof systems are not nearly as interesting as ones for proof systems for which there is a superpolynomial separation.

11.3.3 Implosivity

In practical terms, an even more interesting characteristic for encodings is that of implosivity. Intuitively, an encoding which takes hard formulas for one proof system and converts them into easy ones for another is implosive. In other words, implosive reductions can make otherwise hard instances more accessible to SAT-solvers. Examples of such beneficial reductions are already known to researchers. For example, Kautz and Selman show that SAT encodings of constraint satisfaction problem (CSP) instances can be optimized with respect to local consistency checking and unit propagation [KS03]. In this case the reduction from CSP to SAT actually has beneficial properties, namely that it reduces the proof complexity of its inputs with respect to the consistency conditions. Bailleux and Boufkhad give another good example in [BB03], where they give an encoding that transforms the parity problem, which for many years was considered to be a hard DIMACS instance, into formulas that are easy for DPLL-based solvers.

More formally, the beneficial property of implosivity is defined as follows:

Definition 11.3.6 (Implosive Reduction). *Let α, β, L_1, L_2, and R be as in Definition 11.3.1. If there exists some family of strings $X = \{x_1, x_2, ...\}$, $X \subseteq L_1$ such that for all k and for all $x_i \in X$ there exists a β-proof P_2 of $R(x_i)$ but there exists no α-proof P_1 of x_i such that $|P_1| \leq |P_2|^k$, then we say that the reduction R is (α, β)-implosive on the set X.*

As we saw in Section 11.2.3, the $G_{\frac{n}{2},\frac{n}{2}}$ graphs have exponential NHPS lower bounds, but $H(G_{\frac{n}{2},\frac{n}{2}})_{T,O,1,F}$ has polynomial T-RES upper bounds, making this reduction a formal example of implosivity:

Corollary 11.3.7. *The Hamiltonian Cycle to SAT reduction from Section 8.2.1 which uses the Total, Onto, 1-1, and Function clauses is (NHPS, T-RES)-implosive on the $G_{\frac{n}{2},\frac{n}{2}}$ graphs.*

This gives us a clear and formal example of how different inputs to the same system can be simplified or complicated depending on encoding, because the Total, 1-1, Function version of the reduction is (NHPS, AC^0-Frege)-explosive on the K_n^* graphs.

The other example from Section 11.2.3 which descirbes the addition of redundant clauses is another example of an implosive reduction:

Corollary 11.3.8. *The reduction consisting of the addition of onto clauses is (AC^0-Frege, T-RES)-implosive on the $H(K_n^*)_{T,1,F}$ formulas.*

An interesting potential example of implosivity is the L-RES reduction R from [BOP03] mentioned above. As already stated, R is (RES, L-RES)-stable on the entire SAT language. However, it is unknown whether an exponential separation exists between L-RES and RES. If so, then R is (L-RES, L-RES)-implosive on the inputs which give the separation. Such examples of reflexive implosivity are good candidates for beneficial preprocessing.

Generally speaking, non-trivial implosivity arises when polytime reductions make use of reasoning which is not available to their target proof system. For example, a polytime reduction might add clauses to an input which could not otherwise be derived concisely in RES. Given these clauses, Resolution-based solvers can easily solve the problem, but without them, they require exponential time. In effect, such reductions allow solvers to 'cheat' and do work that cannot be done by their underlying proof systems.

As with explosivity, implosivity is trivially associated with p-simulation. Every example of p-simulation between two proof systems on the same language for which there is a superpolynomial separation implicitly gives an example of (α, β)-implosivity. If proof system β p-simulates proof system α, but α does not p-simulate β, then the trivial reduction of doing nothing is (α, β)-implosive on the set of formulas which gives the separation. For this reason, (α, β)-implosive reductions where α is strictly weaker than β (for example, (T-RES, AC^0-Frege)-implosive reductions) are not nearly as interesting as (α, β)-implosive reductions in which α is strictly stronger than β.

11.3.4 Alternate Hierarchies

Although the proof system hierarchy may prove to be useful for classifying encodings, we could also produce alternative hierarchies for which the notions of explosivity, stability, and implosivity could be used to classify encodings. In order to use the proof system hierarchy for this task we need to perform a fairly robust analysis of the family of problem instances being studied, as we did in Sections 11.2.1 and 11.2.2. A more empirical hierarchy based on the real world performance of specific implementations on families of inputs may be preferred.

11.4 Implications for Proof Complexity

Whenever a relationship between two proof systems on different languages is studied, there must necessarily be a reduction involved. Weak proof systems such as RES and its refinements are not powerful enough to perform polytime reductions. This necessitates the use of a separate polytime algorithm for performing the reduction. Since the details of the reduction can affect the proof complexity of its output, it does not make sense to talk about p-simulation or exponential separation between two weak proof systems over different languages. Rather, one must talk about p-simulation or exponential separation *with respect to a specific reduction and a specific family of inputs.* If a reduction exists which allows one proof system to p-simulate another, then we say that the first proof system effectively p-simulates the second. This notion is formally described in Definition 2.4.2.

We can just as easily consider this definition applied to two proof systems over the same language. This yields a generalization of the normal notion of p-simulation. For example, though L-RES is not known to p-simulate RES, it does effectively p-simulate RES since the polytime reduction in [BOP03] is (RES, L-RES)-stable on the entire SAT language.

11.5 Open Problems & Conjectures Related to Dangerous Reductions

The idea that encodings can inject complexity into a problem is disconcerting. It is worrisome to think that a reduction from one problem to another can negatively affect the proof complexity of the result and potentially make the instance difficult for proof systems which are located several levels higher in the proof complexity hierarchy than the intended system. Furthermore, as we have shown in this chapter, this phenomenon can happen with very natural and even obvious encodings. Even more worrisome is that it does not seem to be at all obvious which types of reductions have this property. With our example, we were lucky enough to see that the input graph was translated to a formula which is very similar to the pigeonhole formulas, but in general we cannot expect to be so lucky. There are probably infinitely many families of formulas which have no short RES proofs and it would not be easy to identify them lurking within the output of an encoding. Random formulas, which are very hard to categorize, as well as other combinatorial problems which have never even been investigated could act very much like the pigeonhole formulas do in our example. If we do not even know what these formulas look like, then it is probably very difficult to predict and avoid reductions which might produce them or something similar to them. Further research is needed in order to characterize which types of reductions have this property.

As a first step towards a characterization, we have outlined a framework for comparing encodings based on the proof complexity hierarchy. The key idea behind the framework is that encodings can affect the proof complexity of the result either beneficially or adversely by overcoming the superpolynomial separation between two proof systems through the use of reasoning that is unavailable to the proof system or by requiring the proof system to derive clauses which cannot be derived concisely.

Nevertheless, the results in this chapter are geared towards classification rather than prediction. Although we were able to design a framework for comparing reductions, this is a tool to be used once the complexities of the reductions are already known, and lacks predictive power. Being able to predict which reductions are beneficial or harmful would be extremely valuable to researchers working with SAT-solvers, and remains an important open problem. However, depending on how this problem is formalized, it may not even be decidable. For example, if we phrase the dangerous reduction prediction problem as taking two Turing Machines (representing competing reductions) as its input and then determining which one produces formulas that are easier for SAT-solvers, then this problem is most likely undecidable for the simple reason that computing almost any non-trivial property of a Turing Machine is undecidable. If we cannot even compute *whether* a Turing Machine outputs a formula, then there isn't much hope in deciding which of two Turing Machines will output a better formula.

However, if we formalize the dangerous reduction prediction problem slightly differently, then it becomes very interesting and is closely related to the research in Part II of this thesis. Instead of taking two competing Turing Machines as inputs, we could take the two formulas which were output by our competing reductions and ask which of these formulas has a shorter RES refutation. However, this is almost exactly the RES size problem which we described in Section 4.9.3 as one of the most interesting open Resolution resource problems. Lemma 4.9.2 shows it to be co$\mathcal{NP}$-Hard, and the best upper bound is a $\mathcal{NEXPTIME}$ algorithm. This unfortunately suggests that the dangerous reduction prediction problem probably does not have a polytime algorithm. In fact, as we saw from Corollary 5.5.4, even the size problem for I-RES is $\mathcal{NP}$-Complete, which does not bode well for our predictive abilities for RES size.

In any case, the RES size problem is a very interesting connection between Parts II and III of this thesis and is probably the most interesting open problems relating to both Resolution resource problem and dangerous reductions.

Yet another interesting avenue of future research for this work has to do with effective p-simulation. We saw that the notion of effective p-simulation is important not only when comparing proof systems for different languages, but also for proof systems on the same language, allowing an apparently weaker proof system to effectively become as powerful as a stronger one. In other words, effective p-simulation has the potential to give us very powerful preprocessing algorithms. A good example of this is work done by Philipp Hertel, Toni Pitassi, Fahiem Bacchus, and Allen Van Gelder [HPBG08] showing that clause learning effectively p-simulates RES.

However, the importance of effective p-simulation runs even deeper than this, and suggests many interesting open problems in the area of proof complexity. For example, can Frege effectively p-simulate E-Frege? Can RES effectively p-simulate AC^0-Frege or any of the more powerful proof systems? Can T-RES effectively p-simulate RES? Is there any automatizable proof system which effectively p-simulates any non-automatizable proof system? All of these are interesting open problems, and positive answers would have a strong impact in the area of automated theorem proving and SAT-solving.

Chapter 12

The Proof Complexity of Intuitionistic Propositional Logic

12.1 Introduction & Motivation

Intuitionistic propositional logic (IPL) is perhaps the best-studied non-classical system of logic, and since it disallows proofs by contradiction, it is associated with constructivism. It contains all of the standard connectives $\wedge, \vee, \supset, \neg$, but uses non-classical semantics that rely on Kripke models [TD88]. The validity problem for intuitionistic logic appears to be intrinsically more complex than the corresponding problem for classical logic. A well-known paper by Statman [Sta79] shows via a natural reduction from QBF that the problem of determining whether a formula is intuitionistically valid is $\mathcal{PSPACE}$-Complete.

The purpose of this chapter is to explore the proof complexity of IPL, considered as a proof system for a $\mathcal{PSPACE}$-Complete set. Since it does not allow proofs by contradiction, intuitionistic logic is weaker than classical logic. Hence it is reasonable to conjecture that there are classical tautologies that are also provable intuitionistically, but whose shortest constructive proofs are superpolynomially longer than their minimal proofs in classical logic. From a complexity-theoretic perspective, IPL is in some regards more tractable than classical logic. For example, it has feasible interpolation [Pud99, BP01], making it reasonable for researchers to suspect that IPL lower bounds might be easier to establish than those for classical logic. This suspicion was recently shown to be correct by Pavel Hrubeš, who in a remarkable paper [Hru07] proved an exponential lower bound for intuitionistic Frege systems.

Motivated in part by the interest generated by Hrubeš's result, the main theorem of this chapter provides more evidence that IPL lower bounds for all proof systems might be within reach. Although we do not show IPL to be weak in an absolute sense, we do show it to be weak relative to Statman's translation. More specifically, we show that unless a variant of the proof system LK called LK$[\vec{E_S}]$ is super (and therefore $\mathcal{NP} = co\mathcal{NP}$), Statman's reduction from QBF to IPL cannot even translate trivial classical instances of the law of excluded middle into intuitionistic formulas which have polynomially-bounded proofs. Since Statman's translation is the obvious and natural reduction to use, this result shows that if a more feasible reduction exists, then it must be quite complicated.

Perhaps even more important than this result itself are the techniques used in proving it. Since these techniques are particularly powerful, there is hope that they might be adapted to solving other related problems.

This chapter is organized as follows: Section 12.7 contains the main result, and the sections preceding it contain various theorems and lemmas which are needed to prove it.

In Section 12.2, we describe LJ, the standard sequent calculus system formulated by Gentzen for IPL. Section 12.3 describes Statman's translation from QBF to IPL and introduces the proof system LJ$[\vec{E_S}]$, an augmented form of LJ which has been strengthened by including the extension variables from the translation as axioms. This is the proof system that we use in our main result.

In Section 12.4 we show that it is possible to take any proof in LJ$[\vec{E_S}]$ and eliminate all cuts not involving extension axioms (thereby producing a new potentially exponential proof) without affecting the closure of the proof. In Section 12.5 we show that any sequent in the closure of a proof can be derived efficiently. Together, Sections 12.4 and 12.5 constitute an important proof technique, since they allow us to wander into the realm of exponentially-large proofs, take advantage of the reasoning which is possible there, and then extract what was learned back in the polynomially-bounded realm.

Section 12.6 contains two critical lemmas which are closely related to the 'disjunction property' of IPL.

Finally, Section 12.7 contains the main result, which is followed by the main theorem's immediate implications, described in Section 12.8, and open problems, described in Section 12.9.

12.2 The System LJ

For the purposes of this chapter, we are dealing with a sequent calculus for IPL in the style of Gentzen [Gen35a, Gen35b, Sza69]. We will use capital letters $A, B, C, ...$ to denote complex formulas, lower-case letters $x, y, z, ...$ to denote atomic formulas, and Greek letters $\Gamma, \Delta, \Theta, ...$ to denote sets.

The proof system that we will be using is essentially the same as Gentzen's system LJ. Our proofs are tree-like, meaning that each sequent can be an input for at most one inference rule. Unlike LK, each sequent in LJ has singular right side; that is, there is at most one formula on the right-hand side of each sequent. In addition, all axioms are of the form $x \mapsto x$, where x is an atomic sentence letter. LJ is formulated as follows:

Axiom:

$$\frac{}{x \mapsto x} \qquad \text{where } x \text{ is atomic}$$

Weakening:

$$\textbf{left} \quad \frac{\Gamma \mapsto A}{B, \Gamma \mapsto A} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \mapsto}{\Gamma \mapsto B}$$

Cut:

$$\frac{\Gamma \mapsto A \quad A, \Delta \mapsto B}{\Gamma, \Delta \mapsto B}$$

$\perp$ Introduction:

$$\textbf{left} \quad \frac{}{\perp \mapsto} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \mapsto}{\Gamma \mapsto \perp}$$

$\neg$ Introduction:

$$\textbf{left} \quad \frac{\Gamma \mapsto A}{\neg A, \Gamma \mapsto} \qquad \text{and} \qquad \textbf{right} \quad \frac{A, \Gamma \mapsto}{\Gamma \mapsto \neg A}$$

$\vee$ Introduction:

$$\textbf{left} \quad \frac{A, \Gamma \mapsto C \quad B, \Delta \mapsto C}{A \vee B, \Gamma, \Delta \mapsto C} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \mapsto A}{\Gamma \mapsto A \vee B} \quad \text{as well as} \quad \frac{\Gamma \mapsto A}{\Gamma \mapsto B \vee A}$$

$\wedge$ Introduction:

$$\textbf{left}\quad \frac{A, B, \Gamma \mapsto C}{A \wedge B, \Gamma \mapsto C} \qquad \text{and} \qquad \textbf{right}\quad \frac{\Gamma \mapsto A \quad \Delta \mapsto B}{\Gamma, \Delta \mapsto A \wedge B}$$

$\supset$ Introduction:

$$\textbf{left}\quad \frac{\Gamma \mapsto A \quad B, \Delta \mapsto C}{A \supset B, \Gamma, \Delta \mapsto C} \qquad \text{and} \qquad \textbf{right}\quad \frac{A, \Gamma \mapsto B}{\Gamma \mapsto A \supset B}$$

One important point of note is that if you remove the requirement of having only a single formula on the right-hand side of a sequent, then the system LJ becomes the system LK for classical logic [Sza69]. In other words, every LJ proof is also an LK proof.

12.3 Statman's Translation & $\mathsf{LJ}[\vec{E_S}]$

In this section we define an augmented form of LJ that includes extension axioms. We then review Statman's reduction from QBF to IPL. Next we define a specific form of LJ augmented with Statman's extension axioms. Finally, we prove that it is easy to extract the ultimate y extension axiom from Statman's translation.

12.3.1 LJ Variant

We define $\mathsf{LJ}[\vec{E}]$ as follows:

Definition 12.3.1 ($\mathsf{LJ}[\vec{E}]$). *We shall use $\mathsf{LJ}[\vec{E}]$ to denote an augmented form of* LJ *where $\vec{E} = E_1, ..., E_n$, and each E_i contains the pair of sequents $A \circ B \mapsto y_{A \circ B}$ and $y_{A \circ B} \mapsto A \circ B$ with the restriction that $A \circ B$ must be a complex formula, and $y_{A \circ B}$ is an atom not appearing in $E_1, ..., E_{i-1}$. We allow the sequents in $\vec{E}$ to be used as axioms in any $\mathsf{LJ}[\vec{E}]$ proof. We refer to $\mathsf{LJ}[\vec{E}]$ as an extended sequent calculus, and the two sequents in each E_i are referred to as extension axioms. This same augmentation can be defined for the system* LK.

12.3.2 Statman's Translation

In his paper showing that IPL is $\mathcal{PSPACE}$-Complete [Sta79], Statman proves this result by providing a reduction from QBF to IPL. This translation is very important to our main result, and proceeds as follows:

1. As input take a QBF formula $F_{QBF} = Q_n x_n, ..., Q_1 x_1 B_0$, where B_0 is a quantifier-free prenex formula, and each $Q_i = \forall$ or $\exists$.

2. Let $y_0, ..., y_n$ be entirely new variables not appearing in F_{QBF}. These are extension variables that are necessary to keep our translation from growing exponentially.

3. Define a series of $B_i^{\checkmark}$ formulas as follows (note that these are not extension variables, but rather are just shorthand for the purposes of this reduction; similarly, $A \leftrightarrow B$ is shorthand for $(A \supset B) \wedge (B \supset A)$:
   - $B_0^{\checkmark} = \neg\neg B_0 \leftrightarrow y_0$
   - $B_k^{\checkmark} = ((x_k \vee \neg x_k) \supset y_{k-1}) \leftrightarrow y_k$ if $Q_k = \forall$
   - $B_k^{\checkmark} = ((x_k \supset y_{k-1}) \vee (\neg x_k \supset y_{k-1})) \leftrightarrow y_k$ if $Q_k = \exists$

4. Output $F_{IPL} = B_0^{\checkmark} \supset (B_1^{\checkmark} \supset (...(B_{n-1}^{\checkmark} \supset (B_n^{\checkmark} \supset y_n))...))$

12.3.3 Proof of Correctness

A full proof of correctness showing that F_{QBF} is true if and only if F_{IPL} is intuitionistically valid does not add to the understanding of this chapter, so we shall refer the interested reader to [Sta79]. However, part of the proof is of interest. Specifically, Statman's proof of correctness implicitly proves that a cut-free tree-like sequent calculus formulation of LJ p-simulates Boolean Truth Trees (BTT), a brute-force tree-like system for QBF.

Boolean Truth Trees

The BTT proof system is defined as follows:

- Every BTT proof is a tree in which every node contains a fully-quantified QBF formula. These formulas may contain constants 0 and 1. The root contains the formula which is to be proven true, and each leaf contains a formula in which all variables have been replaced by constants, and it consequently evaluates to 1, thereby showing that the formula is true. (It is most convenient to picture the proof with the root at the top and the leaves at the bottom).
- Every internal node v in the proof tree has either one or two children. If the outermost quantifier in the formula F at v is $\exists x$, then v has one child containing $F^{'}$, which is F with $\exists x$ removed, and every instance of x replaced by an appropriate constant such that $F^{'}$ is true. If the outermost quantifier in the formula F at v is $\forall x$, then v has two children, one containing F_0, where all instances of x have been replaced by 0, and the other containing F_1, where all instances of x have been replaced by the value 1.

It is easy to see that BTT is sound; if a BTT tree can be built for a QBF formula F, then F is true. It is also easy to see that BTT is complete; every true QBF formula has a BTT proof. This is a brute-force proof system, because it is extremely inefficient; the size of a BTT proof is exponential in the number of $\forall$ quantifiers contained in the formula to be proved true.

BTT Example

Consider the formula $\forall x \exists y \forall z(((\neg x \land \neg y) \lor z) \lor ((x \land y) \lor z))$. The following BTT proof shows that it is a true QBF formula:

$$
{\scriptstyle [0/x]}\ \frac{\forall x \exists y \forall z(((\neg x \land \neg y) \lor z) \lor ((x \land y) \lor z))}{
{\scriptstyle [0/z]}\ \dfrac{\dfrac{\exists y \forall z(((1 \land \neg y) \lor z) \lor ((0 \land y) \lor z))}{\forall z(((1 \land 1) \lor z) \lor ((0 \land 0) \lor z))}\ {\scriptstyle [0/y]}}{\begin{array}{cc} ((1 \land 1) \lor 0) \lor ((0 \land 0) \lor 0) & ((1 \land 1) \lor 1) \lor ((0 \land 0) \lor 1) \\ = 1 & = 1 \end{array}}\ {\scriptstyle [1/z]}
\qquad
{\scriptstyle [0/z]}\ \dfrac{\dfrac{\exists y \forall z(((0 \land \neg y) \lor z) \lor ((1 \land y) \lor z))}{\forall z(((0 \land 0) \lor z) \lor ((1 \land 1) \lor z))}\ {\scriptstyle [1/y]}}{\begin{array}{cc} ((0 \land 0) \lor 0) \lor ((1 \land 1) \lor 0) & ((0 \land 0) \lor 1) \lor ((1 \land 1) \lor 1) \\ = 1 & = 1 \end{array}}\ {\scriptstyle [1/z]}
}\ {\scriptstyle [1/x]}
$$

P-Simulation Result

We shall now make explicit the implicit p-simulation result in Statman's proof of correctness; this provides a deeper insight into the mechanisms at play underlying the reduction.

If we forget about the extension variables, then Statman's translation converts the formula F_{QBF} to an intermediate formula A^+ as follows:

- $B_0^+ = \neg\neg B_0$
- $B_k^+ = (x_k \lor \neg x_k) \supset B_{k-1}^+$ if $Q_k = \forall$
- $B_k^+ = (x_k \supset B_{k-1}^+) \lor (\neg x_k \supset B_{k-1}^+)$ if $Q_k = \exists$

Note that $A^+ = B_n^+$. Statman's proof of correctness proceeds in two parts. He first shows that F_{QBF} is true if and only if A^+ is intuitionistically provable. Next he proves that A^+ is intuitionistically provable if and only if F_{IPL} is. We are only interested in the forward direction of the first part; the implicit p-simulation result is given by the proof that F_{QBF} being true implies that A^+ is intuitionistically provable.

The overall idea of the simulation is to take the brute-force BTT proof tree T_{BTT} for F_{QBF}, and by way of very local transformations, build an exactly analogous cut-free tree-like LJ proof T_{LJ}. In order to prove this result we shall make use of the following Lemma:

Lemma 12.3.2. *Let A be any intuitionistically valid formula, let $l_1, ..., l_n$ be a sequence of literals containing all of the variables in A, and let V be the classical truth assignment which sets all of $l_1, ..., l_n$ to true. If A is true under V, then there exists a cut-free, tree-like* LJ *proof of $l_1, ..., l_n \mapsto A$, and if A is false under V, then there exists a cut-free, tree-like* LJ *proof of $l_1, ..., l_n, A \mapsto$; in either case the size of the proof is linear in the number of logical particles contained in A, and therefore has size $O(n)$.*

The proof is by induction on the number of logical particles in A and is completely straightforward; the basis consists of axioms, which contain no logical particles, and the induction step has eight cases to be considered, four for when A is true, and four for when A is false. An interesting corollary to this Lemma is that the double negation in Statman's translation is unnecessary, and in fact is better omitted.

This brings us to our p-simulation result:

Theorem 12.3.3. *Cut-free tree-like* LJ *p-simulates* BTT*.*

Proof: First build T_{BTT} as follows:

1. The root contains $\vdash_{\mathsf{BTT}} F_{QBF}$, where $F_{QBF} = Q_n x_n, ..., Q_1 x_1 B_0$.
2. For every tree node containing $\vdash_{\mathsf{BTT}} \forall x_i, Q_{i-1} x_{i-1}, ..., Q_1 x_1 B_0$, create two new children $\vdash_{\mathsf{BTT}} Q_{i-1} x_{i-1}, ..., Q_1 x_1 B_0[0/x_i]$ and $\vdash_{\mathsf{BTT}} Q_{i-1} x_{i-1}, ..., Q_1 x_1 B_0[1/x_i]$.
3. For every tree node containing $\vdash_{\mathsf{BTT}} \exists x_i, Q_{i-1} x_{i-1}, ..., Q_1 x_1 B_0$, create one new child node, either $\vdash_{\mathsf{BTT}} Q_{i-1} x_{i-1}, ..., Q_1 x_1 B_0[0/x_i]$ or $\vdash_{\mathsf{BTT}} Q_{i-1} x_{i-1}, ..., Q_1 x_1 B_0[1/x_i]$, depending on which form makes the formula true.

Using T_{BTT} as a template, build T_{LJ} as follows (note that unlike with most LJ proofs which have the root at the bottom, we are building a proof with the root at the top so as to make the relationship with BTT as clear as possible):

1. The root contains $\mapsto A^+$ (which is Statman's translation of F_{QBF} from step 1 of the T_{BTT} construction above).
2. For every tree node containing $l_n, ..., l_{k+1} \mapsto (x_k \vee \neg x_k) \supset B^+_{k-1}$, introduce the following three new nodes:

$$
\begin{array}{c}
l_n, ..., l_{k+1} \mapsto (x_k \vee \neg x_k) \supset B^+_{k-1} \\
| \\
l_n, ..., l_{k+1}, x_k \vee \neg x_k \mapsto B^+_{k-1} \\
\swarrow \qquad \searrow \\
l_n, ..., l_{k+1}, x_k \mapsto B^+_{k-1} \qquad l_n, ..., l_{k+1}, \neg x_k \mapsto B^+_{k-1}
\end{array}
$$

   Note that this requires only one application of $\vee$-*Left*, and one application of $\supset$-*Right* (remember that we are looking at the proof upside-down). This step corresponds exactly to step 2 of the T_{BTT} construction above.

3. For every tree node containing $l_n, ..., l_{k+1} \mapsto (x_k \supset B^+_{k-1}) \vee (\neg x_k \supset B^+_{k-1})$, introduce the following two new nodes:

$$
\begin{array}{c}
l_n, ..., l_{k+1} \mapsto (x_k \supset B^+_{k-1}) \vee (\neg x_k \supset B^+_{k-1}) \\
| \\
l_n, ..., l_{k+1} \mapsto l_k \supset B^+_{k-1} \\
| \\
l_n, ..., l_{k+1}, l_k \mapsto B^+_{k-1}
\end{array}
$$

Note that this requires only one application of $\supset$-*Right*, and one application of $\vee$-*Right*. This step corresponds exactly to step 3 of the T_{BTT} construction above.

This construction halts once we reach the leaves of T_{BTT}. Its 1-leaves correspond to sequents of the form $l_n, ..., l_1 \mapsto \neg\neg B_0$ in T_{LJ}, but $l_n, ..., l_1$ entail $\neg\neg B_0$, so by Lemma 12.3.2, each of these sequents has a size-$O(n)$ cut-free, tree-like LJ proof. In effect, there is a series of local transformations which directly relates the steps in T_{BTT} to the steps in T_{LJ} such that each local step in T_{BTT} grows at most linearly to become a local step in T_{LJ}. It is therefore easy to see that the size of T_{LJ} is linear in the size of T_{BTT}. This completes the p-simulation. □

In effect, the obvious brute-force BTT proof translates into a brute-force cut-free, tree-like LJ proof.

It is interesting to note that the Disjunction Property of IPL (see section 12.6) is what allows for this p-simulation to work. However, it is also the Disjunction Property which allows for the proof of our main theorem, which ultimately shows that the translation is not feasible. The Disjunction Property therefore acts as both an aid as well as a hindrance to Statman's reduction.

12.3.4 The System $\mathsf{LJ}[\vec{E_S}]$

Given that we have the definition of $\mathsf{LJ}[\vec{E}]$ as well as the details of Statman's translation (the full version, with extension variables), we may now define the system $\mathsf{LJ}[\vec{E_S}]$. This is the system which we will use for our main theorem in Section 12.7.

Definition 12.3.4 ($\mathsf{LJ}[\vec{E_S}]$). *We define $\mathsf{LJ}[\vec{E_S}]$ to be LJ augmented with Statman's extension axioms for a formula $F_{QBF} = Q_n x_n, ..., Q_1 x_1 B_0$, where B_0 is a quantifier-free prenex formula, and each $Q_i = \forall$ or $\exists$. More precisely, $\vec{E_S} = E_0, ..., E_n$, where*

- $E_0 = \{\neg\neg B_0 \mapsto y_0,\ y_0 \mapsto \neg\neg B_0\}$
- $E_k = \{(x_k \vee \neg x_k) \supset y_{k-1} \mapsto y_k,\ y_k \mapsto (x_k \vee \neg x_k) \supset y_{k-1}\}$ *if* $Q_k = \forall$, *and*
- $E_k = \{(x_k \supset y_{k-1}) \vee (\neg x_k \supset y_{k-1}) \mapsto y_k,\ y_k \mapsto (x_k \supset y_{k-1}) \vee (\neg x_k \supset y_{k-1})\}$ *if* $Q_k = \exists$

12.3.5 Manipulating the Result of Statman's Translation

For the purpose of our main theorem, instead of working with F_{IPL}, the result of Statman's translation, we will need to access its innermost extension variable. We therefore need to show that this extension variable can be efficiently extracted.

Lemma 12.3.5. *Let P be a size-N $\mathsf{LJ}[\vec{E_S}]$-proof of $\mapsto F_{IPL}$ i.e. of $\mapsto B_0^{\surd} \supset (B_1^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...))$, where N is the number of bits required to encode P. We can produce a size-$O(N^3)$ $\mathsf{LJ}[\vec{E_S}]$ proof P' of $\mapsto y_n$ which contains every possible extension axiom as a sequent.*

Proof: First note that although $\mathsf{LJ}[\vec{E_S}]$ does not have the rule of modus ponens, we can simulate it using cut with nothing more than a polynomial increase in proof-size: Suppose we have proofs of $\Gamma \mapsto A$, $\Gamma \mapsto A \supset B$, $A \mapsto A$, and $B \mapsto B$. From these we can produce a proof of $\Gamma \mapsto B$ as follows:

$$
\begin{prooftree}
\AxiomC{P_1}
\noLine
\UnaryInfC{$\Gamma \mapsto A$}
\AxiomC{P_2}
\noLine
\UnaryInfC{$\Gamma \mapsto A \supset B$}
\AxiomC{P_3}
\noLine
\UnaryInfC{$A \mapsto A$}
\AxiomC{P_4}
\noLine
\UnaryInfC{$B \mapsto B$}
\RightLabel{$\supset$-$Left$}
\BinaryInfC{$A, A \supset B \mapsto B$}
\RightLabel{Cut}
\BinaryInfC{$\Gamma, A \mapsto B$}
\RightLabel{Cut}
\BinaryInfC{$\Gamma \cup \Gamma \mapsto B$}
\noLine
\UnaryInfC{$\vdots$}
\RightLabel{$Weaken$}
\UnaryInfC{$\Gamma \mapsto B$}
\end{prooftree}
$$

Therefore, our size-N $\mathsf{LJ}[\vec{E_S}]$ proof P of $\mapsto B_0^{\surd} \supset (B_1^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...))$ corresponds to the proof P_2 of $\Gamma \mapsto A \supset B$, so to simulate modus ponens, we need only show that we can create proofs of the analogs of $\Gamma \mapsto A$, $A \mapsto A$, and $B \mapsto B$ that are short relative to N, the length of P_2.

In our case, $\Gamma \mapsto A$ is of the form $\mapsto B_i^{\surd}$ for some $B_i^{\surd}$. Each $B_i^{\surd}$ is associated with an E_k, which contains two extension axioms, one of the form $Formula \mapsto Variable$ and the other of the form $Variable \mapsto Formula$. In other words, $B_i^{\surd} = (F \supset v) \wedge (v \supset F)$, and can be proved as follows:

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$F \mapsto v$}
\RightLabel{$\supset$-$Right$}
\UnaryInfC{$\mapsto F \supset v$}
\AxiomC{}
\UnaryInfC{$v \mapsto F$}
\RightLabel{$\supset$-$Right$}
\UnaryInfC{$\mapsto v \supset F$}
\RightLabel{$\wedge$-$Right$}
\BinaryInfC{$\mapsto (F \supset v) \wedge (v \supset F)$}
\end{prooftree}
$$

Therefore we can construct P_1 in our modus ponens simulation using a constant-sized proof. Note that this proof includes both extension axioms associated with $B_i^{\surd}$.

In our case, $A \mapsto A$ is of the form $B_i^{\surd} \mapsto B_i^{\surd}$ for some $B_i^{\surd}$. This can be proved by first proving $\mapsto B_i^{\surd}$, and then using Weakening-L to prove $B_i^{\surd} \mapsto B_i^{\surd}$. Therefore we can construct P_3 in our modus ponens simulation using a constant-sized proof.

The case of $B \mapsto B$ is a little more complicated, and is of the form $B_i^{\surd} \supset (B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...)) \mapsto B_i^{\surd} \supset (B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...))$. As shorthand, we will also refer to this sequent as $F_{IPL} \mapsto F_{IPL}$. The proof P_4 of this sequent is constructed as follows:

$$
\begin{prooftree}
\AxiomC{$\mapsto B_i^{\surd}$}
\AxiomC{$\vdots$}
\noLine
\UnaryInfC{$B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...) \mapsto B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...)$}
\RightLabel{$Weaken$}
\UnaryInfC{$B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...), B_i^{\surd} \mapsto B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...)$}
\RightLabel{$\supset$-$Right$}
\UnaryInfC{$B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...) \mapsto B_i^{\surd} \supset (B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...))$}
\RightLabel{$\supset$-$Left$}
\BinaryInfC{$B_i^{\surd} \supset (B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...)) \mapsto B_i^{\surd} \supset (B_{i+1}^{\surd} \supset (...(B_{n-1}^{\surd} \supset (B_n^{\surd} \supset y_n))...))$}
\end{prooftree}
$$

Repeat this process at most n times, each time stripping off one $B_i^{\surd}$ from each side. This will yield the first line of P_4 to be $y_n \mapsto y_n$, an axiom. We already showed that $\mapsto B_i^{\surd}$ has a constant-sized proof, so the total length of our proof of P_4 will be linear in n, but since F_{IPL} occurs in P, n is $O(N)$, so each P_4 adds at most $O(N^2)$ to the size of our modus ponens simulation.

Since P_1 and P_3 have constant size, P_2 has size N, and P_4 has size-$O(N^2)$, every application of modus ponens adds one P_4 and therefore $O(N^2)$ to the size of our proof. In order to get at y_n, we must apply modus ponens n times, but as we already said, n is $O(N)$, so our entire proof of $\mapsto y_n$ requires size-$O(N^3)$.

Therefore, if given a size-N proof of $\mapsto B_0^{\checkmark} \supset (B_1^{\checkmark} \supset (...(B_{n-1}^{\checkmark} \supset (B_n^{\checkmark} \supset y_n))...))$, we can repeatedly apply modus ponens with the $\mapsto B_i^{\checkmark}$ sequents to produce a size-$O(N^3)$ of $\mapsto y_n$. In addition, since each subproof of $\mapsto B_i^{\checkmark}$ contains both extension axioms, and since all $\mapsto B_i^{\checkmark}$s are present, our overall proof of $\mapsto y_n$ contains each possible extension axiom, as required. □

12.4 Cut-Elimination

In this section we extend the cut-elimination technique developed by Buss and Pudlák in [BP01] (which itself was adapted from [BM99]) so that it holds for any system $\mathsf{LJ}[\vec{E}]$.

12.4.1 Definitions

We shall make use of the following definitions:

Definition 12.4.1 (Proof Closure). *The closure of a proof P, denoted $cl(P)$, is the smallest set of sequents which includes the sequents of P and is closed under both weakening and cut.*

Definition 12.4.2 (Direct Ancestor). *In a proof P, the direct ancestors of a formula A are all instances of A comprising an unbroken path towards the leaves of P from A to the first instance where A was introduced.*

Definition 12.4.3 (Principal Cut). *A principal cut is a cut in which at least one of its two input sequents is an extension axiom.*

12.4.2 Cut-Elimination Theorem

Theorem 12.4.4 (Cut-Elimination). *For any proof P in any system $\mathsf{LJ}[\vec{E}]$, it is possible to eliminate all non-principal cuts from P to produce a pseudo-cut-free proof P' such that $cl(P') \subseteq cl(P)$.*

Proof: We will first show the general technique for eliminating a cut on an arbitrary binary logical connective $\circ$, and will then provide the details specific to the connectives $\supset, \vee, \wedge$. Finally, we will show how to remove cuts on atoms. While reading this proof, please refer to Figure 12.1 below.

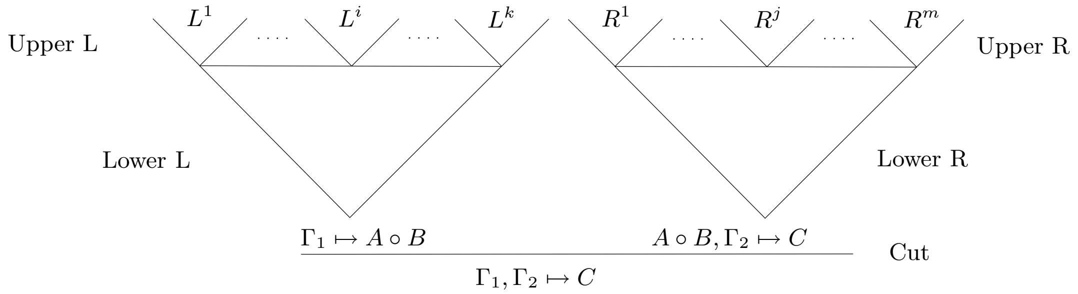

Figure 12.1: The Template For Cut-Elimination

We wish to eliminate a cut on the formula $A \circ B$. Let L be the proof of the left-hand input to the cut, and let R be the proof of the right-hand input. Let 'lower L' be the portion of L which contains sequents with direct ancestors of $A \circ B$ appearing in the succedent. In other words, each sequent in lower L is of the form $\Theta \mapsto A \circ B$. At the upper border of lower L, there are k subproofs labelled L^i, each of which introduces $A \circ B$ for the first time along its branch of the proof. More specifically, each L^i proves the sequent $\Pi_i \mapsto A \circ B$.

Similarly, 'lower R' is the portion of R in which the sequents contain the direct ancestors of $A \circ B$ in the antecedent. In other words, each sequent in lower R is of the form $A \circ B, \Theta \mapsto X$. Along the upper border of lower R, there are m subproofs labelled R^j, each of which introduces $A \circ B$ for the first time along its branch of the proof. More specifically, each R^j proves the sequent $A \circ B, \Delta_j \mapsto D_j$.

In order to eliminate the cut on $A \circ B$, we perform the following steps:

1. For $i = 1$ to k create a proof R^{i*} by taking a copy of R and doing the following: Modify each R^j along the entire border between upper and lower R so that instead of proving $A \circ B, \Delta_j \mapsto D_j$, each R^j now proves $\Pi_i, \Delta_j \mapsto D_j$. In the case where $A \circ B$ was introduced by weakening, simply introduce Π_i via weakening instead. In all other cases, take a copy of L^i (which is where Π_i comes from) and splice it into each R^j along the entire border in order to replace $A \circ B$ with Π_i (the details of this splicing depend on which connective is being eliminated, see below). To be clear, we do this k times; for each of the k separate R^{i*}s that we are building, splice each of the k L^is in once with each of the m R^js appearing in upper R.

2. Next, to complete the construction of each R^{i*}, replace each sequent $A \circ B, \Theta \mapsto X$ in lower R with the sequent $\Pi_i, \Theta \mapsto X$. Each R^{i*} therefore proves $\Pi_i, \Gamma_2 \mapsto C$ instead of $A \circ B, \Gamma_2 \mapsto C$.

3. Modify L: First, replace each L^i in upper L with R^{i*}. Now every sequent along the border between upper and lower L is no longer $\Pi_i \mapsto A \circ B$, it is $\Pi_i, \Gamma_2 \mapsto C$. Next, replace each sequent $\Theta \mapsto A \circ B$ in lower L with $\Theta, \Gamma_2 \mapsto C$. Therefore, instead of proving $\Gamma_1 \mapsto A \circ B$, L now proves $\Gamma_1, \Gamma_2 \mapsto C$, which was precisely the result of our cut, so it has been eliminated.

It is not hard to see that this process of eliminating cuts does not add any new sequents to the closure of the proof; our cut-elimination produced new sequents in only three ways: Firstly, we created new sequents while constructing the $R^{i*}s$ when we replaced each $A \circ B, \Theta \mapsto X$ with the new sequent $\Pi_i, \Theta \mapsto X$. However, each L^i proves the sequent $\Pi_i \mapsto A \circ B$, and when we cut this with $A \circ B, \Theta \mapsto X$, we get $\Pi_i, \Theta \mapsto X$, showing that it was in the original closure of P.

Secondly, we created new sequents while modifying lower L when we replaced each sequent $\Theta \mapsto A \circ B$ in lower L with the new sequent $\Theta, \Gamma_2 \mapsto C$. However, when we cut $\Theta \mapsto A \circ B$ with $A \circ B, \Gamma_2 \mapsto C$ (which was the original result of R), we get $\Theta, \Gamma_2 \mapsto C$, showing that it too was in the original closure of P.

Finally, we created new sequents during the splicing of the L^is into the R^js. Below, alongside the details of how to actually perform the splicing, we will show that each of these new sequents is in the closure of the original proof.

These were the only types of sequents which we introduced during our cut-elimination of binary connectives; all others came from the original proof P. Therefore, the cut elimination technique does not introduce any new sequents to the closure.

All that remains to be shown is how to splice L^i into each R^j to turn $A \circ B, \Gamma_2 \mapsto C$ into $\Pi_i, \Delta_j \mapsto D_j$ for each binary connective, and that the closure condition is satisfied. Each L^i proves the sequent $\Pi_i \mapsto A \circ B$, which is itself an extension axiom, or was introduced by weakening or $\circ$-*Right*. Similarly, each R^j proves the sequent $A \circ B, \Delta_j \mapsto D_j$, which is itself an extension axiom, or was introduced by weakening or $\circ$-*Left*. We must show how to combine each of these potential L^is with each of the R^js to get $\Pi_i, \Delta_j \mapsto D_j$:

Usually replacing $A \circ B$ by Π_i requires splicing, but in the case where R^j introduced $A \circ B$ by weakening, no splicing is required; instead, we weaken to get Π_i rather than $A \circ B$. Each of the remaining combinations is spliced as follows:

Case 1

If both $\Pi_i \mapsto A \circ B$ and $A \circ B, \Delta_j \mapsto D_j$ are extension axioms of the form $y_{A \circ B} \mapsto A \circ B$ and $A \circ B \mapsto y_{A \circ B}$, simply get $y_{A \circ B} \mapsto y_{A \circ B}$ by using a single principal cut.

Case 2

If $\Pi_i \mapsto A \circ B$ is an extension axiom of the form $y_{A\circ B} \mapsto A \circ B$ and $A \circ B, \Delta_j \mapsto D_j$ was introduced by $\circ$-$Left$, again get $y_{A\circ B}, \Delta_j \mapsto D_j$ by using a single principal cut.

Case 3

If $\Pi_i \mapsto A \circ B$ was introduced by $\circ$-*Right* and $A \circ B, \Delta_j \mapsto D_j$ is an extension axiom of the form $A \circ B \mapsto y_{A\circ B}$, once again get $\Pi_i \mapsto y_{A\circ B}$ by using a single principal cut.

Case 4

However, if both $\Pi_i \mapsto A \circ B$ and $A \circ B, \Delta_j \mapsto D_j$ came from introduction rules, then splicing to get $\Pi_i, \Delta_j^\alpha, \Delta_j^\beta \mapsto D_j$ is somewhat more complicated and depends on what connective $\circ$ represents.

Case 4a If $A \circ B$ is the formula $A \supset B$, then L^i is of the form

$$\dfrac{\overset{L^i}{\vdots}\quad \Pi_i, A \mapsto B}{\Pi_i \mapsto A \supset B}\ \supset\text{-}Right$$

and R^j is of the form

$$\dfrac{\overset{R^{j\alpha}}{\vdots}\quad \Delta_j^\alpha \mapsto A \qquad \overset{R^{j\beta}}{\vdots}\quad B, \Delta_j^\beta \mapsto D_j}{A \supset B, \Delta_j^\alpha, \Delta_j^\beta \mapsto D_j}\ \supset\text{-}Left$$

we may therefore splice them together as follows:

$$\dfrac{\dfrac{\overset{R^{j\alpha}}{\vdots}\quad \Delta_j^\alpha \mapsto A \qquad \overset{L^i}{\vdots}\quad \Pi_i, A \mapsto B}{\Pi_i, \Delta_j^\alpha, \mapsto B}\ Cut \qquad \overset{R^{j\beta}}{\vdots}\quad B, \Delta_j^\beta \mapsto D_j}{\Pi_i, \Delta_j^\alpha, \Delta_j^\beta \mapsto D_j}\ Cut$$

to produce $\Pi_i, \Delta_j \mapsto D_j$, as required. Note that $\Pi_i, \Delta_j^\alpha, \mapsto B$ is a new sequent that potentially did not occur in our original proof, but it was produced via cut from sequents from the original proof, so no new sequents are added to the closure.

Case 4b If $A \circ B$ is the formula $A \vee B$, then L^i is of the form

$$\dfrac{\overset{L^i}{\vdots}\quad \Pi_i, \mapsto A}{\Pi_i \mapsto A \vee B}\ \vee\text{-}Right$$

and R^j is of the form

$$\dfrac{\begin{array}{c}R^{j\alpha}\\ \vdots\\ A, \Delta_j^\alpha \mapsto D_j\end{array} \qquad \begin{array}{c}R^{j\beta}\\ \vdots\\ B, \Delta_j^\beta \mapsto D_j\end{array}}{A \vee B, \Delta_j^\alpha, \Delta_j^\beta \mapsto D_j}\ \vee\textit{-Left}$$

we may therefore splice them together as follows:

$$\dfrac{\dfrac{\begin{array}{c}L^{i}\\ \vdots\\ \Pi_i, \mapsto A\end{array} \qquad \begin{array}{c}R^{j\alpha}\\ \vdots\\ A, \Delta_j^\alpha \mapsto D_j\end{array}}{\Pi_i, \Delta_j^\alpha, \mapsto D_j}\ Cut}{\Pi_i, \Delta_j^\alpha, \Delta_j^\beta \mapsto D_j}\ Weaken$$

to produce $\Pi_i, \Delta_j \mapsto D_j$, as required (the case involving $\Pi_i, \mapsto B$ is entirely similar; just use $R^{j\beta}$). Note that $\Pi_i, \Delta_j^\alpha, \mapsto D_j$ is a new sequent that potentially did not occur in our original proof, but it was produced via cut from sequents from the original proof, so no new sequents are added to the closure.

Case 4c If $A \circ B$ is the formula $A \wedge B$, then L^i is of the form

$$\dfrac{\begin{array}{c}L^{i\alpha}\\ \vdots\\ \Pi_i^\alpha, \mapsto A\end{array} \qquad \begin{array}{c}L^{i\beta}\\ \vdots\\ \Pi_i^\beta, \mapsto B\end{array}}{\Pi_i^\alpha, \Pi_i^\beta \mapsto A \wedge B}\ \wedge\textit{-Right}$$

and R^j is of the form

$$\dfrac{\begin{array}{c}R^{j}\\ \vdots\\ A, \Delta_j \mapsto D_j\end{array}}{A \wedge B, \Delta_j \mapsto D_j}\ \wedge\textit{-Left}$$

we may therefore splice them together as follows:

$$\dfrac{\dfrac{\begin{array}{c}L^{i\alpha}\\ \vdots\\ \Pi_i^\alpha, \mapsto A\end{array} \qquad \begin{array}{c}R^{j}\\ \vdots\\ A, \Delta_j \mapsto D_j\end{array}}{\Pi_i^\alpha, \Delta_j, \mapsto D_j}\ Cut}{\Pi_i^\alpha, \Pi_i^\beta, \Delta_j \mapsto D_j}\ Weaken$$

to produce $\Pi_i, \Delta_j \mapsto D_j$, as required (the case involving $B, \Delta_j \mapsto D_j$ is entirely similar; just use $L^{i\beta}$). Note that $\Pi_i^\alpha, \Delta_j, \mapsto D_j$ is a new sequent that potentially did not occur in our original proof, but it was produced via cut from sequents from the original proof, so no new sequents are added to the closure.

Cases $1-3$ above introduce only principal cuts, which do not have to be eliminated. Note that each of the constructions in Case 4 is working to eliminate a cut on a binary connective, but in the process introduces at least one new cut. However, this is all right, since the only cuts being introduced are on simpler formulas, so this process when iterated will inevitably result in a proof in which the only cuts remaining are on atoms. As our last case, we must therefore show how cuts on atoms may be removed without introducing any new cuts at all:

The Atomic Case

The atomic case is fairly simple, and does not involve notions such as upper and lower L or R. Suppose we wish to eliminate a cut on an atomic formula x such as in

$$\cfrac{\begin{matrix} \overset{L}{\vdots} \\ \Gamma_1 \mapsto x \end{matrix} \qquad \begin{matrix} \overset{R}{\vdots} \\ x, \Gamma_2 \mapsto C \end{matrix}}{\Gamma_1, \Gamma_2, \mapsto C} \; Cut$$

Note that x could be a normal variable, or an extension variable $y_{A \circ B}$. All axioms in L are either of the form $x \mapsto x$ or $A \circ B \mapsto y_{A \circ B}$, and every sequent in L is of the form $\Theta \mapsto x$. The atomic cut-elimination proceeds as follows: First, take L, and replace each sequent $\Theta \mapsto x$ by $\Theta, \Gamma_2 \mapsto C$. The result is a proof of $\Gamma_1, \Gamma_2 \mapsto C$ in which all of the leaves are either $x, \Gamma_2 \mapsto C$ or $A \circ B, \Gamma_2 \mapsto C$.

For each leaf of the form $x, \Gamma_2 \mapsto C$, simply place a copy of R, which proves $x, \Gamma_2 \mapsto C$, on top of it. It is not hard to see that the closure of the proof is not affected; we introduced the sequents $\Theta, \Gamma_2 \mapsto C$ by modifying $\Theta \mapsto x$, but when we cut $\Theta \mapsto x$ with $x, \Gamma_2 \mapsto C$ (which was the original result of R), we get $\Theta, \Gamma_2 \mapsto C$, showing that it too was in the original closure of P.

For each leaf of the form $A \circ B, \Gamma_2 \mapsto C$, rather than adding R on top, add the following subproof on top instead:

$$\cfrac{\overline{A \circ B, \mapsto y_{A \circ B}} \qquad \begin{matrix} \overset{R}{\vdots} \\ y_{A \circ B}, \Gamma_2 \mapsto C \end{matrix}}{A \circ B, \Gamma_2 \mapsto C} \; P.Cut$$

The cut is a principal cut, so it will not have to be eliminated. In addition, the axiom $A \circ B, \mapsto y_{A \circ B}$ which we introduced is not new; it was the original sequent which we modified in order to require this subproof in the first place, and therefore is not a new addition to the closure.

In effect, we can eliminate cuts on atoms such that no new non-principal cuts are introduced, thereby concluding the proof that cuts can be eliminated even in the presence of extension axioms. □

12.5 The Proof Closure Property

In this section we formally prove the Proof Closure Property which is stated but never proved in [BM99] and [BP01]. Informally, the Proof Closure Property guarantees that any sequent in the closure of an $\mathsf{LJ}[\vec{E_S}]$ proof P can be derived via a polynomial number of cuts and weakenings from the sequents in P. This property is strongly related to Resolution on Horn clauses. For more information on these topics, please refer to Section 5.2 or [Sch89]. In particular Horn clauses and formulas are described in Defintion 5.2.2.

Lemma 12.5.1. *Every unsatisfiable Horn formula F has a size-$O(n)$ regular* I-RES *refutation, where n is the number of distinct variables in F.*

Proof: This lemma follows directly from Theorem 5.2.5 and Lemma 5.2.8. □

Corollary 12.5.2. *Given any set of Horn clauses Σ and any Horn clause H such that $\Sigma \vdash_{RES} H$, there exists a size-$O(N)$ regular* I-RES *derivation of $D \subseteq H$ from Σ, where N is the number of bits required to encode Σ.*

Proof: Let ϕ be any minimal truth assignment which sets H to false. Since $\Sigma \models_{RES} H$, we know that $\Sigma \lceil_\phi$ is unsatisfiable. Therefore, by Lemma 12.5.1, $\Sigma \lceil_\phi$ has a size-$O(n)$ regular I-RES refutation, call it R, where n is the number of distinct sentence letters in $\Sigma \lceil_\phi$. All of the literals in H were eliminated, so this

RES refutation never resolves on H's literals. It is therefore easy to see that if we create R' by replacing every clause in R that came from $\Sigma\lceil_\phi$ with its corresponding clause in Σ, the literals from H are present and will simply be carried down the proof so that instead of proving $\varnothing$ like R did, R' proves $D \subseteq H$, and its size is clearly bounded by $O(n)$. Since n is bounded above by N, the size of R' is bounded by $O(N)$, as required. $\square$

Lemma 12.5.3 (Proof Closure Property). *Let P be a size-N $\mathsf{LJ}[\vec{E_S}]$ proof, where N is the number of bits required to encode P. If $\Gamma \mapsto A \in cl(P)$ then there exists a size-$O(N^2)$ tree-like and a size-$O(N)$ DAG-like $\mathsf{LJ}[\vec{E_S}]$-proof of $\Gamma \mapsto A$.*

Proof: It is easy to see that sequents are very similar to Horn clauses. A singular right-side sequent $\Gamma \mapsto A$ is interpreted as meaning that a conjunction of all the formulas in Γ implies A. A Horn clause $(\neg x_1 \vee \neg x_2 \vee ... \vee \neg x_i \vee x_j)$ is equivalent to $(x_1 \wedge x_2 \wedge ... \wedge x_i \supset x_j)$. This gives rise to the following translation between sequents and Horn clauses: If given a sequent $A_1, A_2, ..., A_i \mapsto A_j$ where the As are formulas or negated formulas, we convert this sequent to the Horn clause by replacing each A_i with a variable x_i to give $(\neg x_1 \vee \neg x_2 \vee ... \vee \neg x_i \vee x_j)$. It is easy to see that a sequent with no formula on the right-hand side corresponds to a Horn clause containing no positive literal. Furthermore, a cut on two sequents corresponds to the resolution of two Horn clauses.

Therefore, suppose that we have an $\mathsf{LJ}[\vec{E_S}]$ proof P of size N and we know that $\Gamma \mapsto A \in cl(P)$. As shown above, translate the sequents $S_1, S_2, ..., S_k$ in P to Horn clauses. These clauses comprise our initial set of clauses Σ upon which we will resolve. Let H be the Horn clause corresponding to the sequent $\Gamma \mapsto A$. Since $\Gamma \mapsto A \in cl(P)$, we know by Corollary 12.5.2 that there is a size-$O(N)$ linear I-RES derivation R of $D \subseteq H$ from Σ. Translate the clauses in R back into sequents. This corresponds to a size-$O(N)$ derivation of $\Gamma \mapsto A$ from the sequents $S_1, S_2, ..., S_k$ in P using cut. To complete the proof, simply weaken D to get H. If our $\mathsf{LJ}[\vec{E_S}]$ proof is required to be tree-like, then add a proof of each $S_1, S_2, ..., S_k$ immediately above it. Each of these proofs is a sub-proof of P, and therefore has size at most $O(N)$. Since there are at most $O(N)$ S_is, this gives an overall size-$O(N^2)$ $\mathsf{LJ}[\vec{E_S}]$-proof of $\Gamma \mapsto A$ for tree-like proofs. If our $\mathsf{LJ}[\vec{E_S}]$ proof can be DAG-like, then these proofs are not necessary, and we have an overall size-$O(N)$ $\mathsf{LJ}[\vec{E_S}]$-proof of $\Gamma \mapsto A$. $\square$

12.6 The Disjunction & Implication Properties

Normally the Disjunction Property in intuitionistic propositional logic states that if Γ contains no formula containing $\vee$, then $\Gamma \vdash_{IPL} A \vee B$ implies that $\Gamma \vdash_{IPL} A$ or $\Gamma \vdash_{IPL} B$. However, this form of the Disjunction Property fails when applied to certain extension axioms (for example, just take an extension axiom with $\Gamma = y$ on the left where y is an extension variable, and any formula containing $\vee$ as the major logical particle on the right). We must therefore prove a weaker form of the Disjunction Property that is still strong enough to help us prove our main result:

Lemma 12.6.1 (Modified Disjunction Property). *If P is a $\mathsf{LJ}[\vec{E_S}]$-proof of the sequent $l_1, ..., l_k \mapsto A \vee B$ such that the only cuts in P are principal cuts, $l_1, ..., l_k$ are literals which do not include any y extension variables, then there either exists a proof P' of $l_1, ..., l_k \mapsto A$ or of $l_1, ..., l_k \mapsto B$ in which all cuts are principal cuts such that $cl(P') \subseteq cl(P)$.*

Proof: Suppose that P is a $\mathsf{LJ}[\vec{E_S}]$-proof of the sequent $l_1, ..., l_k \mapsto A \vee B$ such that the only cuts in P are principal cuts and $l_1, ..., l_k$ are literals which do not include any y extension variables as above. Since P does not contain normal cuts, $l_1, ..., l_k \mapsto A \vee B$ could only have come from one of the following rules:

1. $\vee$-*Right*
2. *Weaken-Right*
3. *Weaken-Left*
4. *Contraction-Left*
5. *A Principal Cut*

In the first case, the penultimate line in P was either $l_1, ..., l_k \mapsto A$ or $l_1, ..., l_k \mapsto B$, so we are done. In the second case, the penultimate line in P was $l_1, ..., l_k \mapsto$, which can be weakened to get what we want, and again we are done.

The next three cases are more complicated. In the weakening-left case, the only difference is that the left-hand side of the previous sequent contained one fewer literal. In the contraction case, the only difference is that the left-hand side of the previous sequent contained one more duplicated literal. In our final case, if $l_1, ..., l_k \mapsto A \vee B$ came from a principal cut, then this cut must have been on the sequent $l_1, ..., l_k \mapsto y$ and the extension axiom $y \mapsto A \vee B$. The sequent $l_1, ..., l_k \mapsto y$ could only have come from one of the following rules:

1. *Weaken-Right*
2. *Weaken-Left*
3. *Contraction-Left*
4. *A Principal Cut*

In the first case, the previous line was $l_1, ..., l_k \mapsto$, which can be weakened to $l_1, ..., l_k \mapsto A$, and we are done. The weakening and contraction cases are similar to the ones before; each just affects the number of literals on the left by one. Finally, if $l_1, ..., l_k \mapsto y$ came from a principal cut, then this cut must have been on the sequent $l_1, ..., l_k \mapsto A \vee B$ and the extension axiom $A \vee B \mapsto y$, which brings us back to what we started with.

Therefore, in order to avoid the cases in which we are done, let us assume that P contains no $\vee$-*Right* or *Weaken-Right* rules. Hence, P has $l_1, ..., l_k \mapsto A \vee B$ as its last line, preceded by 0 or more weakenings and contractions on the left, preceded by a principal cut on the sequent $l_1, ..., l_j \mapsto y$, which itself was preceded by 0 or more weakenings and contractions on the left, and a principal cut on $l_1, ..., l_i \mapsto A \vee B$, as shown here:

$$\begin{array}{c}\vdots \\ \vdots \\ \dfrac{l_1, ..., l_i \mapsto A \vee B \qquad \dfrac{}{A \vee B \mapsto y}}{l_1, ..., l_i \mapsto y}\ P.Cut \\ \vdots\ \text{0 or more Weaken-L or Contraction-L} \\ \vdots \\ \dfrac{l_1, ..., l_j \mapsto y \qquad \dfrac{}{y \mapsto A \vee B}}{l_1, ..., l_j \mapsto A \vee B}\ P.Cut \\ \vdots\ \text{0 or more Weaken-L or Contraction-L} \\ \vdots \\ l_1, ..., l_k \mapsto A \vee B\end{array}$$

However, this pattern cannot go on indefinitely, since proofs are only finitely long. Note that every sequent P must therefore have either y or $A \vee B$ on the right-hand side. P cannot begin at a sequent $l \mapsto A \vee B$, since we said that none of the l literals are y extension variables, so $l \mapsto A \vee B$ cannot be an axiom. Similarly, the proof cannot begin at a sequent $l \mapsto y$. Alternatively, P cannot contain the sequents $\mapsto A \vee B$ or $\mapsto y$ as axioms, since these are not axioms. Therefore P cannot contain any axioms, a contradiction. In other words, P must contain an application of $\vee$-*Right* or *Weaken-Right*, so the line preceeding this application allows us to easily show that $l_1, ..., l_k \mapsto A \in cl(P)$ or $l_1, ..., l_k \mapsto B \in cl(P)$, as required. $\square$

Lastly, we need one final lemma that is very similar to the modified disjunction property:

Lemma 12.6.2 (Implication Property). *If P is a $\mathsf{LJ}[\vec{E_S}]$-proof of the sequent*

$$l_1, ..., l_k, (x_{k+1} \vee \neg x_{k+1}), ..., (x_j \vee \neg x_j) \mapsto A \supset B$$

such that the only cuts in P are principal cuts, and $l_1, ..., l_k$ are literals which do not include any y extension variables, then there exists a proof P' in which all cuts are principal cuts of $l_1, ..., l_k, (x_{k+1} \vee \neg x_{k+1}), ..., (x_j \vee \neg x_j), A \mapsto B$ such that $cl(P') \subseteq cl(P)$.

Proof: The proof is almost identical to that of Lemma 12.6.1 with $A \supset B$ replacing all instances of $A \vee B$, and the rule $\supset$-*Right* replacing all instances of $\vee$-*Right*. □

12.7 Main Result

We are now ready to prove our main result, which states that if Statman's reduction can translate trivial instances of the law of excluded middle into intuitionistic formulas with polynomially-bounded proofs, then $\mathsf{LK}[\vec{E_S}]$ is a super proof system and $\mathcal{NP} = co\mathcal{NP}$.

Theorem 12.7.1 (Main Theorem). *Let $F_{Prop} = A(x_1, ...x_n)$ be any arbitrary classical propositional tautology containing n distinct variables, and consider the formula $F'_{Prop} = A(x_1, ...x_n) \vee \neg A(x_1, ...x_n)$. Let F_{QBF} be the prenex QBF translation of F'_{Prop} where each quantifier is $\forall$, and let F_{IPL} be Statman's translation of F_{QBF}. If there exists a size-N DAG-like $\mathsf{LJ}[\vec{E_S}]$ proof of F_{IPL}, where N is the number of bits required to encode F_{IPL}, then F_{Prop} has a DAG-like classical $\mathsf{LK}[\vec{E_S}]$ proof of size-$O(N^4)$.*

Proof: Since $F'_{Prop} = A(x_1, ...x_n) \vee \neg A(x_1, ...x_n)$, its quantified form is

$$\forall x_n, ..., \forall x_1 A(x_1, ...x_n) \vee \neg \forall x_n, ..., \forall x_1 A(x_1, ...x_n).$$

Therefore, in order to turn F'_{Prop} into quantified prenex form, we have to rename the variables in $\neg A(x_1, ...x_n)$ to produce $F''_{Prop} = A(x_1, ...x_n) \vee \neg A(x_{n+1}, ...x_{2n})$. This means that

$$F_{QBF} = \exists x_{2n}, ..., \exists x_{n+1}, \forall x_n, ..., \forall x_1 B_0$$

where $B_0 = F''_{Prop} = A(x_1, ...x_n) \vee \neg A(x_{n+1}, ...x_{2n})$, and applying Statman's translation yields

$$F_{IPL} = B_0^{\checkmark} \supset (B_1^{\checkmark} \supset (...(B_{2n-1}^{\checkmark} \supset (B_{2n}^{\checkmark} \supset y_{2n}))...)).$$

The extension axioms associated with F_{IPL} that we will need are:

- $y_0 \mapsto \neg\neg B_0$
- $y_k \mapsto (x_k \vee \neg x_k) \supset y_{k-1}$ for $k \leq n$ (these are the $\forall$ axioms).
- $y_k \mapsto (x_k \supset y_{k-1}) \vee (\neg x_k \supset y_{k-1})$ for $n + 1 \leq k \leq 2n$ (these are the $\exists$ axioms).

Suppose that there exists a size-N $\mathsf{LJ}[\vec{E_S}]$ proof P of $\mapsto F_{IPL}$. We will now show how to build a DAG-like classical $\mathsf{LK}[\vec{E_S}]$ proof of F_{Prop}. By Lemma 12.3.5, there exists a size-$O(N^3)$ proof of $\mapsto y_{2n}$. Note that this proof contains every extension axiom. Cut this sequent with the extension axiom $y_{2n} \mapsto (x_{2n} \supset y_{2n-1}) \vee (\neg x_{2n} \supset y_{2n-1})$ to get $\mapsto (x_{2n} \supset y_{2n-1}) \vee (\neg x_{2n} \supset y_{2n-1})$, and call this entire proof P_1.

By Theorem 12.4.4, we can eliminate all non-principal cuts in P_1 to produce P_2 such that $cl(P_2) \subseteq cl(P_1)$. Note that P_2 may be exponentially large. Since all cuts in P_2 are principal cuts, Lemma 12.6.1 applies, so there exists a proof P_3 of $\mapsto x_{2n} \supset y_{2n-1}$ or of $\neg x_{2n} \supset y_{2n-1}$. Let l_{2n} be the literal such that P_3 proves $\mapsto l_{2n} \supset y_{2n-1}$, and note that $cl(P_3) \subseteq cl(P_2)$. Furthermore, note that all cuts in P_3 are principal cuts.

By Lemma 12.6.2, there exists a proof P_4 of $l_{2n} \mapsto y_{2n-1}$ in which all cuts are principal cuts such that $cl(P_4) \subseteq cl(P_3)$. Cut this sequent with the extension axiom $y_{2n-1} \mapsto (x_{2n-1} \supset y_{2n-2}) \vee (\neg x_{2n-1} \supset y_{2n-2})$ to get $l_{2n} \mapsto (x_{2n-1} \supset y_{2n-2}) \vee (\neg x_{2n-1} \supset y_{2n-2})$. Call this proof P_5. Since $cl(P_4) \subseteq cl(P_3) \subseteq cl(P_2) \subseteq cl(P_1)$, we know that $cl(P_4) \subseteq cl(P_1)$. Furthermore, since P_1 contains every extension axiom, $cl(P_5) \subseteq cl(P_1)$.

Now simply repeat this process. After another iteration, we will have a proof P_6 of $l_{2n}, l_{2n-1} \mapsto y_{2n-2}$ such that $cl(P_6) \subseteq cl(P_1)$. We can repeat this n times in total for all of the $\exists$ extension axioms to get a proof P_7 of $l_{2n}, l_{2n-1}, ..., l_{n+1} \mapsto y_n$ such that $cl(P_7) \subseteq cl(P_1)$, and all cuts in P_7 are still principal cuts.

We now continue cutting with all of the $\forall$ extension axioms: Cut $l_{2n}, l_{2n-1}, ..., l_{n+1} \mapsto y_n$ with $y_n \mapsto (x_n \vee \neg x_n) \supset y_{n-1}$ to get $l_{2n}, l_{2n-1}, ..., l_{n+1} \mapsto (x_n \vee \neg x_n) \supset y_{n-1}$, and call this proof P_8. Note that all cuts in P_8 are principal cuts, and $cl(P_8) \subseteq cl(P_1)$.

By Lemma 12.6.2, there exists a proof P_9 of the sequent $l_{2n}, l_{2n-1}, ..., l_{n+1}, (x_n \vee \neg x_n) \mapsto y_{n-1}$ such that $cl(P_9) \subseteq cl(P_1)$.

Repeat this process. After another iteration, we will have a proof P_{10} of $l_n, l_{2n-1}, ..., l_{n+1}, (x_n \vee \neg x_n), (x_{n-1} \vee \neg x_{n-1}) \mapsto y_{n-2}$, where $cl(P_{10}) \subseteq cl(P_1)$. We can repeat this n times in total for all of the $\forall$ extension axioms to get a proof P_{11} of $l_{2n}, l_{2n-1}, ..., l_{n+1}, (x_n \vee \neg x_n), ..., (x_1 \vee \neg x_1) \mapsto y_0$ such that $cl(P_{11}) \subseteq cl(P_1)$.

Finally, cut this sequent with $y_0 \mapsto \neg\neg B_0$, where $B_0 = A(x_1, ...x_n) \vee \neg A(x_{n+1}, ...x_{2n})$. This gives us a proof P_{12} of $l_{2n}, l_{2n-1}, ..., l_{n+1}, (x_n \vee \neg x_n), ..., (x_1 \vee \neg x_1) \mapsto \neg\neg(A(x_1, ...x_n) \vee \neg A(x_{n+1}, ...x_{2n}))$ such that $cl(P_{12}) \subseteq cl(P_1)$.

Since $cl(P_{12}) \subseteq cl(P_1)$, we know that $l_{2n}, l_{2n-1}, ..., l_{n+1}, (x_n \vee \neg x_n), ..., (x_1 \vee \neg x_1) \mapsto \neg\neg(A(x_1, ...x_n) \vee \neg A(x_{n+1}, ...x_{2n})) \in cl(P_1)$. Therefore by Lemma 12.5.3, since P_1 is a proof of size-$O(N^3)$, there exists a size-$O(N^4)$ DAG-like proof P_{13} of the sequent $l_{2n}, l_{2n-1}, ..., l_{n+1}, (x_n \vee \neg x_n), ..., (x_1 \vee \neg x_1) \mapsto \neg\neg(A(x_1, ...x_n) \vee \neg A(x_{n+1}, ...x_{2n}))$.

Recall that every LJ proof is an LK proof. Therefore P_{13} is a size-$O(N^4)$ DAG-like classical $\mathsf{LK}[\vec{E_S}]$ proof. Restrict P_{13} by setting all of the literals $l_{2n}, l_{2n-1}, ..., l_{n+1}$ to true, and make all obvious simplifications. Since $A(x_1, ...x_n)$ is a tautology, $\neg A(x_{n+1}, ...x_{2n})$ is unsatisfiable. This leaves us with a size-$O(N^4)$ proof P_{14} of the sequent $(x_n \vee \neg x_n), ..., (x_1 \vee \neg x_1) \mapsto \neg\neg A(x_1, ...x_n)$. Since $\mapsto x_i \vee \neg x_i$ is essentially an LK axiom for each i, we can simply introduce n such axioms, one for each i, and repeatedly cut with $(x_n \vee \neg x_n), ..., (x_1 \vee \neg x_1) \mapsto \neg\neg A(x_1, ...x_n)$ to get an $O(N^4)$ proof P_{15} of the sequent $\mapsto \neg\neg A(x_1, ...x_n)$.

Finally, remove the double negation as follows:

$$\dfrac{\dfrac{\vdots\, P_{15}}{\mapsto \neg\neg A(x_1, ...x_n)} \qquad \dfrac{\dfrac{\dfrac{\vdots\, P_x}{A(x_1, ...x_n) \mapsto A(x_1, ...x_n)}}{\mapsto A(x_1, ...x_n), \neg A(x_1, ...x_n)}\ \neg\text{-}Right}{\neg\neg A(x_1, ...x_n) \mapsto A(x_1, ...x_n)}\ \neg\text{-}Left}{\mapsto A(x_1, ...x_n)}\ Cut$$

It is easy to show that any sequent of the form $A(x_1, ...x_n) \mapsto A(x_1, ...x_n)$ has a size-$O(n)$ DAG-like LK proof P_x. Since n is $O(N)$, $\mapsto A(x_1, ...x_n)$ has a classical size-$O(N^4)$ DAG-like $\mathsf{LK}[\vec{E_S}]$ proof, where $A(x_1, ...x_n) = F_{Prop}$ is any arbitrary classical tautology on n variables, as required. $\square$

12.8 Related Complexity Results

The primary implication of our main result is that Statman's reduction is 'dangerous' in the sense that unless $\mathsf{LK}[\vec{E_S}]$ is super, it translates some trivial QBF formulas to very difficult IPL instances. In order to formalize this notion, we take the definitions of what constitute dangerous and safe proof complexity reductions from Section 11.3 of Chapter 11.

Applying Definition 11.3.1 to this Chapter's main result yields the following corollary:

Corollary 12.8.1. *Unless $\mathsf{LK}[\vec{E_S}]$ is super, Statman's reduction is $(\alpha,\mathsf{LJ}[\vec{E_S}])$-explosive for every proof system α for QBF which has polynomially-bounded proofs for every prenex instance of the law of excluded middle.*

Whereas explosive reductions are considered dangerous, stable reductions, described formally in Definition 11.3.4 are safe. We can apply this concept to get another interesting corollary to this chapter's main result:

Corollary 12.8.2. *If Statman's reduction is $(\alpha,\mathsf{LJ}[\vec{E_S}])$-stable for some QBF proof system α which has polynomially-bounded proofs for every prenex instance of the law of excluded middle, then $\mathsf{LK}[\vec{E_S}]$ is a super proof system.*

12.9 Open Problems Related to Intuitionistic Proof Complexity

The main open problem related to this line of research has to do with the reduction in the reverse direction: Does there exist an efficient translation from IPL to QBF which does not map easy instances of IPL to hard instances of QBF? Finding such a translation appears to be difficult, but it is reasonable to conjecture that the answer is yes.

Bibliography

[ABMP01] M. Alekhnovich, S. Buss, S. Moran, and T. Pitassi. Minimum Propositional Proof Length is NP-Hard to Linearly Approximate. *Journal of Symbolic Logic*, 66:171 – 191, 2001.

[ABSRW01] M. Alekhnovich, E. Ben-Sasson, A.A. Razborov, and A. Wigderson. Space Complexity in Propositional Calculus. *SIAM Journal of Computing*, Vol. 31, No. 4:1184 – 1211, 2001.

[AD03] A. Atserias and V. Dalmau. A Combinatorial Characterization of Resolution Width. *Proceedings of the* 18^{th} *IEEE Conference on Computational Complexity*, pages 239 – 247, 2003.

[AJPU07] M. Alekhnovich, J. Johannsen, T. Pitassi, and A. Urquhart. An Exponential Separation Between Regular and General Resolution. *Theory of Computing*, 3:81 – 102, 2007.

[Ajt83] M. Ajtai. Σ_1^1 Formulae on Finite Structures. *Annals of Pure and Applied Logic*, Vol. 2, No. 1:1 – 48, 1983.

[AL86] R. Aharoni and N. Linial. Minimal Non-Two-Colorable Hypergraphs and Minimal Unsatisfiable Formulas. *Journal of Combinatorial Theory, Series A*, 43:196 – 204, 1986.

[ANS80] T. Akiyama, T. Nishizeki, and N. Saito. NP-Completeness of the Hamiltonian Cycle Problem for Bipartite Graphs. *Journal of Information Processing*, Vol. 3 No. 2:73 – 76, 1980.

[APU01] N. H. Arai, T. Pitassi, and A. Urquhart. The Complexity of Analytic Tableaux. *Proceedings of the* 33^{rd} *ACM Symposium on the Theory of Computing (STOC)*, pages 356 – 363, 2001.

[AR02] M. Alekhnovich and A. Razborov. Resolution is Not Automatizable Unless $\mathcal{W}[\mathcal{P}]$ is Tractable. *Proceedings of the* 42^{nd} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[Ats04] Albert Atserias. On Sufficient Conditions for Unsatisfiability of Random Formulas. *Journal of the Association for Computing Machinery*, 51:281 – 311, 2004. Preliminary Version: 17^{th} IEEE Symposium on Logic in Computer Science (LICS), 2002, pages 275 - 284.

[BB03] O. Bailleux and Y. Boufkhad. Efficient CNF Encodings of Boolean Cardinality Constraints. *International Conference on the Principles and Practice of Constraint Programming*, pages 102 – 122, 2003.

[Bea94] P. Beame. A Switching Lemma Primer. Manuscript, 1994.

[BHS90] D. Bauer, S.L. Hakimi, and E. Schmeichel. Recognizing Tough Graphs is NP-Hard. *Discrete Appl. Math.*, Vol. 28:191 – 195, 1990.

[BKS03] P. Beame, H. Kautz, and Ashish Sabharwal. Understanding the Power of Clause Learning. *Proceedings of the* 18^{th} *International Joint Conference on Artificial Intelligence*, 2003.

[BL94] H. Kleine Büning and T. Lettman. *Aussagenlogik: Deduktion und Algorithmen*. B.G. Teubner, Stuttgart, 1994.

[BM76] J.A. Bondy and U.S.R Murty. *Graph Theory With Applications*. Macmillan, London, 1976.

[BM99] S. R. Buss and G. Mints. The Complexity of the Disjunction and Existential Properties in Intuitionistic Logic. *Annals of Pure and Applied Logic*, 99:93 – 104, 1999.

[Bol85] B. Bollobás. *Graph Theory: An Introductory Course*. Springer, New York, 1985.

[BOM07] J. Buresh-Oppenheim and D. Mitchell. Minimum 2CNF Resolution Refutations in Polynomial Time. *Proceedings of the* 10^{th} *International Conference on Theory and Applications of Satisfiability Testing (SAT 2007)*, pages 300 – 313, 2007.

[BOP03] J. Buresh-Oppenheim and T. Pitassi. The Complexity of Resolution Refinements. *Proceedings of the* 18^{th} *Annual IEEE Symposium on Logic in Computer Science*, 2003.

[BP96] P. Beame and T. Pitassi. Simplified and Improved Resolution Lower Bounds. *Proceedings of the* 37^{th} *Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 274 – 282, 1996.

[BP01] S. R. Buss and P. Pudlák. On The Computational Content of Intuitionistic Propositional Proofs. *Annals of Pure and Applied Logic*, 109:49 – 64, 2001.

[BPR97] M. Bonet, T. Pitassi, and R. Raz. Lower Bounds for Cutting Planes Proofs with Small Coefficients. *Journal of Symbolic Logic*, Vol. 62, No. 3, 1997.

[BPR00] M. Bonet, T. Pitassi, and R. Raz. On Interpolation and Automatization For Frege Systems. *SIAM Journal of Computing*, Vol. 29, No. 6, 2000.

[BS02] E. Ben-Sasson. Size Space Tradeoffs For Resolution. *Proceedings of the* 34^{th} *ACM Symposium on the Theory of Computing (STOC)*, pages 457 – 464, 2002.

[BSIW04] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near Optimal Separation of Tree-like and General Resolution. *Combinatorica*, Vol. 24, No. 4:585 – 604, 2004.

[BSW01] E. Ben-Sasson and A. Wigderson. Short Proofs are Narrow -Resolution Made simple. *Journal of the Association for Computing Machinery*, 48:149 – 169, 2001. Preliminary Version: Proceedings of the 31^{st} Annual ACM Symposium on Theory of Computing (STOC), 1999, pages 517 - 526.

[BT88] S. Buss and G. Turán. Resolution Proofs of Generalized Pigeonhole Principles. *Theoretical Computer Science*, 62:311 – 317, 1988.

[CEI96] M. Clegg, J. Edmons, and R. Impagliazzo. Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability. *Proceedings of the* 28^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 174 – 183, 1996.

[Cha70] C. L. Chang. The Unit Proof and the Input Proof in Theorem Proving. *Journal of the Association for Computing Machinery*, Vol. 17, No. 4:698 – 707, 1970.

[Chv85] V. Chvátal. Hamiltonian Cycles. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 403 – 429. John Wiley & Sons Ltd., New York, 1985.

[CK01] P. Clote and E. Kranakis. *Boolean Functions and Computation Models*. Springer-Verlag, Berlin, 2001.

[Coo71] S.A. Cook. The Complexity of Theorem-Proving Procedures. *Proceedings of the* 3^{rd} *Annual ACM Symposium on the Theory of Computation*, pages 151 – 158, 1971.

[Coo73] S. A. Cook. An Observation on Time-Storage Tradeoff. *Proceedings of the* 5^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 29 – 33, 1973.

[Coo75] S.A. Cook. An Exponential Example For Analytic Tableaux. Manuscript, 1975.

[Coo02] S.A. Cook. University of Toronto CS 2404 -Computability and Logic Course Notes, 2002.

[CPT77] R. Celoni, W.J. Paul, and R.E. Tarjan. Space Bounds for a Game on Graphs. *Math. Systems Theory*, 10:239 – 251, 1977.

[CR74] S.A. Cook and R. A. Reckhow. On the Lengths of Proofs in the Propositional Calculus. *Proceedings of the* 6^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, pages 135 – 148, 1974.

[CR79] S.A. Cook and R. A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *The Journal of Symbolic Logic*, Vol. 44, No. 1:36 – 50, 1979.

[CS76] S. Cook and R. Sethi. Storage Requirements for Deterministic Polynomial Time Recognizable Languages. *Journal of Computer & System Sciences*, pages 25 – 37, 1976.

[D'A92] M. D'Agostino. Are Tableaux an Improvement on Truth Tables? *Journal of Logic, Language, and Information*, 1:235 – 252, 1992.

[DDB98] G. Davydov, I. Davydova, and H. Kleine Büning. An Efficient Algorithm for the Minimal Unsatisfiability Problem for a Subclass of CNF. *Annals of Mathematics and Artificial Intelligence*, 23:229 – 245, 1998.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A Machine Program For Theorem Proving. *Communications of the ACM*, 5:394 – 397, 1962.

[DP60] M. Davis and H. Putnam. A Computing Procedure For Quantification Theory. *Communications of the ACM*, 7:201 – 215, 1960.

[Est95] J. Esteban. Complejidad de la Resolucion en Espacio y Tiempo. M.Sc. Thesis, Facultad de Informatica de Barcelona, 1995.

[Est03] J. Esteban. Complexity Measures For Resolution. Ph.D. Thesis, Universitat Politècnica de Catalunya, 2003.

[ET01] J. Esteban and J. Torán. Space Bounds for Resolution. *Information and Computation*, 171:84 – 97, 2001. Preliminary Version: Proceedings of the 16^{th} Annual Symposium on Theoretical Aspects of Computer Science (STACS), 1999, pages 551 - 561.

[ET03] J. Esteban and J. Torán. A Combinatorial Characterization of Treelike Resolution Space. *Information Processing Letters*, 87:295 – 300, 2003. Preliminary Version: Electronic Colloquium on Computational Complexity (ECCC) Report TR03-044, 2003.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Theory of Computing Systems*, Vol. 17, No. 1:13 – 27, 1984.

[FV98] Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study Through Datalog and Group Theory. *Siam Journal on Computing*, 28:57 – 104, 1998.

[Gal77] Z. Galil. On the Complexity of Regular Resolution and the Davis-Putnam Procedure. *Theoretical Computer Science*, 4:23 – 46, 1977.

[Gen35a] G. Gentzen. Untersuchungen Über Das Logische Schließen I. *Mathematische Zeitschrift*, 39:176 – 210, 1935.

[Gen35b] G. Gentzen. Untersuchungen Über Das Logische Schließen II. *Mathematische Zeitschrift*, 39:405 – 431, 1935.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability -A Guide to the Theory of NP-Completeness*. Freeman & Company, New York, 1979.

[GJT76] M.R. Garey, D.S. Johnson, and R.E. Tarjan. The Planar Hamiltonian Circuit Problem is NP-Complete. *SIAM Journal of Computing*, Vol. 5 No. 4:704 – 714, 1976.

[GLT80] J. R. Gilbert, T. Lengauer, and R. E. Tarjan. The Pebbling Problem is Complete in Polynomial Space. *SIAM Journal of Computing*, Vol. 9, No. 3:513 – 524, 1980.

[Hak85] A. Haken. The Intractability of Resolution. *Theoretical Computer Science*, 39:297 – 308, 1985.

[Hås87] J.T. Håstad. *Computational Limitations for Small-Depth Circuits*. ACM Doctoral Dissertation Award (1986), MIT Press, 1987.

[Hei81] F. Meyer Auf Der Heide. A Comparison of Two Variations of a Pebble Game on Graphs. *Theoretical Computer Science*, 13:315 – 322, 1981.

[Her04] A. Hertel. Hamiltonian Cycles In Sparse Graphs. M.Sc. Thesis, University of Toronto, 2004.

[HHU07] A. Hertel, P. Hertel, and A. Urquhart. Formalizing Dangerous Reductions. In *Proceedings of the* 10^{th} *International Conference on Theory and Applications of Satisfiability Testing (SAT 2007)*, pages 159 – 172. Springer, 2007. Lecture Notes in Computer Science 4501.

[HP07] P. Hertel and T. Pitassi. Exponential Time / Space Speedups for Resolution and the PSPACE-Completeness of Black-White Pebbling. *Proceedings of the* 48^{th} *Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, 2007.

[HPBG08] P. Hertel, T. Pitassi, F. Bacchus, and A. Van Gelder. Clause Learning Can Effectively P-Simulate General Propositional Resolution. *Proceedings of the* 23^{rd} *Annual Meeting of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2008.

[Hru07] P. Hrubeš. A Lower Bound for Intuitionistic Logic. *Annals of Pure and Applied Logic*, Vol. 146, No. 1:72 – 90, 2007.

[HU07] A. Hertel and A. Urquhart. Game Characterizations and the PSPACE-Completeness of Tree Resolution Space. In *Proceedings of the* 16^{th} *EACSL Annual Conference on Computer Science and Logic (CSL 2007)*, pages 527 – 541. Springer, 2007. Lecture Notes in Computer Science 4646.

[HU08a] A. Hertel and A. Urquhart. Algorithms & Complexity Results for Input & Unit Resoluution. *Submitted to the Journal on Satisfiability, Boolean Modeling and Computation*, 2008.

[HU08b] A. Hertel and A. Urquhart. The Resolution Width Problem is EXPTIME-Complete. *Submission Withdrawn from Theory of Computing*, 2008. Preliminary Version: Electronic Colloquium on Computational Complexity (ECCC) Report TR06-133, 2006.

[JL77] N.D. Jones and W.T. Laaser. Complete Problems For Deterministic Polynomial Time. *Theoretical Computer Science*, 3:105 – 117, 1977.

[KAI79] T. Kasai, A. Adachi, and S. Iwata. Classes of Pebble Games and Complete Problems. *SIAM Journal of Computing*, Vol. 8, No. 4:574 – 586, 1979.

[KMS96] H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. *Proceedings of the* 5^{th} *International Conference on Knowledge Representation and Reasoning*, 1996.

[KMS97] H. Kautz, D. McAllester, and B. Selman. Ten Challenges in Propositional Reasoning and Search. *Proceedings of the* 15^{th} *International Joint Conference on Artificial Intelligence*, 1997.

[KP03] Phokion G. Kolaitis and Jonathan Panttaja. On the Complexity of Existential Pebble Games. In *Proceedings of the* 12^{th} *EACSL Annual Conference on Computer Science and Logic (CSL 2003)*, pages 314 – 329. Springer, 2003. Lecture Notes in Computer Science 2803.

[Kri75] M.S. Krishnamoorthy. An NP-Hard Problem in Bipartite Graphs. *SIGACT News*, Vol. 7, No. 1:26, 1975.

[KS03] H. Kautz and B. Selman. Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search. *Proceedings of the* 9^{th} *International Conference on Principles and Practice of Constraint Programming*, 2003.

[KV95] Phokion G. Kolaitis and Moshe Y. Vardi. On the Expressive Power of Datalog: Tools and a Case Study. *Journal of Computer and System Sciences*, 51:110 – 134, 1995.

[Lin78] A. Lingas. A PSPACE-Complete Problem Related to a Pebble Game. In *Proceedings of the* 5^{th} *Colloquium on Automata, Languages and Programming*, pages 300 – 321, London, UK, 1978. Springer-Verlag.

[Mar07] B. Martin. The Computational Complexity of the Regular Resolution Width Problem. Available on the Internet: http://www.dcs.kcl.ac.uk/events/LAW07/LAW2007-slides/martin.pdf, 2007.

[Neu27] J. Von Neumann. Zur Hilbertschen Beweistheorie. *Mathematische Zeitschrift*, Vol. 2 No. 1:1 – 46, 1927.

[Nor06] J. Nordström. Narrow Proofs May Be Spacious: Separating Space and Width in Resolution. *Proceedings of the* 38^{th} *ACM Symposium on the Theory of Computing (STOC)*, pages 507 – 516, 2006. Preliminary Version: Electronic Colloquium on Computational Complexity (ECCC) Report TR05-066, 2005.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, New York, 1994.

[PI00] P. Pudlák and Russell Impagliazzo. Lower Bounds for DLL Algorithms for k-SAT. *Proceedings of SODA 2000*, 2000.

[Pip80] N. Pippenger. Pebbling. *Proceedings of the* 5^{th} *IBM Symposium on Mathematical Foundations of Computer Science, Japan (Technical Report RC8528, IBM Watson Research Center)*, 1980.

[Pit02a] Toniann Pitassi. University of Toronto CS 2429 -Propositional Proof Complexity Lecture 4 Scribe Notes. Scribe: Mehrdad Sabetzadeh, 2002.

[Pit02b] Toniann Pitassi. University of Toronto CS 2429 -Propositional Proof Complexity Lecture 9 Scribe Notes. Scribe: Sheikh M.N. Alam, 2002.

[Pla84] D.A. Plaisted. Complete Problems in the First-Order Predicate Calculus. *Journal of Computer and System Sciences*, Vol. 29, No. 1:8 – 35, 1984.

[PU95] T. Pitassi and A. Urquhart. The Complexity of the Hajós Calculus. *SIAM Journal of Discrete Mathematics*, Vol. 8, No. 3:464 – 483, 1995.

[Pud99] P. Pudlák. On The Complexity of Propositional Calculus, Sets and Proofs. In *Logic Colloquium '97*, pages 197 – 218. Cambridge University Press, 1999.

[Rec76] R. A. Reckhow. On the Lengths of Proofs in the Propositional Calculus. Ph.D. Thesis, University of Toronto, 1976.

[Sav70] W. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4:177 – 192, 1970.

[Sch89] U. Schöning. *Logic For Computer Scientists*. Birkhäuser, Berlin, 1989.

[Sta79] R. Statman. Intuitionistic Propositional Logic is Polynomial-Space Complete. *Theoretical Computer Science*, 9:67 – 72, 1979.

[Sto76] L. V. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3:1 – 22, 1976.

[Sub04] K. Subramani. Optimal Length Tree-Like Resolution Refutations for 2SAT Formulas. *ACM Transactions on Computational Logic*, Vol. 5, No. 2:316 – 320, 2004.

[Sza69] M.E. Szabo. *The Collected Papers of Gerhard Gentzen*. North-Holland Publishing Company, Amsterdam, 1969.

[TD88] A.S. Troelstra and D. Van Dalen. *Constructivism in Mathematics, An Introduction*. Elsevier Science Publishers B.V., Amsterdam, 1988.

[Tor99] J. Torán. Lower Bounds for Space in Resolution. In *Proceedings of the* 8^{th} *EACSL Annual Conference on Computer Science and Logic (CSL 1999)*, pages 362 – 373. Springer, 1999. Lecture Notes in Computer Science 1683.

[Tse70] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115 – 125. Consultants Bureau, New York, 1970.

[UF96] A. Urquhart and X. Fu. Simplified Lower Bounds for Propositional Proofs. *Notre Dame Journal of Formal Logic*, 37:523 – 545, 1996.

[Urq87] A. Urquhart. Hard Examples For Resolution. *Journal of the Association For Computing Machinery*, Vol. 34, No. 1:209 – 219, 1987.

[Urq92] A. Urquhart. The Relative Complexity of Resolution and Cut-Free Gentzen Systems. *Annals of Mathematics and Artificial Intelligence*, 6:157 – 168, 1992.

[Urq95] A. Urquhart. The Complexity of Propositional Proofs. *The Bulletin of Symbolic Logic*, Vol. 1, No. 4:425 – 467, 1995.

[Urq03] A. Urquhart. Resolution Proofs of Matching Principles. *Annals of Mathematics and Artificial Intelligence*, 37:241 – 250, 2003.

[Urq06] A. Urquhart. Width Versus Size in Resolution Proofs. *Proceedings of the* 3^{rd} *Annual Conference on Theory and Applications of Models of Computation*, pages 79 – 88, 2006.

[Wig82] A. Wigderson. The Complexity Of The Hamiltonian Circuit Problem For Maximal Planar Graphs. *Technical Report, Princeton University, Dept. of EECS*, Vol. 298, February, 1982.