Proof Complexity

Lecturer: Paul Beame
Notes: Ashish Sabharwal

Contents

IAS/Park City Mathematics Series
Volume ?, 2000

Proof Complexity

Lecturer: Paul Beame
Notes: Ashish Sabharwal

LECTURE 1
An Introduction to Proof Complexity

One of the threads that led to our present notions of computational complexity was research on automated theorem-proving. This influence is evident in the title of Cook's famous paper introducing NP-completeness, "The complexity of theorem-proving procedures" [**28**]. The complexity class NP is characterized by the following simple property: A language L is in NP iff all strings in L have a short, polynomial-time-checkable proof of membership in L. This notion of proof at first seems a little foreign when compared with the familiar notions of proof in mathematical logic but, as the NP-completeness of the satisfiability problem shows, they are intimately connected.

Although formal research on the complexity of proofs begins earlier, at least with work of Tseitin in 1968 [**55**], it was Cook who in the 1970's suggested that proof complexity bounds would give us answers to complexity questions such as separating P from NP. In particular, if one can find a polynomial-size family of tautologies that does *not* have polynomial-size proofs then this will separate NP from $\mathsf{co\text{-}NP}$ and thus separate P from NP. While doing this is certainly quite hard, the theory of proof complexity breaks this problem into smaller, more tractible problems of proving such lower bounds for specific proof systems, that can be seen as steps towards this goal. Part of the attraction of this approach is that these concrete steps are very interesting in their own right and have useful implications for automated theorem-proving.

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

[1]Department of Computer Science, University of Washington, Box 352350, Seattle WA 98195-2350.
E-mail address: `beame@cs.washington.edu`.

of unsatisfiability could be. Some possible proofs are transcript of a failed search for satisfying truth assignment, truth tables, deductive proofs using proof rules (sometimes called Frege-Hilbert proofs), resolution proofs. The question is, can one always make sure that short proofs exist using one of these types? If yes, then $\mathsf{NP} = \mathsf{co}\text{-}\mathsf{NP}$. This leads 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* for proofs of this form. The complexity of a proof system is a measure of how large $|P|$ has to be as a function of $|x|$.[1]

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 defined 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 ([**29**])**.** *There is a polynomially-bounded propositional proof system iff* $\mathsf{NP} = \mathsf{co}\text{-}\mathsf{NP}$.

Proof. We know that SAT is NP-complete. For any formula F,

$$F \in TAUT \Leftrightarrow \neg F \in UNSAT \Leftrightarrow \neg F \notin SAT.$$

It follows that both $TAUT$ and $UNSAT$ are $\mathsf{co}\text{-}\mathsf{NP}$-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 developed a large number of proof systems. Given any two such systems, it is useful to have a way of saying that one is as strong as the other. One natural notion for this is to say one proof system at least as powerful as a second proof system if the former can "simulate" the latter efficiently. (We will see another, possibly weaker, notion later.)

---

[1]Originally Cook and Reckhow [**29**] defined a proof system for a non-empty language L as a polynomial-time-computable function $f : \Sigma^* \overset{onto}{\to} L$, i.e. $f(\Sigma^*) = L$. Observe that this is essentially equivalent to our definition: Given such an f, define a verifier V that on input (x, P) accepts iff $f(P) = x$. Conversely, given a verifier V, let x_0 be any fixed element of L, define $f(\langle x, P\rangle) = \begin{cases} x & \text{if } V \text{ accepts } (x, P) \\ x_0 & \text{otherwise} \end{cases}$. Our definition also covers the case when $L = \emptyset$.

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. Axiom/Inference Systems: Frege Proofs

These systems have a set of axioms such as *excluded middle* which says $\mid (A \vee \neg A)$, 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 $A, (A \rightarrow B) \mid B$, meaning for any formulas A and B, one can derive B if A and $(A \rightarrow B)$ are already present. (Axioms schemas are merely inference rules with no antecedents.)

Each inference rule like *modus ponens* is specified by a sequence of formulas with a fixed set of variables, the last separated from the other by a $\mid$. A formula H follows from formulas $G_1, \ldots, G_k$ using an inference rule σ if there is a consistent substitution of formulas for the variables in σ so that the resulting formulas to the left of the $\mid$ in σ are chosen from $G_1, \ldots, G_k$ and to the right of the $\mid$ in σ is H.

More precisely, a Frege system starts with a finite, *sound* and *implicationally complete* set R of axioms and inference rules each of the form $F_1, F_2, \ldots, F_k | F$ for some $k \geq 0$ where $F_1, \ldots, F_k, F$ are formulas. Axioms correspond to the case $k = 0$. Implicational completeness means that the axioms are sufficient to derive any formula that follows logically from a sequence of formulas. Soundness means that the conclusion of the inference rule logically follows from the premises.

To show that a formula is a tautology we start with the axioms and derive the formula using the inference rules; to show that a formula is unsatisfiable, we start with the formula and keep applying these axioms and inference rules to finally derive FALSE. We concentrate on the latter form to be more in keeping with the other proof systems we will discuss.

More formally, a *Frege refutation* 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. $x \wedge \neg x$.

The size of a Frege proof is the number of symbols in the proof. It is easy to see that a Frege refutation of $\neg F$ can be efficiently converted to a Frege proof of F and vice versa.

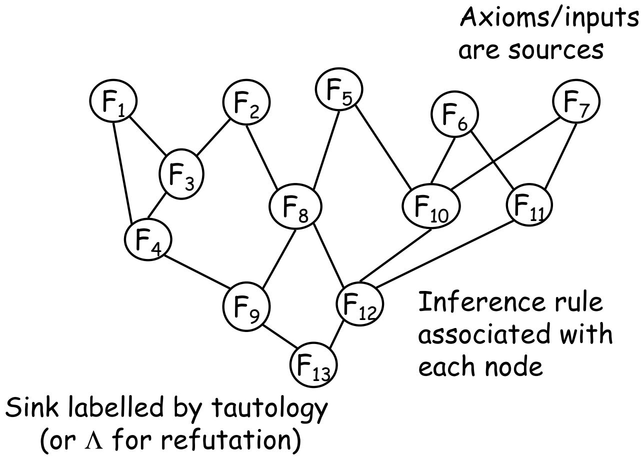

Figure 1. The graph of a Frege proof

We can associate a directed acyclic graph with a Frege refutation in a natural way 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.

Subset of rules
a. **A, (A $\rightarrow$ B) | B**
b. **(A $\wedge$ B) | A**
c. **(A $\wedge$ B) | B**
d. **A, B | (A $\wedge$ B)**

1.	$((x\wedge(x\rightarrow y))\wedge((x \wedge y)\rightarrow\neg x))$	Given
2.	$(x \wedge(x \rightarrow y))$	From 1 by b
3.	$((x \wedge y) \rightarrow \neg x)$	From 1 by c
4.	x	From 2 by b
5.	$(x \rightarrow y)$	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

It seems that there is an infinite variety of possible Frege systems; however, the following fundamental theorem due to Cook and Reckhow shows that it suffices to consider any one of them.

Theorem 1.8. *(*[**29**]*) All Frege systems are polynomially equivalent.*

Proof. Consider two Frege systems given by axiom/inference rule sets R_1 and R_2. The general form of an axiom/inference rule is $G_1, G_2, \ldots, G_k|H$, 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 implicationally complete and R_2 is sound and finite, for every rule σ in R_2 of the form $G_1, G_2, \ldots, G_k|H$, since $(G_1 \wedge G_2 \wedge \ldots \wedge G_k) \rightarrow H$ is a tautology there is a constant size proof in R_1 of the H from $G_1, G_2, \ldots, G_k$. Given any inference 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 by applying the substitution $[x : y]$ to its proof of H from $G_1, \ldots, G_k$. The size of this proof is proportional to the sum of the sizes of $F_1, \ldots, F_k$ and F.

Applying this argument inductively to each line of the the R_2 proof, we get an R_1 proof that is not too large — in fact linear in the size of the R_2 proof. Hence R_1 polynomially simulates R_2. Since R_1 and R_2 were arbitrary, the result follows. □

1.2.3. Gentzen Systems/Sequent Calculus

These are proof systems where statements are of the form $F_1, \ldots, F_k \vdash G_1, \ldots, G_l$, meaning $(F_1 \wedge \ldots \wedge F_k) \rightarrow (G_1 \vee \ldots \vee G_l)$. There are many variations but the following is typical. The only axiom is $F \vdash F$. To prove a formula G, one has to derive $\vdash G$ and to refute it, one has to derive $G \vdash$. This is done using the axioms and rules of the following sort for each connective, plus axioms that allow one to permute and duplicate formulas on a given side of the $\vdash$ symbol:

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

A canonical example of such a system is Gentzen's system LK.

In general, as shown by Cook and Reckhow, sequent calculus systems are equivalent to Frege systems. We can also characterize sub-systems of sequent calculus cleanly based on the kinds of formulas F that are used in the cut rule. Sequent calculus is often used in proof complexity instead of restricted forms of Frege proofs because it allows one to be very precise in the power of system one generates by varying the cut rule. However, reasoning about sequent calculus proofs is cumbersome to write down because of the large numbers of rules and we won't use them here.

One can, in fact, completely eliminate the cut rule from the sequent calculus and still get a complete proof system. This much weaker system is equivalent to tableaux systems which we now we briefly describe.

1.2.4. Tableaux/Model Elimination Systems

This is a refutation system in which one decomposes the given formulas based on the subformulas that might be TRUE simultaneously. One tries to eliminate each of these possibe worlds from consideration. For example, if $\neg(A \rightarrow B)$ is TRUE, then A must be TRUE and B must be FALSE and if $A \leftrightarrow B$ is TRUE then there are two possible situations, both A and B are TRUE or both $\neg A$ and $\neg B$ must be true. Starting with the input formula, we build a tree of possible models based on subformulas and close off each by deriving a contradiction along each branch by finding any formula and its negation along the branch. Because of their potential for nice search strategies in their extensions to handle quantified logics these are among the more popular proof systems in automated theorem proving. However, for propositional logic they are much less interesting since their complexity in the worst case is quite high. The best proofs here can be even larger than truth tables; see [**56**] for a discussion of this system.

1.2.5. Resolution

Resolution forms the basis of the most popular systems for practical theorem proving. Structurally, it is like a Frege system but it 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 lone inference rule, the resolution rule: $(A \vee x), (B \vee \neg x) \,|\, (A \vee B)$. In applying this rule we say that we have *resolved on* variable x and call $(A \vee B)$ the *resolvent*. The goal is to derive the empty clause Λ, which is unsatifiable since it says that some member of the empty set is true.

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.

Resolution can only work with CNF formulas. However, we do not lose much by requiring our input to be in CNF form. This can be seen by the usual 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 procedure was originally used in the proof complexity context by Tseitin [**55**] and is familiar from Cook's argument reducing SAT instances to CNFSAT or 3SAT 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 expressing the fact that y_G takes on the value of G determined by the inputs to the formula. More precisely,

Definition 1.10. Given a Boolean formula F over $\wedge, \vee, \neg$, we create CNF formula F by including the clause (y_F) saying that F is true and the following clauses for each subformula G of F.

- 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 $(y_G \vee y_H)$.

Lemma 1.11. *For any assignment α to the variables of F, α satisfies F iff there exists an assignment β to the variables y_G such that (α, β) satisfies C_F.*

Exercise 1.12. Extend the definition of C_F to include connectives $\rightarrow$ and $\leftrightarrow$ and prove this lemma.

Thus F and C_F are equivalent as far as satisfiability is concerned. Since the size of C_F is polynomial in the size of F, this argument allows us to convert any proof system designed only for CNF formulas into a general proof system and preserve efficiency.

1.2.6. Davis-Putnam (DLL) Procedure

The Davis-Putnam or DLL procedure is both a proof system and a collection of algorithms for finding proofs. As a proof system, we will see that it corresponds to a special case of resolution in which the proof graph forms a tree. A simple David-Putnam algorithm is shown in Figure 3. The formula $F|_{x\leftarrow 1}$ is created from F by eliminating all clauses of F in which the literal x appears and removing $\neg x$ from all clauses in which it appears;

$F|_{x\leftarrow 0}$ is defined dually. Variants of this algorithm (based on different splitting rules) form the most widely used family of complete algorithms for satisfiability. [2]

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
  (a) (Splitting rule) Choose the 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

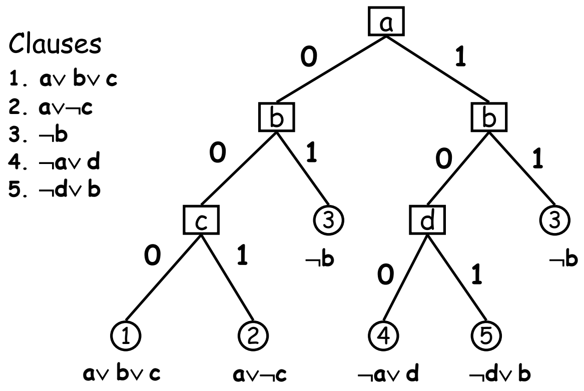

Figure 4. An example of a DLL refutation tree

A DLL refutation forms a tree with a branch at each node based on the value of a variable. A leaf corresponds to an empty clause created from one of the original clauses of F by the partial assignment along that branch. We label each the leaf of the tree by one of the original clauses that is falsified along that branch. Figure 4 shows an example of a DLL refutation tree. (In order to have a convenient tree in this example we have ignored the rule for clauses of size 1, also called unit clauses, and occasionally chosen another variable on which to branch. In general, in drawing the tree when we apply this rule, we will also include a node in the tree with two children, one of them a leaf representing the assignment opposite to the one that satisfies the unit clause.)

There is a straightforward way to 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

[2]As part of a general method for automated theorem-proving, Davis and Putnam [**31**] gave a procedure for finding proofs of unsatisfiability by eliminating variables one by one. Their original procedure proposed eliminating a variable x by computing all possible uses of the resolution rule that eliminate x and then discarding the original clauses involving x. This proved highly impractical because of its space usage so in implementing Davis and Putnam's ideas, Davis, Logeman, and Loveland [**30**] replaced it with a tree search procedure similar to that in Figure 3. The names Davis-Putnam procedure, DLL, and DPLL are all frequently used to describe this class of tree search procedures.

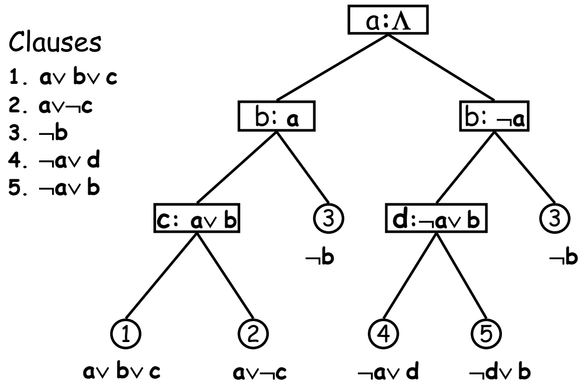

Figure 5. Creating a tree resolution proof from a DLL refutation

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.

The translation from a resolution tree to a DLL tree essentially reverses this process. The only difference is that in general a resolution tree may apply the resolution rule using the same variable more than once on a path from a leaf to the root. It is not hard to see that any such tree may be pruned to produce a smaller tree resolution proof of the same formula. After all such paths are pruned, the conversion is immediate.

Exercise 1.13. Show that any minimal size tree resolution proof resolves on each variable at most once on any path from a leaf to the root.
Hint: If there are two applications of the resolution rule to a variable on a given branch then eliminate the application further from the root and retain only one of the two children of the elminated application.

1.2.7. Nullstellensatz Proof System

We will consider several algebraic proof systems. All such proof systems are based on Theorem 1.14 concerning simultaneous systems of polynomial equations over a field K. This is a restricted form of Hilbert's Nullstellensatz which also lends the name to the simplest of these systems.

Theorem 1.14 (Hilbert's Nullstellensatz)**.** *A system of polynomial equations*

$$\begin{aligned} Q_1(x_1,\ldots,x_n) &= 0 \\ &\ldots \\ Q_m(x_1,\ldots,x_n) &= 0 \end{aligned}$$

over a 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 $\sum_{i=1}^m P_iQ_i \equiv 1$.

Clearly any common root of the original equations is also a root of any linear combination (with polynomial coefficients) of the corresponding polynomials. Therefore, if we can derive a the polynomial 1 in this way then the initial equations cannot have a common root. The other direction is more complex, although it is not hard to prove the special case that we will need directly.

Note that the form of this theorem, which takes a universal statement, the non-existence of a simultaneous zero, and translates it into an existential statement, the existence of the polynomials $P_1, \ldots, P_m$, seems directly suitable for our purposes.

Given any CNF clause C, we can translate it into a polynomial Q_C such that Q_C evaluates to 0 if and only if C is satisfied. More precisely, let Q_C be the product $\prod_{x_i \in C}(1-x_i)\prod_{\neg x_i \in C} x_i$. For example, if $C = (x_1 \vee \neg x_2 \vee x_3)$ is a clause in the instance, we can translate it into the polynomial $Q_C = (1-x_1)x_2(1-x_3)$. Let us also add polynomials $x_i^2 - x_i$ for each variable x_i; this will guarantee only 0-1 solutions so the discussion of extension fields in Theorem 1.14 becomes irrelevant. 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

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

The *size* of a Nullstellensatz proof is the number of monomials in the polynomials P. Its *degree* is the largest total degree of any $P_j Q_j$. [3]

1.2.8. Polynomial Calculus

Polynomial calculus is very similar to the Nullstellensatz proof system. We begin with $Q_1, Q_2, \ldots, Q_{m+n}$ (where $Q_{n+i} = x_i^2 - x_i$) as before. However, instead of trying to combine polynomials and then simplifying, 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. As discussed above, any common root of the original polynomials is also a root of any derived polynomial. Therefore, if we can infer the polynomial 1 then the initial polynomials cannot have had a common root.

The *degree* of a polynomial calculus proof is the maximum of the total degrees of all polynomials appearing in the proof. The *size* of a proof is the number of monomials appearing in it. Any degree d proof has size at most $\binom{n}{d} = n^{O(d)}$ and it is known that one can find a proof of degree d in time $n^{O(d)}$ using Gröebner-basis-like algorithm using linear algebra [**27**].

Exercise 1.15. Show that every unsatisfiable formula over n variables has a proof of degree at most $n + 1$ for Nullstellensatz as well as Polynomial Calculus over any field.

Another (and frequently quite convenient) way of viewing the polynomials in Polynomial Calculus proofs is as multi-linear polynomials representing elements in the ideal generated by the product of the $x_i^2 - x_i$.

Both Nullstellensatz and polynomial calculus proof systems depend on the choice of field. Common choices are $K = GF(p)$ or even the rationals or reals. It is also possible to consider them over rings $\mathbb{Z}_m$, although they no longer necessarily represent complete proof systems.

---

[3] In most discussions of the Nullstellensatz proof system, only the degrees of the coefficient polynomials P are considered. We include the input polynomials in the degree so that the degrees of polynomial calculus proofs are directly comparable. Usually, they differ by only one or two because the coefficients of the $x_i^2 - x_i$ polynomials have the largest degree.

1.3. 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. Obviously, the extra variables x' do not affect the degree of the proof since they could previously have been expressed as $(1 - x)$ but they may reduce the size bounds dramatically by eliminating monomials.

Exercise 1.16. Show that PCR simulates resolution with degree equal to the size of the largest clause in the resolution proof and with no increase in size.

1.3.1. Cutting Planes

The concept of cutting planes was introduced to relate integer and linear programming ([**35, 25**]). 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 in $[0, 1]$. (Since we will consider only integer solutions this is enough to force only 0-1 solutions.) 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 $a_1x_1 + \ldots + a_nx_n \geq \lceil B/c \rceil$.

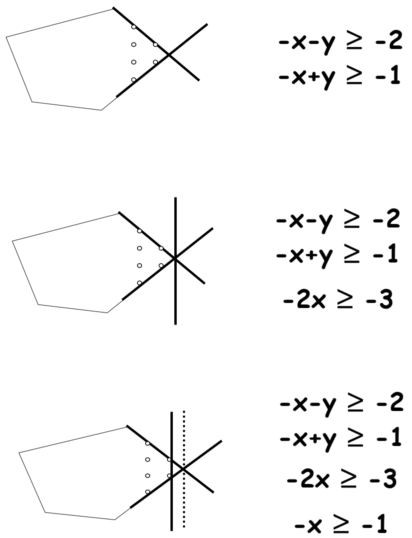

Figure 6. Why is the proof system called cutting planes?

The reason this system is called cutting planes will be clear from Figure 6. The starting inequalities define a region in n-dimensional space which is bounded by lines that might intersect at non-integer points. Adding two such inequalities yields a line that does not pass through any integer point. Applying the rounding in the division rule to such an inequality effectively shifts this line to the nearest integral point in the direction given by the inequality. This way we "cut" into the feasible region making it a smaller one by applying the division rule and taking the ceiling of the constant on the right hand side of the inequality.

Resolution
$$\frac{(a \vee b \vee c \vee \neg d) \quad (\neg a \vee b \vee c \vee \neg f)}{(b \vee c \vee \neg d \vee \neg f)}$$

Cutting Planes
$$\begin{array}{rl} a + b + c + (1-d) \geq 1 & \\ (1-a) + b + c + (1-f) \geq 1 & \\ (1-d) \geq 0 & \\ (1-f) \geq 0 & \\ \hline 2b + 2c + 2(1-d) + 2(1-f) \geq 1 & \text{Addition} \\ \hline b + c + (1-d) + (1-f) \geq 1 & \text{Division} \end{array}$$

Figure 7. Simulating resolution using cutting planes

It is not hard to see how cutting planes can simulate resolution efficiently.

Theorem 1.17. *Cutting planes polynomially simulate resolution.*

Proof. We apply the following construction inductively to each application of the resolution rule. Suppose that the resolution rule derives to $(A \vee B \vee C)$ from $(x \vee A \vee C)$ and $(\neg x \vee B \vee C)$ where the literals in A and B are disjoint. Add the two inequalities that correspond to $(x \vee A \vee C)$ and $(\neg x \vee B \vee C)$ and together with inequality $y \geq 0$ for each positive y in A or B and $(1 - y) \geq 0$ for each negative literal $\neg y$ in A or B. This results in an inequality in which the conversion of each literal in $(A \vee B \vee C)$ appears with coeffcient 2 and applying the division rule with $c = 2$ yields the inequality corresponding to $(A \vee B \vee C)$. An example is shown in Figure 7. $\square$

1.3.2. C-Frege Proof Systems

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

$$\mathsf{C} = \{f : f \text{ is computed by polynomial size circuits with structural property } \mathcal{P}_{\mathsf{C}}\}.$$

In a similar manner, we define a C-*Frege* proof system to be any Frege-style proof systems such that

(1) each line has structural property $\mathcal{P}_{\mathsf{C}}$,
(2) there is a finite set of axioms and inference rules, and
(3) the system is implicationally complete for circuits with property $\mathcal{P}_{\mathsf{C}}$.

Before saying anything more about C-Frege systems, let us quickly review some of the important circuit complexity classes. (See **[20]**, for example, for a more detailed discussion of circuit complexity.)

$\mathsf{P/poly}$: polynomial size circuits
NC^1 : polynomial size formulas = $O(\log n)$ depth fan-in 2 circuits
CNF : polynomial size CNF formulas
AC^0 : constant depth unbounded fan-in polynomial size circuits using $\wedge, \vee, \neg$ gates

$\mathsf{AC}^0[m]$: similar to AC^0 with gates that test the predicate $\equiv 0 \pmod m$ added
TC^0 : similar to AC^0 with threshold gates added

We know the following relationships among these complexity classes:

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

For the field $K = \mathrm{GF}(p)$, polynomial calculus (and PCR) are special cases of $\mathsf{AC}^0[p]$-Frege where all formulas have $\equiv 0 \pmod p$ gates at the top and $\wedge$ gates below that.

Also, the cutting planes proof system is a special case of TC^0-Frege with all formulas having depth 1.

Exercise 1.18. 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 a constant fraction of the nodes in its subtree.)

From the above exercise, and since NC^1 circuits can easily be expanded into trees (formulas) of polynomial size, it follows that ordinary Frege systems are polynomially equivalent to NC^1-Frege.

Resolution is a special case of CNF-Frege.

1.3.3. 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 the 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 appear to increase the power of Frege systems.

Since each extension variable describes a circuit in the input variables, extended Frege is equivalent to $\mathsf{P/poly}$-Frege.

1.4. Proof System Hierarchy

Some of the proof system relationships that we know are summarized in Figure 8. (The proof system ZFC at the top of the diagram is Zermelo-Frankel set theory with the axiom of choice.) For all solid arrows in the diagram, the system at the head of the arrow p-simulates the system at the tail of the arrow. Furthermore, below Frege, the system at the tail of an arrow is provably exponentially weaker than the one at its head. The dotted lines represent incomparable proof systems.

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 their capabilities and limitations. 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.

Although we have placed ZFC at the top of the diagram (since it is capable of formalizing much of mathematical reasoning), we do not know how high the hierarchy goes or even if there is some most powerful proof system sitting at the top of this hierarchy.

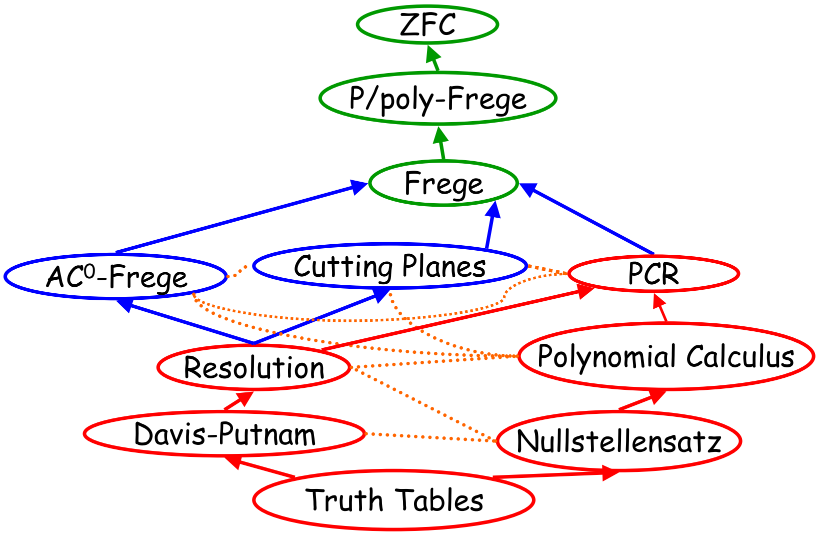

Figure 8. Some proof system relationships

The notion of polynomial simulation gives us one natural way of saying that one proof system is at least as powerful as another. However, in determining the height of the hierarchy we might ask for something a little less.

Definition 1.19. A proof system U *p-dominates* another proof system V iff there is a polynomial $f : \mathbb{N} \to \mathbb{N}$ such that

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

The known hierarchy above is the same if one uses the p-domination criterion instead of p-simulation. We now use some slightly non-standard but, we hope, evocative terminology.

Definition 1.20. U is *super* iff U p-dominates all other propositional proof systems. U is *super-duper* iff U p-simulates all such systems.

Theorem 1.21 ([**40**])**.** *If* $\mathsf{EXP} = \mathsf{NEXP}$ *then super-duper proof systems exist. If* $\mathsf{NEXP} = \mathsf{co\text{-}NEXP}$ *then super proof systems exist*

LECTURE 2
Lower Bounds in Proof Complexity

The first step in proving a lower bound for a proof system is often to find a candidate hard example for that proof system. Consider, for instance, a 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 or even $\mathsf{AC}^0[p]$. This suggests that something counting-related might be hard for AC^0-Frege or even $\mathsf{AC}^0[p]$-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 can 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 for problems based on graph structures.

2.1. The Pigeonhole Principle

The pigeonhole principle PHP^m_n says that there is no 1-1 function from m objects ('pigeons') to n objects ('holes') if $m > n$. The *onto* version of this, $ontoPHP^m_n$, says that there is no 1-1, onto function mapping m pigeons to n holes for $m > n$. This can be easily encoded as an unsatisfiable 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 pigeons to n holes. 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 often leaves out the function clauses because they are frequently redundant for lower bounds. One can derive a refutation of the relational form of this formula from one for the functional form by setting $P'_{ij} = P_{ij} \wedge \neg P_{i1} \wedge \ldots \wedge P_{i(j-1)}$. However, if the number of pigeons m is much larger than n, one does not necessarily need all the clauses and this conversion may be wasteful.

2.1.1. Usual Inductive Proof of PHP^n_{n-1}

The base case that there is no 1-1 and onto mapping from 2 pigeons to 1 hole is trivially true. For the inductive step, suppose there is a 1-1 function f from $f : \{1, \ldots, n\} \rightarrow$

$\{1,\ldots,n-1\}$. If $f(n)=n-1$, then f restricted to $\{1,\ldots,n-1\}$ yields a 1-1 function into $\{1,\ldots,n-2\}$ which is impossible by the inductive hypothesis. Otherwise, define another mapping $g:\{1,\ldots,n-1\}\rightarrow\{1,\ldots,n-2\}$ 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. Now g violates the inductive hypothesis.

2.1.2. Extended Frege Proof of PHP^n_{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 $Q_{ij}\equiv P_{ij}\vee(\neg P_{n(n-1)}\wedge P_{i(n-1)}\wedge P_{nj})$ for $i=1,\ldots,n-1, j=1,\ldots,n-2$. We can easily derive the pigeonhole clauses for the Q_{ij} from those for the $P_{i,j}$. Unwinding the argument, defining new variables at each step, if necessary, and ending with the trivial contradiction PHP^2_1 yields a short extended Frege refutation of PHP^n_{n-1}.

2.1.3. Cutting Planes Proof of PHP^m_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 $i=i,\ldots,m, j=1,\ldots,n$

For each j we first derive $P_{1j}+P_{2j}+\ldots+P_{mj}\leq 1$ inductively. For k from 3 to m, suppose we have $P_{1j}+P_{2j}+\ldots+P_{(k-1)j}\leq 1$. We derive the inequality with $k-1$ replaced by k as follows.

(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.

Since TC^0-Frege can be polynomially simulated by Frege proofs this also implies that there is a polynomial size Frege proof of PHP^n_{n-1}. This latter fact was originally shown by Buss [**24**] who showed this directly. The key idea, which can be developed into the simulation of TC^0-Frege by Frege, is that the usual $O(\log n)$ depth polynomial-size circuits that can compare and compute the sum of n integers are simple to prove correct.

2.1.4. Resolution Proof of PHP^n_{n-1}

Unlike Frege and cutting planes, resolution requires exponential size to prove the pigeonhole principle. This was the first exponential lower bound shown for general resolution.

Theorem 2.1 ([**37, 9**]). *Any resolution proof of PHP^n_{n-1} requires size at least $2^{n/20}$.*

Haken's original proof idea was based on *bottleneck counting*. One views truth assignments as flowing through the proof. Assignments start at Λ and flow out towards input clauses. A clause in the proof only allows the assignments it falsifies to flow through it. A key property that Haken proved is 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 many such clauses to let all the assignments flow through.

We present here a 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

(1) every large clause disappears, and (2) the result is still a $PHP^{n'}_{n'-1}$ proof for some good size n'. We also show that every proof of $PHP^{n'}_{n'-1}$ contains a *medium complexity clause* and further that every medium complexity clause is large. This gets us a lower bound on the proof size of PHP^{n}_{n-1}.

Proof. We say a truth assignment is *i-critical* if it matches all $n-1$ holes to all pigeons but pigeon i. Such an assignment is barely unsatisfying — it always satisfies all 1-1, *onto* and function clauses and all but one of the clauses saying that f is total. The only clause it falsifies is $C_i = (P_{i1} \vee P_{i2} \vee \ldots \vee P_{i(n-1)})$ which says that pigeon i is mapped somewhere. We will only care about the properties of the clauses in the proof when evaluated on critical truth assignments.

The properties of critical truth assignments make it convenient to convert each such clause C to a positive clause $M(C)$ that is satisfied by precisely the same set of critical assignments as C. More precisely to produce $M(C)$, we replace $\neg P_{ij}$ in C with $(P_{i1} \vee \ldots P_{(i-1)j} \vee P_{(i+1)j} \vee \ldots \vee P_{nj})$.

Given a clause C, let $badpigeons(C) = \{i \mid$ there is some i-critical assignment α falsifying $C\}$. Define the complexity of C, $comp(C) = |badpigeons(C)|$. It is not very hard to show that every resolution proof of $\neg PHP^{n}_{n-1}$ contains a clause of complexity t between $n/3$ and $2n/3$. The complexity of each initial clause is at most 1 and the complexity of the final clause Λ is n. Moreover, if clause A and B imply a clause C, then $comp(C) \leq comp(A) + comp(B)$. Therefore, if C is the first clause in the proof with $comp(C) > n/3$, we must have $n/3 < comp(C) \leq 2n/3$. It remains to show that $M(C)$ contains a large number of variables.

For $comp(C) = t$ we claim that $M(C)$ has at least $(n-t)t \geq 2n^2/9$ distinct literals mentioned. Fix some $i \in badpigeons(C)$, and let α be an i-critical truth assignment with $C(\alpha) =$FALSE. For each $j \notin badpigeons(C)$, define the j-critical assignment, α', obtained from α by toggling i and j, that is if α' maps i to the same place l that j was mapped to in α. Now $C(\alpha') =$TRUE since $j \notin badpigeons(C)$ and, further, α and α differ only in that α' maps i to l rather than j to l. Since C and $M(C)$ agree on all critical assignments and $M(C)$ is positive, it must contain the variable P_{il}. This argument may be applied for every $i \in badpigeons(C)$ and $j \notin badpigeons(C)$, yielding the size bound on $M(C)$.

Finally, we 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} to FALSE. This reduces PHP^{n}_{n-1} to $PHP^{n'}_{n'-1}$, 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 there is some resolution proof of PHP^{n}_{n-1} with at most $S < 2^{n/20}$ clauses C such that $M(C)$ is large. 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 a restriction $\log_{9/10} S < 0.329n$ times. The remaining proof proves $PHP^{n'}_{n'-1}$ 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}_{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 $\mathrm{width}(F)$ 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 $\mathrm{proofwidth}(F)$ denote the minimum of all proofs P of F of $\mathrm{width}(P)$. The following theorems due to Ben-Sasson and Wigderson and based on ideas of Clegg, Edmonds, and Impagliazzo relate size lower bounds on P to lower bounds on $\mathrm{width}(P)$.

Theorem 2.2 ([**13**])**.** *Every Davis-Putnam (DLL)/tree-like resolution proof of F of size S can be converted to one of width $\lceil \log_2 S \rceil + \mathrm{width}(F)$.*

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 $\mathrm{width}(P') \le \lceil \log_2 S' \rceil + \mathrm{width}(F')$.

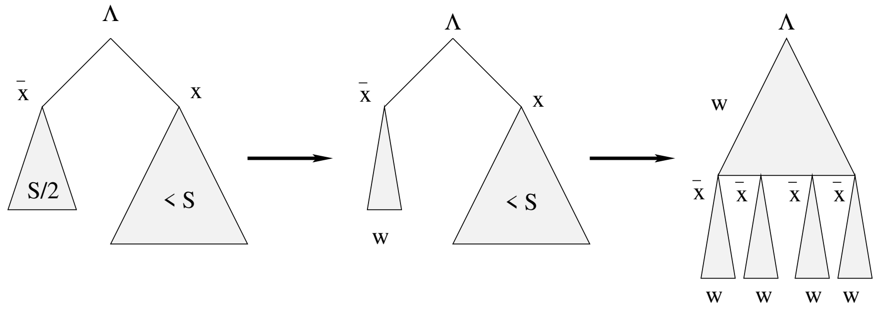

Figure 1. Converting small size proof to one of small width

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 $w - 1 = \lceil \log_2 (S/2) \rceil + \mathrm{width}(F) = \lceil \log_2 S \rceil + \mathrm{width}(F) - 1$. Adding $\neg x$ to each of the clauses in this proof lets us derive $\neg x$ from F in width $w = \lceil \log_2 S \rceil + \mathrm{width}(F)$. 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 $w = \lceil \log_2 S \rceil + \mathrm{width}(F)$.

Now use a copy of the left sub-tree to resolve with each leaf clause of the right subtree that contains an x (see Figure 1). This allows us to eliminate 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 $\lceil \log_2 S \rceil + \mathrm{width}(F)$. This completes the proof. $\square$

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

Theorem 2.4 ([**13**]). *Every resolution proof of F of size S can be converted to one of width $\sqrt{2n \ln S} + \mathrm{width}(F)$.*

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 $(1 - W/2n)^k S \le 1$. By an easy calculation this holds if $k \le \sqrt{2n \ln S}$. We show by induction on n and k that any F with at most S large clauses has a proof of width $\le k + \mathrm{width}(F)$. 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, by what we have already observed, 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 $k-1+\mathrm{width}(F)$. Hence there is a derivation of $\neg x$ from F of width at most $k + \mathrm{width}(F)$.

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 and involving one less variable. The induction hypothesis implies that there is a refutation of $F|_{x\leftarrow 0}$ of width at most $k + \mathrm{width}(F)$.

As in the proof of tree-like resolution case, we use the derivation of $\neg x$ from F in width at most $k + \mathrm{width}(F)$ at each leaf of the proof and resolve $\neg x$ with each clause of F containing x to get $F|_{x\leftarrow 0}$. We now use the refutation of the latter set in width $k + \mathrm{width}(F)$. □

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

We note that this relationship between width and size is nearly optimal for general resolution (at least for formulas with low resolution complexity) as shown by the following result:

Theorem 2.6 ([**17**]). *There is a family of formulas $\{F_n\}$ in n variables with constant clause width that have polynomial-size resolution proofs but require resolution proof width $\Omega(\sqrt{n})$.*

Together with Theorem 2.2, this implies an exponential separation between tree resolution and general resolution.

Corollary 2.7. *There is a family of formulas $\{F_n\}$ in n variables having polynomial-size resolution proofs but requiring Davis-Putnam (DLL)/tree-resolution proofs of size $2^{\Omega(\sqrt{n})}$.*

Such a separation was originally proved in [**16**] using tautologies related to graph pebbling. As shown in [**13**] one can in fact use slightly different tautologies based on graph pebbling other families of graphs and a separate argument to show that Davis-Putnam (DLL)/tree-like resolution can require $2^{\Omega(n/\log n)}$ proofs for formulas that have polynomial size proofs in general resolution.

2.3. Resolution Proofs Based on the Width-Size Relationship

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

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

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

Lemma 2.8 ([**26**]). *If P is a resolution proof of F, then $width(P) \ge e(F)$.*

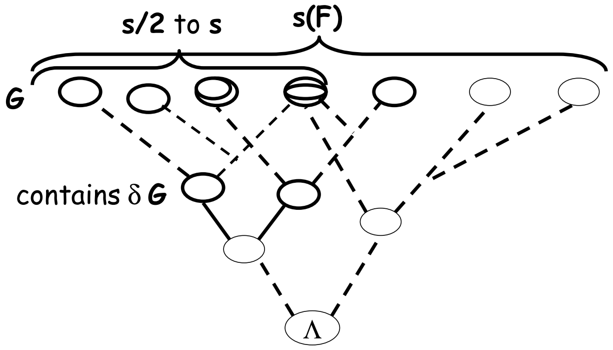

Figure 2. Relating proof width to sub-critical expansion

Proof. Given a clause C in P and a collection of clauses $G \subseteq F$, write $G \Rightarrow_P C$ if all the clauses in G are used in P to derive C. In a resolution proof, if a literal appears in a clause C then the only way it can be removed from clauses derived using C is if the literal is resolved with its negation. Therefore, if $G \Rightarrow_P C$ then every variable in δG appears in C and so $\mathrm{width}(C) \ge |\delta G|$.

Define the complexity, $comp_P(C)$, of a clause C in P to be the size of the set $G \subseteq F$ such that $G \Rightarrow_P C$. By definition $comp_P(\Lambda) \ge s(F)$ and $comp_P(C) = 1$ for any clause in F. Furthermore, $comp_P$ is subadditive, $comp_P(C) \le comp_P(A) \vee comp_P(B)$ if C is a resolvent of A and B. For any $s \le s(F)$, the last clause C in P with $comp_P(C) < s$ satisfies $\mathrm{width}(C) \ge |\delta G|$ for some set $G \subseteq F$ with $s/2 \le |G| < s$. Maximizing over all choices of $s \le s(F)$, we get $\mathrm{width}(P) \ge e(F)$. $\square$

Corollary 2.9. *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

Define a random distribution $F^k_{n,\Delta}$ on the set of k-CNF formulas in n variables by choosing $m = \Delta n$ clauses independently and uniformly from among the $2^k\binom{n}{k}$ clauses of length k. Δ here is the *density* of the formula. For F chosen from this distribution we write $F \sim F^k_{n,\Delta}$. It is well known that the typical satisfiability properties of such random k-CNF formulas are determined by a density threshold [**33**]. Random formulas with density more than the threshold are asymptotically almost surely (a.a.s.) unsatisfiable, whereas those with density below the threshold are a.a.s. satisfiable.

For random formulas whose density is not too large, we can show that any resolution proofs of their unsatisfiability are almost surely super-polynomial, as stated more precisely in the following theorem:

Theorem 2.10. *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 $2^{\frac{n}{\Delta^{4/(k-2)+\epsilon}}}$.*

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 qualitatively the same asymptotic complexity as the best known factoring algorithms, for example.

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 3) 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

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

Hall's Theorem allows us to get lower bounds on $s_H(F)$ for random hypergraphs:

Theorem 2.11 (Hall's Theorem)**.** *A hypergraph F has a system of distinct representatives iff every subgraph of F has density at most 1.*

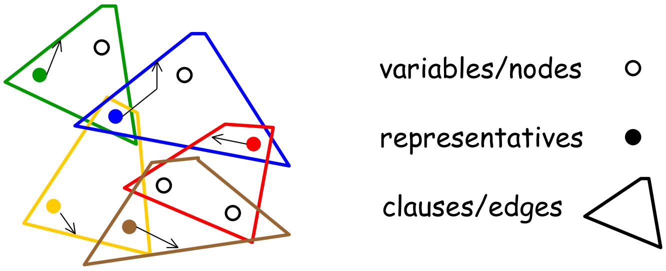

Figure 3. System of distinct representatives

The expansion of a hypergraph F is also related to the densities of its subgraphs, so we can analyze both $s_H(F)$ and $e_H(F)$ by looking at the density of subgraphs of F.

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. With this translation $s_H(F) \leq s(F)$ and $e_H(F) \leq e(F)$.

Lemma 2.12. *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 $\epsilon > 0$.*

Proof. The proof of this lemma 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 its nodes are in its 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 up to s using union bound:

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

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 $q = 2r/(k+\epsilon')$ for all r between $s/2$ and s.

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

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

2.3.1.1. *Upper Bound.* We end this section by giving an upper bound for Davis-Putnam (DLL) proofs random k-CNF formulas. Recall the simple Davis-Putnam (DLL) algorithm shown in Lecture 1, Figure 3.

Theorem 2.13 ([**6**]). *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*

$$2^{O\left(\frac{n}{\Delta^{1/(k-2)}}\right)} n^{O(1)}.$$

We only sketch the argument. 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 in the directed graph of the 2-clauses in the residual formula. But now a few splitting steps will pick one of these almost surely and setting the unit clauses will end the algorithm.

2.4. Nullstellensatz and Polynomial Calculus Lower Bounds

2.4.1. Pigeonhole Principle

The Pigeonhole principle 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 $1 \leq i < j \leq m, j = 1, \ldots n$
f is onto: $P_{1j} + P_{2j} + \ldots + P_{mj} - 1 = 0$, for $j = 1, \ldots, n$

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 $ontoPHP^m_{m-1}$. In general, we have the following bounds:

Theorem 2.14 ([**7**]). *If p^k is the largest power of p dividing $m - n$ and $n > p^{2k}$, then Nullstellensatz proofs of $ontoPHP^m_n$ over $GF(p)$ have degree at least 2^k and at most p^k.*

However, if the onto equations are missing, the degree required can be much larger.

Theorem 2.15 ([**51**]). *Polynomial Calculus proofs of PHP^m_n require degree $n/2$ for any m and any field.*

2.4.2. Counting Principles

Let $Count^{2n+1}_2$ denote the tautology stating that one cannot perfectly match members of an odd size set. More generally, if r does not divide m, let $Count^m_r$ denote the fact that there is no perfect r-partition of m objects. We will encode $Count^m_r$ 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 $e \neq f \in E$ s.t. $e \cap f \neq \phi$

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

2.4.3. Tseitin Tautologies

Let $G(V, E)$ be a given (low-degree) graph with 0-1 *charges* on its nodes. Further, assume that the total charge on the graph is odd. Then by the usual 'hand-shaking theorem' in graph theory 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 these tautologies would be to use equations modulo 2. In fact, it is easy to give a degree 1 proof of the Tseitin tautologies over any field of characteristic 2. However, we will be interested in the size of their proofs in fields on other characteristic. For this we will use the so-called *Fourier basis* $\{1, -1\}$ to represent the charges and weights. If we have a variable x over $\{0, 1\}$, we form an equivalent variable y over $\{1, -1\}$ by setting $y = (-1)^x$, or, equivalently, by the linear transform $y = 1 - 2x$. The equation forcing variables to take legal values now becomes $y^2 - 1 = 0$ and the fundamental contradiction is $1 = -1$. This transformation is convenient for expressing parity: $x_1 \oplus x_2 \oplus \ldots \oplus x_k = 0$ becomes $y_1 y_2 \ldots y_k = 1$.

Exercise 2.17. Show that this transformation to Fourier basis preserves the degrees of proofs.
Hint: Use the fact that transformation is linear.

For Tseitin formulas in the Fourier basis over a field of characteristic different from 2, there is a variable y_e for each $e \in E$. y_e takes values in $\{1, -1\}$ and the constraining equation is $y_e^2 = 1$. The Equation saying that the 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 $v \in V$. The 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 the Nullstellensatz proof system* [**36**],
(2) *and degree $\Omega(n)$ to prove in Polynomial Calculus* [**22**].

These results use expander graphs.

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

We now sketch the proof of Theorem 2.18 using a constant-degree graph G with expansion $\epsilon > 0$. Considering such graphs yields a degree lower bound of $\epsilon n/8$ for Nullstellensatz and Polynomial Calculus proofs of Tseitin tautologies.

Proof. For $S \subset V$, let $E(S) \subseteq E$ be the edges of G with one end-point in S and one outside S. Expansion $\epsilon > 0$ implies $E(S) \geq \epsilon|S| > 0$ for all sets S of size at most $n/2$.

Each input equation for any Tseitin tautology has two terms and is of a special form. We can think of the equation as an equivalence between two monomials where every monomial corresponds to the parity of a subset of edges. Each equivalence corresponds to the parity of the set of edges 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 the Fourier basis is essentially equivalent to equations modulo 2, we will, for simplicity of reasoning, think of the problem as operating on mod 2 equations. Starting with an equation and applying an equivalence between monomials corresponds to adding the equations modulo 2.

Given a set S of vertices, let Σ_S denote the sum of of the original edge variables (in the 0-1 basis) leaving S. We show that if the degree of the proof is small then every equation derived in the proof will be of the form $\Sigma_S = |S|$ (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 equations 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 since a contradiction contains no variables. However, if we start with sets of size at most $n/4$, then this won't happen. By the 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.

We now must translate back to the original polynomials. Each binomial equation corresponds to a parity summation equation with some portion of the equation in each monomial. If each of these monomials has degree less than $\epsilon n/8$, then the corresponding parity equation has fewer then $\epsilon n/4$ terms and as above we cannot reach a contradiction. Hence the degree of any proof has to be more than $\epsilon n/8$ for graphs with expansion ϵ. $\square$

This lower bound for Tseitin tautologies has many implications.

Corollary 2.21 ([**22**])**.** (1) *We can reduce Tseitin tautologies to $Count_2^{2n+1}$. This implies an $\Omega(n)$ degree lower bound for $Count_2^{2n+1}$ for all fields K with $char(K) \neq 2$.*

(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 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 bounds for $Count_p^{pn+1}$ for all fields K with $char(K) \neq p$.*

The degree lower bounds above apply to PCR as well as to the polynomial calculus since any PCR proof can be translated to polynomial calculus proof preserving the degree. The following exercise allows one to derive size lower bounds as well.

Exercise 2.22 ([**2**])**.** Show how the resolution relationships between size and width apply to PCR using size and degree.

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 2.23 ([**12**])**.** *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.*

Proof. We merely sketch the argument. First convert a given k-CNF formula into strengthened 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. $x_1 + x_2 + x_3 = 0 \pmod 2$. Clearly, if the original formula is unsatisfiable then the new system of equations will not have a solution. The goal of an unsatisfiability proof is to derive the contradiction $0 = 1 \pmod 2$ by adding collections of equations modulo 2. The transformation to the Fourier basis is also straightforward provided the field's characteristic is not 2.

As was the case with proofs of the Tseitin tautologies, every polynomial calculus proof can be viewed as substituting equivalent monomials for each other and thus of operating in the realm of parity equations. The degree of the polynomial calculus proof is at least half the number of variables appearing in the longest parity equations. In analogy with the boundary property of the resolution rule, if we have a variable that appears in precisely one of the equations used in deriving a given conclusion, then that variable must appear in the conclusion. Then, by the same reasoning as with resolution, the number of variables in the longest equation in such a proof is at least $e_H(F)$ and thus the degree is at least $e_H(F)/2$. □

This gets us the following lower bounds for random k-CNF formulas for PCR, Polynomial Calculus and Nullstellensatz.

Theorem 2.24. *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 $2^{c_\epsilon n/\Delta^{4/(k-2)+\epsilon}}$.*

LECTURE 3
Automatizability and Interpolation

Our bounds for proof systems have all applied to *non-deterministic* algorithms. Bounding the size of such proofs was useful in relation to the goal of proving $\mathsf{NP} \neq \mathsf{co\text{-}NP}$. The proof system 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 $f(n, S)$-*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.

The results relating size and width allow us to derive weak forms of automatizability for resolution. We recall:

Theorem 3.3 ([**13**])**.** *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 ([**27, 9, 13**])**.** *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 be reduced to polynomial by making the search recursive. $\square$

Theorem 3.5 ([**13**])**.** *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 ([**13, 9**])**.** *General resolution is $2^{O(\sqrt{n \log S} \log n)}$-automatizable.*

It is not hard to see that if one has an upper bound d on the degree of a Nullstellensatz refutation then one solve for the coefficients of the polynomials using linear algebra in time $n^{O(d)}$. Clegg, Edmonds, and Impagliazzo extended this to linear algebra arguments using the more general polynomial calculus. Together with the degree-size relationships of PCR we obtain:

Theorem 3.7 ([**27, 2**]). *Tree-PCR and PCR are $S^{O(\log n)}$-automatizable and $2^{O(\sqrt{n \log S} \log n)}$-automatizable, respectively.*

Other than these results (and the trivial results that very weak systems such as truth tables are automatizable) we do not have much in the way of positive results on automatizability.

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 for this tautology 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 due to Craig:

Theorem 3.9. *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.

Automatizability and interpolation of a proof system V are related provided that V satisfies a very simple property shared by virtually all proof systems: We say that V is *closed under restriction* iff for any unsatisfiable formula $F(x, y)$ and assignment α of truth values to y, given a V-proof P of $F(x, y)$, one can obtain a proof of $F(x, \alpha)$ of at most the same size by subsituting α for y in P.

Lemma 3.12 ([**18**]). *If V is automatizable and closed under restriction 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, since V is closed under restriction, 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 ([39]). *Resolution has feasible (monotone) interpolation.*

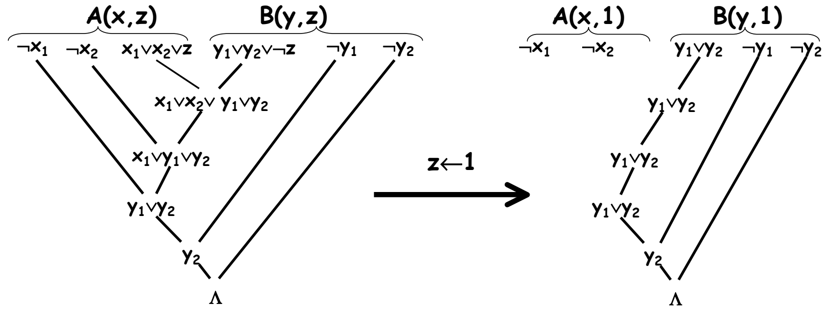

Figure 1. Construction interpolant for resolution

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 and a assignment α to the variables z (in this case only a single variable). We can 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 $z = 1$. It is clear that in general this process will produce a cut that separates either $A(x, \alpha)$ or $B(y, \alpha)$ from the contradiction. Furthermore, which of the two will be remain is a simple computation given the graph of the proof.

Theorem 3.14 ([48]). *Cutting planes has feasible (monotone) interpolation where the interpolants are circuits over the real numbers.*

Theorem 3.15 ([47]). *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 try to 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* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, *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 polynomial size circuit for deciding satisfiability, implying $\mathsf{NP} \subset \mathsf{P/poly}$. (The inequality is strict because $\mathsf{P/poly}$ is known to contain undecidable languages that are not in NP). We have been somewhat vague about precisely how to do the CNF formula encoding of these predicates efficiently but it all can be worked out. $\square$

We can do more than this in the case that the proof system has monotone feasible interpolation using the following *clique-coloring* formulas.

Definition 3.17. For any n and $k < n$ define the unsatisfiable formulas $\textit{clique-color}_{k,n}(x, y, z) = A(x, z) \wedge B(y, z)$, saying that the n-vertex undirected graph $G = G(z)$ defined by the $n(n-1)/2$ variables z_{uv} in z has both a k-clique defined by x and a $k-1$-coloring defined by y. More precisely, we code the clique by kn variables x_{iv} that are TRUE iff vertex v of G is the i^{th} node of the k-clique in G and code the coloring by $(k-1)n$ variables y_{iv} that are TRUE iff vertex v is given the i^{th} color in the valid $(k-1)$-coloring of $G(z)$.

(1) $A(x, z)$, the statement that $G = G(z)$ has a k-clique defined by x contains four types of clauses. Clauses $(\bigvee_v x_{iv})$ for each $i = 1, \dots k$, say that some vertex is chosen as the i^{th} vertex of the k-clique. Clauses $(\neg x_{iv} \vee \neg x_{ju} \vee z_{uv})$ for $u \neq v$ and all $i \neq j$ say that u and v cannot both be chosen in the k-clique unless there is an edge connecting u and v. Clauses $(\neg x_{iv} \vee \neg x_{jv})$ for all v and all $i \neq j$ say that no vertex is counted twice in the clique. Clauses $(\neg x_{iu} \vee \neg x_{iv})$ for all $iand$for $u \neq v$ say that we don't waste vertices.

(2) $B(y, z)$, the statement that $G(z)$ has a $(k-1)$-coloring defined by y has two kinds of clauses. Clauses $(\bigvee_i y_{iv})$ for all v say that each vertex gets a color. Clauses $(\neg z_{uv} \vee \neg y_{ui} \vee \neg y_{vi})$ for all i and all $u \neq v$ say that two vertices that have an edge between them do not get the same color. Clauses $(\neg y_{vi} \vee y_{vj})$ for all v and all $i \neq j$ say that a vertex is given only one color.

Observe that the z_{uv} appear only positively in A and only negatively in B. Also any interpolant of $\textit{clique-color}_{k,n}(x, y, z)$ can distinguish between the cases that $G = G(z)$ has a k-clique or is $(k-1)$-colorable. Thus, the celebrated superpolynomial monotone Boolean circuit lower bounds of Razborov [**50**] for the k-clique problem and their sharpening to exponential lower bounds by Alon and Boppana [**5**] yield lower bounds for any proof system having monotone feasible interpolation by Boolean circuits.

Theorem 3.18. *Any proof system V that has monotone feasible interpolation by Boolean circuits is not polynomially bounded. More precisely they require exponential size proofs of* $\textit{clique-color}_{k,n}$ *for $k = \Theta(n^\alpha)$.*

Pudlák extended the argument of Alon and Boppana for monotone Boolean circuit lower bounds to the case of monotone real-valued circuits which, together with Theorem 3.14, showed unconditional exponential lower bounds for cutting planes proofs.

Theorem 3.19 ([**48**])**.** *Any cutting planes proofs of* $\textit{clique-color}_{k,n}$ *for $k = \Theta(n^\alpha)$ are exponential.*

3.4. Limitations

Under widely believed cryptographic assumptions, sufficiently powerful proof systems do not have feasible interpolation and our technique for proving lower bounds using interpolation becomes useless for such systems. Krajicek and Pudlák showed the first such result based on a fairly general assumption.

Theorem 3.20 ([**40**])**.** *If one-way functions exist, then Frege systems do not have feasible interpolation.*

This was sharpened substantially given more precise assumptions.

Theorem 3.21. *If factoring Blum integers is hard, then any proof system that can polynomially simulate* TC^0*-Frege* [**18, 19**]*, or even* AC^0*-Frege* [**15**]*, does 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 any trapdoor function exists. The actual proof uses Diffie-Hellman secret key exchange which is as hard as factoring Blum integers (integers equal to the product of two distinct primes, each congruent to 3 modulo 4) [**14**].

In the interpolation setting,

(1) the common variables 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 z and the last bit of $key(x, y)$ is 0, and
(3) $B(y, z)$ will say that the player with y correctly computed its side of the conversation z and the last bit of $key(x, y)$ is 1.

Clearly, this is unsatisfiable and any interpolant must be able to compute the low order bit of the key.

We must encode the computation of each player in such a way that the proof system, given only x and z (or only y and z), can prove what the value of the bit is. We can make the task easier by extending x with helper extension variables. In the case of the Diffie-Hellman secret key exchange computing the key requires powering which is definitely not in AC^0 (and probably not in TC^0). However, suitable extension variables can be given to make it easy enough to prove. □

The required hardness for factoring is different in the two cases: to prevent TC^0-Frege interpolation only requires superpolynomial hardness, whereas to eliminate AC^0-Frege interpolation requires sub-exponential hardness.

Corollary 3.22. *If factoring Blum integers is hard, then any proof system that can polynomially simulate* AC^0*-Frege is not automatizable.*

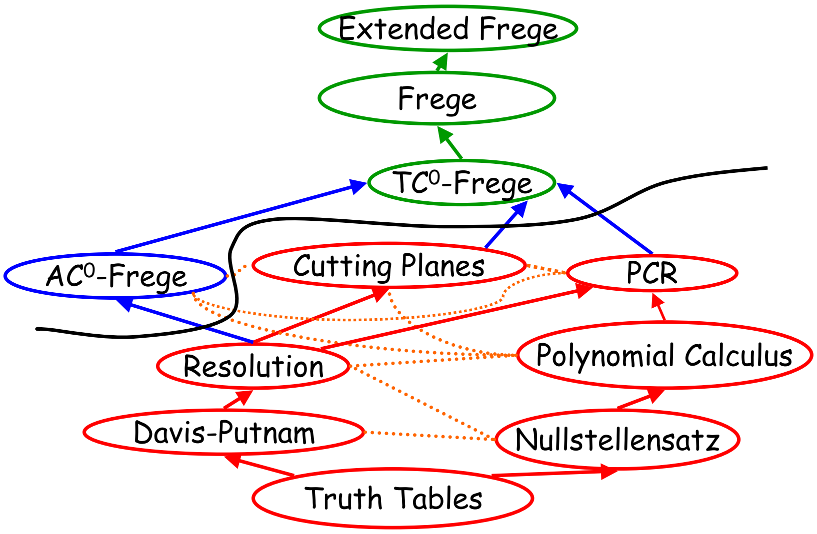

Figure 2. The interpolation line

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 Håstad whose original proof used a restriction argument.

Theorem 4.1 (**[38]**). *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 $S \geq 2^{cn^{1/d-1}}$.*

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

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. $\rho : X \rightarrow \{0, 1, *\}$ 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 $\rho(x_i) \neq *$. Note that in the case of formulas or circuits, as was the case when we created residual formulas based on partial assignments for the DLL procedure, we usually view this as simplifying the structure of the resulting formula or circuit. We will however also need to consider the situation where we do not simplify the formula or circuit but merely plug in values for the inputs. This will be denoted using $F|^\rho$ instead of $F|_\rho$.

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 that small circuits C cannot compute a complex function f, we demonstrate the existence of 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 variable x_i,
(2) leaf nodes are labelled 0 or 1,
(3) edges out of each internal node are labelled 0 or 1, and
(4) no two nodes on a path have the same variable label.

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 $b \in \{0,1\}$, $x_i \leftarrow b$ is in ρ iff on that branch, the out-edge labelled b is taken from a 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.

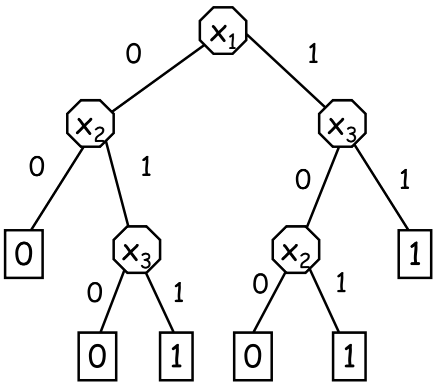

Figure 1. Decision tree for $x_1 + x_2 + x_3 \geq 2$

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 expressed as a DNF formula with term size at most t by associating a term with each branch with leaf label 1. In a similar fashion f can be expressed as a CNF formula with clause size at most t by associating a clause with each branch with leaf label 0.

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 will be the root node). We 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. Otherwise, if this term has t variables that have not yet appeared on 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 for each of the other leaves we solve the problem recursively. The full tree created for our example is shown in Figure 2.

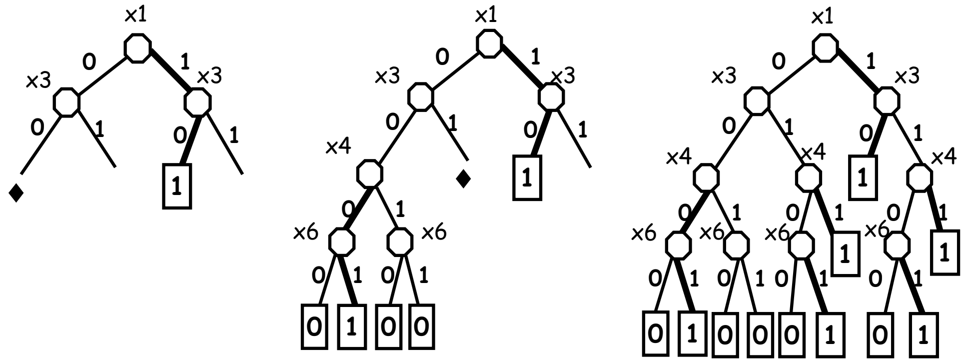

Figure 2. Generating canonical decision tree from DNF formula: $x_1\bar{x}_3 \vee x_3x_4 \vee \bar{x}_4x_6$

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

(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, both parity and its negation require decision trees of height n. (Compare 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.)

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 $g|_{\rho_i}$.

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 $h = (g_1 \vee \ldots \vee g_\ell)$ 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 of 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* unbounded fan-in OR's in the i^{th} layer (or which there are at most S).

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 $\rho_{i+1} = \rho_i\pi$ and by our assumed properties of π, short $T_{i+1}(h)$ exist for all 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 have been choosing π's which guarantee short trees, if the circuit is small, the tree we end up with will be shorter than the number of inputs that

ρ leaves unset. By our earlier observation about restrictions of parity, such a decision tree must be incorrect. This yields the lower bound.

All that remains now is to get the restriction 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 exact details, we here provide a sketch of how the proof works. The idea is to choose a random small π and prove that it 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 will be strictly less than 1, which implies that there must exist a small π that works.

The the argument relies on the following restatement of a result of Håstad [**38**]:

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 maintain trees of height $t = \log_2 S$ and let the number of unset variables decrease 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 $\log_2 S < n/(13 \log_2 S)^d$, i.e., if $\log_2 S < n^{1/(d+1)}/13$. If this happens, the circuit cannot compute parity and we get an exponential lower bound. (Håstad's argument saves two factors of $\log_2 S$ and has a somewhat better factor than 13, but we do not try to optimize the analysis.)

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 (or subformula) 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 a 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 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 decision trees suitable for lower bounds for the pigeonhole principle, we mention some of the main results derived using this method.

Theorem 4.7 ([**1, 46, 41**]). *$ontoPHP_n^{n+1}$ requires exponential size AC^0-Frege proofs.*

Theorem 4.8 ([**1, 8**]). *$Count_2^{2n+1}$ requires exponential size AC^0-Frege proofs even given PHP_m^{m+1} as extra axiom schemas.*

Theorem 4.9 ([**23**]). *$Count_p^{pn+1}$ requires exponential size AC^0-Frege proofs even given $Count_q^{qm+1}$ as axiom schemas for $q \neq p$.*

4.3.1. Matching Decision Trees and the $ontoPHP_n^{n+1}$ Lower Bound

Consider again the pigeonhole principle $ontoPHP_n^{n+1}$ 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 in the lower bounds for resolution proofs of the pigeonhole principle, 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. Thus a decision tree computes a function that is a disjunction of partial matchings. An example is given in Figure 3, along with the DNF formula that it computes.

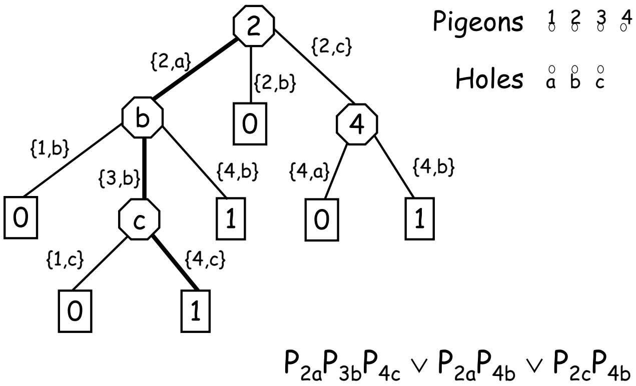

Figure 3. A matching decision tree with path for $P_{2a}P_{3b}P_{4c}$ highlighted

Matching decision trees can only compute certain types of functions exactly. Say that a matching decision tree T *represents* a function f on the P_{ij} variables if for each branch of T, the restriction ρ that corresponds to the branch makes $f|_\rho$ constant and equal to the leaf label on that branch.

Given a refutation $\mathcal{P}$ of PHP_n^{n+1}, a *tree evaluation* of $\mathcal{P}$ associates a matching decision tree $T(g)$ with each distinct subformula g of $\mathcal{P}$ such that:

(1) Every occurance of the same subformula has the same associated tree.
(2) $T(0)$ is a single node with leaf label 0 and $T(1)$ is a single node with leaf label 1.
(3) $T(P_{ij})$ is the tree that queries i and has height 1.
(4) $T(\neg g)$ is $T(g)$ with leaf labels toggled.
(5) The tree $T(h)$ for $h = (g_1 \vee \ldots \vee g_\ell)$, represents the the DNF formula $F_h = T(g_1) \vee \ldots \vee T(g_\ell)$.

The *height* of a tree evaluation is the height of the largest tree in the evaluation.

Given a DNF formula F_h whose terms correspond to partial matchings, the canonical conversion of F_h into a matching decision tree results in a tree that represents F_h. This is very similar to the canonical conversion for ordinary decision trees: We go term by term left to right simplifying future terms based on partial assignments. The major difference is that for each term, we query both endpoints of every variable in that term rather than simply querying the variable itself.

We say that g *evaluates to true under tree evaluation* T if and only if every leaf label of $T(g)$ is 1 and *false* if every leaf label of $T(g)$ is 0.

Lemma 4.10. *Let cP be a proof in Frege system each of whose rules has at most s distinct subformulas. If $st \leq n$ and T is a tree evaluation, of height at most t, of the set of subformulas of $\mathcal{P}$ then all formulas in $\mathcal{P}$ evaluate to true under T.*

On the other hand one can show:

Lemma 4.11. *If $t+1 \leq n < m$ and T is a tree evaluation of the subformulas in a proof of $ontoPHP_n^m$ then $ontoPHP_n^m$ evaluates to false under T.*

As was the case with the parity argument, however, in general a canonical conversion to a matching decision tree does not produce small height trees. However, if a matching restriction is applied then there is an analog of Håstad's switching lemma saying that the result of the conversion will probably have small height. Although such results were first proved in [**41, 46**], the precise formulation is from [**7**].

Lemma 4.12. *Let F_h be a DNF formula, each of whose terms is a matching of size s on the variables of $ontoPHP_n^{n+1}$. If $t \geq 0$ and $10 \leq k \leq (n/s)^{1/2}/10$ then, for ρ chosen uniformly at random from $R^{k,n}$, the probability that the canonical conversion of $F_h|_\rho$ results in a matching decision tree of height at least t is less than $(1.5k^2\sqrt{s/n})^t$.*

The overall structure then is like the parity argument. We apply restrictions of additional variables from among those remaining at each level as we go up through the proof. In particular, given a matching decision tree T and a partial matching ρ such that the height of T is at most $n-|\rho|$, the matching decision tree $T|_\rho$ is obtained by contracting all edges of T whose label is in ρ and deleting all edges of T (and their associated subtrees) whose labels disagree with ρ.

One final ingredient is needed to make this work. In the case of circuit lower bounds where restricted circuits were represented exactly it sufficed that if one applied a restriction ρ to a tree T computing f then $T|_\rho$ computes the same function as $f|_\rho$. Here, because of the tree evaluation must also be consistent in how it associates trees with different instances of the same subformula, we do not simplify formulas g in the usual way when we apply a restriction ρ to produce $g|_\rho$. Instead, we simply plug in the constants at the leaves to produce $g|^\rho$. As functions, the two are the same but this allows us to preserve some of the distinctions between subformulas that were distinct prior to restriction. This allows us to say that restrictions preserve tree evaluations.

Lemma 4.13. *If T is a tree evaluation of height t of a set Γ of formulas on the variables of $ontoPHP_n^{n+1}$ that is closed under subformulas and ρ is a matching restriction with $|\rho|+t \leq n$ then the map T^ρ given by $T^\rho(g) = T(g)|_\rho$ is a tree evaluation of $\Gamma|^\rho$.*

Furthermore one must strengthen Lemma 4.11 to:

Lemma 4.14. *If ρ is a matching restriction and $t+|\rho|+1 \leq n < m$ and T is a tree evaluation of the subformulas in a proof of $ontoPHP_n^m|^\rho$ then $ontoPHP_n^m|^\rho$ evaluates to false under T.*

The exponential lower bound follows by maintaing tree evaluations of height $t = \log_2 S$. In order to do this, the number of holes n_i whose matching is undetermined after each level i is given by the recurrence $n_0 = n$, $n_{i+1} = (n_i/9\log_2 S)^{1/4}$. This yields exponential lower bounds of the form $2^{n^{\epsilon_d}}$ for depth d proofs, where ϵ_d decays exponentially in d.

4.3.2. Beyond PHP_n^{n+1}

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.

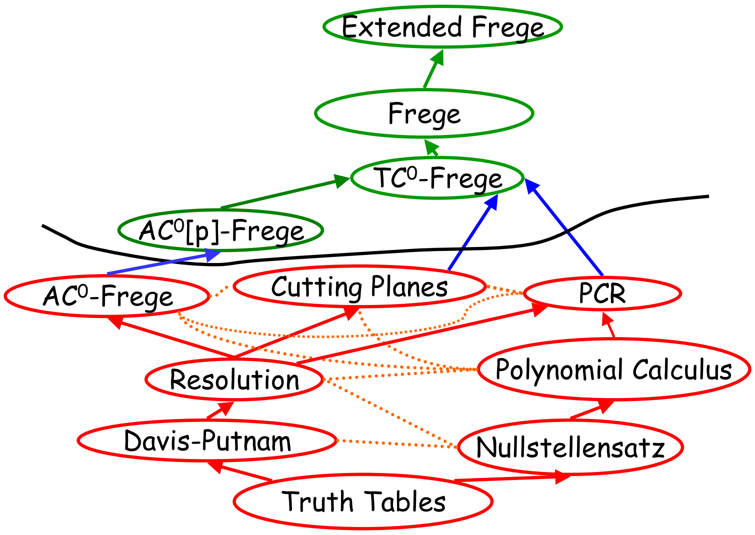

Figure 4. The frontier of super-polynomial proof system lower bounds

LECTURE 5
Other research and open problems

Here we give a brief outline of other research and open problems on a number of topics in proof complexity. The reader is also referred to [**56**] and [**10**] for complementary surveys of proof complexity and a list of additional open problems.

Subsystems of Resolution. There are a number of subsystems of resolution other than tree-resolution that seem to have relevance for proof search algorithms. These include *regular* resolution, first studied in Tseitin's foundational 1968 paper, in which any path in the proof cannot resolve on the same variable more than once, and *linear* resolution, in which each resolution step must use the clause created in the previous step. Exponential lower bounds that separate these subsystems from tree resolution and from general resolution exist for each of them (see [**3, 21, 17**]).

Weak Pigeonhole Principle. For PHP_n^m, the resolution lower bounds proved here are non-trivial only when $m < n^2/\log n$. What happens when $m \gg n$, e.g. $m = 2^{n^\alpha}$? This is a fundamental question since proof systems unable to prove these weak versions of the pigeonhole principle cannot express the difference between polynomial-size and exponential-size quantities. Exponential lower bounds on the resolution proof size of this very weak pigeonhole principle were first given for regular resolution [**45**], then extended to general resolution [**49**] and improved in [**52, 54**]. Following a framework given in [**51**], this shows that resolution cannot prove natural formalizations of $\mathsf{NP} \not\subset \mathsf{P/poly}$, since in particular it cannot prove that SAT is not computable by a polynomial-size DNF formula. In the other direction, it is known that PHP_n^m has quasi-polynomial size depth 2 Frege proofs for $m \geq (1+\epsilon)n$ [**44, 43**] and it is not known whether polynomial-size AC^0-Frege proofs of this weak pigeonhole principle are possible. For much more on these questions, see the excellent survey [**53**].

Polynomial Calculus and PCR. The only methods given for explicit bounds for these proof systems in Lecture 2 were for binomial systems of equations. Also, the lower bounds for random formulas based on parity equations did not work for characteristic 2. Some general criteria for polynomial calculus lower bounds have been given in [**4**] and these also yield exponential lower bounds for random formulas in characteristic 2.

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 or 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. It seems reasonable, however, to conjecture that random formulas are hard even for Frege systems.

Space in Proof Complexity. Natural notions of the space required to produce proofs in resolution and PCR are given in [**32, 2**]. In particular, a tight relationship is shown between tree-resolution proof size and clause space. Using this relationship, the lower bound for DLL proofs of random formulas given in Lecture 2 has been improved to roughly match the upper bound [**11**]. This does not seem to improve the bounds for general resolution.

AC^0**-Frege.** We are far from a general method for AC^0-Frege lower bounds. The only method we described here for AC^0-Frege lower bounds has involved restrictions and switching lemmas. Recently, Ben-Sasson has given a useful notion of reduction among tautologies applicable to AC^0-Frege and used it to show exponential lower bounds for the Tseitin tautologies for random graphs.

Cutting Planes. Our only lower bounds for this system are based on interpolation and the tautologies involved are not as natural as in other lower bounds. Is there another method that can be applied to this system? Can one prove lower bounds for arbitrary Tseitin tautologies?

Lovasz-Schriver Proof Systems. [**42**]. These systems are like cutting planes but based on 01-programming. Initial inequalities and the goal 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 adding the square of any linear quantity. This system polynomially simulates resolution and also can prove PHP_n^{n+1} efficiently. It has feasible interpolation and hence is not polynomially bounded given $\mathsf{NP} \not\subset \mathsf{P/poly}$. However, no hard tautology is known for it. One might try to prove $Count_2^{2n+1}$ is hard for these systems. A related question is to see if these systems can polynomially simulate cutting planes.

Improved Proof Search. Can we build better algorithms to beat the DLL algorithms in practice by using properties of PCR? Are there other restrictions of resolution that can be made more effective in practice?

Harder Questions. Can one prove lower bounds for $\mathsf{AC}^0[p]$-Frege? It is natural to conjecture that $Count_q^{qn+1}$ is hard for prime $q \neq p$. Can one prove lower bounds for TC^0-Frege or Frege in general? A candidate for this, suggested by Cook, could be the fact that $AB = I \Rightarrow BA = I$ for Boolean matrix multiplication.

BIBLIOGRAPHY

1. Miklós Ajtai. The complexity of the pigeonhole principle. In *29th Annual Symposium on Foundations of Computer Science*, pages 346–355, White Plains, NY, October 1988. IEEE.
2. M. Alekhnovich, E. Ben-Sasson, A. A. Razborov, and A. Wigderson. Space complexity in propositional calculus. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 358–367, Portland,OR, May 2000.
3. M. Alekhnovich, J. Johannsen, T. Pitassi, and A. Urquhart. An exponential separation between regular and general resolution. Technical Report TR01-56, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2001.
4. M. Alekhnovich and A. A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. In *Proceedings 42nd Annual Symposium on Foundations of Computer Science*, pages 190–199, Las Vegas, Nevada, October 2001. IEEE.
5. Noga Alon and Ravi Boppana. The monotone complexity of Boolean functions. *Combinatorica*, 7(1):1–22, January 1987.
6. P. Beame, R. Karp, T. Pitassi, and M. Saks. On the complexity of unsatisfiability of random k-CNF formulas. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 561–571, Dallas, TX, May 1998.
7. Paul Beame and Søren Riis. More on the relative strength of counting principles. In Paul W. Beame and Samuel R. Buss, editors, *Proof Complexity and Feasible Arithmetics*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 13–35. American Mathematical Society, 1998.
8. Paul W. Beame and Toniann Pitassi. An exponential separation between the parity principle and the pigeonhole principle. *Annals of Pure and Applied Logic*, 80:197–222, 1996.
9. Paul W. Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *Proceedings 37th Annual Symposium on Foundations of Computer Science*, pages 274–282, Burlington, VT, October 1996. IEEE.
10. Paul W. Beame and Toniann Pitassi. Propositional Proof Complexity: Past, Present, Future. *EATCS Bulletin*, 1998. To appear.
11. E. Ben-Sasson and N. Galesi. Space complexity of random formulae in resolution. In *ProceedingsSixteenth Annual IEEE Conference on Computational Complexity*, pages 42–51, Chicago, IL, June 2001.

12. E. Ben-Sasson and R. Impagliazzo. Random CNF's are hard for the polynomial calculus. In *Proceedings 40th Annual Symposium on Foundations of Computer Science*, pages 415–421, New York,NY, October 1999. IEEE.
13. E. Ben-Sasson and A. Wigderson. Short proofs are narrow – resolution made simple. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 517–526, Atlanta, GA, May 1999.
14. E. Biham, D. Boneh, and O. Reingold. Breaking generalized Diffie-Hellmann modulo a composite is no easier than factoring. *Information Processing Letters*, 70(2):83–87, 1999.
15. M. L. Bonet, C. Domingo, R. Gavalda, A. Maciel, and T. Pitassi. Non-automatizability of bounded-depth Frege proofs. In *ProceedingsFourteenth Annual IEEE Conference on Computational Complexity (formerly: Structure in Complexity Theory Conference)*, pages 15–23, Atlanta, GA, May 1999.
16. M. L. Bonet, J. L. Esteban, N. Galesi, and J. Johansen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000.
17. M. L. Bonet and N. Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *Proceedings 40th Annual Symposium on Foundations of Computer Science*, New York,NY, October 1999. IEEE.
18. M. L. Bonet, T. Pitassi, and R. Raz. No feasible interpolation for TC^0 frege proofs. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE, October 1997.
19. M. L. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for Frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000.
20. R. Boppana and M. Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science (Volume A), Ed. Jan van Leeuwen*, pages 757–804. Elsevier and MIT Press, 1990.
21. J. Buresh-Oppenheim, D. Mitchell, and T. Pitassi. Linear and negative resolution are weaker than resolution. Technical Report TR01-074, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2001.
22. S. Buss, D. Grigoriev, R. Impagliazzo, and T. Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 547–556, Atlanta, GA, May 1999.
23. S. Buss, R. Impagliazzo, J. Krajíček, P. Pudlák, A. A. Razborov, and J. Sgall. Proof complexity in algebraic systems and bounded depth Frege systems with modular counting. *Computation Complexity*, 6(3):256–298, 1997.
24. Samuel R. Buss. Polynomial size proofs of the pigeonhole principle. *Journal of Symbolic Logic*, 57:916–927, 1987.
25. V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4, 1973.
26. V. Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.
27. M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 174–183, Philadelphia, PA, May 1996.
28. Stephen A. Cook. The complexity of theorem proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158,

Shaker Heights, OH, May 1971.

29. Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1977.
30. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
31. M. Davis and H. Putnam. A computing procedure for quantification theory. *Communications of the ACM*, 7:201–215, 1960.
32. J. L. Esteban and J. Toran. Space bounds for resolution. In *(STACS) 99: 16th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 530–539, Trier, Germany, March 1999. Springer-Verlag.
33. E. Friedgut. Sharp thresholds of graph properties, and the k-sat problem. *Journal of the American Mathematical Society*, 12:1017–1054, 1999.
34. O. Gabber and Z. Galil. Explicit constructions of linear size superconcentrators. In *20th Annual Symposium on Foundations of Computer Science*, pages 364–370, New York, NY, 1979. IEEE.
35. R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.
36. D. Grigoriev. Tseitin's tautologies and lower bounds for Nullstellensatz proofs. In *Proceedings 39th Annual Symposium on Foundations of Computer Science*, pages 648–652, Palo Alto, CA, November 1998. IEEE.
37. A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–305, 1985.
38. Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20, Berkeley, CA, May 1986.
39. J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, June 1997.
40. J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *J. Symbolic Logic*, 54(3):1063–1079, 1989.
41. J. Krajíček, P. Pudlák, and A Woods. Exponential lower bounds to the size of bounded depth Frege proofs of the pigeonhole principle. *Random Structures and Algorithms*, 7(1), 1995.
42. L. Lovasz and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM J. Optimization*, 1(2):166–190, 1991.
43. A. Maciel, T. Pitassi, and A. Woods. A new proof of the weak pigeonhole principle. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 368–377, Portland,OR, May 2000.
44. J.B. Paris, A. J. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *Journal of Symbolic Logic*, 53:1235–1244, 1988.
45. T. Pitassi and R. Raz. Lower bounds for regular resolution proofs of the weak pigeonhole principle. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 347–355, Hersonissos, Crete, Greece, July 2001.
46. Toniann Pitassi, Paul W. Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3(2):97–140, 1993.

47. P. Pudlák and J. Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In Paul W. Beame and Samuel R. Buss, editors, *Proof Complexity and Feasible Arithmetics*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 279–295. American Mathematical Society, 1998.
48. Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, September 1997.
49. R. Raz. Resolution lower bounds for the weak pigeonhole principle. Technical Report TR01-021, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2001.
50. A. A. Razborov. Lower bounds on the monotone complexity of some boolean functions. *Doklady Akademii Nauk SSSR, n.s.*, 281(4):798–801, 1985. In Russian; English translation in *Soviet Math. Dokl. 31*, 354–357.
51. A. A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998.
52. A. A. Razborov. Improved resolution lower bounds for the weak pigeonhole principle. Technical Report TR01-055, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2001.
53. A. A. Razborov. Proof complexity of pigeonhol principles. In *Proceedings of the Fifth International Conference on Developments in Language Theory*, Vienna, Austria, July 2001.
54. A. A. Razborov. Resolution lower bounds for the weak functional pigeonhole principle. Technical Report TR01-075, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2001.
55. G. S. Tseitin. On the complexity of derivation in the propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*. 1968.
56. A. Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, December 1995.