Proof Complexity

Lecturer: Paul Beame
Notes: Ashish Sabharwal

Contents

Proof Complexity 1

Lecture 1. An Introduction to Proof Complexity 1
- 1.1. Proof Systems 1
- 1.2. Examples of Propositional Proof Systems 2
- 1.3. Proof System Hierarchy 9

Lecture 2. Lower Bounds in Proof Complexity 11
- 2.1. The Pigeonhole Principle 11
- 2.2. Width vs. Size of Resolution Proofs 13
- 2.3. Resolution Proofs Based on Width-Size Relation 15
- 2.4. Nullstellensatz and Polynomial Calculus Lower Bounds 19
- 2.5. Polynomial Calculus with Resolution - PCR 21

Lecture 3. Automatizability and Interpolation 23
- 3.1. Automatizability 23
- 3.2. Interpolation 23
- 3.3. Lower Bounds using Interpolation 25
- 3.4. Limitations 26

Lecture 4. The Restriction Method 29
- 4.1. Decision Trees 29
- 4.2. Restriction Method in Circuit Complexity 30
- 4.3. Restriction Method in Proof Complexity 32

Lecture 5. Open Problems 35

Bibliography 37

IAS/Park City Mathematics Series
Volume ?, 2000

Proof Complexity

Lecturer: Paul Beame
Notes: Ashish Sabharwal

LECTURE 1
An Introduction to Proof Complexity

NP is characterized by a following simple property. For $L \in \mathrm{NP}$, all strings in L have a short, polynomial time checkable proof of membership in L. This immediately gives one way of proving $\mathrm{NP} \neq \mathrm{coNP}$ that Cook originally came up with in 1970's – find a language in coNP that does *not* have short proofs. Separating NP from coNP this way would also separate P from NP, and might therefore be hard. The theory of proof complexity gives us a way of breaking this problem into smaller, more tangible ones. The research in this area has led to lots of nice, very useful smaller steps towards answering the big question of $\mathrm{P} \neq \mathrm{NP}$.

1.1. Proof Systems

Consider the boolean formula satisfiability problem, SAT. For formulas in SAT, there is always a short proof of satisfiability – a satisfying truth assignment – and therefore SAT is trivially in NP. However, for formulas not in SAT, it is not that clear what a proof of unsatisfiability could be. Some possible proofs are transcript of failed search for satisfying truth assignment, truth tables, Frege-Hilbert proofs and resolution proofs. The question is, can these proofs always be short? If yes, then $\mathrm{NP} = \mathrm{coNP}$. This leads us to the definition of a proof system.

Definition 1.1. A *proof system* for a language L is a polynomial time algorithm V such that for all inputs x, $x \in L$ iff there exists a string P such that V accepts input (x, P).

We think of P as a *proof* that x is in L and V as a *verifier* of this proof. The complexity of a proof system is a measure of how large $|P|$ has to be as a function of $|x|$.

[1]Department of Computer Science, Box 352350, University of Washington, Seattle WA 98195-2350.
E-mail address: beame@cs.washington.edu.
Received by the editors October 31, 2000.

Definition 1.2. The *complexity* of a proof system V is a function $f : \mathbb{N} \to \mathbb{N}$ defined by

$$f(n) = \max_{x \in L, |x|=n} \min_{P: V \text{ accepts } (x,p)} |P|$$

We say V is *polynomially-bounded* iff $f(n)$ is bounded above by a polynomial function of n.

In this terminology, NP can be redefined to be precisely the set of languages that have a polynomially-bounded proof system.

Definition 1.3. A *propositional proof system* is a proof system for the set TAUT of propositional logic tautologies, i.e. a polynomial time algorithm V such that for all formulas F, F is a tautology iff there exists a string P such that V accepts input (P, F).

The existence of a proof for each tautology is called *completeness* of the proof system. The fact that existence of a proof implies the given formula is a tautology is called *soundness*. Since a formula is unsatisfiable iff its negation is a tautology, we can give the following equivalent definition of propositional proof systems.

Definition 1.4. A *propositional proof system* is a proof system for the set UNSAT unsatisfiable propositional logic formulas, i.e. a polynomial time algorithm V such that for all formulas F, F is unsatisfiable iff there exists a string P such that V accepts input (P, F).

Theorem 1.5 ([**CR77**])**.** *There is a polynomially-bounded propositional proof system iff NP = coNP.*

Proof. We know that SAT is NP-complete. For any formula F, $F \in$ TAUT iff $\neg F \in$ UNSAT iff $\neg F \notin$ SAT. It follows that both TAUT and UNSAT are coNP-complete. From what we said above, there exists a polynomially-bounded proof system for TAUT and UNSAT (i.e. a propositional proof system) iff both these languages belong to NP. □

Over the years, people have come up with a large number of proof systems. Given any two such systems, it is useful to have a way of saying if one is *better* than the other. A natural notion for this is to consider one proof system at least as powerful as a second proof system if the former can "simulate" the latter efficiently. We give a more formal statement of this in the following.

Definition 1.6. A proof system U *polynomially simulates* (or p-simulates) a proof system V iff

1. Both U and V prove the same language L, i.e.
$$\exists P.V \text{ accepts } (x, P) \iff \exists P'.U \text{ accepts } (x, P')$$
2. Proofs in V can be efficiently converted into proofs in U, i.e. there is a polynomial-time computable function f such that
$$V \text{ accepts } (x, P) \iff U \text{ accepts } (x, f(P))$$

Definition 1.7. Proof systems U and V are said to be *polynomially equivalent* iff either of them can polynomially simulate the other.

1.2. Examples of Propositional Proof Systems

1.2.1. Truth Tables

One naive way of proving that a formula computes a certain function is to give a completely filled truth table for it. Whether the table is correct or not can be checked quickly relative

to the size of the table. However, the proof size itself is exponential, which renders this proof system practically useless.

1.2.2. Tableaux/Model Elimination Systems

The idea here is to search through sub-formulas of the given formulas that might be TRUE simultaneously. For example, if $\neg(A \rightarrow B)$ is TRUE, then A must be TRUE and B must be FALSE. Starting with the input formula, we build a tree of possible models based on subformulas and derive a contradiction in each branch. This system is equivalent to sequent calculus (described later) without the cut rule. In terms of complexity, proofs here can be even larger than truth tables.

1.2.3. Axiom/Inference Systems: Frege Proofs

These systems have a set of axiom schemas such as *excluded middle* which says $|(A \vee \neg A)$, meaning for any formula A, one can derive $(A \vee \neg A)$ from nothing. It further has a bunch of inference rules such as *modus ponens* which says $A, (A \rightarrow B)|B$, meaning for any formulas A and B, one can derive B if A and $(A \rightarrow b)$ are already present. To show a given formula is unsatisfiable, we start with the formula and keep applying these axioms and inference rules to finally derive FALSE.

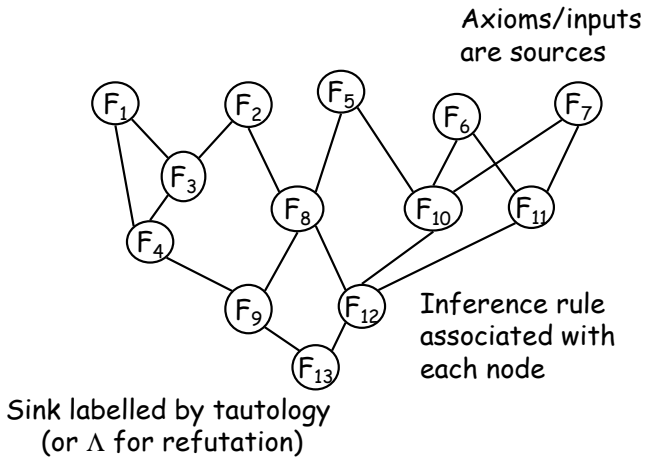

Figure 1. The graph of a Frege proof

More precisely, Frege systems start with a finite, implicationally complete set R of axioms and inference rules. A Frege refutation (or proof of unsatisfiability) of a formula F is a sequence $F_1, \ldots, F_r$ of formulas (called lines of the proof) such that

1. $F_1 = F$,
2. each F_j follows from an axiom in R or follows from previous formulas via an inference rule in R,
3. $F_r =$ FALSE trivially, e.g. $x \wedge \neg x$.

We can associate in a natural way a directed acyclic graph with a Frege refutation as shown in figure 1, where the implicit direction of each edge is from top to bottom. Each node has an associated inference rule and is derived using this rule from the formulas that point to it. An example of Frege refutation is shown in figure 2.

Theorem 1.8. *All Frege systems are polynomially equivalent.*

Subset of rules
a. $A, (A \to B) \mid B$
b. $(A \wedge B) \mid A$
c. $(A \wedge B) \mid B$
d. $A, B \mid (A \wedge B)$

1. $((x \wedge (x \to y)) \wedge ((x \wedge y) \to \neg x))$	Given
2. $(x \wedge (x \to y))$	From 1 by b
3. $((x \wedge y) \to \neg x)$	From 1 by c
4. x	From 2 by b
5. $(x \to y)$	From 2 by c
6. y	From 4,5 by a
7. $(x \wedge y)$	From 4,6 by d
8. $\neg x$	From 6,3 by a
9. $(x \wedge \neg x) = \Lambda$	From 4,8 by d

Figure 2. Sample Frege refutation with 4 inference rules

Proof. Consider two Frege systems given by axiom/inference rule sets R_1 and R_2. The general form of an axiom/inference rule would be $G_1, G_2, \ldots, G_k | H$, meaning that given $G_1, G_2, \ldots, G_k$, one can derive H in a single step. The case $k = 0$ makes this rule an axiom. We will show how to efficiently simulate R_2 with R_1.

Since R_1 is complete and R_2 is sound and finite, for every schema σ in R_2 as above, there is a constant size proof in R_1 of the tautology $(G_1 \wedge G_2 \wedge \ldots \wedge G_k) \to H$. Given any deduction of F from $F_1, F_2, \ldots, F_k$ in R_2 using σ (i.e. $F_i = G_i[x : y], F = H[x : y]$ for some substitution $[x : y]$), we can get a corresponding deduction in R_1 as follows:

1. Derive $(F_1 \wedge F_2 \wedge \ldots \wedge F_k)$ which has a constant size proof from $F_1, F_2, \ldots, F_k$ in R_1.
2. Copy the R_1 proof of σ but use the substitution $[s : y]$ at the start to prove $(F_1 \wedge F_2 \wedge \ldots \wedge F_k) \to F$.
3. Derive F from $(F_1 \wedge F_2 \wedge \ldots \wedge F_k)$ and $(F_1 \wedge F_2 \wedge \ldots \wedge F_k) \to F$ again in constant size.

Starting with the R_2 proof and doing this for every deduction in the proof, we get an R_1 proof that is not too large. Hence R_1 polynomially simulates R_2. Since R_1 and R_2 are arbitrary, the result follows. □

1.2.4. Gentzen/Sequent Calculus

This is a proof system where statements are of the form $F_1, \ldots, F_k \to G_1, \ldots, G_l$, meaning $(F_1 \wedge \ldots \wedge F_k) \to (G_1 \vee \ldots \vee G_l)$. Axioms in this system are $F \to F$. To prove a formula G, one has to derive $\to G$ and to refute it, one has to derive $G \to$. This is done using the axioms and the following rules:

1. $\Gamma, F \to \Delta$ and $\Gamma, G \to \Delta$ imply $\Gamma, (F \vee G) \to \Delta$
2. $\Gamma \to \Delta, F$ implies $\Gamma \to \Delta, (F \vee G)$
3. $\Gamma \to \Delta, F$ implies $\Gamma, \neg F \to \Delta$
4. $\Gamma, F \to \Delta$ implies $\Gamma \to \Delta, \neg F$
5. **Cut rule**: $\Gamma \to \Delta, F$ and $\Pi, F \to \Sigma$ imply $\Gamma, \Pi \to \Delta, \Sigma$

We can characterize sequent calculus cleanly based on what kinds of formulas F are used in the cut rule. This system is often used in proof complexity but the proofs are cumbersome to write down and we won't use it here. However, we should mention two things about sequent calculus. First, if we take away the cut rule, we still get a complete

proof system, though a much weaker one. Second, it is known that sequent calculus is polynomially equivalent to Frege systems.

1.2.5. Resolution

Resolution forms the basis of most popular systems for practical theorem proving. It is like Frege systems but uses only CNF clauses. We start with the original input clauses of a CNF formula F and repeatedly pick pairs of clauses to apply the resolution rule: $(A \vee x), (B \vee \neg x) | (A \vee B)$. The goal is to derive the empty clause Λ.

Exercise 1.9. Show that resolution may be simulated by sequent calculus where we start with one sequent per clause and all cuts are on literals.

This proof system can only work with CNF formulas. However, we do not loose much by requiring input to be in CNF form. This can be seen by the following procedure which efficiently converts any given formula to one in CNF form (or DNF form in case we are interested in proving the formula to be a tautology). This uses the trick that is used in [**T68**] to reduce SAT instances to CNFSAT instances. The idea is to add an extra variable y_G corresponding to each sub-formula G of the input propositional formula F. C_F then includes clauses (or terms in the DNF case) expressing the fact that y_G takes on the value of G determined by the inputs to the formula. More precisely,

- If $G = H \vee J$, then C_F includes clauses $(\neg y_H \vee y_G), (\neg y_J \vee y_G)$ and $(\neg y_G \vee y_H \vee y_J)$.
- If $G = H \wedge J$, then C_F includes clauses $(\neg y_G \vee y_H), (\neg y_G \vee y_J)$ and $(\neg y_H \vee \neg y_J \vee y_G)$.
- If $G = \neg H$, then C_F includes clauses $(\neg y_G \vee \neg y_H)$ and $(y_G \vee y_H)$.

C_F also contains clause $\{y_F\}$ expressing the truth value of F. The claim, which can be easily verified, is that for any assignment α to the variables of F, α satisfies F iff there exists an assignment β to variables y_G such that (α, β) satisfies C_F. Thus F and C_F are equivalent as far as satisfiability is concerned.

1.2.6. Davis-Putnam (DLL) Procedure

Davis-Putnam or DLL procedure is both a proof system and a collection of algorithms for finding proofs. As a proof system, it forms a special case of resolution where the proof graph forms a tree. A simple David-Putnam algorithm is shown in figure 3. Variants of this algorithm form the most widely used family of complete algorithms for satisfiability.

Refute(F)

1. While F contains a clause of size 1
   (a) Set variable to make that clause TRUE
   (b) Simplify all clauses using this assignment
2. If F has no clauses then
   (a) Output "F is satisfiable" and HALT
3. If F does not contain an empty clause then (Splitting rule)
   (a) Choose smallest-numbered unset variable x
   (b) Run Refute($F|_{x \leftarrow 0}$)
   (c) Run Refute($F|_{x \leftarrow 1}$)

Figure 3. Simple Davis-Putnam Algorithm

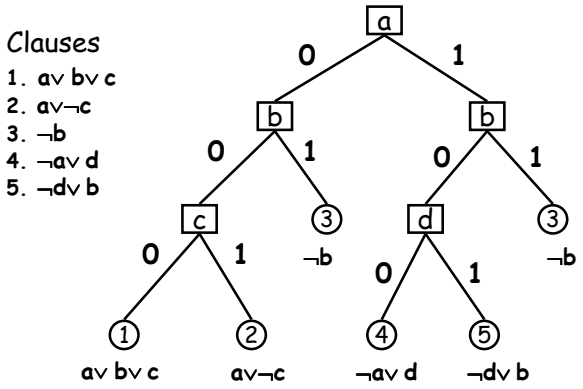

Figure 4. An example of a DLL refutation tree

A DLL refutation is essentially a tree where we branch at each node based on the value of a variable. The leaves are labelled with one of the original clauses that is falsified by the assignment represented by the branch from the root to this leaf. Figure 4 shows an example of a DLL refutation tree.

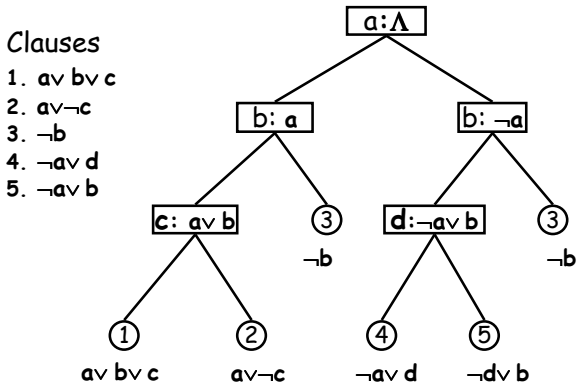

Figure 5. Creating resolution proof from DLL refutation

There is a straightforward way one can translate a DLL refutation to a tree resolution proof. The result of this translation for the DLL example above is shown in figure 5. We associate each leaf node with the input clause it is labelled with in the DLL refutation. For each node both whose children have associated clauses, we resolve these two clauses on the variable this node was branched on in the DLL refutation. The resulting clause is associated with this node. We keep doing this until we finally reach the root and associate with it the empty clause Λ. This gives a resolution tree deriving Λ from the input clauses.

1.2.7. Nullstellensatz Proof System

This proof system is based on Theorem 1.10 about polynomials equations over a field K. The idea is that any common root of the original equations is also a root of any weighted sum of the corresponding polynomials. Therefore, if we can derive a constant non-zero polynomial in this way, then the initial equations couldn't have had a common root. If we have a language whose inputs can be encoded as a set of equations which do not have

a common root iff the input is in the language, then Nullstellensatz can be used to prove membership in that language.

Theorem 1.10 (Hilbert's Nullstellensatz)**.** *A system of polynomials equations $Q_1(x_1, \ldots, x_n) = 0, \ldots, Q_m(x_1, \ldots, x_n) = 0$ over field K has no solution in any extension field of K iff there exist polynomials $P_1(x_1, \ldots, x_n), \ldots, P_m(x_1, \ldots, x_n)$ in $K[x_1, \ldots, x_n]$ such that $\sum_{i=1}^{n} P_i Q_i \equiv 1$.*

Consider an instance of 3SAT. If $C_j = (x_1 \vee \neg x_2 \vee x_3)$ is a clause in the instance, we can translate it into the equation $Q_{C_j} = (1-x_1)x_2(1-x_3) = 0$. Let us also add equations $x_i^2 - x_i = 0$ for each variable x_i. This will guarantee only 0-1 values. If $C_1, \ldots, C_m$ are all the clauses in the input formula, then it follows from Hilbert's Nullstellensatz that the formula is unsatisfiable iff there exists polynomials $P_1, \ldots, P_{m+n}$ such that

$$\sum_{j=1}^{m} P_j Q_{C_j} + \sum_{i=1}^{n} P_{m+i}(x_i^2 - x_i) \equiv 1$$

1.2.8. Polynomial Calculus

Polynomial calculus is very similar to Nullstellensatz proof system. We begin with $Q_1, Q_2, \ldots, Q_{m+n}$ as before. However, instead of trying to combine polynomials, we start deriving new polynomials according to the following rule. Given polynomials R and S, we can infer $aR+bS$ for any $a, b \in K$. We can also infer $x_i R$ for any variable x_i. The goal is to derive the constant polynomial 1. The idea is that any common root of the original polynomials is also a root of any derived polynomial. Therefore, if we can infer a constant non-zero polynomial, then the initial polynomials couldn't have had a common root.

The *degree* of a polynomial calculus proof is the maximum of the degrees of all polynomials appearing in the proof. It is known that we can find a proof of degree d in time $n^{O(d)}$ using Gröebner basis-like algorithm from linear algebra [**CEI96**]. We should note here that polynomial calculus is a special case of $\mathrm{AC}^0[p]$-Frege if $K = \mathrm{GF}(p)$, and we only need depth 1.

Exercise 1.11. Show that every unsatisfiable formula over n variables has a proof of degree at most $n + 1$ for Nullstellensatz as well as Polynomial Calculus.

1.2.9. Cutting Planes

The concept of cutting planes was introduced to relate integer and linear programming ([**G58, C73**]). As a proof system, this is a special case of TC^0-Frege with depth 1. The objects here are linear integer inequalities. For instance, a clause $(x_1 \vee \neg x_2 \vee x_3)$ becomes the inequality $x_1 + (1 - x_2) + x_3 \geq 1$. To these, we add inequalities $x_i \geq 0$ and $1 - x_i \geq 0$ to force each variable x_i to have a value between 0 and 1, both inclusive. The goal is to derive the contradiction $0 \geq 1$ using the following three rules:

- **Addition:** From $a_1x_1 + \ldots + a_nx_n \geq A$ and $b_1x_1 + \ldots + b_nx_n \geq B$, one can derive $(a_1 + b_1)x_1 + \ldots + (a_n + b_n)x_n \geq A + B$.
- **Multiplication by positive integer:** From $a_1x_1 + \ldots + a_nx_n \geq A$ and any positive integer c, one can derive $ca_1x_1 + \ldots + ca_nx_n \geq cA$.
- **Division by positive integer:** From $ca_1x_1 + \ldots + ca_nx_n \geq B$, one can derive $a_1x_1 + \ldots + a_nx_n \geq \lceil B/c \rceil$.

The reason this system is called cutting planes will be clear from figure 6. The starting inequalities define a region in the n-dimensional space which is bounded by lines that might intersect at non-integer points. Adding two such inequalities gets us a line that

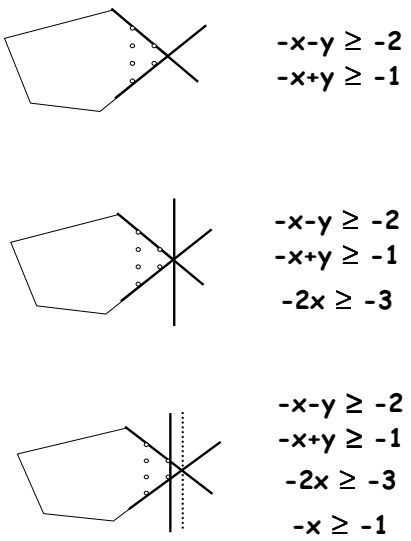

Figure 6. Why is the proof system called cutting planes?

doesn't pass thru any integer point. Applying division rule to such an inequality effectives moves this line to the nearest integral one in the direction given by the inequality. This way we "cut" in the feasible region into a smaller one by applying the division rule and taking the ceiling of the constant on the right hand side of the inequality.

Theorem 1.12. *Cutting planes polynomially simulate resolution.*

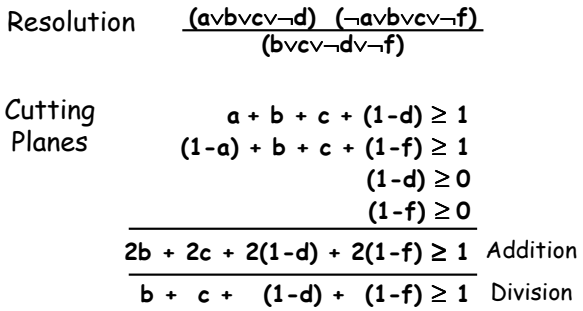

Figure 7. Simulating resolution using cutting planes

It is not hard to see how cutting planes can simulate resolution efficiently. An example is shown in figure 7. We first convert each input clause to the corresponding inequality in the standard way. Resolving two clauses then is simply adding them along with certain inequalities of the form $x_i \geq 0$ and $1 - x_i \geq 0$ so that all coefficients are 2. Dividing the resulting inequality by 2 gives one that corresponds to the resolvant of the two original clauses.

1.2.10. C-Frege Proof Systems

Many circuit complexity classes such as non-uniform $NC^1, AC^0, AC^0[p], ACC, TC^0$ and P/poly are defined as follows:

$C = \{f : f$ is computed by polynomial size circuits with structural property $P_C\}$.
In a similar manner, we define C-Frege to be the p-equivalence class of Frege-style proof systems such that

1. each line has structural property P_C,
2. there is a finite set of axioms and inference rules, and
3. the system is complete for circuits with property P_C.

Before saying anything more about C-Frege systems, let us quickly review some of the important circuit complexity classes.

P/poly : polysize circuits
NC_1 : polysize formulas = $O(\log n)$ depth fan-in 2 circuits
CNF : polysize CNF formulas
AC^0 : constant depth unbounded fan-in polysize circuits using $\wedge, \vee, \neg$ gates
$AC^0[m]$ similar to AC^0 with $(= 0 \mod m)$ tests added
TC^0 : similar to AC^0 with threshold gates added

We know the following relationships among these complexity classes:

- $CNF \subset AC^0 \subset AC^0[p] \subset TC^0$ for p prime
- $TC^0 \subseteq NC^1 \subseteq$ P/poly $\subseteq$ NP/poly
- $AC^0[m] \subset \#P$

Exercise 1.13. Show that every formula may be re-balanced to an equivalent one of logarithmic depth. (*Hint*: First find a node in the formula that has constant fraction of the nodes in its subtree.)

From the above exercise, it follows that Frege systems are polynomially equivalent to NC^1-Frege because NC^1 circuits can be expanded into trees (formulas) of polynomial size. Another result about C-Frege systems is that resolution is a special case of CNF-Frege. CNF-Frege however is not strong enough to express the p-simulation among Frege systems.

1.2.11. Extended Frege Systems

Extended Frege proofs are like Frege proofs plus extra extension steps that define new propositional variables to stand for arbitrary formulas on current set of variables. These new variables are like the variables y_G in the conversion of arbitrary formulas to CNF. However, they can be defined for any formula in the proof, not only for the input formulas. These extension variables allow one to write formulas more succinctly and increase the power of Frege systems. Since each extension variable describes a circuit in the input variables, extended Frege is equivalent to P/poly-Frege.

1.3. Proof System Hierarchy

It is useful to understand the relationship between the large number of proof systems that we know. The notion of polynomial simulation we defined earlier can be used to say that one proof system is at least as powerful as a second one. However, fast simulation might be too hard a condition to satisfy. One weaker notion of what it means to be at least as powerful as another proof system is polynomial domination, which is defined now.

Definition 1.14. A proof system U *p-dominates* another proof system V iff there is a polynomial $f : \mathbb{N} \to \mathbb{N}$ such that

$$\exists P.V \text{ accepts } (x, P) \iff \exists P'.|P'| \leq f(|P|) \text{ and } U \text{ accepts } (x, P')$$

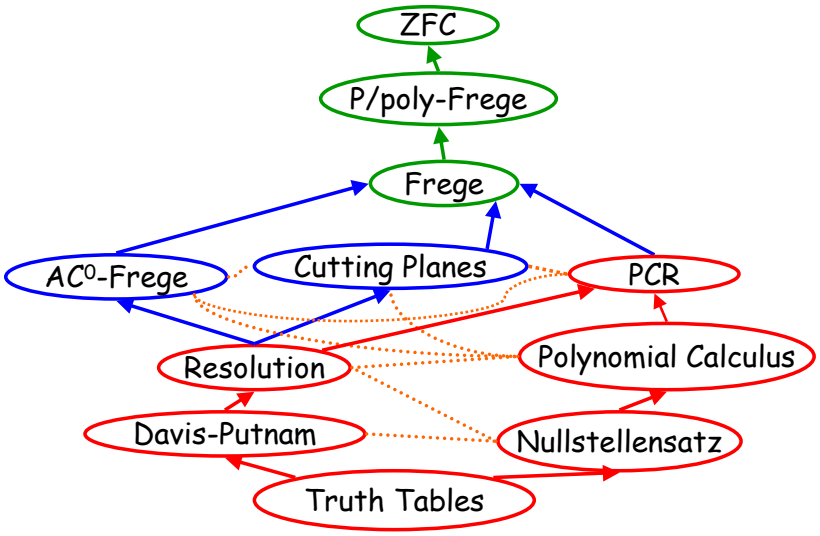

Figure 8. Some proof system relationships

Some of the proof system relationships that we know are summarized in figure 8. One might ask why we keep working with weaker proof systems when there are stronger ones we know. We do this for several reasons. First, different proof systems formalize different types of reasoning that we use and it should be useful to understand these types. Second, many weaker proof systems such as Davis-Putnam, Nullstellensatz and polynomial calculus have better associated proof search strategies and are therefore more useful as theorem proving techniques than other stronger systems. Third, there is a natural correspondence between proof system hierarchy and circuit complexity classes. As in circuit complexity, analyzing systems working upwards in proof strength helps one gain insight for useful techniques.

We do not know how high the hierarchy goes or if there is some proof system sitting at the top of this hierarchy. We formally define below what it means to be at the top of the hierarchy and say what implications the existence of such a proof system has.

Definition 1.15. U is *super* iff U p-dominates all other propositional proof systems. U is *super-duper* iff U p-simulates all such systems.

Theorem 1.16 ([**KP89**])**.** *Super-duper proof systems exist implies NEXP = coNEXP. Super proof systems exist implies NEXPEXP = coNEXPEXP.*

LECTURE 2
Lower Bounds in Proof Complexity

The first step in proving a lower bound for a proof system is to find a hard example for that proof system. Consider, for instance, the C-Frege system. A tautology seems likely to be hard to prove in C-Frege if the *natural* proof of it requires concepts that are not computable in circuit complexity class C. For example, Majority is not computable in $AC^0[p]$. This suggests that something counting-related might be hard for $AC^0[p]$-Frege. A second place to look for hard examples is randomly chosen tautologies or unsatisfiable formulas. These might be hard to prove because they simply have no particular structure that could be exploited to get a really short proof. We begin with one of the basic counting principles, the pigeonhole principle, and later show lower bounds for random formulas and graphs structures.

2.1. The Pigeonhole Principle

The pigeonhole principle $PHP^{m\to n}$ says that there is no 1-1 function from m things to n things if $m > n$. The *onto* version of this, $ontoPHP^{m\to n}$, says that there is no 1-1, onto function mapping m things to n things for $m > n$. This can be easily encoded as a propositional formula over variables P_{ij} which represent pigeon i mapping to hole j. The clauses ensure that any satisfying assignment to these variables corresponds to a valid 1-1, onto function from m things to n things. There are four kinds of clauses:

f is total: $(P_{i1} \vee P_{i2} \vee \ldots \vee P_{in})$, for $i = 1, \ldots m$
f is 1-1: $(\neg P_{ij} \vee \neg P_{kj})$, for $1 \leq i < k \leq m, j = 1, \ldots, n$
f is onto: $(P_{1j} \vee P_{2j} \vee \ldots \vee P_{mj})$, for $j = 1, \ldots, n$
f is a function: $(\neg P_{ij} \vee \neg P_{ik})$, for $i = 1, \ldots, m, 1 \leq j < k \leq n$

We note here that one usually leaves out the function clauses because they are redundant for a lower bound. One can derive the relational form of this mapping from the functional form by setting $P'_{ij} = P_{ij} \wedge \neg P_{i1} \wedge \ldots \wedge P_{i(j-1)}$.

2.1.1. Usual Inductive Proof of $PHP^{n\to n-1}$

The base case $PHP^{2\to 1}$ is trivially false. For the inductive steps, consider the hole pigeon n maps to under a mapping f. If $f(n) = n-1$, then f on $\{1, \ldots, n-1\}$ also violates $PHP^{n-1\to n-2}$ and we are done by inductive hypothesis. Otherwise define another mapping $g : \{1, \ldots, n-1\} \to \{1, \ldots, n-2\}$ by $g(i) = f(i)$ if $f(i) \neq n-1$ and $g(i) = f(n)$

otherwise. It is easy to check that g is 1-1 and onto iff f is. Further, g is a mapping from $n-1$ things to $n-2$ things and must therefore violate $PHP^{n-1\to n-2}$.

2.1.2. Extended Frege Proof of $PHP^{n\to n-1}$

The inductive proof we gave above can be easily translated into an extended Frege proof. P_{ij} are our original variables as usual. To perform the inductive step, we define new variables $Q_{ij} = P_{ij} \vee (\neg P_{n(n-1)} \wedge P_{i(n-1)} \wedge P_{nj})$ for $i = 1, \ldots, n-1, j = 1, \ldots, n-2$. This gets us a short extended Frege proof of $PHP^{n\to n-1}$.

2.1.3. Cutting Planes Proof of $PHP^{m\to n}$

The problem can be reformulated for cutting planes as follows. The constraints are

- $P_{i1} + P_{i1} + \ldots + P_{in} \geq 1$, for $i = 1, \ldots m$
- $P_{ij} + P_{kj} \leq 1$, for $1 \leq i < k \leq m, j = 1, \ldots n$
- $P_{ij} \geq 0$; $P_{ij} \leq 1$, for $i = i, \ldots, m, j = 1, \ldots, n$

The goal is to derive $P_{1j} + P_{2j} + \ldots + P_{(k-1)j} \leq 1$. We do this as follows. For k from 3 to m, do

1. Add $(k-2)$ copies of $P_{1j} + P_{2j} + \ldots + P_{(k-1)j} \leq 1$ and one each of $P_{1j} + P_{kj} \leq 1$ to get $(k-1)P_{1j} + (k-1)P_{2j} + \ldots + (k-1)P_{kj} \leq 2k-3$
2. Apply division rule to get $P_{1j} + P_{2j} + \ldots + P_{kj} \leq 1$.

Summing these inequalities $P_{1j} + P_{2j} + \ldots + P_{mj} \leq 1$ over all j gives that the sum of all P_{ij}'s is at most n. Moreover, summing up the first set of input inequalities gives us that the sum of all P_{ij}'s is at least m. Together these two imply $m \leq n$ and we get a contradiction.

2.1.4. Resolution Proof of $PHP^{n\to n-1}$

Unlike Frege and cutting planes proofs, pigeonhole principle requires exponential size resolution proofs.

Theorem 2.1 ([**?, ?**]). *Any resolution proof of $PHP^{n\to n-1}$ requires size at least $2^{n/20}$.*

The original proof idea was based on bottleneck counting. We view truth assignments flowing through the proof. Assignments start at Λ and flow out torwards input clauses. A clause in the proof lets only the assignments it falsifies to flow through it. The key thing is to prove that at a *middle* level in the proof, clauses must talk about lots of pigeons. Such a middle level clause falsifies only a few assignments and thus there must be lots of them to let all the assignments flow through.

A present here a much simplified argument which goes as follows. We show that a partial assignment to the variables, called a restriction, can be applied to every small proof so that one, every large clause disappears, and two, the result is still a $PHP^{n'\to n'-1}$ proof for some good size n'. We next show that every proof of $PHP^{n'\to n'-1}$ contains a medium complexity clause and further that every medium complexity clause is large. This get us a lower bound on proof size of $PHP^{n\to n-1}$.

We say a truth assignment is *critical* if it matches all $n-1$ holes to all but one of the pigeons. Such an assignment is barely unsatisfying – it always satisfies all 1-1, *onto* and function clauses. The only input clauses that may not be true under a critical truth assignment are $C_i = (P_{ia} \vee P_{i2} \vee \ldots \vee)_{in})$ which say that pigeon i is mapped somewhere. This allows us to modify each of the clauses in the proof to a positive one without blowing up the size of the proof. More precisely, we replace $\neg P_{ij}$ with $(P_{i1} \vee \ldots P_{(i-1)j} \vee P_{(i+1)j} \vee$

$\ldots \vee P_{nj}$). It is easy to see that these new clauses let precisely the same critical truth assignments through as they the original ones did.

To prove that any PHP proof has medium a complexity clause, we do the following. Given a positive clause C and $I \subseteq \{1, \ldots, n\}$, we say I *implies* C iff whenever for all critical truth assignments α, $\forall i \in I, C_i(\alpha) = \text{TRUE} \Leftrightarrow C(\alpha) = \text{TRUE}$. The *complexity* of C, denoted $comp(C)$, is the minimum $|I|$ such that I implies C. It is not very hard to show that every resolution proof contains a clause of complexity m between $n/3$ and $2n/3$. We do this by looking at clause complexities. We know that Λ has complexity n and the input clauses have complexity at most 1. Moreover, if clause A and B imply a clause C, then $comp(C) \leq comp(A) + comp(B)$. If we walk backwards in the proof from Λ, clause complexities decrease only in a sub-additive way. Hence they can't jump over the $(n/3, 2n/3)$ region. Thus there must always be a clause whose complexity is between $n/3$ and $2n/3$.

Now we prove that clauses with medium complexity (between $n/3$ and $2n/3$) must be big. Suppose I implies C and $|I| = m = comp(C), n/3 \leq m \leq 2n/3$. Since I is minimal, for each $i \in I$, there is a critical truth assignment α_i such that $C_i(\alpha_i) = C(\alpha_i) =$ FALSE. For each $j \notin I$, toggle α_i to yield $\alpha_i j$ where the latter assignment maps pigeon j to the hole k to which i was mapped earlier. This new assignment satisfies I and must therefore satisfy C_i also. Since $C_i(\alpha_i j) =$ TRUE, literal P_{jk} must be in the clause C since it is the only new TRUE variable since α_i. Therefore for each i and k, we have a variable P_{ik} in C, implying that C is big if it is minimally implied by a medium complexity clause. The exact bound is $m(n-m) \geq 2n^2/9$ because k can be anything not in I.

We finally describe the restriction argument that gets us the desired result. Restrictions in this case are partial assignments that map certain pigeons to certain holes. To map a pigeon i to hole j, we set P_{ij} to TRUE and set all other P_{ik} or P_{kj} are set to FALSE. This reduces $PHP^{n \to n-1}$ to $PHP^{n' \to n'-1}$, where $n' = n - 1$. To complete the proof, let us call a positive clause *large* iff it has at least $n^2/10$ literals. Assume, for a proof by contradiction, that some resolution proof of $PHP^{n \to n-1}$ has a resolution proof with at most $S < 2^{n/20}$ large clauses. On average, restricting a P_{ij} to TRUE will satisfy $S/10$ of all large clauses because large clauses each have $1/10$ of all variables. Choose a P_{ij} that satisfies the most large clauses. This restriction decreases the number of large clauses by a factor of 9/10. Now repeat such restriction $\log_{9/10} S < 0.329n$ times. The remaining proof proves $PHP^{n' \to n'-1}$ for some n' such that $2(n')^2/9 > n^2/10$ and does not have any large clauses. This is a contradiction because such a refutation, from what we saw in the previous paragraph, must have a clause of size at least $2(n')^2/9$ which qualifies as a large clause even for $PHP^{n \to n-1}$.

2.2. Width vs. Size of Resolution Proofs

Let F be a set of clauses over variables $x_1, \ldots, x_n$ and $w(F)$ be the number of literals in the largest clause in F. If P is a resolution proof of F, width(P) is the number of literals in the largest clause in P. Let width(F) denote the minimum of all proofs P of F of width(P). The following theorems due to Ben-Sasson and Wigderson relate size lower bounds on P to lower bounds on width(P).

Theorem 2.2 ([**BW99**]). *Every Davis-Putnam (DLL)/tree-like resolution proof of F of size S can be converted to one of width $\lceil \log_2 S \rceil + w(F)$.*

Proof. We show this by induction on the size of the resolution proof. Clearly, the claim holds for $S = 1$. Assume that for all sets F' of clauses with a tree-like resolution refutation

of size $S' < S$, there is a tree-like resolution proof P' of F' with width$(P') \leq \lceil \log_2 S' \rceil + w(F')$.

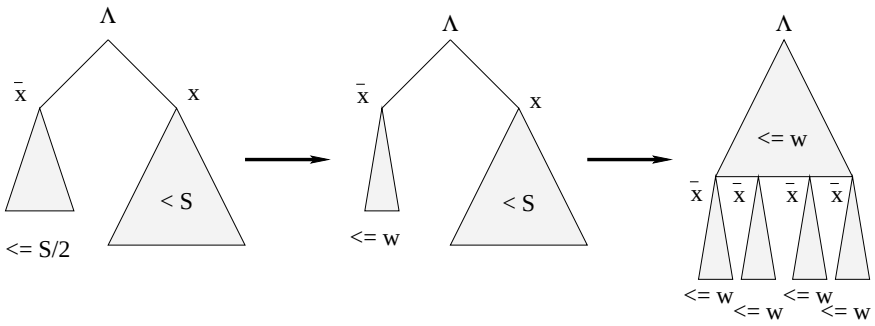

Figure 1. Converting small size proof to one of small width

Now consider a tree-like resolution refutation of size S of a set F of clauses and let x be the last variable resolved on to derive the empty clause Λ. Clearly, one of the two subtrees at the top has size at most $S/2$ and the other has size strictly smaller than S. W.l.o.g. let these be the left and the right subtree, respectively. Also assume $\neg x$ comes from the left subtree and x from the right as in figure 1.

Since we can prove $\neg x$ from F in size at most $S/2$, we can also prove Λ from $F|_{x \leftarrow 1}$ in size at most $S/2$. The induction hypotheses now implies that we can also derive Λ from $F|_{x \leftarrow 1}$ in width at most $\lceil \log_2(S/2) \rceil + w(F) = \lceil \log_2 S \rceil + w(F) - 1$. Adding $\neg x$ to each of the clauses in this proof lets us derive $\neg x$ from F in width $\lceil \log_2 S \rceil + w(F)$. In a similar way, starting with the right subtree, which is of size strictly smaller than S, we can derive Λ from $F|_{x \leftarrow 0}$ in width at most $\lceil \log_2 S \rceil + w(F)$.

Let us take the left subtree and reconstruct it so that it derives $\neg x$ in width $\lceil \log_2 S \rceil + w(F)$. Now plug this tree at the bottom of each leaf of the right subtree (see figure 1). This allows us to resolve x right at the bottom of the right subtree, and we are effectively left with $F|_{x \leftarrow 0}$. From what we said before, we can now derive Λ from this in width $\lceil \log_2 S \rceil + w(F)$. This completes the proof. □

Corollary 2.3. *Any Davis-Putnam (DLL)/tree-like resolution proof of F requires size at least $2^{\Omega(\text{width}(F) - w(F))}$.*

Theorem 2.4 ([BW99]). *Every resolution proof of F of size S can be converted to one of width $O(\sqrt{n \log S}) + w(F)$.*

Proof. The key idea behind this proof is to repeatedly find the most popular literals appearing in large clauses in the given resolution proof. Resolving on these literals at the very beginning allows us to keep the width of the whole proof small.

Let us call a clause *large* if it has width at least $W = \sqrt{2n \ln S}$. Since there are at most $2n$ literals and at least W of them appear in any large clause, an average literal must occur in at least $W/2n$ fraction of large clauses. Let k be such that $(1 - W/2n)^k S \leq 1$. We show by induction on n and k that any F with at most S large clauses has a proof of width $\leq k + w(F)$. The base case is trivial. Assume now that the theorem holds for all smaller values of n and k.

Choose the literal x that occurs most frequently in large clauses and set it to 1. This, from what we observed before, will satisfy at least a $W/2n$ fraction of large clauses. What

we get as a result is a refutation of $F|_{x\leftarrow 1}$ with at most $S(1 - W/2n)$ large clauses. By our induction hypothesis, $F|_{x\leftarrow 1}$ has a proof of width at most $k - 1 + w(F)$. Hence there is a derivation of $\neg x$ from F of width at most $k + w(F)$.

Now consider $F|_{x\leftarrow 0}$. If we restrict the proof of F which has at most S large clauses, we get a proof of $F|_{x\leftarrow 0}$ with at most S large clauses over one less variable. The induction hypothesis implies that there is a refutation of $F|_{x\leftarrow 0}$ of width at most $k + w(F)$.

As in the proof of tree-like resolution case, we derive $\neg x$ from F in width at most $k + w(F)$ and resolve this with each clause of F to get $F|_{x\leftarrow 0}$. Now we refute this in width $k + w(F)$. □

Corollary 2.5. *Any resolution proof of F requires size at least $2^{\Omega(\frac{(\mathrm{width}(F)-w(F))^2}{n})}$.*

We note here that this relationship between width and size is optimal for general resolution as shown by the following result:

Theorem 2.6 ([**?**])**.** *There are tautologies with constant input size and polynomial-size proofs that require width $\Omega(n)$.*

[**?**] and [**BW99**] use graph pebbling and width-based lower bounds to show that Davis-Putnam (DLL)/tree-like resolution can require exponentially larger ($2^{\Omega(n/\log n)}$ size) proofs than general resolution.

2.3. Resolution Proofs Based on Width-Size Relation

Given F, a set of unsatisfiable clauses, let $s(F)$ be the size of the minimum subset of F that is unsatisfiable. Define *boundary* δF of F as the set of variables appearing in exactly one clause of F. Let the *sub-critical expansion* of F be

$$\max_{s\leq s(F)} \min\{|\delta G| : G \subseteq F, s/2 \leq |G| < s\}$$

The following lemma, which is depicted in figure 2, relates proof width to sub-critical expansion of F.

Lemma 2.7 ([**CS88**])**.** *If P is a resolution proof of F, then $width(P) \geq e(F)$.*

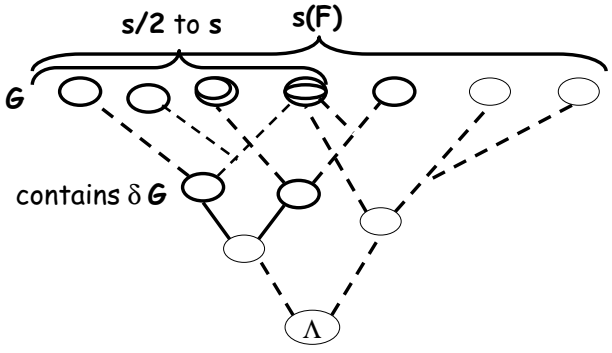

Figure 2. Relating proof width to sub-critical expansion

Corollary 2.8. *Any Davis-Putnam/DLL proof of F requires size at least $2^{e(F)}$ and any resolution proof requires size at least $2^{\Omega(e^2(F)/n)}$.*

2.3.1. Random k-CNF Formulas

Let distribution $F^k_{n,\Delta}$ over random k-CNF formulas be defined as making $m = n\Delta$ independent choices of one of the $2^k\binom{n}{k}$ clauses of length k. Δ here is called the density of the graph. Let F be chosen from this distribution (also written $F \sim F^k_{n,\Delta}$). It is well known that satisfiability of such random k-CNF formulas is determined by a density threshold. As shown in figure 3 for $k = 3$, random formulas with density more than a certain threshold are asymptotically almost surely unsatisfiable, whereas those with density below are threshold are a.a.s. satisfiable. For random graphs with density above the threshold, resolution proofs of satisfiability are almost surely super-polynomial, as stated in the following theorem:

Theorem 2.9. *For $F \sim F^k_{n,\Delta}$, almost certainly for any $\epsilon > 0$,*

1. *Any Davis-Putnam (DLL) proof of F requires size at least $2^{\frac{n}{\Delta^{2/(k-2)+\epsilon}}}$.*
2. *Any resolution proof of F requires size at least $2^{\frac{n}{\Delta^{4/(k-2)+\epsilon}}}$.*

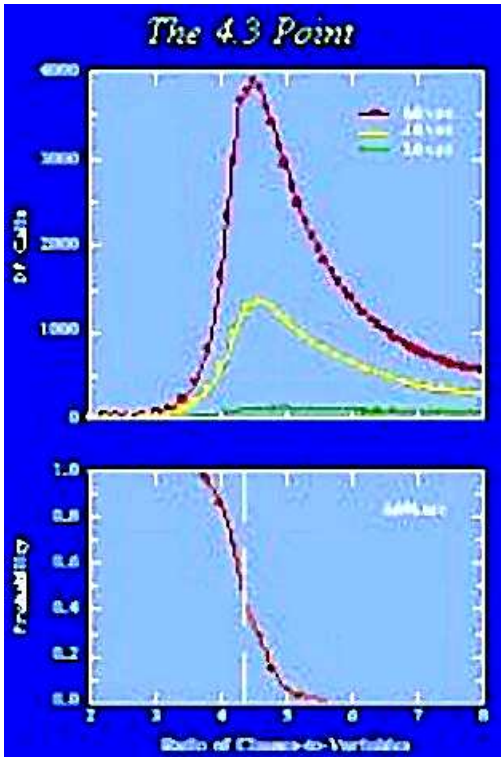

Figure 3. Threshold behavior of random 3SAT formulas

This result implies that random k-CNF formulas are provably hard for the most common proof search procedures which are DLL type. In fact, this hardness extends well beyond the threshold. Even at density $\Delta = n^{1/3}$, current algorithms for random 3-CNF have asymptotically the same running time as the best factoring algorithms.

The proof of this theorem is based on properties of random hypergraphs. Let F be a hypergraph. Denote by δF the *boundary* of F, which is the set of degree 1 vertices of

F. The density of F is the ratio of the number of hyperedges to the number of vertices. We say that a subset of F has a system of distinct representatives (see figure 4) iff with each hyperedge in F, we can associate a unique vertex (a representative) belonging to that hyperedge. Let $s_H(F)$ be the size of minimum subset of F that does not have a system of distinct representatives. Define the *sub-critical expansion*, $e_H(F)$, of F as

$$e_H(F) = \max_{s \leq s_H(F)} \min\{|\delta G| : G \subseteq F, s/2 \leq |G| < s\}$$

The following theorem allows us to get lower bounds on $s_H(F)$ and $e_H(F)$ for random hypergraphs:

Theorem 2.10 (Hall's Theorem)**.** *A hypergraph F has a system of distinct representatives iff every subgraph of F has density at most 1.*

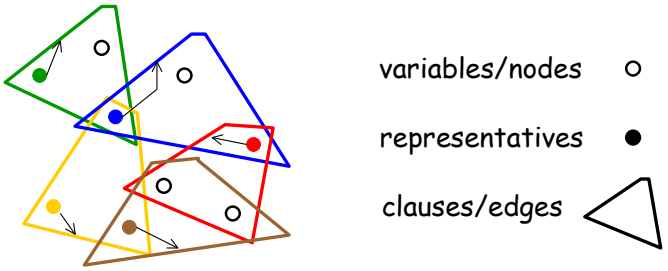

Figure 4. System of distinct representatives

A k-CNF formula can be associated with hypergraphs in a natural way, where each variable becomes a vertex and each clause becomes an edge. This mapping discards the distinction between a variable and its negation, but is sufficient for proving useful results. It is easy to see that if the hypergraph has a system of distinct representatives, then the corresponding k-CNF formula is satisfiable and a satisfying assignment can be obtained by setting each representative to satisfy the clause which it represents. If we define notions $s(F)$ and $e(F)$ for the k-CNF formula corresponding to a hypergraph, then graph theoretic lower bounds on $s_H(F)$ and $e_H(F)$ allow us to get lower bounds on $s(F)$ and $e(H)$.

Lemma 2.11. *If $F \sim F^k_{n,\Delta}$, then almost certainly*

1. $s(F) = \Omega\left(\frac{n}{\Delta^{1/(k-2)}}\right)$, *and*
2. $e(F) = \Omega\left(\frac{n}{\Delta^{2/(k-2)+\epsilon}}\right)$ *for any $\epsilon > 0$.*

The proof of this theorem is based on the fact that a k-uniform hypergraph of density bounded below $2/k$, say $2/k - \epsilon$, has average degree bounded below 2. This implies that a constant fraction of nodes are in the boundary. Fix a set S of vertices/variables of size r. The probability p that a single edge/clause lands in S is at most $(r/n)^k$. Therefore the probability that S contains at least q edges is at most

$$\Pr[B(\Delta n, p) \geq q] \leq \left(\frac{e\Delta np}{q}\right)^q \leq \left(\frac{e\Delta r^{k-1}}{n^{k-1}}\right)^q$$

To get a bound on $s(F)$, we apply this for $q = r + 1$ for all r upto s using union bound:

$$\begin{aligned}\Pr[s(F) \le s] &\le \sum_{r=k}^{s} \binom{n}{r} \left(\frac{e\Delta r^{k-1}}{n^{k-1}}\right)^{r+1} \\ &\le \sum_{r=k}^{s} \left(\frac{ne}{r}\right)^{r} \left(\frac{e\Delta r^{k-1}}{n^{k-1}}\right)^{r+1} \\ &\le \sum_{r=k}^{s} \frac{r}{en} \left(\frac{e^2\Delta r^{k-2}}{n^{k-2}}\right)^{r+1}\end{aligned}$$

This quantity is $o(1)$ in n for $s = O(n/\Delta^{1/(k-2)})$. In a similar way, we get a bound on $e(F)$ by summing the probability for $q = 2r/k$ for all r between $s/2$ and s.

$$\begin{aligned}\Pr[e(F) \le s] &\le \sum_{r=s/2}^{s} \binom{n}{r} \left(\frac{e\Delta r^{k-1}}{n^{k-1}}\right)^{2r/k} \\ &\le \sum_{r=s/2}^{s} \left(\frac{ne}{r}\right)^{r} \left(\frac{e\Delta r^{k-1}}{n^{k-1}}\right)^{2r/k} \\ &\le \sum_{r=s/2}^{s} \left(\frac{e^{1+k/2}\Delta r^{k-1-k/2}}{n^{k-1-k/2}}\right)^{2r/k}\end{aligned}$$

This is $o(1)$ in n for $s = \Theta(n/\Delta^{2/(k-2)})$.

2.3.1.1. *Upper Bound.* We end this section by giving a tight upper bound for Davis-Putnam (DLL) proofs random k-CNF formulas. The simple Davis-Putnam algorithm shown in figure 5 achieves the bound we give.

Refute(F)

1. While F contains a clause of size 1
   (a) Set variable to make that clause TRUE
   (b) Simplify all clauses using this assignment
2. If F has no clauses then
   (a) Output "F is satisfiable" and HALT
3. If F does not contain an empty clause then (Splitting rule)
   (a) Choose smallest-numbered unset variable x
   (b) Run Refute($F|_{x\leftarrow 0}$)
   (c) Run Refute($F|_{x\leftarrow 1}$)

Figure 5. Simple Davis-Putnam Algorithm

Theorem 2.12 ([**BKPS98**]). *For $F \sim F^k_{n,\Delta}$ and Δ above the satisfiability threshold, the simple Davis-Putnam (DLL) algorithm almost certainly finds a refutation of size $2^{O\left(\frac{n}{\Delta^{1/(k-2)}}\right)} n^{O(1)}$.*

The idea of the proof is to look at 2-clauses $(x \vee y)$ as edges $(\bar{x}, y)$ and $(\bar{y}, x)$ in a directed graph with literals of the formula as vertices. The formula is unsatisfiable is there is a contradictory cycle, i.e. one that contains both x and $\bar{(x)}$ for some variable x. It can be shown that after setting $\Omega\left(\frac{n}{\Delta^{1/(k-2)}}\right)$ variables, at least half the variables left are almost certainly in contradictory cycles of the 2-clause digraph. But now a few splitting steps will pick one of these almost surely and setting clauses of size 1 will end the algorithm.

2.3.2. Random Graph k-Colorability

Consider a random graph $G \sim \mathcal{G}_{n,p}$ where each edge occurs independently with probability p. There is a sharp threshold for whether such a graph is k-colorable or not. For example, for $k = 3$, the threshold is around $4.6/n$. The following theorem states that random graphs almost surely require large proofs of non-k-colorability. The basic outline is the same as that for k-CNF. The notion of boundary of a sub-graph is the set of vertices of degree less than k. These are the vertices that can be trivially colored because they have at most $k-1$ neighbors. $s(G)$ in this case is the smallest non-k-colorable subgraph of G.

Theorem 2.13 ([**BCM00**]). *Non-k-colorability almost surely requires exponentially large proofs for random graphs.*

2.4. Nullstellensatz and Polynomial Calculus Lower Bounds

2.4.1. Pigeonhole Principle

$ontoPHP^{m\to n}$ has a natural representation in terms of polynomials. If f is a mapping from pigeons to holes and P_{ij} is a variable saying pigeon i is mapped to hole j, then the following equations ensure that any good assignment to P_{ij}'s is a valid mapping.

f is total: $P_{i1} + P_{i2} + \ldots + P_{in} - 1 = 0$, for $i = 1, \ldots, n$
f is 1-1: $P_{ij}P_{kj} = 0$, for $1 \le i < j \le m, j = 1, \ldots n$
f is onto: $P_{1j} + P_{2j} + \ldots + P_{mj} - 1 = 0$, for $j = 1, \ldots, n$

If $m = n + 1$, we can simply sum up all the total equations and subtract the onto equations to get $0 = 1$. This gives a degree 1 Nullstellensatz proof of $ontoPHP^{m\to m-1}$. In general, we have the following bounds:

Theorem 2.14 ([**BR98**]). *If $m = n + p^k$ and $n > p^{2k}$, then Nullstellensatz proofs of $ontoPHP^{m\to n}$ over $GF(p)$ have degree at least 2^k. For p does not satisfy these conditions, Nullstellensatz proofs of $ontoPHP^{m\to n}$ over $GF(p)$ are of small degree.*

Theorem 2.15 ([**R98**]). *Polynomial Calculus proofs of $PHP^{m\to n}$ (without onto clauses) require degree $n/2$ for any m and any field.*

2.4.2. Counting Principles

Let $Count_2^{2n_1}$ denote the fact that one cannot perfectly match members of an odd size set. More generally, let $Count_r^m$ denote the fact that there is no perfect r-partition of m things if r does not divide m. We will encode $Count_r^m$ as a set of polynomial equations. Let $E = \{1, \ldots, m\}^r$ be the set of all size r subsets of $\{1, \ldots, m\}$. In other words, E forms a complete r-uniform hypergraph over m vertices. For each $e \in E$, we have a variable x_e. Then there are two sets of equations:

1. Every point is covered: $1 - \sum_{e, i\in e} x_e = 0$, for $i = 1, \ldots, m$
2. Edges are disjoint: $x_e x_f = 0$, for all $e \ne f \in E$ s.t. $e \cap f \ne \phi$

Exercise 2.16. Prove that $Count_r^m$ is easy to refute over $\mathbb{Z}_r$.

2.4.3. Tseitin Tautologies

Let $G(V, E)$ be a given lowe degree graph with 0-1 *charges* on its nodes. Further, assume that the total charge on the graph is odd. Then there is no way to put 0-1 weights on the edges of the graph such that the charge on each vertex is the parity of the weights on the edges touching that vertex.

A natural way to represent this is to use $mod2$ equations. however, we will use Fourier basis for this. If we have a variable x over $\{0,1\}$, we form an equivalent variable y over the Fourier basis $\{1,-1\}$ by setting $y=(-1)^x$, or as a linear transform $y=1-2x$. The equation forcing variables to take legal values now becomes $y^2-1=0$ and a contradiction is $1=-1$. This transformation is convenient for expressing parity: $x_1\oplus x_2\oplus\ldots\oplus x_k=0$ becomes $y_1y_2\ldots y_k=1$.

Exercise 2.17. Show that this transformation to Fourier basis, being linear and invertible, preserves degrees of proofs.

For Tseitin formulas in Fourier basis, there is a variable y_e for each $e\in E$. y_e takes values in $\{1,-1\}$ and the constraining equation is $y_e^2=1$. Equation saying parity of edge weights is equal to the charge of the vertex are: $\prod_{e,v\in e} y_e = (-1)^{charge(v)}$ for every $v\in V$. Degree of these polynomial equations equals the degree of the graph. We have the following lower bounds for Tseitin tautologies expressed in this form.

Theorem 2.18. *There is a constant degree graph G such that a Tseitin tautology for G with all charges 1 requires*

1. *degree $\Omega(n)$ to prove in Nullstellensatz* [**?**]
2. *degree $\Omega(n)$ to prove in Polynomial Calculus* [**BGIP99**]

These results use expander graphs, which we define below.

Definition 2.19. Let $G=(V,E)$ be a graph. G has *expansion* ϵ iff every subset S of at most $|V|/2$ vertices has at least $(1+\epsilon)|S|$ neighbors.

Theorem 2.20 ([**?, ?**])**.** *Constant degree regular bipartite graphs with constant expansion $\epsilon>0$ exist.*

Let $E(S)\subseteq E$ be edges of G with one end-point in S and one outside S. Expansion ϵ implies $E(S)\geq\epsilon S>0$ for all sets S of size at most $n/2$. Considering such graphs gets us a degree lower bound of $\epsilon n/8$ for Nullstellensatz and Polynomial Calculus proofs of Tesitin tautologies.

We give a general overview of the proof. Every input equation for Tseitin tautologies has two terms. We can think of the equation as an equivalence of monomials where every monomial corresponds to the parity of a subset of edges. Each equivalence corresponds to the parity of the set of ecdges leaving a small non-empty set of vertices. We initially start with just a single vertex and then use expansion properties of G to increase the size of this set of vertices. Since Fourier basis is essentially equivalent to equations mod 2, we will, for simplicity of reasoning, think of the problem as mod 2 equations.

Given a set S of vertices, let Σ denote the sum of the original edge variables leaving S. Every equation is of the form $\Sigma=|S|$ (mod 2). We start with $S=\{v\}$ and all charges are 1. If we add two equations $\Sigma_S=|S|$ (mod 2) and $\Sigma_{S'}=|S'|$ (mod 2), then combining these two sets gets us $\Sigma_{S\Delta S'}=|S\Delta S'|$ (mod 2), where Δ is the set difference operator. If we always have $|S\Delta S'|\leq n/2$, then $|E(S\Delta S')|>0$. This means there will be an edge going out of $S\Delta S'$ and we will not reach a contradiction. However, if we start with sets of size at most $n/4$, then this won't happen. By expansion property of G, sets of size more than $n/4$ have at least $\epsilon n/4$ edges leaving them. Hence, if one is working with sums of fewer than $\epsilon n/4$ terms, one won't see such sets. Each binomial equation corresponds to a parity summation equation with some portion of the equation in each monomial. If each of these monomial has degree at most $\epsilon n/8$, then we cannot reach a contradiction. Hence the degree of any proof has to be more than $\epsilon n/8$ for graphs with expansion ϵ.

This lower bound for Tseitin tautologies has many implications.

1. We can reduce Tseitin tautologies to $Count_2^{2n+1}$. This implies a $\Omega(n)$ degree lower bound for $Count_2^{2n+1}$ for all fields K with $char(K) \neq 2$[1].
2. We can generalize Tseitin tautologies to $Tseitin(p)$ where we encode in extension fields having p^{th} roots of unity instead of using the Fourier basis. This gives us similar binomial degree lower bounds if $char(K) \neq p$.
3. We can reduce $Tseitin(p)$ to $Count_p^{pn+1}$. This implies $\Omega(n)$ degree lower bound for $Count_p^{pn+1}$ for all fields K with $char(K) \neq p$.

2.5. Polynomial Calculus with Resolution - PCR

As the name suggests, Polynomial Calculus with Resolution combines the power of the two underlying proof systems. For each atomic proposition x, a PCR proof has two variables x and x', where x' stands for $\neg x$. The equations governing values taken by x and x' are: $x + x' - 1 = 0, x^2 - x = 0$ and $(x')^2 - x' = 0$. A clause $(X_1 \vee \neg x_2 \vee x_3)$ for resolution translates as $(1 - x_1)x_2(1 - x_3) = 0$ or equivalently as $x_1' x_2 x_3' = 0$. Proof rules are the same as those for Polynomial Calculus.

Exercise 2.21. Show that PCR simulates resolution with degree = width and no increase in size.

Exercise 2.22. Show how the resolution relationships between size and width apply to PCR using size and degree.

Exercise 2.23. Show that binomial equations work just as in Polynomial Calculus if characteristic of the underlying field is not 2.

We derived resolution lower bounds for random k-CNF formulas using sub-critical expansion $e_H(F)$. Those bounds also translate to systems with polynomial equations.

Lemma 2.24 ([**BI99**])**.** *The degree of any PCR, Polynomial Calculus or Nullstellensatz proof of unsatisfiability of F is at least $e_H(F)/2$ if the characteristic of the underlying field is not 2.*

We first convert a given k-CNF formula into parity equations in a natural way. For example, clause $(x_1 \vee \neg x_2 \vee x_3)$ translates to $x_1 + (x_2 + 1) + x_3 = 1 \pmod 2$, i.e. $x_1 + x_2 + x_3 = 0 \pmod 2$. The goal is to derive the contradiction $0 = 1 \pmod 2$ by adding collections of equations modulo 2. The transformation to Fourier basis is also straightforward. The corresponding connection with sub-critical expansion is that the number of variables in the longest equation is at least $e_H(F)$. This gets us the following lower bounds for random k-CNF formulas for PCR, Polynomial Calculus and Nullstellensatz.

Theorem 2.25. *For random k-CNF formulas chosen from $\mathcal{F}_{n,\Delta}^k$, almost certainly for any $\epsilon > 0$, any PCR, Polynomial Calculus or Nullstellensatz refutation over a field K with $char(K) \neq 2$ requires degree at least $n/\Delta^{2/(k-2)+\epsilon}$ and size at least $2^{c_\epsilon n/\Delta^{4/(k-2)+\epsilon}}$.*

[1] For $char(K) = 2$, the conversion to Fourier basis does not work.

LECTURE 3
Automatizability and Interpolation

We defined a notion of complexity of a proof system, which essentially said how big a proof in that proof system has to be for showing membership in a given language. Lower bounding the size of such a proof was a step towards our big goal of proving NP $\neq$ coNP. This definition, however, didn't say anything about how costly it is to find a short proof in the given proof system. Whereas short proofs might exist, finding them may not be easy.

3.1. Automatizability

Definition 3.1. Given a proof system V for a language L and a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, we say that V is $f(n, X)$-*automatizable* iff there is an algorithm A_V such that given any input x with $|x| = n$, if $x \in L$, then A outputs a proof P in V of this fact in time at most $f(n, S)$, where S is the size of the shortest proof of V of the fact that $x \in L$.

Definition 3.2. We say that V is *automatizable* iff it is $f(n, S)$-automatizable for some function f that is $n^{O(1)}S^{O(1)}$, i.e. it is possibly to find a proof in time polynomial in the size of the smallest one.

Theorem 3.3 ([**BW99**])**.** *Every Davis-Putnam (DLL) or tree-like resolution proof of size S for a CNF formula F can be converted to one of width $\lceil \log_2 S \rceil + \mathrm{width}(F)$.*

Corollary 3.4 ([**CEI96, BP98, BW99**])**.** *Tree-like resolution is $S^{O(\log n)}$-automatizable.*

Proof. There are only $2^{\log S}\binom{n}{\log S} = n^{O(\log S)} = S^{O(\log n)}$ clauses of size at most $\log S$. We can run a breadth-first resolution only deriving clauses of width at most $\log S$. Space requirements can also be kept down by making the search recursive. □

Theorem 3.5 ([**BW99**])**.** *Every resolution proof of size S for a CNF formula F can be converted to one of width $O(\sqrt{n \log S}) + \mathrm{width}(F)$.*

Corollary 3.6. *General resolution is $2^{O(\sqrt{n \log S} \log n)}$-automatizable.*

Theorem 3.7. *Tree-PCR and PCR are $S^{O(\log n)}$-automatizable and $2^{O(\sqrt{n \log S} \log n)}$-automatizable, respectively.*

3.2. Interpolation

Let $A(x, z)$ denote a formula over variables x and z, and let $B(y, z)$ denote one over y and z.

Definition 3.8. If $A(x,z) \vee B(y,z)$ is a tautology then an *interpolant* C is a function such that for any truth assignment α to z,

1. $C(\alpha) = 0$ implies $A(x,\alpha)$ is a tautology, and
2. $C(\alpha) = 1$ implies $B(y,\alpha)$ is a tautology.

The origin of the term interpolant for such a function can be understood by looking at the following property of interpolants:

Theorem 3.9 (Craig??)**.** *If $A(x,z) \rightarrow B(y,z)$ is a tautology then there is an interpolant C with only free variables z such that $A(x,z) \rightarrow C(z)$ and $C(z) \rightarrow B(y,z)$.*

We can also give a dual definition of an interpolant for the case when $A(x,z) \wedge B(y,z)$ is known to be unsatisfiable. Given any assignment α to variables z, the interpolant says which one of $A(x,\alpha)$ and $B(y,\alpha)$ is unsatisfiable.

Definition 3.10. If $A(x,z) \wedge B(y,z)$ is unsatisfiable then an *interpolant* C is a function such that for any truth assignment α to z,

1. $C(\alpha) = 0$ implies $A(x,\alpha)$ is unsatisfiable, and
2. $C(\alpha) = 1$ implies $B(y,\alpha)$ is unsatisfiable.

Definition 3.11. Given a propositional proof system V and a function $f : \mathbb{N} \rightarrow \mathbb{N}$, we say that V has *f-interpolation* iff given an unsatisfiable formula of the form $A(x,z) \wedge B(y,z)$ with proof size S in V, there is a circuit of size at most $f(S)$ computing an interpolant C for $A(x,z) \wedge B(y,z)$.

Such a V is said to have *feasible interpolation* iff f is polynomial. We say that V has *monotone f-interpolation* iff whenever the variables z occur only negatively in B and only positively in A, the circuit C is a monotone circuit.

Lemma 3.12 ([**BPR97**])**.** *If V is automatizable then V has feasible interpolation.*

Proof. Let f be the polynomial function such that V is f-automatizable and let A_V be the associated algorithm. Given an unsatisfiable formula $A(x,z) \wedge B(y,z)$ and an assignment α to z, run A_V on input $A(x,z) \wedge B(y,z)$ to get a proof P of size $S' \leq f(S)$, where S is the size of its optimal proof in V. Now run A_V on input $A(x,\alpha)$ for $f(S')$ steps. If it finds a proof, set $C(\alpha) = 0$. Otherwise set $C(\alpha) = 1$. The key thing to note here is that if $B(y,\alpha)$ has a satisfying assignment β, then plugging β, α into the proof P yields a proof of size S' of unsatisfiability of $A(x,\alpha) \wedge B(\beta,\alpha) = A(x,\alpha)$. □

Theorem 3.13 (Krajicek)**.** *Resolution has feasible (monotone) interpolation.*

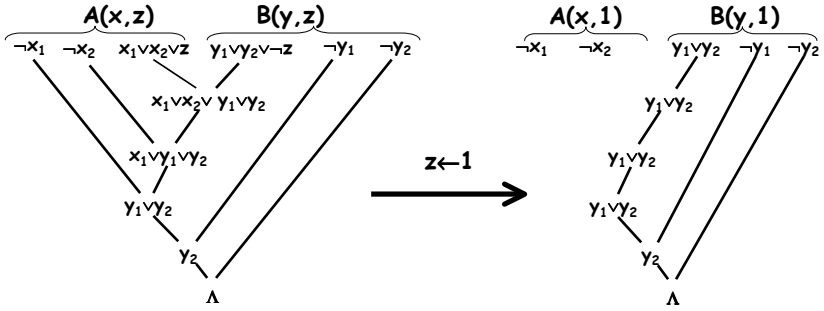

Figure 1. Construction interpolant for resolution

The key idea in the proof of this theorem is that the structure of a resolution proof allows one to easily decide which clauses cause unsatisfiability under a particular assignment. Consider the refutation tree given in the left portion of figure 1. We simply go through each possible assignment α to the variables z (in this case only a single variable) and restrict the proof accordingly to determine which of $A(x,\alpha)$ and $B(y,\alpha)$ is false. For instance, setting $z = 1$ in our example simplifies the clause $(y_1 \vee y_2 \vee \neg z)$ to $(y_1 \vee y_2)$. The clause $(x_1 \vee x_2 \vee z)$ similarly gets simplified to 1. Since the original clause $(x_1 \vee x_2 \vee y_1 \vee y_2)$ derived from these two clauses now contains variables that appear in neither of its parents, it can be simplified to $(y_1 \vee y_2)$ and we no longer need the clause (1) to derive it. This simplification goes on until we finally get to the tree on the right hand side of figure 1, which gives us a refutation of $B(y,1)$ and says that $B(y,z)$ was the reason the original formula became unsatisfiable when we set $z = 1$. Doing this for each assignment to z gives us a polynomial time way of determining the interpolant completely.

Theorem 3.14 ([**P97**])**.** *Cutting planes has feasible (monotone) interpolation where the interpolants are circuits over the real numbers.*

Theorem 3.15. *Polynomial calculus has feasible interpolation.*

3.3. Lower Bounds using Interpolation

If we are given a class of circuits for which we know lower bounds and a proof system whose interpolants are in that circuit class, then we can build a formula whose interpolant will be a circuit for a hard problem in the circuit class.

Theorem 3.16. *If a proof system V has feasible interpolation and NP $\not\subset$ P/poly, then V is not polynomially bounded.*

Proof. (Sketch) Suppose V does have feasible interpolation and is also polynomially bounded with bound p. Consider a formula $A(x,z) \wedge B(y,z)$ where z represents a CNF formula, $A(x,z)$ says that assignment x satisfies z, and $B(y,z)$ says that y, of length $p(|x|)$, is a proof in V that z is unsatisfiable. Feasible interpolation for this formula corresponds to a polynomial size circuit that, for each CNF formula z, tells us which of $A(x,z)$ and $B(y,z)$ is unsatisfiable. In other words, it is a polysize circuit for deciding satisfiability, implying NP $\subset$ P/poly (the inequality is strict because P/poly is known to contain languages that are not in NP).

The way we have stated this proof, it is not clear how one could efficiently encode $z, A(x,z)$ and $B(y,z)$. As an example, suppose we restrict z to represent clique-coloring formulas. For a given graph G over n variables,

1. z contains the $\frac{n(n-1)}{2}$ variables z_{uv} representing the existence of the corresponding edges.
2. $A(x,z)$ is the statement that $G = G(z)$ has a k-clique. The variables x_{iv} are TRUE iff vertex v of G is the i^{th} node of the k-clique. Clauses $(\bigvee_v x_{iv})$ say that some vertex is chosen as the i^{th} vertex of the k-clique. Clauses $(\neg x_{iv} \vee \neg x_{ju} \vee z_{uv})$ say that both u and v are not chosen in the k-clique if there is an edge connecting them. Clauses $(\neg x_{iv} \vee \neg x_{jv})$ say that no vertex is counted twice in the clique. Clauses $(\neg x_{iu} \vee \neg x_{iv})$ say that we don't waste vertices.
3. $B(y,z)$ is the statement that $G(z)$ is $(k-1)$-colorable. The variables y_{iv} are TRUE iff vertex v is given the i^{th} color in some fixed valid $(k-1)$-coloring of $G(z)$. Clauses $(\bigvee_i y_{iv})$ say that each vertex gets a color. Clauses $(\neg z_{uv} \vee \neg y_{ui} \vee \neg y_{vi})$

say that two vertices that have an edge between them do not get the same color. Clauses $(\neg y_{vi} \vee y_{vj})$ say that a vertex is given only one color.

□

Theorem 3.17. *Any proof system V that has monotone feasible interpolation is not polynomially bounded.*

Theorem 3.18. *Any cutting planes proofs of clique-coloring formulas are exponential.*

Proof. Follows from the result of Pudlak we saw earlier saying that cutting planes has feasible monotone interpolation. □

3.4. Limitations

Under widely believed assumptions, sufficiently powerful proof systems do not have feasible interpolation and our technique for proving lower bounds using interpolation becomes useless for such systems.

Theorem 3.19 ([**KP89**])**.** *If one-way functions exist, then Frege systems do not have feasible interpolation.*

Proof. (Idea) Suppose one has a method of *key agreement*, i.e. given two people, one with x and one with y, they can exchange messages and agree on a secret key $key(x, y)$ so that even listening to their conversation without knowing x or y, it is hard to figure out what even a single bit of $key(x, y)$ is. Such methods exist if one-way functions do. Going to the interpolation setting,

1. our common varaibles z will represent the transcript of their conversation,
2. $A(x, z)$ will say that the player with x correctly computed its side of the conversation and the last bit of $key(x, y)$ is o, and
3. $B(y, z)$ will say that the player with y correctly computed its side of the conversation and the last bit of $key(x, y)$ is 1.

We must encode the computation of each player in such a way that the proof system (Frege in this case), given x and z, can prove what the value of the bit is. We can make the task easier by extending x with helper extension variables. The actual proof uses Diffie-Hellman secret key exchange which is as hard as factoring. It requires powering which is not in TC^0. However, the extension variables make it easy enough to prove. □

Theorem 3.20 ([**BPR97**])**.** *If factoring Blum integers is hard, then any proof system that can polynomially simulate TC^0-Frege, or even AC^0-Frege, does not have feasible interpolation.*

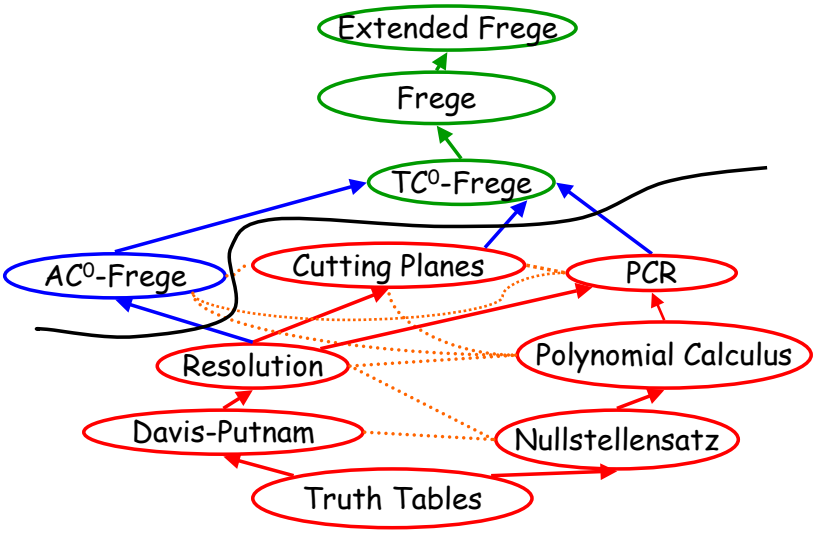

Figure 2. The interpolation line

LECTURE 4
The Restriction Method

The restriction method is a very useful tool for proving lower bounds in both circuit complexity and proof complexity. We will motivate this lecture by a result of Hastad whose original proof used the restriction argument.

Theorem 4.1 ([**H86**])**.** *The n-bit parity function $x_1 \oplus x_2 \oplus \ldots \oplus x_n$ cannot be computed by unbounded fan-in circuits of size S and depth d unless $S \geq 2^{cn^{1/d}}$.*

Corollary 4.2. *Polynomial-size circuits for parity require $\Omega(\log n / \log\log n)$ depth. In particular, Parity $\notin AC^0$.*

Definition 4.3. Given a set X of Boolean variables, a *restriction* ρ is a partial assignment of values to the variables of X, i.e. $\rho : X \to \{0, 1, *\}$ where $\rho(x_i) = *$ indicates that the variable x_i is not assigned any value by this restriction.

If F is a function, formula or circuit, we write $F|_\rho$ for the result of substituting $\rho(x_i)$ for each x_i such that $\rho(x_i) \neq *$.

In what follows, we will allow circuits to have unbounded fan-in but restrict connectives to $\vee$ and $\neg$. The *depth* of a formula F (circuit C) is then defined as the maximum number of $\vee$'s on any path from an input to an output. Formulas or circuits in standard CNF or DNF form, for instance, have depth 2.

Restrictions simplify functions, circuits or formulas that we have. Given $F = (\bigvee_i x_i \vee \bigvee_j \neg x_j)$, a single assignment $\rho(x_i) = 1$ or $\rho(x_j) = 0$ makes $F|_\rho$ a constant. Thus the simplification we obtain by restricting a small set of variables is typically substantially more than the number of variables we set. To prove a lower bound saying small circuits C cannot compute a complex function f, we demonstrate a restriction ρ such that $f|_\rho$ is still complicated but $C|_\rho$ is so simple that it obviously cannot compute $f|_\rho$.

4.1. Decision Trees

We begin by introducing the concept of decision trees that we will associate with each gate of a circuit or each formula appearing in a proof when using the restriction method.

Definition 4.4. A *Boolean decision tree* T is a binary rooted tree such that

1. Each internal node is labelled by some x_i
2. Leaf nodes are labelled 0 or 1
3. Edges out of each internal node are labelled 0 or 1
4. No two nodes on a path have the same variable label

From now on, we will write decision tree to actually mean a Boolean decision tree. It is easy to see that every root to leaf path (or branch) of a decision tree corresponds to a restriction ρ of the input variables. More precisely, for $b \in \{0,1\}, x \leftarrow b$ is in ρ iff on that branch, the out-edge labelled b is taken from the node in the branch labelled x_i.

Definition 4.5. A decision tree T computes a function f iff for every branch B of T, the restriction ρ corresponding to branch B has the property that $f|_\rho$ equals the leaf label of B.

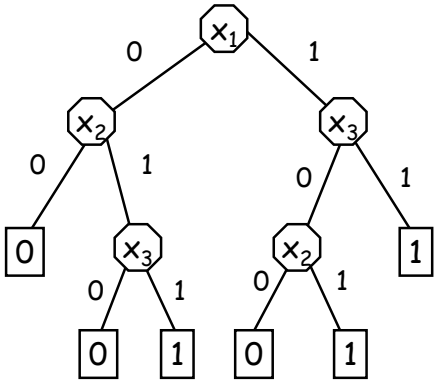

Figure 1. Decision tree for $x_1 + x_2 + x_3 \geq 2$

A Boolean decision tree computing the function $f(x)$ which is 1 if $x_1 + x_2 + x_3 \geq 2$ and 0 otherwise is shown in figure 1.

Decision trees give a natural way of describing the function they compute as a CNF or DNF formula. Suppose we have a decision tree of height t computing a function f. Then f can be described in CNF form with clause size at most t by associating a clause with each branch with leaf label 0. In a similar fashion, f can also be expressed as a DNF formula with term size at most t by associating a term with each branch with leaf label 1.

In the other direction, there is a canonical conversion from any DNF formula to a decision tree computing the same function. We describe this conversion with an example, $F = x_1\bar{x}_3 \vee x_3x_4 \vee \bar{x}_4x_6$. We first create an unlabelled root node. At any stage of the algorithm, we pick the deepest and leftmost unlabelled leaf node (which would for now be the root node). We now select the first term of F from the left that is not falsified by the assignments in the path from the root to this unlabelled node (in this case, $x_1\bar{x}_3$). If there is no such term, the node is labelled 0 and we continue looking for another unlabelled node. Otherwise, if this term has t variables that have not appeared yet in the path from the root to this node (here $t = 2$), we generate a complete binary tree on the t variables appearing in this term and make it a subtree of the current node. The leaf that corresponds to the term is labelled 1 and we continue searching for the next unlabelled leaf node. The tree created thus for our example is shown in figure 2.

4.2. Restriction Method in Circuit Complexity

We will call an unbounded fan-in circuit of size at most S and depth at most d an (S,d)-circuit. For functions f, we will be interested in lower bounds saying that no (S,d)-circuit computes f. The key idea will be to find a set $R_{S,d}(f)$ of restrictions such that

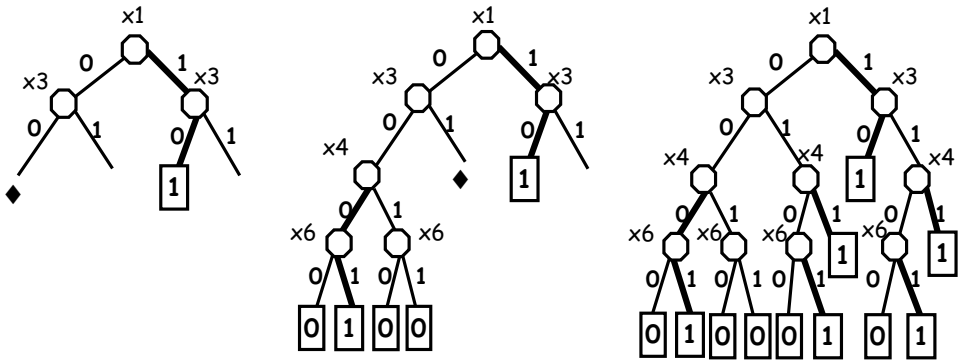

Figure 2. Generating canonical decision tree from DNF formula: $x_1\bar{x}_3 \vee x_3x_4 \vee \bar{x}_4x_6$

1. For any (S, d)-circuit C, there is a restriction $\rho \in R_{S,d}(f)$ for which we can associate a short Boolean decision tree $T(g)$ with each gate g of C such that $T(g)$ computes $g|_\rho$.
2. For any $\rho \in R_{S,d}(f)$, $f|_\rho$ cannot be computed by any short decision tree.

Here, by "short" we mean short relative to the number of variables unset by ρ.

Let us try to find such a set of restrictions for the parity function. We will first note a few important properties of parity. For any restriction ρ, Parity$|\rho$ is either parity or its negation on the variables that are still not assigned a value. Moreover, one of parity and its negation must require a decision tree of height at least n. Comparing this with OR of n bits, any decision tree for OR also requires height n but most restrictions of it are constant and therefore only need height 0. Parity, in this sense, is a more complex function and better suited for getting a good lower bound.

To find restrictions for parity, we start at the inputs of the circuit and work upwards one layer at a time. As we go along, we maintain a current restriction ρ_i and a decision tree $T_i(g)$ for each gate g in the first i layers such that $T_i(g)$ computes $g|\rho$.

For layer 0, the gates are input variables, ρ_0 is empty and all decision trees have height 1. As we move up from layer $i-1$ to layer i, any new gate h is either a negation or an OR. If $h = \neg g$, we let $T_i(h)$ be $T_i(g)$ with the labels on its leaves flipped from 0 to 1 and vice versa. The case when $h = (g_1 \vee \ldots \vee g_t)$ is more complex. It might happen that $h|\rho_i$ requires tall decision trees even if all $T_i(g_j)$ are short. We therefore look for a further small restriction π to the inputs in the hope if simplifying $h|\rho_i$ so that we might get a shorter tree. We would like to choose *one* π that simultaneously does this for *all* (possibly S) unbounded fan-in OR's in the i^{th} layer.

Let's postpone the details of how we might find such a π and first see what we would do if we did have one. We will set $\rho_{i+1} = \rho_i\pi$. By our assumed properties of π, short $T_{i+1}(h)$ exists for gates h in this layer. For all gates g below this layer, we will set $T_{i+1}(g) = T_i(g)|_\pi$. We now continue upward in normal fashion and end by setting $\rho = \rho_d$ for the depth d circuit. Since we had been choosing π's which guaranteed short trees, the tree we end up with will be shorter than the number of inputs that ρ leaves unset. By our earlier observation about parity, such a decision tree cannot compute parity correctly and this gets us our lower bound.

All that remains now is to get the restrictions π. We won't give a way of finding such a π but only show that one exists using the standard probabilistic method. Instead of going into the exactly details, we here provide a sketch of how the proof works. We show that

a randomly chosen small π fails to shorten the decision tree for any single OR gate h in a given layer with probability less than $1/S$. Since there are at most S OR gates in any layer, the probability that there exists an OR gate in this layer which is not shortened by π is strictly less than 1, which implies that there must exist a small π that works.

The rest of the argument relies on the following result of Hastad:

Lemma 4.6 (Hastad's Switching Lemma)**.** *Let f be a DNF formula in variables $x_1, \ldots, x_n$ with terms of size at most t. Let $R_{k,n}$ be the set of all restrictions to variables $x_1, \ldots, x_n$ that leave precisely k variables unset. For π chosen uniformly at random from $R_{k,n}$, if $n > 12tk$, then the probability that the canonical decision tree for $F|_\pi$ has height at least t is less than 2^{-t}.*

Given this lemma, we will maintain trees of height $t = \log_2 S$. The number of variables then decreases by a factor of $13t = 13\log_2 S$ per layer. The height of the tree will therefore be less than the number of variables if $\log_2 S < n/(13\log_2 S)^d$, or if $\log_2 S < n^{1/(d+1)}/13$. If this happens, our circuit cannot compute parity. We note here that by being careful in the analysis, we can save one power of $\log_2 S$.

4.3. Restriction Method in Proof Complexity

In circuit complexity, for each gate g of a given circuit, we defined decision trees $T(g)$ that precisely computed each $g|\rho$ in the circuit. The obvious analog for proof complexity would be to define a decision tree for each formula that appears in the proof. However, this cannot possibly work because every formula in the proof is a tautology and hence computes the constant function 1.

We get around this problem by using a different notion of decision trees that *approximates* each formula so that

1. The bigger the proof needed for tautology, the worse approximation we get.
2. Decision trees are well-behaved under restrictions.
3. Approximation is particularly bad for the goal formula F. In fact, we try to show that any short approximating decision tree for F looks like FALSE, one for an axiom looks like TRUE, and one for any formula with a short proof looks like TRUE.

As in the circuit complexity case, we define these decision trees for each subformula in the proof and tailor decision trees and restrictions to F. Before we go on to describe this in detail for the bipartite matching case, we mention some of the main results derived using this method.

Theorem 4.7 ([**A94, PBI93, KPW91**])**.** *$ontoPHP^{n+1\to n}$ requires exponential size AC^0-Frege proofs.*

Theorem 4.8 ([**A94, BP93**])**.** *$Count_2^{2n+1}$ requires exponential size AC^0-Frege proofs even given $PHP^{m+1\to m}$ as extra axiom schemas.*

Theorem 4.9 ([**BIKPP94**])**.** *$Count_p^{pn+1}$ requires exponential size proofs even given $Count_q^{qm+1}$ as axiom schemas for $q \neq p$.*

4.3.1. Matching Decision Trees

Consider again the pigeonhole principle $PHP^{n+1\to n}$ from $n+1$ pigeons to n holes. Let P_{ij} be a variable which is TRUE iff pigeon i is mapped to hole j. As before, restrictions here are partial matchings. Let $R^{k,n}$ be the set of all partial matching restrictions that leave exactly k holes unset. We will construct a bipartite matching decision tree where

queries are either the name of a pigeon, in which case the answer is the mapping edge for that pigeon, or the name of a hole, in which case the answer is the mapping edge for that hole. We do not repeat any name that was already used higher in the tree and every path corresponds to a partial matching between pigeons and holes. The leaves are labelled 0 or 1, depending on whether.

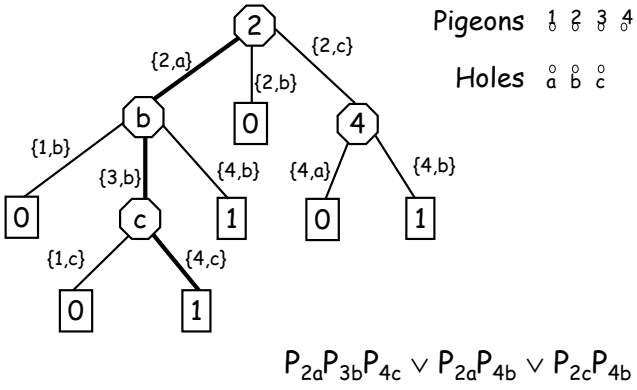

Figure 3. A matching decision tree with path for $P_{2a}\,P_{3b}\,P_{4c}$ highlighted

Given a refutation of $PHP^{n+1\to n}$, we associate a matching decision tree with each formula of the proof as follows.

1. $T(P_{ij})$ is the tree that queries i and has height 1
2. $T(\neg g)$ is $T(g)$ with leaf labels toggled
3. To get the tree for $h = (g_1 \vee \ldots \vee g_t)$
   (a) Take DNF formula $F_h = T(G_1) \vee \ldots \vee T(g_t)$
   (b) Do canonical conversion of F_h into a matching decision tree.

The canonical conversion into a matching decision tree is essentially the same as canonical conversion for ordinary decision trees. We go term by term left to right simplifying future terms based on partial assignments. For each term, we query both endpoints of every variable in that term.

4.3.2. Ideas for $PHP^{n+1\to n}$ Lower Bound

The lower bound for $PHP^{n+1\to n}$ we stated in the previous section is proved using the restriction method. There is an anolog of Hastad's switching lemma for canonical conversion of DNF formulas to matching decision trees. If one has a small proof of the pigeonhole principle, the corresponding trees can bbe made short. A matching decision tree of height less than n has all 0's on its leaves for $PHP^{n+1\to n}$, has all 1's on its leaves for an axiom, and preserves this property of all 1's on the leaves underinference rules.

One can add extra axioms and get the same sorts of restrictions and matching decision trees. To be able to use these extra axioms, one must also prove that they convert to trees with all 1's on their leaves. Surprisingly, this follows from Nullstellensatz degree lower bounds for the extra axioms.

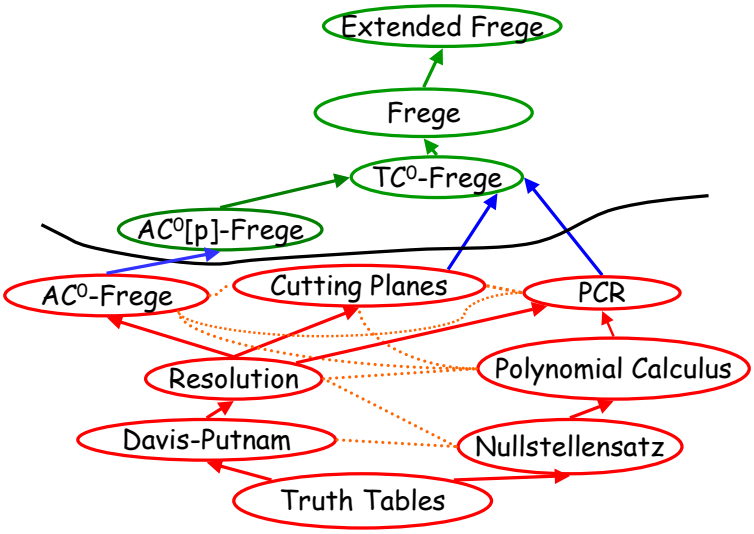

Figure 4. The frontier of proof system relationships

LECTURE 5
Open Problems

Random Formulas: Random formulas have been shown to be hard for resolution. An open problem is to show they are hard even for cutting planes and for depth 2 Frege systems. The problem with the latter is that for AC^0-Frege, all we know is the restriction method and restriction families seem to almost certainly falsify random formulas. The big conjecture, though, is that random formulas are hard for Frege systems.

Weak Pigeonhole Principle: For $PHP^{m \to n}$, resolution lower bounds are non-trivial only when $m < n^2/\log n$. What happens when $m \gg n$, e.g. $m = 2^{n^c}$? A lower bound in this direction would have applications to bounded arithmetic (existence of infinitely many primes) and provability of NP $\not\subset$ P/poly. In the other direction, it is known that $PHP^{m \to n}$ has quasi-polynomial size depth 2 Frege proofs for $m \geq (1+\epsilon)n$.

Lovasz-Schriver Proof Systems: These systems are like cutting planes but based on 01-programming. Initial inequalities and goals are like those in cutting planes. In addition, one can substitute x for x^2 anywhere. The division rule, however, is not present. One can create non-negative degree two polynomials by multiplying two non-negative linear quantities or squaring any linear quantity. This system polynomially simulates resolution and can therefore prove PHP. It has feasible interpolation and hence is not polynomially bounded given NP $\not\subset$ P/poly. However, no hard tautology is known for it. One might try to prove $Count_2^{2n+1}$ is hard for these systems.

The Bigger Questions:

- Prove lower bounds for AC^0-Frege, e.g. $Count_q^{qn+1}$ is hard.
- Prove lower bounds for TC^0-Frege or Frege in general. A candidate for this could be $AB = I \Rightarrow BA = I$ for Boolean matrix multiplication.

Proof Search for PCR: Can we build better algorithms to beat the Davis-Putnam/DLL algorithms in practice by using some PCR ideas?

The reader is referred to [**BP98**] for a list of more open problems.

BIBLIOGRAPHY

[A88] Ajtai, M., *The complexity of the pigeonhole principle*, Proceedings of the Twenty-Ninth Annual IEEE Symposium on Foundations of Computer Science, pages 346–355, White Plains, NY, Oct 1988.

[A94] Ajtai, M., *The independence of the modulo p counting principles*, Proceedings of Twenty-Sixth Annual ACM Symposium on Theory of Computing, pages 402–411, Montréal, Québec, May 1994.

[BCM00] Beame, P. and Cullberson, J. and Mitchell D., *The resolution complexity of random graph k-colorability*.

[BGIP99] Buss, S. and Grigoriev, D. and Impagliazzo, R. and Pitassi, T., *Linear gaps between degrees for the polynomial calculus modulo distinct primes*, Proceedings of Thirty-First Annual ACM Symposium on Theory of Computing, pages 547-556, Atlanta, GA, May 1999.

[BI99] Ben-Sasson, E. and Impagliazzo, R., *Random CNF's are hard for the polynomial calculus*, Proceedings of Fortieth Annual IEEE Symposium on Foundations of Computer Science, pages 415–421, New York, NY, Oct 1999.

[BIKPP94] Beame, P. and Impagliazzo, R. and Krajíček, J. and Pitassi, T. and Pudlák, P., *Lower bounds on Hilbert's Nullstellensatz and propositional proofs*, Proceedings of Thirty-Fifth Annual IEEE Symposium on Foundations of Computer Science, pages 794–806, Santa Fe, NM, Nov 1994.

[BKPS98] Beame, P. and Karp, R. and Pitassi, T. and Saks, M., *On the complexity of unsatisfiability of random k-CNF formulas*, Proceedings of Thirtieth Annual ACM Symposium on Theory of Computing, pages 561–571, Dallas, TX, May 1998.

[BP93] Beame, P. and Pitassi, T., *An Exponential Separation between the Matching Principle and the Pigeonhole Principle*, Proceedings of the Annual IEEE Symposium on Logic in Computer Science, pages 308–319, Montréal, Québec, Jun 1993.

[BP98] Beame, W. and Pitassi, T., *Propositional Proof Complexity: Past, Present, Future*, Bulletin of the European Association for Theoretical Computer Science, volume 65, pages 66–89, June 1998.

[BPR97] Bonet, M. and Pitassi, T. and Raz, R., *No Feasible Interpolation for TC^0 Frege Proofs*, Proceedings of Thirty-Eighth Annual IEEE Symposium on Foundations of Computer Science, pages 254–263, Miami Beach, Florida, Oct 1997.

[BR98] Beame, P. and Riis, S., *More on relative strength of counting principles*, Proof Complexity of Feasible Arithmetic, Editors Beame, P. and Buss, S., volume 39 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 13–35, American Mathematical Society, 1998.

[BW99] Ben-Sasson, E. and Wigderson, A., *Short proofs are narrow – resolution made simple*, Proceedings of Thirty-First Annual ACM Symposium on Theory of Computing, pages 517–526, Atlanta, GA, May 1999.

[C73] Chvátal, V., *Edmonds polytopes and a hierarchy of combinatorial problems*, Journal of Discrete Mathematics, volume 4, pages 305–337, 1973.

[CEI96] Clegg, M. and Edmonds, J. and Impagliazzo, R., *Using the Gröebner basis algorithm to find proofs of unsatisfiability*, Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pages 174–183, Philadelphia, PA, May 1996.

[CR77] Cook, Stephen A. and Reckhow, Robert A., *The relative efficiency of propositional proof systems*, Journal of Symbolic Logic, volume 44, number 1, pages 36–50, 1977.

[CS88] Chvátal, V. and Szemerédi, Endre, *Many Hard Examples for Resolution*, Journal of the ACM, volume 35, number 4, pages 759–768, 1988.

[G58] Gomory, G. B., *Outline of an Algorithm for Integer Solutions to Linear Programs*, Bulletin of the American Mathematical Society, volume 64, pages 275–278, 1958.

[H86] Håstad, J., *Almost Optimal Lower Bounds for Small Depth Circuits*, Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, pages 6–20, Berkeley, CA, May 1986.

[IPS99] Impagliazzo, R. and Pudlák, P. and Sgall, J., *Lower bounds for the polynomial calculus and the Gröebner basis algorithm*, Journal of Computational Complexity, volume 8, number 2, pages 127–144, 1999.

[KP89] Krajíček, J. and Pudlák, P., *Propositional proof systems, the consistency of first order theories and the complexity of computations*, Journal of Symbolic Logic, volume 54, number 3, pages 1063–1079, 1989.

[KPW91] Krajíček, J. and Pudlák, P. and Woods, A., *Exponential Lower Bounds to the size of bounded depth Frege proofs of the pigeonhole principle*, Random Structures and Algorithms, volume 7, number 1, 1995.

[PBI93] Pitassi, T. and Beame, P. and Impagliazzo, R., *Exponential Lower Bounds for the Pigeonhole Principle*, Journal of Computational Complexity, volume 3, number 2, pages 97–140, 1993.

[P97] Pudlák, P., *Lower bounds for resolution and cutting plane proofs and monotone computations*, Journal of Symbolic Logic, volume 62, number 3, pages 981–998, Sep 1997.

[R98] Razborov, A. A., *Lower Bounds for Polynomial Calculus*, Journal of Computational Complexity, volume 7, number 4, pages 291–324, 1998.

[RR97] Razborov, A. A. and Rudich, S., *Natural Proofs*, Journal of Computer and System Sciences, volume 55, number 1, pages 24–35, August 1997.

[SML96] Selman, B. and Mitchell, D. and Levesque, H., *Generating Hard satisfiability problems*, Journal of Artificial Intelligence, volume 81, pages 17–29, 1996.

[T68] Tseitin, G. S., *On the complexity of derivation in the propositional calculus*, Studies in Constructive Mathematics and Mathematical Logic, Part II, Editor Slisenko, A. O., 1968.

[U95] Urquhart, A., *The complexity of propositional proofs*, Bulletin of Symbolic Logic, volume 1, number 4, pages 425–467, Dec 1995.