Propositional Proof Complexity: A Depth Oral Survey

Alexander Hertel *

June 29th, 2005

1 Introduction

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 classes $\mathcal{NP}$ 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 published the first major proof complexity paper in 1966 [Tse70]. This work predated complexity theory by a few years, but Tseitin nevertheless included 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. We will follow their definitions closely.

1.1 Motivation

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 (ie. whether $\mathcal{NP}$ is closed under complement) is probably the second most important one. However, the motivation for studying propositional proof systems 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 $\mathcal{P} \neq \mathcal{NP}$. The proof is immediate: $\mathcal{P}$ is closed under complement, so if $\mathcal{NP}$ is not, then they cannot be the same sets. These would of course be major results.

In addition, the results in proof complexity can translate into lower bound results for algorithms. Creating better SAT solvers has turned into an industry, 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.

1.2 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 1.1. *A **proof system** for a language $L \subseteq \Sigma^*$ is a poly-time computable onto function $f : \Sigma_p^* \rightarrow L$ where Σ_p is some alphabet.*

Intuitively, Σ_p is an alphabet of 'proof symbols', and f maps proofs to the elements in L which 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 computable. Even more importantly, the words 'poly-time computable' enforce our intuitive notion that when given a proof, we should be able to

*This research supported by NSERC and the University of Toronto Department of Computer Science

tell (compute) what exactly it is that the proof is trying to prove within a feasible 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 in practice f is also total, since syntactically incorrect proofs are usually all mapped to a dummy element in L. In other words, no proof is mapped outside of 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 1.2. $\mathcal{NP} = \{\mathcal{L} \mid \mathcal{L}$ *is decidable by a NTM in polynomial time*$\}$

Definition 1.3. $\mathcal{NP} = \{\mathcal{L} \mid$ *Each* $x \in \mathcal{L}$ *has a poly-length certificate that can be verified by a DTM in polytime*$\}$

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 1.4. *co*$\mathcal{NP} = \{\overline{\mathcal{L}} \mid \mathcal{L} \in \mathcal{NP}\}$

1.3 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 1.5. *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.*

So 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 the truth assignment which satisfies it. All invalid proofs are mapped to an arbitrary formula in SAT, say the sentence letter p. In 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; ie. it contains only the unsatisfiable formulas. In any case, these 'garbage' strings are decidable by a DTM in polytime, so this is not a problem.

The two co$\mathcal{NP}$-Complete languages that we are most interested in are UNSAT and TAUT. 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, the language consisting of all logical tautologies. 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 1 below.

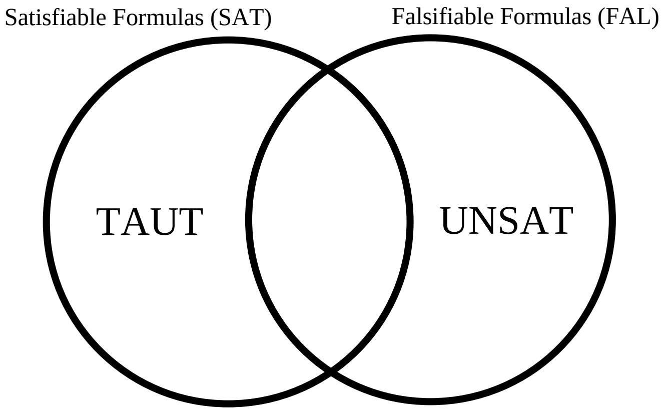

Figure 1: The relationship between SAT, FAL, UNSAT, and TAUT

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 1.6. *$\mathcal{NP} = co\mathcal{NP}$ 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 A Description of Major Proof Systems

Many propositional proof systems have been studied and categorized into a hierarchy based on their strengths as proof systems. This hierarchy is shown in Figure 2 and is based on the concept of p-simulation, which allows us to objectively compare two proof systems.

Definition 2.1. *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 that 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' g. 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. When the adjectives 'strong' and 'weak' are omitted, by default we mean strong p-simulation. Although the definition above requires the two proof systems to agree on the same language, proof systems for different languages can also p-simulate each other. We shall see examples of this when we discuss 'non-propositional' proof systems.

The following diagram gives a preview of the proof systems to be discussed as well as their relative strengths. 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 there is an exponential separation between the two systems. Question marks indicate open problems, and when multiple proof systems appear in the same box, it means that they belong to the same p-equivalence class.

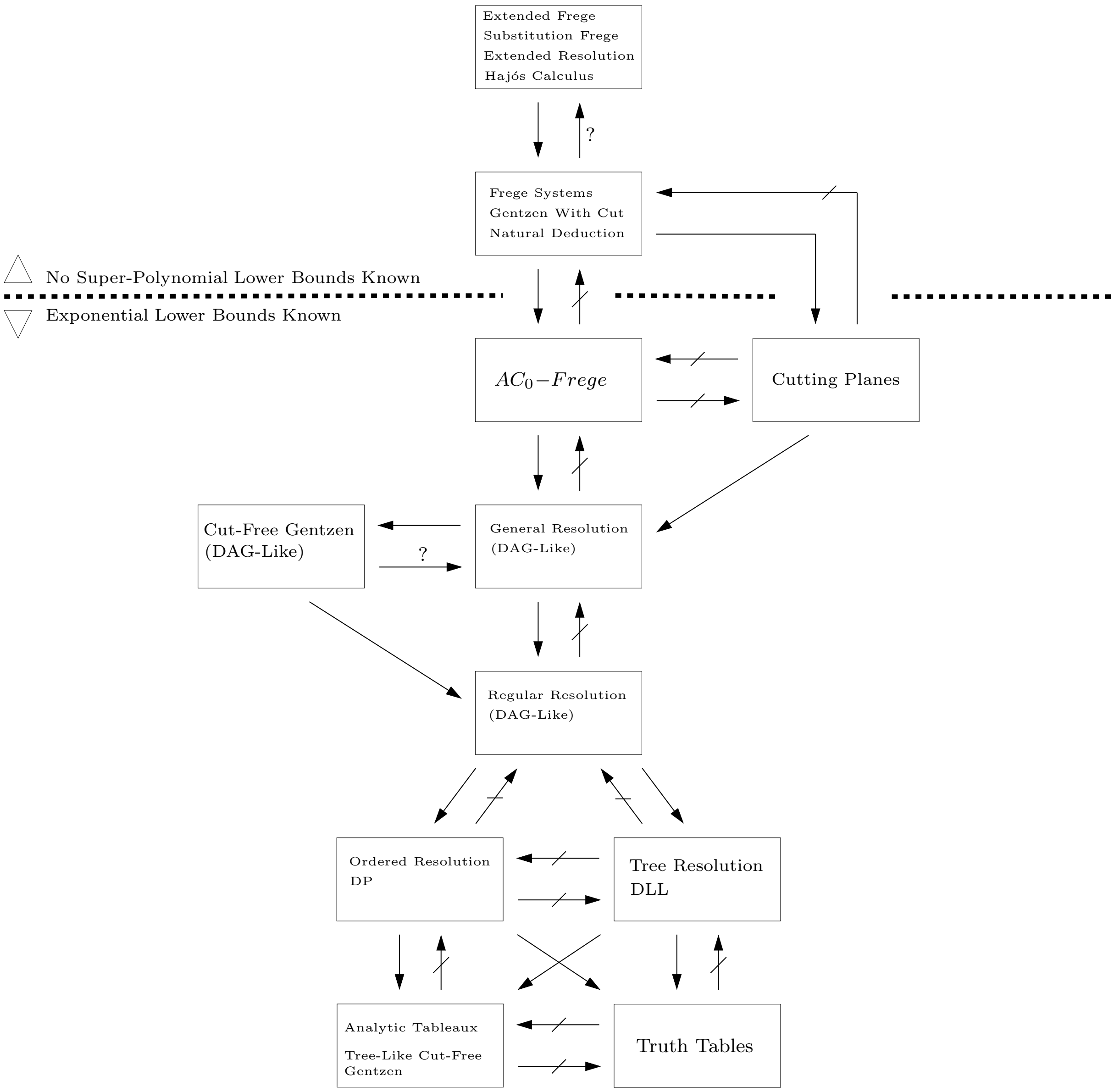

Figure 2: The Proof System p-Simulation Hierarchy

2.1 Truth Tables

Truth tables (TT) are perhaps the most obvious way of proving formulas to be tautologies (logically true) or unsatisfiable (logically false). A truth table for a formula F on n sentence letters is simply 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 can 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.

2.2 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 such that each child is labeled 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 branch 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 p-simulate TT [Urq95]. Similarly, TT cannot p-simulate AT.

2.3 Resolution

Resolution in all of its many forms is perhaps the best-studied propositional proof system. Much like AT, Resolution is a refutation system for unsatisfiable CNF formulas. The resolution rule is quite simple: 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. The vertices of these graphs are labeled with clauses. 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.

2.3.1 Tree Resolution

If the underlying graph of a resolution proof is 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 Resolution in its weakest form, and its exponential lower bounds are easily understood.

One particularly intuitive way of proving T-RES lower bounds is through a 'Prover / Delayer' game, described in [BSIW04]. The Prover / Delayer game is an adversarial game that is played using an unsatisfiable CNF formula F. The prover's goal is to prove that the formula is unsatisfiable, whereas the delayer's goal is to delay the prover for as long as possible. The game proceeds in a number of rounds. The prover chooses a variable and queries the delayer for its value. The delayer has three possible responses: 'true', 'false', and 'you choose', at which point the prover is allowed to assign the value. The delayer wins a point whenever 'you choose' is said, and the game ends as soon as a clause is falsified. This game provides lower bounds on T-RES size according to the following theorem:

Theorem 2.2. *Given an unsatisfiable CNF formula F, if the delayer has a strategy which is guaranteed to score at least r points, then the number of clauses in the smallest* T-RES *refutation of F is at least 2^r.*

This is a useful tool, since it allows T-RES lower bounds to be proved under the intuitive setting of the game. Numerous examples of formulas requiring exponential T-RES proofs are known.

As already mentioned, there is considerable interest in SAT solvers. One such algorithm, called DPLL after its authors [DLL62], is widely-used and successful. It turns out that the DPLL algorithm corresponds almost exactly to T-RES as a proof system. DPLL works by building a tree in which the root contains the original formula, each edge corresponds to a restriction, and every other node corresponds to the resulting restricted formula. DPLL terminates once every leaf contains a clause that has been falsified. It is not hard to see that with at most a little bit of pruning, the tree built by the DPLL algorithm forms an inverted T-RES proof for exactly the same formula. Lower bounds for T-RES therefore translate directly into algorithmic lower bounds for DPLL. This is a good example of the interaction between proof complexity and algorithms.

2.3.2 Ordered Resolution

In Ordered Resolution (O-RES), each variable x is systematically eliminated by performing all possible resolutions involving it. The problem 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 correlate to algorithmic lower bounds for DP.

2.3.3 Regular Resolution

Let $C_1, C_2, ..., C_{k-1}, C_k$ be a path within a branch of a Resolution refutation where each C is a clause. If C_1 and C_k contain a variable x but $C_2, ..., C_{k-1}$ do not, then that path is said to be an 'irregularity', and the refutation is said to be irregular. Regular Resolution (R-RES) is resolution in which the underlying structure is a DAG, but irregularities are disallowed. In what was to become the first major proof complexity lower bound, Tseitin proved superpolynomial 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 particularly 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 the irregularity while shortening the proof [Urq95]. 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 the other two Resolution systems.

2.3.4 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:

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

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

Since the PHP is clearly true, the formula 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 [Sab02]. Although slightly improved, Haken's result remains essentially the same:

Theorem 2.3. *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 Resolution 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 so as to eliminate all large clauses. The resulting restricted proof is a refutation of $PHP^{n'}_{n'-1}$, but $\frac{2}{9}n'^2$ 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:

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

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:

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

Intuitively, since they contain more clauses and these clauses could only help when trying to find shorter proofs, the functional PHP and onto PHP should be easier for proof systems to deal with. Nevertheless, Haken's original proof generalizes easily to show that even the onto PHP requires 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, 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 [BSIW04], Ben-Sasson et. al. use pebbling graphs to prove a near-optimal separation of T-RES and RES. Pebbling graphs are directed graphs which contain the additional vertex subsets S and T. Each vertex has at most in-degree 2, and one can think of pebbling graphs as circuits where the vertices represent gates. The idea with pebbling graphs is to place a pebble on some target node in T according to the following rules: we have an unlimited number of red and blue pebbles. We may remove a pebble at any time, and we may place a pebble on any source node in S at any time. For every other node not in S, we may pebble it only if both of its predecessors have pebbles on them. The pebbling number $P_G(S,T)$ for a graph G with source and terminal sets S and T is the minimum number of pebbles required in order to place a pebble on some vertex in T.

The colours of the pebbles come into play when translating a pebbling graph into its corresponding formula $Peb_{G,S,T}$. The variables are of the form $x_{v,c}$, which are interpreted as meaning vertex v has a pebble of colour c on it. For each $s \in S$, we create a clause saying that it either has a red or blue pebble on it. For every other vertex v, we create clauses saying that if both of their predecessors are pebbled (it does not matter what colours the pebbles are), then v either has a red or blue pebble on it. Finally, we create two singleton clauses for each $t \in T$ which together state that t neither has a red nor a blue pebble on it. It is not hard to see that the resulting formula is unsatisfiable.

Ben-Sasson et. al. use the Prover / Delayer game to show that for every pebbling graph G, S, T, the size of the smallest T-RES refutation of $Peb_{G,S,T}$ is $2^{P_G(S,T)}$. Combined with the result that there exist pebbling graphs with pebbling numbers at least $\Omega(n/log\ n)$ [CPT77], this yields a T-RES lower bound of $2^{\Omega(n/log\ n)}$. 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, proving a near-optimal separation.

2.3.5 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 refutation proof systems for TAUT can easily be turned into systems for UNSAT (and vice-versa) by simply negating the formula in question. For example, Resolution can be used to prove tautologies, even if they use logical connectives that Resolution cannot handle; just take a tautology F and negate it to produce an unsatisfiable DNF formula $\neg F$. Convert this formula to its equivalent CNF form, and Resolution 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 subformula of $\neg F$. For example, let us assume that we want to use Resolution 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 Resolution to prove arbitrary tautologies, regardless of the basis. Of course, the introduction of extension variables is by no means restricted to Resolution.

2.4 Gentzen's System PK

Gentzen systems refer to the logical proof systems due to Gerhard Gentzen. Originally published in German, his work has since been translated into English. Good overviews can be found in both [Coo02] and [CK01]. Gentzen's original system, called LK, was formulated for predicate logic. We are interested in LK's propositional fragment, called PK. PK is called a sequent calculus because every line in a PK 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 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 $(A_1 \wedge A_2 \wedge ... \wedge A_k) \supset (B_1 \vee B_2 \vee ... \vee B_k)$.

PK 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 PK inference rules:

weakening:

$$\textbf{left} \quad \frac{\Gamma \mapsto \Delta}{A, \Gamma \mapsto \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \mapsto \Delta}{\Gamma \mapsto \Delta, A}$$

exchange:

$$\textbf{left} \quad \frac{\Gamma_1, A, B, \Gamma_2 \mapsto \Delta}{\Gamma_1, B, A, \Gamma_2 \mapsto \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \mapsto \Delta_1, A, B, \Delta_2}{\Gamma \mapsto \Delta_1, B, A, \Delta_2}$$

contraction:

$$\textbf{left} \quad \frac{\Gamma_1, A, A, \Gamma_2 \mapsto \Delta}{\Gamma_1, A, \Gamma_2 \mapsto \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \mapsto \Delta_1, A, A, \Delta_2}{\Gamma \mapsto \Delta_1, A, \Delta_2}$$

$\neg$ 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 PK 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 Resolution, PK can be characterized as being either DAG-like or tree-like depending on whether derived sequents are allowed to be reused or not. Unlike Resolution, however, tree-like and DAG-like PK 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. PK with cut is p-equivalent to Frege systems, whereas cut-free DAG-like PK can be p-simulated by RES, and cut-free tree-like PK is very weak; it is only p-equivalent to AT. Whether cut-free PK can p-simulate RES is an open problem.

2.5 Frege Systems

Frege systems (Frege), also called Hilbert-style systems, are a group of robust 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 $p \to p$, then a substitution of the formula $(q \vee r)$ for p yields the formula $(q \vee r) \to (q \vee r)$, which can be introduced at any point in a proof. In effect, Frege systems have an infinite number of axioms. Much like Gentzen's system PK with cut, tree-like and DAG-like Frege systems are p-equivalent [CK01, p.372]. It appears that tree-like proof systems are weaker than their DAG-like counterparts only in the weaker proof systems.

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 attributed to Von Neumann [CR79].

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 $\Gamma \cup A \vdash B$ 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.

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 them is a major open problem.

2.6 Bounded-Depth Frege Systems

We do have lower bounds for a special 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. *The* depth *of a formula F is the minimum height among all unbounded fan-in circuits equivalent to F.*

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

Bounded-Depth Frege systems are alternatively called AC^0-Frege Systems because constant depth circuits comprise the class AC_0. Exponential lower bounds for AC^0-Frege Systems were proved using techniques based on circuit complexity lower bounds for parity. 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 [Ala02], is stated as follows:

Lemma 2.5. *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.

3 More On Powerful Proof Systems

We conclude our survey with a few remarks about proof systems that appear to be even more powerful than Frege systems as well as a few comments on 'non-propositional proof systems'.

3.1 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].

3.2 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 that they are in the same equivalence class.

3.3 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 Resolution 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 [Urq95].

3.4 Non-Propositional Proof Systems

As it turns out, the phrase 'Propositional Proof Complexity' has become a bit of a misnomer because proof systems are not necessarily proof systems for propositional logic. For example, Cutting Planes (CP) is a proof system for refuting inconsistent systems of linear inequalities [BP01, CK01]. Similarly, different algebraic proof systems such as Nullstellensatz (NS) and Polynomial Calculus (PC) [CK01, p.306] are of interest to proof complexity researchers. Exponential lower bounds are known for all of these proof systems.

Another interesting non-propositional proof system is the Hajós Calculus (HC). It is a graph theoretic proof system for non-3-colourability, which is co$\mathcal{NP}$-Complete. Its complexity was an open problem for many years before Pitassi and Urquhart showed that as a proof system it surprisingly belongs in the same equivalence class as E-Frege [PU95].

4 Acknowledgements

- First and foremost I'd like to thank my supervisor Alasdair for the many hours he spent patiently teaching me the ins and outs of proof complexity, and for guiding me through the depth oral process. His teaching was extremely valuable to me, and it was also extremely appreciated.
- I'd also like to thank Toni for her meetings with me as well as her input into what I should be reading as well as guidance with respect to the depth oral process.
- Thanks to Philipp for sharing his notes.
- Thanks to Tanya for her continued help and support.
- A nod of appreciation to Alan for sending me his PK inference rules.
- And finally thank you to NSERC and the U of T Dept. of Computer Science for their support.

References

[Ala02] Sheikh M.N. Alam. CS 2429 -Propositional Proof Complexity Lecture 9 Scribe Notes. Lecturer: Toniann Pitassi, 2002.

[APU01] N. H. Arai, T. Pitassi, and A. Urquhart. The Complexity of Analytic Tableaux. *Proceedings of the 33rd ACM Symposium on the Theory of Computing*, pages 356 – 363, 2001.

[Bea94] P. Beame. A Switching Lemma Primer. Manuscript, 1994.

[BP96] P. Beame and T. Pitassi. Simplified and Improved Resolution Lower Bounds. *Proceedings of the 37th Annual IEEE Symposium on the Foundations of Computer Science*, 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.

[BSIW04] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near Optimal Separation of Tree-like and General Resolution. *Combinatorica*, Vol. 24, Issue 4:585 – 604, 2004.

[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 Third Annual ACM Symposium on the Theory of Computation*, pages 151 – 158, 1971.

[Coo75] S.A. Cook. An Exponential Example For Analytic Tableaux. Manuscript, 1975.

[Coo02] S.A. Cook. 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 Sixth Annual ACM Symposium on the Theory of Computing*, 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.

[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.

[Gal77] Z. Galil. On the Complexity of Regular Resolution and the Davis-Putnam Procedure. *Theoretical Computer Science*, 4:23 – 46, 1977.

[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.

[PU95] T. Pitassi and A. Urquhart. The Complexity of the Hajós Calculus. *SIAM Journal of Discrete Mathematics*, Vol. 8, No. 3:464 – 483, 1995.

[Rec76] R. A. Reckhow. *On the Lengths of Proofs in the Propositional Calculus.* PhD thesis, University of Toronto, 1976.

[Sab02] Mehrdad Sabetzadeh. CS 2429 -Propositional Proof Complexity Lecture 4 Scribe Notes. Lecturer: Toniann Pitassi, 2002.

[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.

[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.