Propositional Proof Complexity

Instructor: Alexander Razborov, University of Chicago.
razborov@cs.uchicago.edu
Course Homepage:
`http://people.cs.uchicago.edu/~razborov/teaching/winter09.html`

Winter, 2009 and Winter, 2014

Contents

Lecture 1
Scribe: Raghav Kulkarni, University of Chicago.
Date: January 6, 2009

1 Global Picture

Question: Intuitively, what makes a proof easy or hard ?
a) Length of the proof (e.g., 5 pages vs 10 pages)
b) Logical Complexity (e.g., number of quantifiers used)
c) Conceptual Complexity (e.g., complexity of the mathematical concepts used).

Types of Proof:
1. Proof that uses Quantifiers (e.g., first-order theories)
2. Quantifier-free proof (e.g., propositional proof).

In the first part of this course we will briefly study the proofs of type 1 but the majority of this course is intended to study the proofs of type 2. We will present one particular result in the course which will bridge the part 1 with part 2. (This result is roughly of the flavor that "the circuits and the Turing machines are exactly the same in some precise sense.")

Outreach:
a) Lovasz-Schrijver rank : This notion of rank is associated with a positive semidefinite relaxation of integer linear programs. One consecutively shrinks the polytope by applying some rules. The rank is roughly the number of such shrinks.
b) Applications in Learning.

We begin with a "movie trailer" of the second part.

2 Size-Width Tradeoff for Resolution

Let $x_1, \ldots, x_n$ be *propositional variables*, i.e., $x_i \in \{0, 1\}$.
A *literal* is either x_i or $\neg x_i$ (sometimes we write $\neg x$ as $\bar{x}$).

Let $\epsilon \in \{0, 1\}$. $x_i^\epsilon = x_i$ if $\epsilon = 1$ otherwise $\neg x_i$.
A *clause C* is a disjunction of literals: $x_{i_1}^{\epsilon_1} \vee \ldots \vee x_{i_w}^{\epsilon_w}$.
The empty clause is denoted by 0.
A *conjunctive normal form* (CNF) formula τ is a conjunction of clauses: $C_1 \wedge \ldots \wedge C_m$. A CNF-formula is *unsatisfiable* if no assignment of its variables evaluates to 1 (1 = True). For example, the empty clause 0 is unsatisfiable. An unsatisfiable formula is also called a *contradition*.

2.1 Resolution Proof System

Question: How does one prove that a CNF-formula τ is unsatisfiable?
We prove that τ is unsatisfiable by deducing the empty clause 0 from its clauses by a series of rules of the following two types:

$$\text{Resolution Rule: } \frac{C \vee x \qquad D \vee \bar{x}}{C \vee D} \qquad\qquad \text{Weakening: } \frac{C}{C \vee D}.$$

Note that one can arrange the intermediate formulae either in a directed acyclic graph (DAG) or unfold the DAG to form a *tree-like* proof system.

(Soundness Theorem) If one can deduce 0 from τ by applying the above rules, then τ is unsatisfiable.

(Completeness Theorem) If τ is unsatisfiable then $\tau \vdash 0$, i.e., one can deduce 0 from τ. (Homework 1.)

Let P be a resolution(refutation) of τ. We denote by $|P|$ the total number of clauses in P.

$S_R(\tau) = \min\{|P| \ : \ P \ is \ a \ refutation \ of \ \tau\}$.

Question: Are there short CNF-formulae which require large refutation ?

Historical background:

Tseitsin [68] - answered the question for *regular* resolutions.

Haken [85] - constructed τ such that all of its refutations are exponential.

Chvatal-Szemeredi [88] - random CNF is almost surely unsatisfiable and has only exponentially long refutation proofs.

The above results are extremely nice and important but somewhat ad hoc.

2.2   Width: Another Complexity Measure

The width of a clause: $w(x_1^{\epsilon_1} \vee \ldots \vee x_{i_w}^{\epsilon_w}) = w$.

$w(\tau) = \max\{w(C) \ : \ C \in \tau\}$.

$w(P) = \max\{w(C) \ : \ C \in P\}$.

$w_R(\tau) = \min\{w(P) \ : \ P - refutation \ of \ \tau\}$.

Question: What is the relation between $w(\tau)$ and $w_R(\tau)$?

If we use all clauses in τ then $w_R(\tau) \geq w(\tau)$ but it is not true in general. One might not use all clauses of τ for its refutation, one might be able to deduce 0 from a subset of its clauses.

Question: What is the relation between $S_R(\tau)$ and $w_R(\tau)$?

It is easy to see that $S_R(\tau) \leq (2n)^{w_R(\tau)}$.

Question: Does the above inequality hold in the other direction ?

Unfortunately, $w_R(\tau) \leq log_n S_R(\tau)$ is not true exactly, but the following theorem is known.

Theorem 1 (Ben-Sasson Wigderson 99)**.** For any n-variable contradiction τ, $w_R(\tau) = O(\sqrt{n \log S_R(\tau)})$.

If τ is a contradiction and $w_R(\tau) = \Omega(n)$ then $S_R(\tau) = \exp(\Omega(n))$.

4

Remark: Bonet, Galesi [99] - the bound in Theorem 1 is almost tight. In particular, for every n, there is an n-variable contradiction τ_n such that $S_R(\tau_n) = n^{O(1)}$ but $w_R(\tau_n) = \Omega(\sqrt{n})$.

2.3 Restrictions

For a clause C, its *restriction* to $x = \epsilon$ is defined as follows:
$C|_{x=\epsilon} = C$ if $x \notin C$; 1 if $C = D \vee x^{\epsilon}$; D if $D = D \vee x^{1-\epsilon}$.
If $\tau = C_1 \wedge \ldots \wedge C_m$, then $\tau|_{x=\epsilon} = C_1|_{x=\epsilon} \wedge \ldots \wedge C_m|_{x=\epsilon}$.

Exercise: If τ is a contradiction then so is $\tau|_{x=\epsilon}$. Moreover, the restriction of a refutation (of τ) is again a refutation (of $\tau|_{x=\epsilon}$).

Lemma 2.1 (Restriction is simpler to refute). $w_R(\tau|_{x=\epsilon}) \leq w_R(\tau)$.

It is interesting that one can prove the following lemma in the other direction.

Lemma 2.2. $w_R(\tau) \leq \max\{w_R(\tau|_{x=0}), w_R(\tau|_{x=1})\} + 1$.

Exercise: Prove Lemma 2.1 and Lemma 2.2.

Next time, we will show that a slightly stronger version of the above lemma holds. In particular, we will prove that:
if $w_R(\tau|_{x=\epsilon}) \leq w - 1$ and $w_R(\tau|_{x=1-\epsilon}) \leq w$, then $w_R(\tau) \leq w$.
We will see in the next lecture that this crucial inequality leads to the proof of Theorem 1.

Lecture 2
Scribe: Raghav Kulkarni, University of Chicago.
Date: January 8, 2009

Lemma 2.3. *If $w_R(\tau|_{x=\epsilon}) \leq w - 1$ and $w_R(\tau|_{x=1-\epsilon}) \leq w$, then $w_R(\tau) \leq w$.*

Proof: The idea is to do the refutations sequentially. First deduce $x^{1-\epsilon}$ from τ using clauses of width at most w. (Follow closely the deduction from

5

$\tau|_{x=\epsilon}$ to 0, which uses clauses of length at most $w - 1$. Add $x^{1-\epsilon}$ to every clause in such a deduction.) Now, from $x^{1-\epsilon}$ and τ, deduce $\tau|_{x=1-\epsilon}$ by using the resolution rule. This step does not introduce any clause of width more than w. Finally, deduce 0 from $\tau|_{x=1-\epsilon}$ using clauses of width at most w.

Fat Clause: C is called *fat* if $w(C) > d$. (d is a parameter to be specified later)

Lemma 2.4. *If P is a refutation of τ that contains less than a^b fat clauses then $w_R(\tau) \leq d + b$ where $a = (1 - \frac{d}{2n})^{-1}$.*

    Proof: We do induction on b and n.
Base case: $b = 0 \Rightarrow$ no fat clauses $\Rightarrow w_R(\tau) \leq d$.
Let $C_1, C_2, \ldots, C_k$ be all the fat clauses in P. Note that, out of the $2n$ literals, there exists x^ϵ such that x^ϵ is contained in $\frac{d}{2n} \times k$ clauses. Without loss of generality, let $x^\epsilon = x$ be such a literal. By induction on b we have $w_R(\tau|_{x=1}) \leq d + b - 1$ and by induction on n we have $w_R(\tau|_{x=0}) = d + b$. From Lemma 2.3, $_R(\tau) \leq d + b$. $\qquad\square$

Now we begin the proof of Theorem 1. *Proof:* Let $a^b = S_R(\tau)$
$\Rightarrow b = \frac{\log S_R(\tau)}{\log a} = O(\frac{n}{d} \log S_R(\tau))$
$\Rightarrow w_R(\tau) = O(d + \frac{n}{d} \log S_R(\tau))$.
Choose $d = \sqrt{n \log S_R(\tau)}$ to minimize the RHS. $\qquad\square$

Corollary: $w_R(\tau) = \Omega(n) \Rightarrow S_R(\tau) = \exp(\Omega(n))$.

Applications of Theorem 1:
Usually it is easier to prove a lower bound on width than on size. From Theorem 1, one can deduce a lower bound on the size from a lower bound on the width. The above theorem is used in proving lower bounds for the size of the refutations of the following:
a) Pigeon Hole Principle
b) Random 3-CNF
c) Tseitin tautologies (introduced in 1968).
    For an alternate way of converting width lower bounds into size lower bounds see Appendix A.1.

2.4   Edge Expansion Gives a Width Lower Bound

Let G be a fixed graph on n vertices. Let $\{x_e \mid e \in E(G)\}$ be the set of variables. Let $f : V(G) \to \{0,1\}$ be such that $\oplus_{v \in V(G)} f(v) \equiv 1 \bmod 2$. Let

$\tau(G, f) \equiv (\forall v \in V(G))(\oplus_{e \ni v} x_e = f(v))$. Suppose we want to refute $\tau(G, f)$. Let $d = \max\{deg_G(v) \mid v \in V(G)\}$. It is easy to see that $\tau(G, f)$ is a d-CNF with at most $\frac{dn}{2}$ variables and at most $n \times 2^{d-1}$ clauses.

Edge Expansion: $e(G) = \min_{\frac{n}{3} \leq |V| \leq \frac{2n}{3}} e(V, V(G) \backslash V)$.

Theorem 2 (Ben-Sasson and Wigderson 99)**.** For any connected G, $w_R(\tau(G, f)) \geq e(G)$.

    Axioms:
$\oplus_{v \in V(G)} f(v) \equiv 1 \ mod \ 2$.
$A_v \ : \ \oplus_{e \in v} x_e \equiv f(v) \ mod \ 2$.

A Measure on Clauses:
For a clause C, $\mu(C) = \min\{|V| \ : \ \{A_v \mid v \in V\} \models C\}$.

Exercise: If $C \in \tau(G, f)$ then $\mu(C) \leq 1$.

Exercise: If G is connected then $\mu(0) = n$.

Exercise: $(\mu(C \vee x) \leq a \& \mu(D \vee \bar{x}) \leq b) \Rightarrow \mu(C \vee D) \leq a + b$.

Exercise: *(Intermediate Clause:)* $(\exists C)(\frac{n}{3} \leq \mu(C) \leq \frac{2n}{3})$.

Choose any intermediate C. Let V' be a *minimal* set such that $\{A_v \mid v \in V'\} \models C$. Since C is inetrmediate, we have: $\frac{n}{3} \leq |V'| \leq \frac{2n}{3}$. Let $e' = (v', v) \in (V', V \backslash V')$.
Claim: x_e appears in C.
Proof of Claim: By the minimality of V', there exists an assignment $\{X_e \mid e \in E(G)\}$ which satisfies all the axioms in V' except $A_{v'}$ and falsifies C. Suppose for the sake of contradiction that $x_{e'} \notin C$. Note that flipping $X_{e'}$ satisfies all the axioms of V' and thus satisfies C. This is a contradiction because the new assignment still agrees with the old assignment for all variables of C, whereas the old assignment does not satisfy C. $\qquad \square$
Thus, $w(C) \geq e(G)$, which completes the proof of Theorem 2. $\qquad \square$
    We note that there exist explicit connected graphs G with $d = 3$ and $e(G) \geq \Omega(n)$. For any such graph, $\tau(G, f)$ is exponentially hard to refute in Resolution.

3 Bounded Arithmetic: Theory of Feasible Proofs

In the next two lectures, we will do the *witnessing theorems* connecting the following two:
a) Logical Complexity
b) Conceptual Complexity
For the purpose of this course, Conceptual Complexity $\equiv$ Computational Complexity, i.e., intuitively, the concepts that are *easy to compute* are simple. Moreover, most often *easy to compute* means poly-time computable.

Example: $(\forall x)(1 < x < p \Rightarrow x \nmid p) \ \Rightarrow (\forall y)(1 < y < p \Rightarrow y^{p-1} \equiv 1 \ mod \ p)$. In the above statement, all the concepts are poly-time computable. For example, y^{p-1} can be computed in poly-time by iterated squaring. Thus the above statement is *feasible.*
On the other hand, the standard proof of already $ord_y(p) = p - 1$ is strikingly not feasible. We will see later on that, unless FACTORING is easy, Fermat's Little Theorem does not possess *any* feasible proof in the strict sense we are defining.

First Order Theory: A first order theory contains the following:
$\neg, \wedge, \vee, \Rightarrow, (\forall x), (\exists x)$ and arithmetic over integers.
Language: The language of bounded arithmetic consists of the following:
$S, 0, +, \ \cdot \ , |x|, \lfloor x/2 \rfloor, \#, \leq$, where S is the successor function, i.e., $S(x) = x + 1$, $|x| = $ "bit-size of x" and $\#(x,y) = 2^{|x| \cdot |y|}$.

Lecture 3
Scribe: Joshua A. Grochow, University of Chicago.
Date: January 13, 2009

In these two lectures we continue with bounded arithmetic. We start by noting that without the $\#$ operation, if t is any term, then $t(x_1, \ldots, x_n) < (x_1 + \cdots + x_n)^{O(1)}$, corresponding to linear time. So in order to capture polynomial time, we need some way to increase bit-sizes faster, which is exactly what $\#$ allows us to do. Note that $|x \# y| = |x| |y|$; with $\#$, we get $|t(x_1, \ldots, x_n)| \leq (|x_1| + \cdots + |x_n|)^{O(1)}$, corresponding to polynomial time, as desired.

3.1 Axioms for Bounded Arithmetic

All bounded arithmetics include the axioms BASIC$_2$: these are 32 axioms that describe the properties of the logical symbols S, 0, $+$, $\cdot$, $|x|$, $\lfloor x/2 \rfloor$, $\#$,

and $\leq$. Note that these axioms are all quantifier-free (which semantically implies they are implicitly universally quantified, but we don't want to count such implicit quantifiers when we talk about quantifier alternations). Rather than list out all the axioms, we ran a PCP protocol with the following outputs:

$$\begin{aligned}
\text{Axiom 17} \quad &\text{is} \quad |x| = |y| \Rightarrow x\#z = y\#z \\
\text{Axiom 2} \quad &\text{is} \quad x \neq Sx \\
\text{Axiom 32} \quad &\text{is} \quad x = \lfloor y/2 \rfloor \Rightarrow (2 \cdot x = y \vee S(2 \cdot x) = y).
\end{aligned}$$

Throughout, we'll make use of some standard abbreviations for quantifiers, namely, the bounded quantifiers:

$$\left. \begin{aligned}
(\forall x \leq t)A(x) \quad &= \quad (\forall x)(x \leq t \Rightarrow A(x)) \\
(\exists x \leq t)A(x) \quad &= \quad (\exists x)(x \leq t \wedge A(x))
\end{aligned} \right\} \text{ ``bounded quantifiers''}$$

and the sharply bounded quantifiers:

$$\left. \begin{aligned}
(\forall x \leq |t|)A(x) \quad &= \quad (\forall x)(x \leq |t| \Rightarrow A(x)) \\
(\exists x \leq |t|)A(x) \quad &= \quad (\exists x)(x \leq |t| \wedge A(x))
\end{aligned} \right\} \text{ ``sharply bounded quantifiers''}$$

Bounded arithmetics also have induction axioms, which are of the form

$$\text{IND: } [A(0) \wedge (\forall x)(A(x) \Rightarrow A(Sx))] \Rightarrow (\forall x)A(x)$$

In fact, this an axiom *scheme*, as there is one axiom for each formula A. Which types of formulae we allow for A determines the strength of the induction axiom scheme we use.

The class Σ_i^b consists of those formulae that start with an existential quantifier, and have i quantifier alternations, where we ignore sharply bounded quantifiers. Π_i^b is defined similarly, except that the formulae in Π_i^b begin with a universal quantifier. $\Sigma_k^b - \text{IND}$ is the induction axiom scheme for all formula $A \in \Sigma_k^b$.

The bounded arithmetic T_2^k consists of the basic axioms plus $\Sigma_k^b\text{-IND}$. This type of induction is *not* polynomial however. In order to prove $A(n)$, we have to prove $A(0), A(1), \ldots, A(n-1)$ first, which gives proofs of length $O(n)$, but that is *exponential* in the bit-length of n. To remedy this, we introduce a polynomial induction scheme, which allows induction based on the bit representation, as follows:

$$\text{PIND: } [A(0) \wedge (A(\lfloor x/2 \rfloor) \Rightarrow A(x))] \Rightarrow (\forall x)A(x)$$

The bounded arithmetic we will be interested in is S_2^k, which consists of the basic axioms plus $\Sigma_k^b\text{-PIND}$.

3.2 The Main Theorem of Bounded Arithmetic

The main theorem of bounded arithmetic essentially says that S_2^1 captures polynomial-time computations:

Theorem 3.

(a) If $f(x_1, \ldots, x_k)$ can be computed in polynomial time, then there exists a Σ_1^b formula $A(x_1, \ldots, x_k, y)$ such that $\mathbb{N} \models A(n_1, \ldots, n_k, f(n_1, \ldots, n_k))$ for all $n_1, \ldots, n_k \in \mathbb{N}$ and $S_2^1 \vdash \forall x_1, \ldots, x_k \exists! y A(x_1, \ldots, x_k, y)$. That is, S_2^1 proves that A is total and single-valued.

(b) If $A(x_1, \ldots, x_k, y)$ is a Σ_1^b formula and $S_2^1 \vdash \forall x_1, \ldots, x_k \exists y A(x_1, \ldots, x_k, y)$, then there is a polynomial-time computable function f such that $\models A(n_1, \ldots, n_k, f(n_1, \ldots, n_k))$, and furthermore, $S_2^1 \vdash A(n_1, \ldots, n_k, f(n_1, \ldots, n_k))$.

Theorems of this sort are called "witnessing theorems." Note that in (b), we only require that S_2^1 prove for each $x_1, \ldots, x_k$, that there is some y satisfying A, not necessarily a unique one.

Recall Fermat's Little Theorem, which we can write as the Σ_1^b formula: $x^{n-1} \not\equiv 1 \pmod{n} \Rightarrow (\exists y \leq n)(y|n)$. As a corollary to the above witnessing theorem, we have:

Corollary 4. If integer factoring is hard (not in polynomial time), then S_2^1 does not prove Fermat's Little Theorem.

Outline of proof of main theorem. To prove (a), use the definition of P based on limited iteration (due to Cobham, 1965). Consider the following class of functions $f\colon \mathbb{N}^r \to \mathbb{N}$:

- The constant function 0

- The successor function $x \mapsto Sx$

- Bit-shift $x \mapsto \lfloor x/2 \rfloor$

- Multiplication by 2, $x \mapsto 2x$

- Conditional Choice$(x, y, z) = y$ if $x > 0$ and z if $x = 0$

- The characteristic function of $\leq$, $\chi_{\leq}(x, y)$

Cobham showed that the closure of these functions under composition and limited iteration (or recursion) is exactly equal to the class of polynomial-time functions. Before defining limited iteration, we define (unlimited) iteration: given $g\colon \mathbb{N}^r \to N$ and $h\colon \mathbb{N}^{r+2} \to \mathbb{N}$, we can define $\tau\colon \mathbb{N}^{r+1} \to \mathbb{N}$ by

$$\tau(x, 0) = g(x) \text{ and } \tau(x, n+1) = h(x, n, \tau(x, n)).$$

Using limited iteration, such τ can be constructed iff there are polynomials p and q such that $(\forall n \leq p(|x|))(|\tau(x, n)| \leq q(|x|)$. This forbids constructions like $\tau(0) = 2$ and $\tau(n+1) = \tau(n)^2$, whose lengths blow up super-polynomially.

Using Cobham's recursive definition, we can prove (a) first for the functions listed above, and then show that if (a) holds for some functions, then it also holds for their closure under composition and limited iteration.

(Historical aside: Cobham's definition gave rise to so-called "machine-independent complexity theory" and really got logicians interested in complexity.)

We will finish the proof of (a) and outline the proof of (b) next lecture.

$\square$

Lecture 4
Scribe: Joshua A. Grochow, University of Chicago.
Date: January 15, 2009

This lecture will be devoted to finishing the proof of the main witnessing theorem of bounded arithmetic (that S_2^1 exactly captures polynomial-time computation). Before continuing, we need a few technical constructions.

3.2.1 Bootstrapping by Introducing New Symbols

Adding new symbols to our language by bootstrapping is essentially only a matter of convenience, but it makes the proofs much more intelligible. If a theory T proves $(\forall x)(\exists! y)A(x, y)$, then we can add a new function symbol $f_A(x)$ and axioms $(\forall x)A(x, f_A(x))$ without altering what can be proved in the theory. That is, if a formula B does not contain f_A and $T + (\forall x)A(x, f_A(x)) \vdash B$, then $T \vdash B$.

When we add a new symbol, we have to be careful how we use induction. In particular, Σ_1^b-PIND only allows polynomial induction for formulae in the original language, and not formulae with the new symbols. The following lemma takes care of this technicality, as well as most other technicalities we might worry about when introducing new symbols in this way:

11

Lemma 3.1. *Let $A \in \Sigma_1^b$ be such that $S_2^1 \vdash (\forall x)(\exists! y)A(x,y)$, and let $T = S_2^1 + (\forall x)A(x, f_A(x))$. Then for every $B \in \Sigma_1^b(f_A)$, there is a $B^* \in \Sigma_1^b$ such that $T \vdash (B \equiv B^*)$.*

Note: we can actually drop the uniqueness from the above statement, but the lemma as stated is all we'll need.

Hint of proof. We'll see later that $S_2^1 \vdash (\forall x)(\exists! y)A(x,y)$ actually lets us put a bound on y, that is $S_2^1 \vdash (\forall x)(\exists! y \leq t)A(x,y)$.

The idea is to remove each occurrence of f in B one at a time. Suppose B looks like $blah_1 f(t_1, \ldots, t_r)blah_2$. Then an equivalent formula is

$$\exists z \leq t(t_1, \ldots, t_r)[A(t_1, \ldots, t_r, z) \wedge (blah_1 \quad z \quad blah_2)]$$

We've reduced the number of occurences of f by one, and we haven't increased the complexity since we only added an existential quantifier. (Note that, since Σ_1^b formulae start with an existential quantifier, adding another one does not change the complexity of the formula.) Wash, rinse, repeat. $\square$

For example, suppose we wish to add the "monus" symbol:

$$x \dotminus y = \begin{cases} x - y & \text{If } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

Then $A(x, y, z) \equiv (y + z = x) \vee (x \leq y \wedge z = 0)$ is a $\Sigma_0^b \subseteq \Sigma_1^b$ formula. Existence and uniqueness can be expressed as $[A(x,y,z) \wedge A(x,y,z')] \Rightarrow z = z'$ and $(\exists z)A(x,y,z)$, respectively. Both of these follow from the basic axioms of bounded arithmetic. However, it is hard to show bounded uniqueness in the form $(\exists z \leq x)A(x,y,z)$; it is easy to prove in T_2^1, but proving this in S_2^1 seems nearly as hard as proving the whole witnessing theorem. This is actual a more general phenomena: the more powerful the theory, the easier it is to bootstrap. When people want to make this harder (to see if it can still be done), they consider theories such as $I\Delta_0$, which is $S_2 = \bigcup_k S_2^k$ but without $\#$.

3.2.2 Finite Sequences

Now we discuss the encoding of finite sequences in bounded arithmetic, e.g. $\langle x_1, \ldots, x_n \rangle$ where $n \leq |t|$ for some term t. For this we introduce (after a fair amount of bootstrapping in the style of the previous section) three functions to our language:

- UniqSeq(w): a predicate indicating that w is a valid code of a sequence;

- $\beta(i, w)$: projection function, $\beta(i, \langle x_1, \ldots, x_n \rangle) = x_i$; and

- Len(w): the length of w (that is, n).

To complete the proof that all of Cobham's hierarchy (and hence all of P) is captured by S_2^1, we essentially need to show two basic facts about finite sequences. First, that they are determined by their projections, namely:

$$(\text{UniqSeq}(A) \wedge \text{UniqSeq}(B) \wedge \text{Len}(A) = \text{Len}(B) \wedge (\forall i \leq \text{Len}(A))(\beta(i, A) = \beta(i, B))) \Rightarrow A = B$$

and second, we need to show Σ_1^b-*replacement* (which is essentially the bounded version of Axiom of Choice). Namely, we'd like to show that for any $A(x, y) \in \Sigma_1^b$, S_2^1 proves

$$(\forall x \leq |t|)(\exists y \leq s)(A(x, y)) \Rightarrow (\exists w \leq \text{SeqBd}(s, t))(\forall x \leq |t|)(A(x, \beta(x, w)) \wedge \beta(x, w) \leq s)$$

where SeqBd is the (bound on) largest w representing a sequence of length $\leq |t|$ and whose elements are all at most s.

With these powerful tools at our disposal, the proof of part a) becomes easy.

3.2.3 Hentzen Calculus

To prove part (b) of the main witnessing theorem, we will use cut elimination and Hentzen natural proof systems. In the Hentzen calculus, an implication such as $(A_1 \wedge \cdots \wedge A_k) \Rightarrow (B_1 \vee \cdots \vee B_\ell)$ is called a sequent, and we use the notation $A_1, \ldots, A_k \longrightarrow B_1, \ldots, B_\ell$.

The Hentzen calculus has one axiom scheme, $A \longrightarrow A$, and several rules of logical inference. Here are several examples of such rules:

$$\frac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta, A} \qquad \text{(contraction right)}$$

$$\frac{A, \Gamma \longrightarrow \Delta \qquad B, \Gamma \longrightarrow \Delta}{(A \vee B), \Gamma \longrightarrow \Delta} \qquad (\vee \text{ left})$$

$$\frac{a \leq t, \Gamma \longrightarrow \Delta, A(a)}{\Gamma \longrightarrow \Delta, (\forall x \leq t)A(x)} \qquad (\forall \leq \text{ right})$$

In the latter rule, note that a must not appear in either Δ or Γ. As an interesting point, without this one rule (and its "left" equivalent), first-order logic would be decidable.

Finally, we come to the (in)famous *cut rule*:

$$\frac{\Gamma \longrightarrow \Delta, A \qquad A, \Pi \longrightarrow \Lambda}{\Gamma, \Pi \longrightarrow \Delta, \Lambda}$$

Why is this rule different from all other rules? What sets this rule apart is the fact that the formula A can be arbitrarily complex. In particular, the premise of the inference rule can be *more* complicated than the consequence, which is unlike any of the other inference rules. This makes Hentzen's Theorem all the more surprising:

Theorem 5 (Hentzen). The cut rule can be eliminated from first order logic.

This proof of this theorem uses the technique of *cut elimination*, which is (luckily) available in bounded arithmetic. However, if we add PIND naively to S_2^1, the cut elimination technique does not work. We carefully add the inference rule:

$$\frac{\Gamma, A(\lfloor a/2 \rfloor) \longrightarrow A(a), \Delta}{\Gamma, A(0) \longrightarrow A(t), \Delta} \qquad \text{PIND}$$

(in which, as usual, a must not appear in Δ, Γ). Then we get limited cut elimination (in which A must be a sub-formula of one of $A(0)$, $A(t)$ or basic axioms, which in particular implies $A \in \Sigma_1^b \cup \Pi_1^b$).

Finally, we can outline the proof of part (b) of the witnessing theorem:

Outline of proof of witnessing theorem part (b). $S_2^1 \vdash \forall x \exists y A(x, y)$, which we've already noted implies $(\forall x)(\exists y \leq t)A(x, y)$.

First, if $A_i, B_j \in \Sigma_1^b$, then the sequent $A_1, \ldots, A_k \longrightarrow B_1, \ldots, B_\ell$ is equivalent to

$$(\exists y_1 \leq t_1)C_1, \ldots, (\exists y_k \leq t_k)C_k \longrightarrow (\exists z_1 \leq t_1')D_1, \ldots, (\exists z_\ell \leq t_\ell')D_\ell$$

for some terms t_i, t_i' and formula $C_i, D_j \in \Sigma_0^b$. Using the limited form of cut elimination mentioned above, we can assume that all sequents in our proof are of this form.

We prove (by induction on the inference in the Hentzen Calculus) that, for every sequent as above appearing in our proof there is a polynomial-time function $f \colon \mathbb{N}^k \to (i, b)$ such that if $f(y_1, \ldots, y_k) = (i, b)$ then taking $z_i = b$ gives a witness for B_i, whenever $y_1, \ldots, y_k$ are witnesses for $A_1, \ldots, A_k$, respectively. Note that the f constructed here has access to all the parameters t_i, t_i', C_i, D_j. $\qquad \square$

Lectures 5 and 6

Scribe: Alex Rosenfeld, University of Chicago.
Date: January 20 and 22, 2009

3.3 RSUV isomorphism

With many different ideas of how and in what language proofs should proceed (and how to determine their complexity), it would be nice to have some sort of gateway between them. This gives us the concept of "robustness", i.e. given different idea of how our proofs should operate, we should arrive at the same version of feasible provability. To that end, Takeuti and Razborov in 1992 (or is it 1892?) proved a connection between S_1^2 and a system based on treating words as sets.

The first concern in defining such a class is "what do the integers represent?". Instead of treating integers as a binary expansion (as we always did before), we will use them to identify *positions* in a binary string. Since we are not using binary expansion, we no longer need $\#, |x|$, and $\lfloor x/2 \rfloor$. However, we now have the problem that since we encode integers by position, we are no longer able to use binary strings to define words. Thus, we instead use sets of integers to encode words.

Since we are denoting integers by position, it is natural that we get bounded quantifiers for free. However, now we have the problem of talking about words. So, we resort to the language of second-order logic and introduce *comprehension axioms* as $\exists \alpha (\forall x \leq t)(A(x) \equiv \alpha(x))$ where α is a second-order variable (interpreted as a word) and A is some predicate not containing either α or any quantifiers on second-order variables.

Since we get bounded quantifiers for free, we need a new way of organizing the complexity of formulas. Now, we count quantifiers over words, using the notation $\Sigma_k^{1,b}$ in place of Σ_k^b. And we define V_1^k as the system that in addition to the $\Sigma_0^{1,b}$-CA (comprehension axiom above) also has $\Sigma_k^{1,b} - IND$ and a dozen of simple BASIC axioms describing properties of the remaining symbols $S, 0, +, \ . \ , \ \leq$.

It turns out that this system is more or less equivalent to the system we used before as shown by the following theorem:

Theorem 6. There are isomorphisms $\flat : V_1^k \to S_2^k$ and $\sharp : S_2^k \to V_1^k$ such that for any second order formula A, $\Sigma_k^{1,b} - CA \vdash A^{\flat\sharp} \equiv A$ and $S_2^k \vdash A^{\sharp\flat} \equiv A$ for any first-order A. We also get that if A is $\Sigma_k^{1,b}$ then $A^\flat$ is Σ_k^b.

15

Let's go further. Forget about even using integers at all; let's only consider binary strings. Ferreira in 1990 made a (firs-order) version of bounded arithmetics with only strings, 0, 1, concatenation, and a prefix operator. It turns out that again, we get an isomorphism with the Buss system.

So, it turns out that no matter which path we take, we always end up in Rome.

3.4   The bounded arithmetic hierarchy

Given the previous section, we can work within Buss's bounded arithmetic and do fine. However, can we get a way to compare the strengths of these systems? The following theorem develops a hierarchy of these systems.

Theorem 7. $S_2^1 \subseteq T_2^1 \subseteq S_2^2 \subseteq T_2^2 \subseteq \ldots$

Proof: Well, $S_2^k \subseteq T_2^k$ mostly follows from definition. $T_2^k \subseteq S_2^{k+1}$, however, is much less trivial to prove.

So, let's take a predicate $A(x)$ in Σ_k^b. We need to prove that $S_2^{k+1} \vdash A(0) \wedge \forall x (A(x) \Rightarrow A(x+1)) \Rightarrow A(t)$.

So, let's look at a more general form of this statement: $\forall u \leq v \leq t (((v - u) \leq y \wedge A(u)) \Rightarrow A(v))$. In other words, for a given distance y, for any u and v within that distance, $A(u)$ implies $A(v)$. And so, if we can prove this statement when $y = t$, we have our original statement.

So, how do we prove this statement, while only using polynomial induction? Well, we take a u and v with distance less than t. We find a midpoint of those numbers, w and apply our polynomial induction hypothesis on the two smaller chunks $[u, w]$ and $[w, v]$. And in this way we go from $y/2$ to y, so it really takes only polynomially many steps to go from $y = 1$ to $y = t$. Thus, we get induction in S_2^{k+1} and so $T_2^k \subseteq S_2^{k+1}$. $\square$

As a technical note, what we actually proved induction for Π_k^b statements, but that's okay since $S_2^{k+1} \vdash \Pi_{k+1}^b - PIND$ (see Homework # 1).

3.5   Does this hierarchy collapse?

The natural question given this hierarchy is whether or not it collapses. However, this problem turns out to be at least as difficult to solve as whether or not the polynomial time hierarchy collapses. Thus, we seem to know little. But the following remarkable result due to Krajicek, Pudlak and Takeuti (1991) proves that it is *at most* that difficult:

16

Theorem 8. If PH does not collapse then

$$S_2^1 \overset{\neq}{\subseteq} T_2^1 \overset{\neq}{\subseteq} S_2^2 \overset{\neq}{\subseteq} T_2^2 \ldots$$

Well, computer scientists have encountered a little bit of difficulty in proving PH doesn't collapse. However, we do know that there is an oracle C such that PH^C does not collapse, and we can perform a similar proof in our systems of bounded arithmetic.

Remember our second order formulas we used earlier? Well, an oracle for a proof system is a new predicate symbol α we can use in proofs (note that unlike in Section 3.3, we now do *not* allow quantifiers on α). And in the same paper Krajicek, Pudlak and Takeuti showed that without any assumptions we have:

Theorem 9.
$$S_2^1(\alpha) \overset{\neq}{\subseteq} T_2^1(\alpha) \overset{\neq}{\subseteq} S_2^2(\alpha) \overset{\neq}{\subseteq} T_2^2(\alpha) \ldots$$

An open question is whether we can separate the levels of this hierarchy by formulas of bounded complexity , i.e. show that $S_2^k(\alpha) \neq T_2^k(\alpha)$ via, say, a Σ_1^b-formula.

Let us give a simple combinatorial proof of the first separation in this theorem. First we need to extend Buss's theorem to polynomial time machines with oracles:

Theorem 10. If $S_2^1(\alpha) \vdash \forall x \exists y A(\alpha, x, y)$ then $\exists M \in P$ such that for any oracle L $A(L, x, M^L(x))$.

Using this we can prove (the difference between a new predicate symbol α and a new function symbol f is cosmetic and is in Homework #1):

Theorem 11. Let f be a new (undefined, i.e. with no added axioms) function symbol. $S_2^1(f) \neq T_2^1(f)$.

Proof: Let $\mathrm{PHP}(f) \equiv (\forall x < 2a)(f(x) < a) \Rightarrow (\exists x_1, x_2 < 2a)(x_1 \neq x_2 \wedge f(x) = f(y))$. This is an encoding of the classic pigeon-hole principle.

Paris, Wilkie, and Woods proved in 1988 that $T_2^2(f) \vdash \mathrm{PHP}(f)$ and Maciel, Pitasci, and Wodds in 1999 extended this into $T_2^1(f) \vdash \mathrm{PHP}(f)$. However, $S_1^2(f) \nvdash \mathrm{PHP}(f)$ as we will show by a standard adversary argument.

17

Let M^f be a machine that asks for f at x then tries to find another point that collides with x. Since this machine works in $(\log a)^{O(1)}$, we may make sure it never gets to the point where we must have a collision by answering its queries appropriately. Thus, $S_1^2(f) \nvdash \text{PHP}(f)$. $\square$

3.6 More witnessing theorems

PLS-problems were devised by Johnson, Papadimitrious, and Yannakakis in 1988. Given a finite length binary string x and an optimization program P, we define three functions, F_P, c_P, and N_P, which simulate the process of searching for its optimum. The first function, $F_P(x)$, is a function that returns feasible solutions to problem P with input x. We require three properties for such a function. First, we require that if s is a solution, then $|s| \leq p(|x|)$ for some polynomial p, i.e. our solutions are not more complicated than our input. Second, we require that $\emptyset$ be in $F_P(x)$ for technical reasons, which will be made clear later in the proof. Third, we require that "$s \in F_P(x)$" is poly-time computable. So, intuitively, our function F_P should be able to recognize useful solutions in a short amount of time. The second function, $c_P(x, s)$, is a cost function. It quantifies how good a solution s is for input x. Our "ideal" goal is to find the best solution to x, i.e. a global maximum to c_P. The third function, $N_P(x, s)$, is in some sense a "next solution" function. It takes an input and a solution and it tries to find a better solution to our input. Whenever it fails, it just stays at the current s. And so, it follows the rule that $\forall s \in F_P(x)(N_P(x, s) = s \vee c_P(x, N_P(x, s)) > c_P(x, s))$ and that N_P is poly-time computable.

With this definition, Buss and Krajicek in 1991 proved the two following theorems:

Theorem 12. Let P be a program with associated PLS problem (F_P, c_P, N_P). If $T_2^1 \vdash \forall s \in F_P(x)(N_P(x, s) = s \vee c_P(x, N_P(x, s)) > c_P(x, s))$, then $T_2^1 \vdash \forall x \exists s \in F_P(x)(N_P(x, s) = s)$.

Note that this only states the existence of a local optimum and not a global optimum. However, in T_2^1 we can actually find even a global optimum. Let $A(a)$ be $\exists s \in F_P(x)(c_P(x, s) \geq a)$. Since we defined F_P to contain $\emptyset$, we get that $A(0)$ is true. Remember that each solution has bitlength polynomial in the bitlength of our input, so we can't have an infinite amount of solutions. Thus, it is not true that $\forall x A(x)$ and so, by induction, there is

an a such that $A(a)$ is true, but not $A(a+1)$. This a is our global optimum. □

So, given a PLS problem, we can define a Σ_1^b formula. However, we can also get the reverse.

For ease of notation, let's define the predicate $opt_P(x,s) \equiv N_P(x,s) = s$, which says that s is a locally optimal solution for P with x. We can prove that:

Theorem 13. Let $A(x,y)$ be a Σ_1^b formula. If $T_2^1 \vdash \forall x \exists y A(x,y)$ then there exists a PLS problem such that $T_2^1 \vdash \forall s \in F_P(x)(N_P(x,s) = s \lor c_P(x, N_P(x,s)) > c_P(x,s))$ and $T_2^1 \vdash \forall x(\forall s \in F_P(x) \land opt_P(x,s) \Rightarrow A(x,s))$. (This trivially works for S_2^1).

Proof: Like in the proof of Buss's witnessing theorem, we first remove all essential cuts and then use induction on the inference. We consider only the most interesting case of the induction rule without side formulas:

$$\frac{A(a) \longrightarrow A(a+1)}{A(0) \longrightarrow A(t)}.$$

Well, $A(x)$ is Σ_1^b so there is a Σ_0^b formula $B(x,y)$ such that $A(x) \equiv \exists y \leq s B(x,y)$.

For notation's sake, let s_a be a witness for a, i.e. $B(a, s_a)$. By the inductive hypothesis, there exists a program P that takes $< a, s_a >$ as inputs and witnesses the assumption (that is, every local optimum s_{a+1} satisfies $B(a+1, s_{a+1})$).

We can use this P to construct a program Q that witnesses the conclusion. Let $F_Q = \{< a, s_a, s_{a+1} >: 0 \leq a \leq x \land B(a, s_a) \land s_{a+1} \in F_P(< a, s_a >)\}$, $c_Q(< a, s_a, s_{a+1} >) = c_P(< a, s_a >, s_{a+1}) + f(a)$ where f is some "offset" function that makes the witnesses of a cost less than the witnesses of $a+1$. For $N_Q(< a, s_a, s_{a+1} >)$, we applying N_P if it can further improve the solution s_{a+1} or output $< a+1, s_{a+1}, 0 >$ otherwise. (In the latter case s_{a+1} will be a witness for $a+1$). This program satisfies the theorem. □

There are many witnessing theorems out there, however, this one is particularly interesting, because it comes from optimization instead of proof complexity.

Lectures 7 and 8
Date: January 27 and 29, 2009

3.7 Transition to propositional logic

We discussed two translations from first-order logic to propositional logic, they can be found in [19, Section 6] (all references are given w.r.t the course page `http://people.cs.uchicago.edu/~razborov/teaching/winter09.html`). Along this way we reminded about the most important concrete proof systems: Frege (F), Extended Frege (EF) and bounded-depth Frege (F_d).

4 Interpolation and Automatizability

We first gave the general definition of a propositional proof system in the sense of Cook and Reckhow (74) and pointed out the connection to the NP vs. $co - NP$ question.

4.1 Feasible Interpolation

We discussed at length the concept of a disjoint NP-pair [20] and the closely related concept of Feasible Interpolation. We proved Feasible Interpolation Theorem for Resolution (in Lecture 14 we will prove a result from [28,29] that is stronger but nonetheless much more intuitive). We ended up with proving that Feasible Interpolation is not possible for Extended Frege unless factoring is easy [21].

Lectures 9, 10 and 11
Scribe: Eric Purdy, University of Chicago.
Date: Fbruary 5, 10 and 12, 2009

4.2 Automatizability

In AI, people try to determine whether a CNF is satisfiable using the Davis-Longeman-Loveland (DLL) procedure. (Certain modifications are also known as Davis-Putnam, or Davis-Putnam-Longeman-Loveland.)

The performance of Algorithm 1 depends heavily on the choice of the pivot variable - some choices lead to more simplication than others. Thus researchers in this field have tried all sorts of heuristics for choosing pivot variables. As an impossibility result, we would like to give a lower bound on the runtime of Algorithm 1, *regardless of the heuristic used.* This is captured in:

Algorithm 1 Davis-Longeman-Loveland procedure

Input: CNF τ
  if τ contains an empty clause **then**
    return false
  end if
  if τ contains a singleton clause $x_i^{\epsilon_i}$ **then**
    Set $x_i = \epsilon_i$
  end if
  Pick a pivot variable x
  Let $\tau_\epsilon = \tau|_{x=\epsilon}$ for $\epsilon = 0, 1$.
  Recur on τ_0, τ_1
  if either τ_0 or τ_1 is satisfiable **then**
    return true
  else
    return false
  end if

Theorem 14 (DLL is no better than treelike resolution)**.** The number of CNF's produced by Algorithm 1 is at least $S_{tree-R}(\tau)$, the minimal size of a treelike resolution refutation. Refuting an unsatisfiable CNF with Algorithm 1 thus takes at least $S_{tree-R}(\tau)$ steps.

Recall the definition of treelike resolution: you can only use a clause C in an application of the resolution rules once. If you want to reuse it, you have to rederive it. The graph of a treelike resolution refutation (where each vertex is a clause, and its children are the antecedents used to prove it) thus looks like a tree, where the root is the empty clause and the leaves are the axioms. The graph of a standard resolution refutation will be a DAG.

Proof. We can think of the DLL procedure as generating a tree T_{DLL}, where each node is a CNF ϕ whose children are the two restrictions of ϕ that we consider next. We recursively map T_{DLL} to a treelike resolution refutation which has as many clauses as T_{DLL} has restrictions of τ. This produces a treelike resolution refutation; thus $S_{tree-R}(\tau)$ is at most the number of nodes in T_{DLL}, which is the number of clauses produced by Algorithm 1.

If in T_{DLL} we have a restricted CNF $\tau' = \tau|_{x_1=\epsilon_1,\ldots,x_r=\epsilon_r}$, we map this to the clause $C = x_1^{1-\epsilon_1} \vee \cdots \vee x_r^{1-\epsilon_r}$, which represents the assertion "this assignment can not be extended to a satisfying assignment for τ". Inducting up from the leaves of T_{DLL}, when we want to produce the clause $x_1^{1-\epsilon_1} \vee \cdots \vee x_r^{1-\epsilon_r}$ via resolution, we use the fact that we have already produced two

children of τ' corresponding to different settings of some variable x_j. This means that we have produced, via resolution, the two clauses

$$x_j^0 \vee x_1^{1-\epsilon_1} \vee \cdots \vee x_r^{1-\epsilon_r}$$

$$x_j^1 \vee x_1^{1-\epsilon_1} \vee \cdots \vee x_r^{1-\epsilon_r}.$$

Applying the resolution rule to these two clauses (with x_j being the pivot variable), we derive the desired clause. (We are using the special case of resolution $C \vee x, C \vee \bar{x} \implies C$.)

Our base case is the leaves of T_{DLL}, which correspond to restrictions of τ which violate some clause C of τ. These restrictions are mapped to a clause C' which contains all the variables of C (and possibly others). C' can thus be derived in one step from C (which is part of our input) by the weakening rule. □

It is easy to see that if in Algorithm 1 we do not insist on the particular choice of the pivot variable when τ contains a singleton clause (or, perhaps, even if we do), then the argument becomes reversible. That is, $S_{tree-R}(\tau)$ will be *precisely* the miminal number of clauses generated by any implementation of Algorithm 1.

From lecture 2, we know that some CNF's τ have $S_R(\tau)$ which is exponential in the number of variables, so Algorithm 1 will not refute them efficiently using any heuristic. We would still like to find a heuristic that causes Algorithm 1 to be efficient when τ does have a small refutation. More generally, we would like to efficiently find a proof that is not much longer than the shortest proof. This motivates the following definition:

Definition 4.1 (Bonet, Pitassi, Raz 2000)**.** A proof system f is *automatizable* if there exists an algorithm that outputs an f-proof of any tautology τ, and runs in time $S_f(\tau)^{O(1)}$, i.e., polynomially in the size of the smallest f-proof. (We will let $p(S_f(\tau))$ be this polynomial for future reference.)

A proof system is *quasi-automatizable* if the same holds, except with a quasi-polynomial time bound.

This property is somewhat weird in that it is not *monotone*: making a proof system more powerful can make it less automatizable, if we gain the ability to make proofs significantly shorter without the ability to find these short proofs.

Definition 4.2. A proof system f is *normal* if

$$S_f(\tau|_\rho) \leq S_f(\tau)^{O(1)},$$

22

i.e., applying a restriction to a CNF does not make it much harder to prove in f.

A proof system is *strictly normal* if

$$S_f(\tau|_\rho) \leq S_f(\tau).$$

i.e., applying a restriction to a CNF does not make it any harder to prove in f.

Well-behaved systems will be strictly normal. Resolution is strictly normal, since we can apply ρ to every clause in the proof without invalidating any of the deductive steps, and possibly making some of them unnecessary.

Recall:

Definition 4.3. A proof system f has *feasible interpolation* if, given an unsatisfiable CNF τ of the form $A(y) \wedge B(z)$, and an f-refutation of τ of size s, we can decide in time $s^{O(1)}$ which of A and B is unsatisfiable. (If neither is satisfiable, we can answer with either one of them.)

Theorem 15. For any normal proof system f, Automatizability implies Feasible Interpolation.

This theorem is unfortunate, because we have just proved that, unless factoring is easy, we do not have Feasible Interpolation for Extended Frege, Frege, and other nice systems. Thus we do not have Automatizability for them, either.

Proof. We will only go through the proof for strictly normal proof systems. It is completely straightforward to adapt it to normal proof systems.

Suppose f is automatizable. We then have an algorithm Auto which, given an unsatisfiable CNF τ, outputs a refutation of τ in time $p(S_f(\tau))$. If τ is satisfiable, Auto outputs `ERROR` (without any time restrictions).

We can use Auto to perform feasible interpolation using Algorithm 2.

Claim 4.4. The correctness of Auto ensures that we answer correctly in line 5.

Claim 4.5. If Auto does not finish in $p(s)$ steps (line 3) or outputs something weird (line 7), then B is unsatisfiable.

Suppose $B(z)$ were satisfiable, and let $z = b$ be a satisfying assignment. Then, by strict normality, we have

$$S_f(A(y) \wedge B(z)|_{z=b}) \leq s = S_f(A(y) \wedge B(z))$$
$$S_f(A(y) \wedge \texttt{true}) \leq s$$
$$S_f(A(y)) \leq s$$

23

Algorithm 2 Feasible Interpolation Algorithm

Input: $A(y) \wedge B(z)$, $s = S_f(A(y) \wedge B(z))$
1: Run Auto on $A(y)$ for $p(s)$ steps.
2: **if** Auto did not finish in $p(s)$ steps **then**
3:    **print** "B is unsatisfiable"
4: **else if** Auto returns a valid refutation **then**
5:    **print** "A is unsatisfiable"
6: **else**
7:    **print** "B is unsatisfiable"
8: **end if**

---

so $A(y)$ had a refutation of length at most s, and Auto should have been able to find some refutation in time at most $p(s)$. $\square$

This theorem establishes that many proof systems of interest are not automatizable, but what about resolution? Both resolution and treelike resolution admit feasible interpolation, so maybe they are automatizable. We could try to automatize them with the following Algorithm 3 based on refutation width: (this corresponds to a widely-used algorithm in AI)

Algorithm 3 Bounded Search

Input: τ, a 3-CNF
  for $w = 0, \ldots, n$ **do**
    Infer all possible clauses C of width $\leq w$.
    if We have inferred $\emptyset$ **then**
      Halt
    end if
  end for

The runtime of this algorithm is $n^{O(w_R(\tau))}$.

Recall that for standard resolution, $w_R(\tau)$ is closely related to $S_R(\tau)$ (see Theorem 1 on page 4). We now complement that result with an even tighter (and easier to prove, too!) relation between width and *tree-like* refutation size.

Theorem 16 (Ben-Sasson, Wigderson). $w_R(\tau) \leq \log_2 S_{tree-R}(\tau)$.

Proof.

Claim 4.6. For any b, $S_{tree-R}(\tau) \leq 2^b \implies w_R(\tau) \leq b$.

Suppose that τ has a treelike refutation of size less than or equal to 2^b. The last deductive step in this refutation will be $x, \bar{x} \implies$ `false`, and via the restriction-refutation equivalence for treelike resolution, this implies that we have a resolution refutation of $\tau_0 = \tau|_{x=0}$ and $\tau_1 = \tau|_{x=1}$. Because our resolution is treelike, the size of the refutation of τ is the sum of the sizes of the refutation of τ_0 and the refutation of τ_1. Their sum is at most 2^b, so one of them (say τ_0) has size at most 2^{b-1}. By induction, we can assume that $w_R(\tau_0) \leq b - 1$ (induction on b), and $w_R(\tau_1) \leq b$ (an inner induction on the size of τ). By Lemma 2.3 on page 5, this yields $w_R(\tau) \leq b$, which is what we wanted. $\qquad\square$

Putting this bound together with Algorithm 3, we find that we can find a resolution refutation for τ in time $n^{O(w_R(\tau))} = n^{O(\log(S_{tree-R}(\tau)))} = S_{tree-R}(\tau)^{O(\log n)}$. This algorithm is quasi-polynomial, so we are tempted to say that treelike resolution is quasi-automatizable. This is not true, because the refutation produced by Algorithm 3 will not be treelike.

Question 4.7. Neither resolution nor treelike resolution is quasi-automatizable, under any mildly reasonable complexity assumption.

A practical message emerging from all this is that Algorithm 3 is as good as *any* DLL procedure, up to a factor of $n^{O(w_R(\tau))}$. If you wanted to improve this to a polynomial factor, you most likely can't, because:

4.3   Resolution is Probably not Automatizable

Theorem 17 (Alekhnovich Razborov 01)**.** Resolution is not automatizable unless W[P] is tractable.

This shows that automatizability is (most likely) a stronger property than feasible interpolation.

This isn't a class on parameterized complexity, so we do not even attempt to define W[P]. Instead, we indicate a concrete complete problem in this complexity class.

Problem 4.8 (Minimum Weight Monotone Assignment (MWMA))**.**
 Instance: C, a monotone circuit on n variables (monotone: using only $\wedge$ and $\vee$ gates, no negation)
Solution: $k(C)$, the minimum Hamming weight of a satisfying assignment a.

Problem 4.9 (h-Gap MWMA)**.**
An algorithm A (receiving as its input a monotone circuit C and an integer k) solves the h-Gap MWMA problem if

- $k(C) \leq k \implies A(\langle C, k \rangle) = \texttt{YES}$

- $k(C) \geq hk \implies A(\langle C, k \rangle) = \texttt{NO}$

Lemma 4.10. *For any constant $h > 1$, and any function $f : \mathbb{N} \to \mathbb{N}$, if we can solve the h-Gap MWMA problem in time $f(k) \cdot |C|^{O(1)}$, then* $\mathsf{W[P]}$ *is tractable.*

Assumption 4.11. For any constant $h > 1$, and any function $f : \mathbb{N} \to \mathbb{N}$, no algorithm A that solves the h-Gap MWMA problem can work in time $f(k) \cdot |C|^{O(1)}$.

We introduce a new parameter m, which we will assign a value to later.

Lemma 4.12 (Main Lemma). *There is a reduction $\tau(\cdot, \cdot)$ that maps a monotone circuit C and a parameter m to a contradiction (= an unsatisfiable CNF) $\tau(C, m)$, satisfying*

$$m^{\Omega(\min\{k(C), \log m\})} \leq s_R(\tau(C, m)) \leq |C| \cdot m^{O(\min\{k(C), \log m\})}.$$

Moreover, $\tau(C, m)$ can be computed efficiently, in time polynomial in $|C|$ and m.

Assuming the Main Lemma for now, we can prove Theorem 17.

Proof of Theorem 17. If we have an automatizing algorithm Auto which runs in time $s_R(\tau)^{O(1)}$ and outputs a refutation of size $S_{Auto}(\tau)$, we can utilize it to solve h-Gap MWMA efficiently, using Algorithm 4. We must prove that we output $\texttt{YES}$ on $\texttt{YES}$ instances, that we output $\texttt{NO}$ on $\texttt{NO}$ instances, and that Algorithm 4 obeys our time bounds.

We have to specify values for several parameters.

ϵ, h_0 The implied constants in the Main Lemma:

$$m^{\epsilon \min\{k(C), \log m\}} \leq s_R(\tau(C, m)) \leq |C| \cdot m^{h_0 k(C)}$$

h_1 Suppose Auto runs in time at most $s_R(\tau(C, m))^{h_1}$.

h We pick h so that $h^2 > \frac{h_1}{\epsilon}(h_0 h + 1)$

m We pick $m = 2^{h \max\{k, \frac{1}{k} \log |C|\}} = \max\left\{2^{hk}, |C|^{\frac{h}{k}}\right\}$. Note that $m^{\log m} \geq m^{hk}$.

Algorithm 4 Using Auto to solve h-Gap MWMA

Input: $\langle C, k \rangle$

1: $m \leftarrow m(k, |C|, h_1, \epsilon)$ as specified in the text.
2: $S_{\max} \leftarrow (|C| \cdot m^k)^{h_1}$
3: $\tau \leftarrow \tau(C, m)$ as described in the Main Lemma.
4: Run Auto on $\tau(C, m)$ for $S_{\max}$ steps.
5: **if** Auto did not finish in $S_{\max}$ steps **then**
6:     **return** NO
7: **else if** $S_{\text{Auto}}(\tau(C, m)) \leq S_{\max}$ **then**
8:     **return** YES
9: **else**
10:     **return** NO
11: **end if**

Claim 4.13. When $k(C) \leq k$, algorithm 4 outputs YES.

$$S_{\text{Auto}}(\tau(C, m)) \leq \text{runtime of Auto} \leq s_R(\tau)^{h_1}$$
$$\leq (|C| \cdot m^{h_0 k(C)})^{h_1} = S_{\max}$$

So Auto finishes in line 4, and $S_{\text{Auto}}(\tau(C, m)) \leq S_{\max}$ on line 7.

Claim 4.14. When $k(C) \geq hk$, algorithm 4 outputs NO.

$$S_{\text{Auto}}(\tau(C, m)) \geq s_R(\tau(C, m))$$
$$\geq m^{\epsilon \cdot \min\{k(C), \log m\}}$$
$$\geq m^{\epsilon \cdot hk}$$
$$= m^{\epsilon h^2 \cdot \frac{k}{h}}$$
$$> m^{h_1(hh_0 + 1) \cdot \frac{k}{h}}$$
$$= m^{h_1 h_0 k} m^{\frac{h_1 k}{h}}$$
$$\geq m^{h_1 h_0 k} \cdot |C|^{\frac{h}{k} \frac{h_1 k}{h}}$$
$$= (|C| m^{h_0 k})^{h_1} = S_{\max}$$

Therefore, even if we get to line 7, we output NO.

Claim 4.15. Algorithm 4 obeys the time bound in Assumption 4.11.

The two operations that take superconstant time are creating τ in line 3, and running Auto in line 4. Line 3 takes at most $(|C| \cdot m)^{O(1)}$, according to the Main Lemma 4.12. Line 4 takes at most $S_{\max}$ steps, or we stop running Auto. Since $S_{\max} = |C|^{O(1)} \cdot m^{O(k)}$, and $m \leq 2^{hk}$, this results in a time bound of the form $|C|^{O(1)} \cdot f(k)$. $\qquad \square$

4.3.1 Constructing the contradiction

The Combinatorial Principle

Consider the following situation: we have our circuit C on n variables $x_1, \ldots, x_n$, and a set $\mathcal{A}$ of "admissible" m-vectors, where $|\mathcal{A}| = m$. We will say that an $m \times n$ matrix M is admissible if every column of M is admissible. Note that we allow the same column to be repeated multiple times in M.

Each row of M represents a possible assignment to the variables $x_1, \ldots, x_n$.

Definition 4.16 ($\mathcal{P}_{C,\mathcal{A}}$)**.** In every admissible matrix M, there is some i such that row_i satisfies C.

This principle only holds for some combinations of $\mathcal{A}$ and C. We will pick $\mathcal{A}$ later, but as we go through the proof, we will note what properties of $\mathcal{A}$ will be required.

Definition 4.17. For a set of d vectors (not necessarily distinct), a *one-position* is a coordinate i such that all the vectors have a one in it.

$d_1(\mathcal{A})$ is the maximum d such that every set of d admissible vectors has some one-position i.

Lemma 4.18 (Our tautology is a tautology)**.** *If $k(C) \leq d_1(\mathcal{A})$, then $\mathcal{P}_{C,\mathcal{A}}$ is true (every admissible matrix has a row which satisfies C).*

Proof. There is some satisfying assignment $x_1, \ldots, x_n$ for C which has exactly $k = k(C)$ ones. Let $j_1, \ldots, j_{k(C)} \in [n]$ be the positions where x_j is one. Since C is monotone, having ones in these positions is sufficient to ensure that C outputs one.

Let us look at these "sufficient" columns $\mathrm{col}_{j_1}, \ldots, \mathrm{col}_{j_{k(C)}}$; since $k \leq d_1(\mathcal{A})$, they must share some one-position i. row_i has a one in all of the sufficient columns, and is thus a satisfying assignment. $\qquad \square$

This is the first desired property of $\mathcal{A}$:

$$\boxed{d_1(\mathcal{A}) \geq k(C)}$$

28

We choose this principle because we want $\mathcal{P}_{C,\mathcal{A}}$ to have complexity roughly $|\mathcal{A}|^{k(C)} = m^{k(C)}$. We have a simple approach: searching over all possible placements of admissible vectors in the columns corresponding to the smallest satisfying assignment for C. We need to show that this is basically the only approach, and that no significantly shorter proof exists.

The Circuit

For our purposes, a circuit is a sequence of statements of the form $v \leftarrow v' * v''$ where: v is a new variable that has not been used previously; $*$ is either $\vee$ or $\wedge$; and v', v'' are either input variables x_j, or have already received assignments.

We will have a special variable `out` (which receives its assignment in the same way) that represents the output of the circuit.

The straightforward encoding

We consider evaluating a copy of C on every row of our matrix M. We have the following variables:

a_{ij} is the i,j-th entry of M.

$\vec{a}_j$ is the admissible vector in the j-th column of M.

$z_{i,v}$ is the value at node v in the i-th copy of C.

This encoding is too straightforward: Resolution might be able to do something more clever than the bruteforce approach (at least, we don't know how to prove the contrary). Therefore, we use a scrambled encoding. We will still refer to the straightforward encoding to motivate our proofs.

The scrambled encoding

We use redundant encodings of the straightforward variables. We introduce two functions $F : \{0,1\}^s \to \mathcal{A}$, and $f : \{0,1\}^s \to \{0,1\}$ (to be specified later), and variables $x_j^1, \ldots, x_j^s,\ y_{i,v}^1, \ldots, y_{i,v}^s$, so that

$$F(x_j^1, \ldots, x_j^s) = \vec{a}_j$$
$$f(y_{i,v}^1, \ldots, y_{i,v}^s) = z_{i,v}.$$

We want to specify constraints on these variables, so we will introduce the following abbreviations:

$[\mathrm{True}_{v,i}]$**:** $f(y_{i,v}^1, \ldots, y_{i,v}^s) = 1$, written as a CNF in $y_{i,v}^1, \ldots, y_{i,v}^s$.

$[Column_j = \vec{a}]$**:** $F(x_j^1, \ldots, x_j^s) = \vec{a}$, written as a CNF in $x_j^1, \ldots, x_j^s$.

Remark 4.19. At the end of the day, s will be $O(\log m)$, and, therefore, these CNFS (as well as those resulting from the axioms listed below) will have size $m^{O(1)}$. Also, in the proof of Lemma 4.40 we will actually need a slightly more general version in which the functions used to encode $[Column_j = \vec{a}]$ can be different for different j, and the same is true for $[True_{v,i}]$. None of our proofs, however, is affected by this generalization.

Our CNF τ has the following axioms:

$[Column_j = \vec{a}] \implies [True_{x_j,i}]$ (for $j = 1, \ldots, m$, for every $\vec{a} \in \mathcal{A}$, and for every i such that $a_i = 1$).

> We will call these the **input axioms**, since they ensure that each copy of C is receiving as its input a value which is *at least* the correct row of M (since C is monotone, we do not care about the opposite direction – see Remark 4.21 below). Note that each of these axioms belongs to a specific row and column of the matrix.

$([True_{v',i}] * [True_{v'',i}]) \implies [True_{v,i}]$ for $i = 1, \ldots, m$, and whenever $v \leftarrow v' * v''$ is a gate in C, for $* = \wedge, \vee$.

> We will call these the **gate axioms**, since they force the node variables $z_{i,v}$ to correctly emulate C at every gate. Note that each of these axioms belongs to a specific row of the matrix.

$\neg[True_{\mathbf{out},i}]$ for $i = 1, \ldots, m$. We will call these the **output axioms**, since they force each copy of C to output zero. Note that each of these axioms belongs to a specific row of the matrix.

We then define

Definition 4.20 $(\tau(C, \mathcal{A}, F, f))$**.**

$$\tau(C, \mathcal{A}, F, f) = \bigwedge_{j=1}^{m} \bigwedge_{\{i|a_i=1\}} \left[[Column_j = \vec{a}_j] \implies [True_{x_j^i,i}] \right] \wedge$$
$$\bigwedge_{i=1}^{n} \bigwedge_{v \leftarrow v' * v''} \left[([True_{v',i}] * [True_{v'',i}]) \implies [True_{v,i}] \right] \wedge$$
$$\bigwedge_{i=1}^{n} \neg[True_{\mathbf{out},i}]$$

30

Remark 4.21. As we already indicated above, we are allowing assignments to cheat in one direction: in terms of the straightforward encoding, we are allowing $(z_{i,x_j} = 1 \land a_{ij} = 0)$ and $(z_{i,v'} * z_{i,v''} = 0 \land z_{i,v} = 1)$. This does not cause problems: we claim that any satisfying assignment for $\tau(C, \mathcal{A}, F, f)$ would produce a genuine counterexample to $\mathcal{P}_{C,\mathcal{A}}$.

We can produce this counterexample by setting $z_{i,x_j} = 0$ whenever $a_{ij} = 0$, and then repeatedly changing $z_{i,v}$ to zero when its children would make it zero in C. Since C is acyclic and monotone, this will eventually correct all errors of the two forms we allow. Since we never change any variable from zero to one, this cannot cause any copy of C to be satisfied.

Thus, (the negation of) $\tau(C, \mathcal{A}, F, f)$ is equivalent to $\mathcal{P}_{C,\mathcal{A}}$. Allowing this cheating gives us some needed flexibility later in the proof.

Lemma 4.22. *We can efficiently construct a CNF encoding of $\tau(C, \mathcal{A}, F, f)$ as long as we can efficiently compute F, f, and as long as we can efficiently enumerate $\mathcal{A}$. (Here efficiently means in time polynomial in $|C|$ and m.)*

Exercise: Prove this.

4.3.2 The Upper Bound on s_R

Lemma 4.23. *If $k(C) \leq d_1(\mathcal{A})$, then $s_R(\tau(C, \mathcal{A}, F, f)) \leq O(|C| \cdot 2^{s(k(C)+1)})$.*

Proof. There is some satisfying assignment $x_1, \ldots, x_n$ for C which has exactly $k(C)$ ones. Let $J_1 = \{j_1, \ldots, j_{k(C)}\}$ be a set of columns where x_j is one. We then use the Canonical Refutation in Algorithm 5 on these columns.

We can find an appropriate i in line 3 because $|J| \leq d_1(\mathcal{A})$. We can succesfully deduce $[True_{\text{out},i}]$ in line 7 because $(\forall j \in J_1, x_j = 1) \implies C(x_1, \ldots, x_n) = 1$.

$\square$

Exercise: Show that the Canonical Refutation can be encoded in a Resolution refutation using $O(|C| \cdot 2^{s(k(C)+1)})$ clauses.

Remark 4.24. Theorem 17 is also true for tree-like resolution, but the proof is more complicated. The reason is that since C is a circuit rather than a formula, the Canonical Refutation need not necessarily be tree-like. This problem is fixed by introducing an even more elaborate encoding (see the paper for the details).

Algorithm 5 The Canonical Refutation

Input: $J_1 = \{j_1, \ldots, j_k\}$
1: **for all** $(\vec{a}_{j_1}, \ldots, \vec{a}_{j_k}) \in \mathcal{A}^k$ **do**
2:   **Assume:** $\wedge_{j \in J_1}[Column_j = \vec{a}_j]$
3:   Pick i so that $(\vec{a}_j)_i = 1$ for all $j \in J_1$.
4:   **for** $j \in J_1$ **do**
5:     Derive $[\text{True}_{x_j,i}]$ using the input axioms.
6:   **end for**
7:   Use the gate axioms to derive a set of $[\text{True}_{v,i}]$ including $[\text{True}_{\text{out},i}]$.
8:   Recall that $\neg[\text{True}_{out,i}]$ is an (output) axiom.
9:   **Conclude:** $\bigwedge_{j \in J_1}[Column_j = \vec{a}_j]$ is false.
10: **end for**
11: **Conclude:** $\neg \tau(C, \mathcal{A}, F, f)$

4.3.3 The Lower Bound on s_R

Definition 4.25. A function $F : \{0,1\}^s \to D$ is called *r-surjective* if, for any restriction ρ that assigns at most r variables, $F|_\rho$ is surjective.

Definition 4.26. Let $d_0(\mathcal{A})$ be the maximal d such that, for every d positions $i_1, \ldots, i_d$, there exists an $\vec{a} \in \mathcal{A}$ with $a_{i_1} = \cdots = a_{i_d} = 0$.

Lemma 4.27 (Lower bound on resolution width). *Let r be the largest r such that F and f are both r-surjective. Then*

$$w_R(\tau(C, \mathcal{A}, F, f)) \geq \frac{r}{2}\min\{k(C), d_0(\mathcal{A})\}$$

We follow the same strategy as in the proof of Theorem 2 on page 7.

When we defined the axioms of $\tau(C, \mathcal{A}, F, f)$, we noted that each of the axioms refers to a single row i. Let Ax_i be the set of axioms that refer to row i, and let $\text{Ax}_I = \bigcup_{i \in I} \text{Ax}_i$ for a set of rows I.

Definition 4.28. For a clause D, let $\mu(D) = \min\{|I| : \text{Ax}_I \models D\}$.

Claim 4.29. $\mu(D) = 1$ when D is an axiom, since D comes from some row i.

Claim 4.30. $\mu(\emptyset) > d_0(\mathcal{A})$

A set of axioms entails the empty set if and only if it is unsatisfiable. Therefore, we need to show that when $|I| \leq d_0(\mathcal{A})$, we can find an assignment

that satisfies Ax_I. Since $|I| \leq d_0(\mathcal{A})$, some $\vec{a} \in \mathcal{A}$ has $(\vec{a})_i = 0$ for all $i \in I$. We can then satisfy Ax_I by setting

$$M = [\vec{a} \cdots \vec{a}] \tag{1}$$

$$z_{i,v} = 0 \text{ for } i \in I \tag{2}$$

In the I rows, all variables are zero, which is consistent with the gate and output axioms. The input axioms are satisfied because $\vec{a}$ is also zero in these rows.

Claim 4.31 ("Continuity of μ"). In the deduction

$$\frac{D_1 \qquad D_2}{D},$$

$\mu(D) \leq \mu(D_1) + \mu(D_2)$. This is because $I_1 {\models} D_1, I_2 {\models} D_2 \implies I_1 \cup I_2 {\models} D$.

Definition 4.32. An *intermediate clause* is a clause D satisfying

$$\frac{1}{2} d_0(\mathcal{A}) \leq \mu(D) \leq d_0(\mathcal{A}).$$

Claim 4.33. R can be rewritten so that it contains an intermediate clause D.

Consider the set of clauses in R which satisfy condition c: $\mu(D) < \frac{1}{2} d_0(\mathcal{A})$. The conclusion ($\emptyset$) does not, so there must be a first clause D which violates c. D cannot be an axiom, so it must have been derived.

If D was derived using the resolution rule on clauses D_1 and D_2, then D_1 and D_2 appear before D, and thus satisfy c. By the continuity property, $\mu(D) \leq \frac{1}{2} d_0(\mathcal{A}) + \frac{1}{2} d_0(\mathcal{A}) = d_0(\mathcal{A})$.

If D was derived through weakening, then it was derived from some D_1 that satisfies c. We can then break the weakening into two steps, $D_1 \implies D' \implies D$, where D' is an intermediate clause.

Claim 4.34. If D is an intermediate clause, it cannot satisfy both

$$w_R(\tau(C, \mathcal{A}, F, f)) < \frac{r}{2} k(C) \tag{3}$$

$$w_R(\tau(C, \mathcal{A}, F, f)) < \frac{r}{2} d_0(\mathcal{A}) \tag{4}$$

Let I be a set realizing the minimum $\{|I| : \text{Ax}_I {\models} D\}$. If we remove any element $i_0 \in I$, we will get a set I' for which $\neg(\text{Ax}_{I'} {\models} D)$. This means that there is some assignment $x_j^t = \alpha_j^t, y_{i,v}^t = \beta_j^t$ that satisfies $\text{Ax}_{I'}$ but violates D.

Claim 4.35. We can modify this assignment to satisfy Ax_I, without modifying any variable that appears in D. (Thus continuing to violate D.)

How would we build such an assignment? We want to modify α, β to satisfy Ax_{i_0} without messing up any of the other rows. Our goal is to achieve the following assignment to the straightforward variables:

- Set the $\vec{a}_j$ to be zero in all of I in any column j where this is possible without modifying D's variables. We will say that a column is *free* is we can put any $\vec{a}$ there without disturbing D. We will say that the other columns are *fixed*.

- Leave the $z_{i,v}$ alone for $i \in I'$.

- We will set the $z_{i_0,v}$ as they would be while correctly emulating C on the input $\vec{x_0}$ which is one in fixed columns and zero in free columns. We will say that a row is free if we can set the $z_{i_0,v}$ arbitrarily without disturbing D.

All that remains is to choose a free row i_0, and to prove that most columns are free. Let $w_i(D)$ be the number of $y_{i,v}^t$ that appear in D. Let $w^j(D)$ be the number of x_j^t that appear in D.

We choose i_0 to minimize $w_i(D)$, so that we have maximum freedom to change this row's variables without disturbing D. By the r-surjectivity of f, a row i is free if at most r of the $y_{i,v}^1, \ldots, y_{i,v}^s$ appear in D. We have $\sum_i w_i(D) \leq w(D) < \frac{r}{2} d_0(\mathcal{A})$, and $\sum_i w_i(D) \geq \sum_{i \in I} w_i(D) \geq |I| w_{i_0}(D)$. Since $|I| \geq \frac{1}{2} d_0(\mathcal{A})$, we have $w_{i_0}(D) \leq r$, and thus i_0 is a free row.

By the r-surjectivity of F, a column is free if D contains at most r of the $x_j^1, \ldots, x_j^s$. We have $\sum_j w^j(D) \leq w(D) < \frac{r}{2} k(C)$. Let $J_0 = \{j \mid w^j(D) > r\}$. Then $r \cdot |J_0| \leq \sum_j w^j(D)$, so $|J_0| < \frac{1}{2} k(C)$. Thus any column outside of J_0 is free.

Claim 4.36. For any free column $j \notin J_0$, we can choose $\bar{\alpha}$ to set $(\vec{a}_j)_i = 0$ simultaneously for all $i \in I$, without disturbing D.

Since $|I| \leq d_0(\mathcal{A})$, there is an admissible vector $\vec{a}$ satisfying this condition. We can then put $\vec{a}$ in any free column by modifying α.

Claim 4.37. We can choose $\bar{\beta}$ to set row_i to $\vec{x_0}$ without disturbing D.

Claim 4.38. The input axioms from Ax_I are satisfied by $\bar{\alpha}, \bar{\beta}$.

Claim 4.39. The gate and output axioms from Ax_{i_0} are satisfied by $\bar{\alpha}, \bar{\beta}$.

Proof. We have modified the $y_{i_0,v}^t$ to make the $z_{i_0,v}$ correctly emulate C rejecting input $\vec{x_0}$. $\square$

This completes the proof of Claim 4.34 and hence of the width lower bound, but we are more interested in a lower bound on the size of the proof:

Lemma 4.40 (Lower bound on resolution size). $s_R(\tau(C, \mathcal{A}, F, f)) \geq e^{\Omega(\frac{r^2}{s} \cdot \min\{k(C), d_0(\mathcal{A})\})}$.

The naive way to try to deduce it from Lemma 4.27 would be to apply the general trade-off between width and size (Theorem 1 on page 4). Unfortunately, the parameters do not work out nicely (or at least we do not know how to make them work). Instead, we are using the method of *random restrictions* (later on we will see its much more elaborated version).

All our variables are split into blocks $\{X_j, Y_{iv}\}$ according to their subscripts, s variables in each block. Let us choose independently at random $(r/2)$ variables in each block and assign them to $0, 1$ (also independently at random). What will happen to our refutation? The first nice thing is that every individual fat clause will be eliminated with high probability.

Lemma 4.41. *For a clause C with $w(C) \geq \frac{r}{4} \min\{k(C), d_0(\mathcal{A})\}$, the probability it is not eliminated by a random restriction is at most $e^{-\Omega(\frac{r^2}{s} \cdot \min\{k(C), d_0(\mathcal{A})\})}$.*

Exercise: Prove this.

And then, since we have assigned just $(r/2)$ variables in each block, $F|_\rho, f|_\rho$ will still be $(r/2)$-surjective (although they can become different in each block – cf. Remark 4.19!). So, we will get a refutation of a contradiction that still has the form $\tau(C, \mathcal{A}, F|_\rho, f|_\rho)$, and we can apply to it Lemma 4.27 to conclude that in fact not all fat clauses have been eliminated. Comparing this with Lemma 4.41 and applying the union bound completes our proof.

4.3.4   Combinatorial Voodoo

For a matrix $\mathcal{A}$, we define $d_1(\mathcal{A})$ ($d_0(\mathcal{A})$respectively) to be the maximum number of columns (rows, resp.) such that there is some row (column, resp.) which contains only ones (zeros, resp.) in the intersection of the row and columns. For our proof, we need a 0-1 matrix $\mathcal{A}$ such that $d_1(\mathcal{A}), d_0(\mathcal{A}) \geq \Omega(\log m)$. Explicit examples of such matrices are well-known, e.g.:

Definition 4.42 (The Paley Matrix). Let m be a prime, and, for $i, j \in \mathbb{F}_m$, $a_{ij} =$ iff $j \neq i$ and $(j - i)$ is a quadratic residue mod m.

F and f need to be r-surjective functions on s bits, where s and r are $\Theta(\log m)$. For this, we use linear codes. Let h be a sufficiently large constant and $L \subseteq \{0,1\}^{h \log_2 m}$ be an $\mathbb{F}_2$-linear subspace of dimension $\log m$ that does not contain non-zero vector of weight $< \log m$. There are many explicit examples of such L. Now, fix in L an arbitrary basis $x_1, \ldots, x_{\log m}$, and let $F : \{0,1\}^{h \log m} \to \{0,1\}^{\log m}$ be the linear mapping defined by $F(y) = (\langle x_1, y \rangle, \ldots \langle x_{\log m}, y \rangle)$. Then F is $(\log m)$-surjective.

4.3.5 The Upper Bound when $k(C)$ is too large

Choosing $s = \Theta(\log m)$, we have proved that

$$m^{\Omega(\min\{k(C), \log m\})} \leq s_R(\tau(C, \mathcal{A}, F, f)) \leq O(|C| \cdot m^{O(k(C))}).$$

We still need to prove that $s_R(\tau) \leq |C| \cdot m^{O(\log m)}$. Our construction thus far cannot possibly supply this, since we have been assuming all along that $k(C) \leq d_1(\mathcal{A})$. When $k(C) > d_1(\mathcal{A})$, it is even possible that $\tau(C, \mathcal{A}, F, f)$ is satisfiable.

We need explicit examples of unsatisfiable CNF's with $s_R(\tau_m) = m^{\Theta(\log m)}$. We note that the Tseitin tautologies provide τ_t with $s_R(\tau_t) = 2^{\Theta(t)}$, so if we take $t = (\log m)^2$, these will suffice.

We define $\tau(C, m) = \tau(C, \mathcal{A}, F, f) \wedge \tau_m$, where the τ_m is operating on completely new variables. Now resolution can choose between disproving $\tau(C, \mathcal{A}, F, f)$ and disproving τ_m, so we have $s_R(\tau(C, m)) \leq s_R(\tau_m) \leq m^{O(\log m)}$.

Our lower bound is still valid because of feasible interpolation. If we have a refutation for $\tau(C, \mathcal{A}, F, f) \wedge \tau_m$ of size s, then we must have a refuation of size $s^{O(1)}$ for $\tau(C, \mathcal{A}, F, f)$ or τ_m. This will only weaken the constant in $s_R(\tau(C, m)) \geq m^{\Omega(\min\{k(C), \log m\})}$.

This completes the proof of the Main Lemma, and thus the theorem.

<div align="center">

Lectures 12 and 13

Scribe: Gabriel Bender, University of Chicago.

Date: February 17 and 19, 2009

</div>

5 Applications to Learning

Definition: $k(C)$ is the minimum number of ones in a satisfying assignment of a circuit C.

PAC ("Probable, Approximate, Correct") is a model in learning theory in which we want to learn from examples generated randomly by nature with an unknown distribution D.

- *Input:* A list of pairs $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$ for an unknown function f generated accordingly to D.

- *Output:* A circuit C that will perform "reasonable well" in trying to predict the output of f on future inputs according to the same distribution.

"Exact Learnability" is a simple model which (under certain conditions that are always fulfilled in our situation) implies PAC learnability.

Definition: Given $\{0, 1\}^n$, a *Concept Class* is a class of Boolean expressions $\mathcal{C}$ which represents Boolean functions $f : \{0, 1\}^n \to \{0, 1\}$. For instance, $\mathcal{C}$ could be the set of all DNFs in n variables.

5.1   Decision Trees

One concept class of particular interest is the set of all decision trees under the following construction:

Definition: A *decision tree* in n variables is a binary tree in which every node has a label, as well as two edges labeled 0 and 1, and every leaf is labeled "accept" or "reject."

Definition: A decision tree is *balanced* if every leaf is at the same depth.

Definition: The *height* of a tree, $h(T)$, is the length of the maximal path from the root to any leaf.

Problem: Given a system of equations $F(a_1) = \epsilon_1, \ldots, F(a_m) = \epsilon_m$, we would like to solve it in the concept class $\mathcal{C}$ consisting of all decision trees. That is, we want to find $\hat{F} \in \mathcal{C}$ that satisfies $\hat{F}(a_i) = \epsilon_i$ for $1 \le i \le m$. We would like to do this as efficiently as possible, and want our solution to be "not much longer" than the size s of the minimal solution $F \in \mathcal{C}$.

Definition: By *proper learning theory*, we mean (as above) that an output hypothesis $\hat{F}$ must be in the same concept class as our original function F.

Theorem 18 (Alekhnovich et al.)**.** DNF formulas of size s can be exactly learned in time $m^{O(1)}n^{O(\sqrt{n \log s})}$. Notice that this is subexponential, but not polynomial-time.

Theorem 19. Decision trees are not PAC-learnable in polynomial time unless SAT is computable in randomized time which is subexponential.

Proof of Theorem 18: This proof has close parallels to the proof that "Short proofs are narrow" (Theorem 1 on page 4). We replace $w_R(P)$, the refutation width of a proof tree P, with the "fanning" width of a DNF expression, ie. the maximum number of variables that appears in any of its terms. In the original analysis of proof trees, a clause $\mathcal{C}$ was *fat* if $w(\mathcal{C}) > d$ for some predetermined d. Here, a term is *fat* if it involves at least d variables.

Induction Base: Assume our system S has a solution without fat terms. Then we claim that there exists an algorithm BoundedSearch(S) that solves the problem in allocated time. Since there are no fat clauses, each expression has at most d variables. Hence, there are at most $O(n)^d$ possible terms. We simply take, in the allotted time, all those terms that are consistent with all negative examples $F(a) = 0$, and check if they "cover" all positive examples $F(a) = 1$.

Inductive step: Suppose we have f fat clauses. In the proof of Theorem 1, we took $2n$ restrictions. By the Pigeon-Hole Principle, one of those restrictions was guaranteed to produce at most αf fat clauses, where $\alpha = (1 - \frac{2d}{n})$. However, when we try to apply that approach here, we run into trouble because we don't know which restriction will sufficiently simplify our system. The solution is to run our algorithm recursively on all $2n$ restrictions in parallel:

Algorithm: Search(S): Run the following restrictions in parallel:

$$\text{Search}(S \mid_{x_1=0}), \text{Search}(S \mid_{x_1=1}), \ldots, \text{Search}(S \mid_{x_n=1})$$

We wait for at least one search to terminate. Assume W.L.O.G. that it is the restriction $x_i = \epsilon$. Let $T(n, f)$ denote the amount of time needed to find a solution with n variables and f fat clauses. Then $T(n, f)$ satisfies $T(n, 0) \leq n^{O(\sqrt{n \log s})}m^{O(1)}$ by the base case. Furthermore, $T(n, f) \leq (2n)T(n, \alpha f) + T(n-1, f)$. $T(n, \alpha f)$ represents the time needed to run the search algorithm on our "good" restriction $x_i = \epsilon$. The time is multiplied by $2n$ because we're checking all $2n$ possible restrictions in parallel. Once one of our searches has

terminated, we need to run the algorithm on the complementary restriction $x_i = 1 - \epsilon$; this accounts for the additional term $T(n-1, f)$.

The fact that, despite the extra $2n$ term, this recursion still resolves to $T(n, s) \le m^{O(1)} n^{O(\sqrt{n \log s})}$ is in Homework # 2.

6 Concrete Lower Bounds

Let F_d denote the depth-d Frege system (in which every clause has depth at most d). Then we know that

$$F_1 \le Res(k) \le F_2$$

(recall that $Res(k)$ is the natural proof system operating with k-DNFs).

6.1 Random 3-CNFs

Suppose we have a CNF τ with n variables and Cn clauses chosen at random. If C is big then τ is almost surely unsatisfiable. We may therefore ask: what is the minimum refutation size in various proof systems?

Extended Frege: We believe that $S_{EF}(\tau) \ge \exp(n^{\Omega(1)})$. Intuitively, this means you can't do much better for random 3-CNFs than a brute-force search through solutions. But the strongest system for which we can really *prove* this is $\text{Res}(k)$:

Theorem 20 (Alekhnovich 05). $S_{\text{Res}(\sqrt{\log n / \log \log n})} \ge \exp(n^{\Omega(1)})$.

6.2 The Pigeon-Hole Principle

We introduce the notation: $[m] = \{1, 2, \ldots, m\}$ for an integer m.

Observation: Let our domain $D = [m]$ and our range $R = [n]$. If $m > n$ then there is no bijection from $[m]$ to $[n]$.

We create Pigeon-Hole contradiction as follows:

- For each $i \in [m], j \in [n]$, we introduce a new variable x_{ij}. Intuitively, this corresponds to the statement "Pigeon i is in hole j."

- For each $i \in [m]$, we set $Q_i = \bigvee_{j=1}^{n} x_{i,j}$. Intuitively, Q_i represents the statement "Pigeon i is in some hole."

- For $j \in [n]$ we let Q_j represent the statement "Hole j contains some pigeon:" $Q_j = \bigvee_{i=1}^{n} x_{i,j}$

- For each $i_1 \neq i_2 \in [m]$ and $j \in [n]$, $Q_{i_1,i_2,j} = \overline{x_{i_1,j}} \vee \overline{x_{i_2,j}}$. This corresponds to the statement "no two pigeons share the same hole."

- For each $i \in [m]$ and $j_1, j_2 \in [m]$, we set $Q_{i,j_1,j_2} = \overline{x_{i,j_1}} \vee \overline{x_{i,j_2}}$. This represents the statement "no pigeon is placed in two (or more) holes."

Definitions: Let $i, i_1, i_2 \in [m]$ and $j, j_1, j_2 \in [n]$. We define the following CNFs by listing their sets of clauses:

- $\mathrm{PHP}_n^m = \{Q_i\} \cup \{Q_{i_1,i_2,j}\}$

- $\mathrm{FPHP}_n^m = \mathrm{PHP}_n^m \cup \{Q_{i,j_1,j_2}\}$

- $\mathrm{onto\text{-}PHP}_n^m = \mathrm{PHP}_n^m \cup \{Q_j\}$

- $\mathrm{onto\text{-}FPHP}_n^m = \mathrm{FPHP}_n^m \cup \{Q_j\}$

Theorem 21 (Maciel, Pitassi, Woods 99). $S_{Res((\log n)^{O(1)})}(PHP_n^{2n}) \leq n^{O(\log n)}$.

It is an open question whether we can say the same thing about $\mathrm{Res}(2)$.

Theorem 22 (Krajicek, Pudlak, Woods, Pitassi, Beame, Impagliazzo 93). $S_{F_d}(\mathrm{onto\text{-}FPHP}_n^{n+1}) \geq \exp(n^{\epsilon_d})$ with a constant $\epsilon_d > 0$.

Theorem 23 (Raz, Razborov 01). $S_R(\mathrm{onto\text{-}FPHP}_n^{\infty}) \geq \exp(\Omega(n^{1/3}))$

There is a number of interesting problems about the complexity of PHP that are still open (see the course Web page).

6.2.1 Matching Restrictions

Suppose ρ is a restriction. Let $m = n + 1$ and $l = n^{\epsilon}$. Then $\mathcal{M}_{D \times R}^{l}$ is the set of *partial matchings* assigning pigeons to *all but l* holes.

Formally, we start with a collection of pairs (i_ν, j_ν). We say that

$$\rho(x_{ij}) = \begin{cases} 1, \text{if } i = i_\nu \text{ and } j = j_\nu \text{ for some } \nu \\ 0, \text{if } i = i_\nu \text{ for some } \nu \text{ and } j \neq j_\nu \\ 0, \text{if } j = j_\nu \text{ for some } \nu \text{ and } i \neq i_\nu \end{cases}$$

6.2.2 Matching Decision Trees

A *Matching Decision Tree* is a tree. At each node, we can either query a pigeon i or we can query a hole j. If we query the node of a pigeon i then one of the variables $x_{i,1}, \ldots, x_{i,n}$ will be 1. Hence, our "pigeon" node will have n successors corresponding precisely to those variables. The intermediate edges will have the labels $j = 1, \ldots, j = n$ respectively.

Similarly, if we query a hole j then we can determine which of m pigeons $x_{1,j}, \ldots, x_{m,j}$ sits in it. Hence, a "hole" node will have m successors with edges labeled $i_1, \ldots, i_m$, corresponding to the cases $x_{i,1} = 1, \ldots, x_{i,m} = 1$ respectively.

All leaves in the decision tree are labeled by 0 or 1. Let $Br(T)$ denote the set of all leaves, $Br_0(T)$ be the set of leaves labeled by 0, and $Br_1(T)$ be the set of leaves labeled by 1. Every leaf naturally corresponds to a matching restriction.

Definition: A matching decision tree T *represents* F if

- $(\forall \rho \in Br_0(T))(F \restriction_\rho \equiv 0)$

- $(\forall \rho \in Br_1(T))(F \restriction_\rho \equiv 1)$

6.2.3 Putting it all together

Define $h(T)$ to be the maximum length over all paths from the root of T to any node.

As a first attempt, we might say that all formulas can be represented by low-height matching decision trees with only accepting branches. However, this fails for the depth-2 formula that is the conjunction of axioms: it is easily deducible but never accepts.

More specifically, let $(\bigwedge_i Q_i) \wedge (\bigwedge_{i_1,i_2,j} Q_{i_1,i_2,j})$ be the conjunction of all of the PHP_n^m axioms. It is a depth-2 formula, and is identically zero in the real world, since there are more pigeons than holes in our setup.

Let Γ be a set of all subformulas that appear in our proof P.

Definition Consider the set of formulas over the language consisting of the constants $\{0, 1\}$ and the operators $\neg$ and $\vee$. The *k-evaluation* is a map

$F \mapsto T_F$ that takes formulas to matching decision trees of height at most k. It is defined as follows:

- T_0 consists of a tree with a single node labeled 0, ie. always reject

- T_1 consists of a tree with a single node labeled 1, ie. always accept

- $T_{\neg F} \equiv T_F^c$, ie. take the tree T_F and reverse all the labels.

- $F = \bigvee_{i \in I} F_i$. We wish to express T_F in terms of the T_{F_i} values. Ideally we would like to let T_F represent $\bigvee_{i \in I} \text{Disj}\ (T_{F_i})$, where $\text{Disj}\ (T) = \bigvee_{\rho \in Br_1(T)} t_\rho$ is a matching disjunction; $t_\rho \equiv \bigwedge_{i \in dom(\rho)} x_{i,\rho(i)}$.

We will discuss how to construct a k-evaluation later. For the time being let us notice that it suffices for our purposes.

Lemma 6.1. *Suppose $k < \frac{n}{3}$. Assume Γ is an arbitrary set of formulas that is closed under subformulas and that has a k-evluation. Then there is no refutation of the Pigeon-Hole Principle in which all lines belong to Γ.*

Lectures 14 and 15
Scribe: Paolo Codenotti, University of Chicago.
Date: February 24 and 26, 2009

Let us finish the discussion on how to construct k-evaluations. Let Γ be a set of formulas closed under subformulas. Let $\Gamma_i \subseteq \Gamma$ be the set of formulas in Γ of depth at most i. So we have

$$\Gamma_0 \subseteq \Gamma_1 \subseteq \Gamma_2 \subseteq \cdots \subseteq \Gamma_d = \Gamma.$$

By induction, we will construct k_i-evaluations for $\Gamma_i|_{\rho_i}$, where ρ_i are restrictions we will construct as we go along, with ρ_{i+1} an extension of ρ_i.

It is easy to see that restrictions are a natural and robust concept. In particular this means that $\Gamma_i|_{\rho_{i+1}}$ will still be a k_i-evaluation, independently of our choice of ρ_{i+1} (since ρ_{i+1} extends ρ_i). The only case we need to take care of is when $F = \bigvee_{i \in I} F_i$, where the F_i are already evaluated formulas. In particular, we have $F = \bigvee_{i \in I} t_i$, where the t_i are matching terms with at most r variables. We can assume without loss of generality that ρ_i is trivial (since we can assume it is already fixed). In order to finish our construction, and show we can get a restriction ρ_{i+1} for which we can get k_{i+1}-evaluations for some small k_{i+1}, we use the following lemma.

Lemma 24 (Switching Lemma)**.** Let F be a matching r-disjunction, and let $10 \leq \ell \leq \epsilon\sqrt{n/r}$, where ϵ is a constant (something like 1/10). Let ρ be a random restriction that assigns all but $\ell + 1$ pigeons. Then $F|\rho$ can be represented by a height s matching decision tree with probability at least $1 - \exp\left(-\Omega(\ell^2\sqrt{r/n})\right)^s$.

In a slightly different context, this lemma was first stated and proved by Håstad in 1986. A simple combinatorial proof was given 10 years later by Razborov.

Once we have Lemma 24, the proof is completed by a simple application of the union bound: for every individual $F \in \Gamma_d \setminus \Gamma_{d-1}$, the probability that $F|_{\rho_{k+1}}$ does not get simplified as necessary is $\ll |\Gamma_d|^{-1}$.

7 Algebraic and Geometric proof systems

Assignments to formulas are points on the boolean cube. There are two ways to translate formulas so that variables become numbers (i.e. elements of some field K). The two schools of thought differ in how clauses are represented.

1. In the *algebraic* approach clauses are represented by equations.

2. In the *geometric* approach clauses are represented by inequalities.

Let us show how this is done by example. The clause $x_1 \vee \overline{x_2} \vee x_3$ can be written as $(1 - x_1)x_2(1 - x_3) = 0$ (algebraic), or $x_1 + (1 - x_2) + x_3 \geq 1$, which simplifies to $x_1 + x_3 \geq x_2$ (geometric).

7.1 Algebraic proof systems

We will have the axioms $x_i^2 - x_i = 0$ (which force $x_i = 0, 1$). This means that we consider
$$\Lambda_n = K[x_1, \ldots, x_m]/(x_i^2 - x_i),$$
which is a ring of multilinear polynomials over our field K.

7.1.1 Nullenstellensatz proof systems

A CNF reduces to a system of polynomial equations $f_1 = 0, \ldots, f_m = 0$. It is easy to see that there is no solution to the above system of equations if and only if there exist polynomials Q_i such that

$$1 = Q_1 f_1 + \cdots + Q_m f_m$$

(note that due to the presence of the axioms $x_i^2 - x_i = 0$, we do not have to require that the field K is algebraically closed). Therefore the polynomials Q_i are a refutation for the CNF. This is called the *Nullstellensatz proof system*. The degree of a refutation is defined as

$$d(1 = Q_1 f_1 + \cdots + Q_m f_m) = \max_i \deg (Q_i).$$

We define $d_{NS}^K(\tau)$ to be the minimum degree of any refutation of a polynomial translation of τ.

7.1.2 Polynomial Calculus

Polynomial Calculus(PC) is a proof system that is a "dynamical version" of Nullstellensatz. The inference rules in Polynomial Calculus are the following:

$$\frac{f = 0 \qquad g = 0}{\alpha f + \beta g = 0} \quad \forall \alpha, \beta \in K \qquad \qquad \frac{f = 0}{f \cdot x_i = 0} \quad \forall i \in [n]$$

We define the degree of the first rule to be $\max\{\deg(f), \deg(g)\}$, and the degree of the second rule to be $\deg(f) + 1$. A PC refutation is a sequence of inferences proving $1 = 0$. The degree of a refutation of a contradiction τ is $d_{PC}^K(\tau)$, the minimum degree of a PC refutation of τ. We can make the following two observations about $d_{PC}^K(\tau)$:

- $d_{PC}^K(\tau) \leq w_R(\tau)$ (we can simulate resolution rules with polynomial calculus).

- $d_{PC}^K(\tau) \leq d_{NS}^K(\tau) + O(1)$.

There are many separation results for Polynomial Calculus and Nullstellensatz. For example Buss and Pitassi showed the following.

Theorem 25 (Buss, Pitassi 1998)**.** Let τ be the least induction principle. $x_1 = 1$, $x_i \Rightarrow x_{i+1}, x_n = 0$. τ is obviously unsatisfiable and has a PC refutation of degree 2 (or 3). However, $d_{NS}^K(\tau)$ is linear in n over any field.

Proposition 7.1. *PC refutations of degree O(1) are automatizable (see lecture 9 for the definition).*

Proof. The proof is essentially the Groebner basis algorithm. The idea is as follows. Let X_d be the linear space of polynomials of degree d you can infer. We iteratively look at all polynomials we can infer in the set of polynomials of degree $d + 1$, and intersect it back with the set of polynomials of degree d. $\qquad \square$

Efficient Interpolation Property: As a corollary, PC refutations of constant degree have the *efficient interpolation property*. That is, let $\tau = \tau', \tau'' = \tau'(x,y), \tau''(x,z)$ be unsatisfiable with $d_{PC}^K(\tau) \leq O(1)$. Then, given x, we can point out in polynomial time which of τ' and τ'' is not satisfiable.

Polynomial Calculus with Resolution: We would like our system to have the property that terms and their negations have the same *size*. However, this is far from true in PC: the negation of the clause $x_1 \cdot \cdots \cdot x_n$ is $(1 - x_1)(1 - x_2) \ldots (1 - x_n)$, which has much larger size, even though the degree is the same. *Polynomial Calculus with Resolution (PCR)* solves this problem. In PCR, we allow linear combinations of all clauses. Note that $d_{PC} = d_{PCR}$. To prove lower bounds for PCR, we can follow the same strategy as in proving lower bounds for resolution. In particular, we can take a PCR proof, hit it with a random restriction, this kills all fat clauses, so we end up with a PCR refutation of low degree, and then we prove that such a thing does not exist. Since the degree of PCR is the same as the PC degree, lower bounds on the PC degree are the key for lower bounds for PCR.

Lower Bounds for PC Refutations for which PC lower bounds are known include the following:

- Random CNF (Ben-Sasson, Impagliazzo, 99).

- Pigeon Hole Principle (PHP):

  Theorem 26 (Razborov, 95)**.**

  $$d_{PC}^k(PHP_n^m) \geq n/2.$$

  The proof technique is the "pigeon dance."

- Tsetin tautologies: we will prove the following result in class.

  Theorem 27 (Buss, Grigoriev, Impagliazzo, Pitassi, 99)**.**

  $$\text{char}(K) \neq 2 \Rightarrow d_{PC}^K(\tau(G,f)) \geq e(G)/2.$$

Recall from lecture 2 the definition of Tsetin Tautologies, and edge expansion. Given a graph G on n vertices, and a function $f : V(G) \to \{0,1\}$ such that $\oplus_{v \in V(G)} f(v) = 1$. Let $\{x_e \mid e \in E(G)\}$ be the set of variables. The Tsetin tautology is defined as $\tau(G,f) \equiv (\forall v \in V(G))(\oplus_{e \ni v} x_e = f(v))$. the

edge expansion of a graph is $e(G) = \min_{n/3 \le |V| \le 2n/3} |E(G) \cap (V \times \text{co-}V)|$. Recall the Ben-Sasson, Wigderson result from lecture 2:

$$w_R(\tau(G, f)) \ge e(G).$$

Proof of Theorem 27. The overall strategy of the proof is the following:

$$d_{PC}^K(\tau(G, f)) \ge w_g(\tau(G, f))/2 \ge w_R(\tau(G, f))/2 \ge e(G)/2.$$

Here $w_g(\tau)$ is the Gaussian width: the width of a refutation using the rule

$$\frac{f = \epsilon \qquad g = \delta}{f \oplus g = \epsilon + \delta}, \quad \epsilon, \delta \in \{0, 1\}$$

Note that $\tau(G, f)$ is a system of linear equations mod 2. We will show that for any system of linear equations mod 2, $d_{PC}^K(\tau) \ge w_g(\tau)/2 \ge w_R(\tau)/2$. The Ben-Sasson, Wigderson result from lecture 2 will conclude the proof.

In fact, it is known that $w_g(\tau) = w_R(\tau)$, but we will only show the inequality required in the proof ($w_g(\tau) \ge w_R(\tau)$). The other direction is left as homework. Given a Gaussian refutation, we want to construct a resolution refutation of the same width. We will achieve this by encoding each equation in the Gaussian refutation by clauses. Given an equation $x_1 \oplus x_2 \oplus x_3 = 0$, we can encode this by 4 clauses: $(x_1 \vee x_2 \vee \overline{x_3})$, $(x_1 \vee \overline{x_2} \vee x_3)$, $(\overline{x_1} \vee x_2 \vee x_3)$, $(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$. Clearly this transformation preserves width, and applying this encoding line by line to a Gaussian refutation, we obtain a resolution refutation.

We are left to show that $d_{PC}^K(\tau) \ge w_g(\tau)/2$ for any system of linear equations mod 2. We will switch from additive to multiplicative notation. In particular our variables $x_i \in \{0, 1\}$ will correspond to variables $y_i \in \{-1, 1\}$. Note that we make key use of the fact that $\text{char}(K) \ne 2$. The transformation is as follows:

$$x_i \to \frac{1 - y_i}{2} \quad (y_i = \exp(i\pi x_i))$$

$$x_1 \oplus x_2 \oplus \cdots \oplus x_\ell = \epsilon \quad \to \quad y_1 y_2 \ldots y_\ell = \epsilon'$$

$$x_i^2 = x_i \quad \to \quad y_i^2 = 1.$$

The main idea of the proof is about binomials. Consider a binomial b of the form $m_1 \pm m_2$, where m_1 and m_2 are monomials. Since for any monomial m, $m^2 = 1$, we have:

$$m_1 \pm m_2 = 0 \quad \Longleftrightarrow \quad m_1 m_2 = \pm 1$$

Therefore by reversing the map back to the x_i, we can associate a linear equation mod 2 to every binomial.

Given a PC refutation P of degree at most d, our goal is to construct a Gaussian refutation of degree at most $2d$. Let $f \in P$, our goal will be to show that we can write f in the form $f = \alpha_1 b_1 + \cdots + \alpha_t b_t$, where $\deg(b_i) \leq d$, and every b_i has a binomial (i.e., the one in which every line is a binomial) proof from our binomial axioms. Once we have this, it is an obvious exercise to prove that we have a Gaussian refutation of degree at most $2d$, since $\deg(m_1 m_2) \leq 2d$.

We will prove by induction on f the following. Given f such that $\deg(f) \leq d_0 \leq d$, we can write f as $f = \alpha_1 b_1 + \cdots + \alpha_t b_t$, where $\deg(b_i) \leq d_0$, and every b_i has a binomial proof from our binomial axioms of degree at most d.

The statement is obvious for the axioms. As long as $\deg(f) \leq d - 1$, we can apply the multiplication rule:

$$\frac{f = 0}{f \cdot x_i = 0}$$

The addition rule:

$$\frac{f = 0 \qquad g = 0}{\alpha f + \beta g = 0}$$

requires a bit more work. By induction, we are assuming that we can write

$$f = \alpha_1 b_1 + \cdots + \alpha_t b_t,$$

where $\deg(f) \leq d_0$, $\deg(b_i) \leq d_0$, and every b_i has a binomial proof of degree at most d. We can similarly decompose g. A problem can arise if $\deg(\alpha f + \beta g)$ is lower: all terms of degree $> d_0$ get cancelled. However, we can decompose h as $h = (\alpha f + \beta g) = \alpha_1 b_1 + \cdots + \alpha_t b_t$, where $\deg(b_i) \leq d$. Suppose $\deg(h) \leq d_0 < d$, and let M_d be the set of monomials of degree at most d, and M_{d_0} the set of monomials of degree at most d_0. Suppose there is some monomial $m \in M_d \setminus M_{d_0}$ such that m cancels out. Then let us look at all binomials containing m in the decomposition of h:

$$\alpha_1(m \pm m_1) + \alpha_2(m \pm m_2) + \cdots + \alpha_r(m \pm m_r) \tag{5}$$

We know that $\alpha_1 + \cdots + \alpha_r = 0$, since m cancels out. Therefore the polynomial (5) can be re-written as $\sum \alpha_{i,j}(m_i \pm m_j)$. Therefore if h has monomials of degree $> d_0$, we can eliminate them one by one, and by induction we can eliminate all monomials of high degree. During the whole process, we can

use the same trick to keep the degree $\leq d_0$ when applying the multiplication rule.

The only thing left to prove is that we can do the same thing in a Gaussian refutation. But this is straightforward, and is left as an exercise.

$\square$

This theorem leaves open the question of $\operatorname{char}(K) = 2$. Alekhnovich, Razborov (2002) prove lower bounds over any field (in particular for a random 3-CNF). Their techniques are based upon the notion of a pseudo-random generator in proof complexity that we will discuss at the end of this course.

7.1.3 Algebraic calculus and logic

Here is a list of different proof systems that combine algebraic calculus and logic, and the status of lower bounds for these systems.

- PCR: Polynomial Calculus and Resolution.

- $F_d + \operatorname{Count}_p$: lower bound by Beane and Riis (1998). Here F_d is bounded depth Frege. Count_p is the set of axioms that describe the following fact: if V is a set of size n, with p not dividing n, then there is no partition of V into sets of size p. More formally, there is a variable for each subset of V of size p, and the axioms say that we can pick two sets with non-empty intersection, and every point is covered by a selected set.

- $F_d[\operatorname{MOD}_p]$, where MOD_p means we are allowed to count modulo p. For this proof system we have no lower bounds, even for $p = 2$. Note that we have lower bounds for circuits of this form, but no lower bounds in proof complexity.

7.2 Geometric proof systems

For geometric proof systems, we will be working over $\mathbb{R}$. The clauses are transformed into inequalities as follows:

$$x_1 \vee \overline{x_2} \vee x_3 \quad \rightarrow \quad x_1 + (1 - x_2) + x_3 \geq 1.$$

After this transformation, a CNF τ corresponds to a polytope P, and τ is unsatisfiable if and only if $P \cap \{0,1\}^n = \emptyset$. We can compute P and test whether or not it is empty in polynomial time. The problem arises if $P \neq \emptyset$:

we need to show that it contains no $\{0, 1\}$ points. The obvious axioms we have are:

$$x_i \geq 0, \quad \text{and}$$

$$-x_i \geq -1 \quad (x_i \leq 1).$$

But the problem is that these describe the continuous cube $[0, 1]^n$, not $\{0, 1\}^n$ as needed for our purposes.

7.2.1 Cutting planes proof system

Every line of a cutting planes(CP) proof system looks like:

$$\sum_{i=1}^{n} \alpha_i x_i \geq t,$$

where a_i, t are integers. We will denote $\sum \alpha_i x_i$ by $\ell(x)$. The rules for CP are the *addition rule*:

$$\frac{f \geq 0 \qquad g \geq 0}{\alpha f + \beta g \geq 0},$$

and the *division rule*:

$$\frac{d\, \ell(x) \geq t}{\ell(x) \geq \lceil t/d \rceil}.$$

It is left as an exercise to prove that this proof system is complete, i.e. if $P \cap \{0, 1\}^n = \emptyset$, then we can infer $1 \leq 0$. (Hint: prove it contains resolution).

Good News: we have lower bounds for this proof system.
Bad News: we do not have any combinatorial lower bounds.

Feasible Interpolation Theorem: We did this for resolution, this is a stronger system, but the proof is cleaner.

Let $A(x, y)$ and $B(x, z)$ be two systems of clauses,

$$A \equiv \sum_k a_{i,k}\, x_k + e_i(y) \geq s_i \quad (i \in I)$$

$$B \equiv \sum_k b_{j,k}\, x_k + e'_j(z) \geq t_j \quad (j \in J)$$

Theorem 28 (standard interpolation theorem)**.** Given a size s CP refutation for $\{A, B\}$, there is a polynomial time algorithm that given x, will point out one of A or B that is unsatisfiable.

Proof. We are given an assignment $x_k = \alpha_k \in \{0, 1\}$. This gives us the following new systems of equations:

$$e_i(y) \geq s_i - \sum_k a_{i,k}\, \alpha_k = D_i$$

$$e_j'(z) \geq t_j - \sum_k b_{j,k}\, \alpha_k = D_j'$$

Now let us look at a line in the CP refutation (note that there are no more x variables, because they have been assigned). We will show that any line can be split as

$$\ell(y) + \ell'(z) \geq D \quad \Rightarrow \quad \ell(y) \geq D_0, \ \ell'(z) \geq D_1, \ \text{with} \ \ D \leq D_0 + D_1$$

in such a way that the inequalities $\ell(y) \geq D_0, \ \ell'(z) \geq D_1$ also have CP refutations of the same size.

The addition rule is easy. What about the division rule? Look at a line $d\,\ell(y) + d\,\ell'(z) \geq D$. By inductive assumption, we get the following inferences:

$$\frac{d\,\ell(y) \geq D_0}{\ell(y) \geq \lceil D_0/d \rceil} \qquad \frac{d\,\ell'(z) \geq D_1}{\ell'(z) \geq \lceil D_1/d \rceil}.$$

The only thing left to prove is the following inequality, which is left as an exercise:

$$\lceil D_0/d \rceil + \lceil D_1/d \rceil \geq \lceil (D_0 + D_1)/d \rceil.$$

$\square$

Actually we have a different interpolation theorem (with the same proof).

Theorem 29. If $a_{i,k} \geq 0$ (A is monotone), and $b_{j,k} \leq 0$ (B is antimonotone), then A and B are separated by a *real monotone circuit* (will be defined in the next lecture).

Feasible Interpolation for the Lovász-Schrijver system is discussed in Appendix A.2.

Lectures 16, 17 and 18

Date: March 3,5 and 10, 2009

We gave the application of the feasible interpolation theorem promised in the previous lecture: the CLIQUE-COLORING contradiction is hard for Cutting Planes. We reviewed a few more geometric proof systems, such as Positivestellensatz and Lovász-Schrijver (see Appendix A.2 below).

In the context of proof systems of the latter type, we discussed tight connections between propositional proof complexity and such important topics in linear and positive semi-definite optimization as LS-rank and integrality gaps. We actually proved one strong lower bound on the LS-rank for Tseitin tautologies.

And then we wrapped up this course with an informal discussion of deep connections between feasible provability of circuit lower bounds, feasible interpolation and pseudorandom function generators. That discussion closely followed the lines of Introduction to the paper
`http://people.cs.uchicago.edu/~razborov/files/res_k.ps`

A   Added in 2014

A.1   Proofs of Resolution Lower Bounds based on Restrictions

Scribe: Pooya Hatami, University of Chicago.
Date: February 18, 2014

The idea of a random restriction is to randomly assign each variable to $0, 1$ or $*$ where $*$ means that the variable remains untouched. Random restrictions are broadly used in proving lower bounds in complexity theory, since most computational models behave nicely according to random restrictions. For example, both the resolution rule $\dfrac{C \vee x \qquad D \vee \bar{x}}{C \vee D}$ and the weakening rule $\dfrac{C}{C \vee D}$ are preserved under any restrictions.

Here we will use the random restriction technique to prove refutation size lower bound for Tseitin tautologies.

For a set of vertices $S \subseteq V$, denote by $E(S)$ the set of all edges with at least one endpoint in S, and by $E(S, \overline{S})$ the set of edges with exactly one endpoint in S.

Definition. A graph G is an (r, γ)-*expander* if for every set of vertices S with $|S| \leq r$, $|E(S, \overline{S})| \geq \gamma |S|$.

Thus, when $r = n/2$, this is just the ordinary edge expansion and (the minimal possible) γ is the *Cheeger constant* of G.

Theorem 30. Let $G = (V, E)$ be a bounded degree (r, γ)-expander with $\gamma > 1$ and $r = \Omega(n)$, and let $f : V(G) \to \{0, 1\}$ be an odd labeling. Then $S_R(\tau(G, f)) \geq \exp(\Omega(n))$.

The assumption $\gamma > 1$ is not absolutely necessary in this theorem, but it will simplify the argument a bit.

Proof idea: Assume for contradiction that the refutation size is small and therefore the number of wide clauses is small. Then the idea is to show through a union bound that after randomly restricting a constant fraction of variables we can kill all the wide clauses, then combined with the width lower bound Theorem 2 conclude with a contradiction. A problem with this plan is that a random restriction will in fact kill the Tseitin formula. In order to resolve this we will need a slightly more complicated restriction.

Proof. Let $G = (V, E)$ be an (r, γ)-expander, and $f : V \to \{0, 1\}$ be an odd labeling of its vertices.

Choose $A \subseteq V$, $|A| \leq \epsilon r$, uniformly at random where $\epsilon > 0$ is a constant to be chosen later. The following lemma shows that A is contained in a set B of size at most r for which $G \backslash B$ is still a good enough expander ($\Delta(G)$ is the maximal degree of G).

Lemma 31. Let $d > 0$ be a constant and $\gamma > \gamma^*$ be constants. Let $G = (V, E)$ be a (r, γ)-expander, with $\Delta(G) \leq d$. Then there exists $\epsilon = \epsilon(d, \gamma, \gamma^*)$ such that the following holds: For every $A \subseteq V$ with $|A| \leq \epsilon r$ there exists $B \supseteq A$ such that $G \backslash B$ is $(\epsilon r, \gamma^*)$-expander, and $|B| \leq r$.

Proof. Let $B_0 := A$. Now at step $i \geq 0$, if $G \backslash B_i$ is an $(\epsilon r, \gamma^*)$-expander we stop with $B := B_i$. Otherwise, there must be a set $A_{i+1} \subseteq V(G) \backslash B_i$, $|A_{i+1}| \leq \epsilon r$ with expansion (in $G \setminus B_i$) less than γ^*. We will let $B_{i+1} := B_i \cup A_{i+1}$, however if $|B_{i+1}| > r$, then we stop with $B := B_i$.

This process will stop after $t < \infty$ steps, as the size of the sets B_t is strictly increasing. If $G - B = G - B_t$ is an expander of the desired form we are done. Otherwise, we know that $r - \epsilon r \leq |B_t| \leq r$. We will estimate the number of edges going out of $B_t = A \cup A_1 \cup \ldots \cup A_t$, $|E(B_t, \overline{B}_t)|$, in two ways.

$$r(1 - \epsilon)\gamma \leq |E(B_t, \overline{B}_t)| \leq \epsilon r d + r \gamma^*. \tag{6}$$

The first inequality follows from the fact that G is an (r, γ)-expander, and the second inequality follows from the fact that there are at most $\epsilon r d$ edges going out from A, and at most $(|A_1| + \ldots + |A_t|)\gamma^* \leq r\gamma^*$ from $A_1 \cup \ldots \cup A_t$. We conclude by choosing $\epsilon < \frac{\gamma - \gamma^*}{d + \gamma}$ which would contradict (6). $\qquad \square$

Call a partial assignment to the variables B-satisfying if it satisfies all the constraints for the vertices in B. Let R be the $\mathbb{F}_2$-affine space of all B-satisfying restrictions which assigns values to all the edges in $E(B)$. We will pick an assignment from R uniformly at random. We claim that with high probability a wide clause, which is a clause of width $> cn$, for some constant c, will be killed, meaning that at least one of its literals will be

assigned true. Let C be a wide clause. Notice that C contains a matching of size at least $cn/2d$, where $d = \Delta(G) \leq O(1)$. Moreover, the randomly selected set A will with high probability contain a constant fraction of this matching. Call this part of the matching M, $|M| = \Omega(n)$.

We claim that each assignment to M can be extended to equally many elements of R. This then would imply that the wide clause C will be assigned to true with exponentially high probability, which would then finish our proof by a union bound and Theorem 2.

Notice that R is the space of the solutions to the system of equations $\sum_{e \ni v} x_e = f(v)$ $(v \in B)$ over $\mathbb{F}_2$ (see Section 2.4). Any linear combination of these equations has the form $\sum_{e \in E(S, \bar{S})} x_e = \sum_{v \in S} f(v)$ for some $S \subseteq B$. We observe that $E(S, \bar{S})$ can not be a subset of M. This follows from the fact that M is a matching and that $\gamma > 1$. This implies that every assignment can be extended to same number of solutions in R finishing the proof. $\quad\square$

A.2 Feasible Interpolation for the Lovász-Schrijver Proof System

Scribe: Denis Pankratov, University of Chicago.
Date: February 20, 2014

Let $C_1, C_2, \ldots, C_k$ be clauses of a given CNF. Furthermore, suppose $C_i = \bigvee_{j=1}^{|C_i|} x_{ij}^{\epsilon_{ij}}$, where $\epsilon_{ij} \in \{0, 1\}$ is an indicator of whether the variable x_{ij} appears negated in the clause (cf. page 3). Like any geometric proof system (see more informal description at the beginning of Section 7.2), Lovász-Schrijver proof system starts out with the following axioms:

1. (integrality axioms) for each variable x_i we have $x_i^2 = x_i$ (note: these are quadratic constraints),

2. (clause axioms) for each clause C_i there is a corresponding axiom is $\sum_{j=1}^{|C_i|} x_{ij}^{1-\epsilon_{ij}} (1 - x_{ij})^{\epsilon_{ij}} \geq 1$ (note: these are linear constraints),

3. (bounded domain axioms) for each variable x_i we have $0 \leq x_i \leq 1$.

A proof in Lovász-Schrijver proof system proceeds deriving new statements in two steps called "lift" and "project". Corresponding to these two steps are the following two derivation rules:

1. (lifting step) $\dfrac{f(x) \geq 0 \qquad g(x) \geq 0}{f(x)g(x) \geq 0}$, where f and g are linear,

2. (projection step) $\dfrac{f_1(x) \geq t_1 \quad \ldots \quad f_k(x) \geq t_k}{\sum_{i=1}^{k} a_i f_i(x) \geq \sum_{i=1}^{k} a_i t_i}$, where $a_i \geq 0$, f_i
   can be either linear or quadratic, and the result $\sum_{i=1}^{k} a_i f_i(x)$ is linear.

This can be viewed as a refutation proof system - it starts out with an unsatisfiable CNF and the goal is to derive the inequality $0 \geq 1$. Or, as is often the case in combinatorial optimization, the original CNF can be *satisfiable*, and we are interested in deriving inequalities of the form $g \geq t$, where g is a fixed linear "goal" function, for as large values of t as possible.

The above proof system is denoted by LS_*. Note: this is a stronger notion than the original Lovász-Schrijver proof system (denoted by LS) where in the lifting step $f(x)$ is restricted to be of the form x_i or $(1 - x_i)$.

In the rest of this section we shall prove that LS_* has the feasible interpolation property. Feasible interpolation was previously discussed in these notes in Subsection 4.1, Definitions 4.3, and Theorems 15 and 28.

Theorem 32. There exists a polynomial time algorithm M with the following properties:

Input: • an unsatisfiable CNF formula $A(x, y) \wedge B(x, z)$, where x, y, z are disjoint sets of variables,

• P - an LS_* proof of unsatisfiability of $A(x, y) \wedge B(x, z)$,

• $a \in \{0, 1\}^{|x|}$ - a truth assignment to x.

Output: • A if $A(a, y)$ is unsatisfiable, or

• B if $B(a, z)$ is unsatisfiable.

Note: if both are unsatisfiable the algorithm can output either of the above answers. Also, like in all other known results of this kind, the algorithm actually outputs an *efficient* proof of the unsatisfiability of either $A(a, y)$ or $B(a, z)$ in *the same proof system LS_**. The latter property, however, is seldom used for applications.

Proof. Without loss of generality, assume that $|x| = 0$. Otherwise we can substitute a for x and simplify the entire proof P (i.e., we apply the restriction $x \leftarrow a$ to the formulas and the proof). Obviously, this can be done in polynomial time.

Therefore our refutation P is of the form

$$U_1(y, z) \geq t_1, U_2(y, z) \geq t_2, \ldots, U_m(y, z) \geq t_m,$$

where $U_m(y, z) = 0$, $t_m = 1$, and each U_i is either an axiom or follows from $\{U_j \mid j < i\}$ by inference rules. Furthermore, we can assume that each

$U_i(y, z)$ is linear (except for integrality axioms) if we allow performing both lifting and projection as a single step. This assumption will simplify the proof, and it can be made without loss of generality, because linear rule is required to kill all quadratic terms anyway.

Our algorithm M will split the above proof P into two:

$$P^{(1)} = U_1^{(1)}(y) \geq t_1^{(1)}, U_2^{(1)}(y) \geq t_2^{(1)}, \ldots, U_{2m}^{(1)}(y) \geq t_{2m}^{(1)},$$

$$P^{(2)} = U_1^{(2)}(z) \geq t_1^{(2)}, U_2^{(2)}(z) \geq t_2^{(2)}, \ldots, U_{2m}^{(2)}(z) \geq t_{2m}^{(2)},$$

(compare with a similar argument for Cutting Planes in Section 7.2.1) such that $P^{(1)}$ consists of valid inferences from the axioms of $A(y)$, $P^{(2)}$ consists of valid inferences from the axioms of $B(z)$, and $t_i^{(1)} + t_i^{(2)} \geq t_i$ and $U_i^{(1)}(y) + U_i^{(2)}(z) = U_i(y, z)$.

Observe that the above implies that $U_{2m}^{(1)}(y) = U_{2m}^{(2)}(z) = 0$ and at least one of $t_{2m}^{(1)}$ or $t_{2m}^{(2)}$ is strictly positive. Thus, at least one of $P^{(1)}$ or $P^{(2)}$ represents a valid refutation of A or B, respectively. Once our algorithm constructs proofs $P^{(1)}$ and $P^{(2)}$, it can tell which of the two formulas A or B is unsatisfiable by examining the last inequality in the proof.

We shall demonstrate how to perform the splitting of P efficiently by induction on i - the index of statements in P. Notation: $P_{<i} := \{U_j(y, z) \geq t_j \mid j < i\}$, $P_{<i}^{(1)} := \{U_j^{(1)}(y) \geq t_j^{(1)} \mid j < 2i - 1\}$, $P_{<i}^{(2)} := \{U_k^{(2)}(z) \geq t_k^{(2)} \mid k < 2i - 1\}$.

Base case. If the current statement of P is an axiom then it either involves only variables of A or only variables of B, since the two do not have any variables in common. Suppose that it only involves the variables of A, then we set $U_i^{(1)}(y) = U_i(y, z)$, $t_i^{(1)} = t_i$, $U_i^{(2)}(z) = 0$, $t_i^{(2)} = 0$. For the case when it involves only variables of B we proceed similarly.

Induction step. In general $U_i(y, z)$ is linear and is obtained via the inference rules from $\{U_j \mid j < i\}$. By induction we can represent $U_i(y, z)$ as $U_i(y, z) = T_1(y) + T_2(z) + T_3(y, z)$, where

$$T_1(y) = \sum_{j,k<i} \alpha_{jk}(U_j^{(1)}(y) - t_j^{(1)})(U_k^{(1)}(y) - t_k^{(1)}) + \sum_{j<i} \beta_j y_j (1 - y_j),$$

$$T_2(z) = \sum_{j,k<i} \gamma_{jk}(U_j^{(2)}(z) - t_j^{(2)})(U_k^{(2)}(z) - t_k^{(2)}) + \sum_{j<i} \delta_j z_j (1 - z_j),$$

$$T_3(y, z) = \sum_{j,k<i} \phi_{jk}(U_j^{(1)}(y) - t_j^{(1)})(U_k^{(2)}(z) - t_k^{(2)}).$$

Since our projection step in the inference rules requires the result to be linear, the quadratic terms in the above expressions disappear. In particular, $T_1(y)$ can be derived from $P_{<i}^{(1)}$ and $T_2(z)$ can be derived from $P_{<i}^{(2)}$; we take them as $U_{2i-1}^{(1)}(y) \geq t_{2i-1}^{(1)}$ and $U_{2i-1}^{(2)}(y) \geq t_{2i-1}^{(2)}$, respectively. It is left to show that $T_3(y, z)$ can be split into two distinct parts depending only on y and z each having a derivation from $P^{(1)}$ and $P^{(2)}$, respectively.

We know that $T_3(y, z)$ can be represented as $T_3(y, z) = R(y) + S(z) - t$, where R and S are linear and $t \in \mathbb{R}$. Let n_Y denote the number of y-variables and n_Z denote the number of z-variables. Therefore, for all $(y, z) \in \mathbb{R}^{n_Y + n_Z}$ we have the inequality

$$(\forall j, k < i) \left(U_j^{(1)}(y) \geq t_j^{(1)} \wedge U_k^{(2)}(z) \geq t_k^{(2)} \right) \Rightarrow R(y) + S(z) \geq t. \quad (7)$$

Now, consider the following linear program (LP):

$$\begin{aligned} \min_{y,z} \quad & R(y) + S(z) \\ \forall j < i \quad & U_j^{(1)}(y) \geq t_j^{(1)} \\ \forall k < i \quad & U_k^{(2)}(z) \geq t_k^{(2)} \end{aligned}$$

By (7), the value of this LP is at least t. By LP duality the value of the dual is at least t too. Let a_j, b_k be an optimal solution to the dual. Then we can derive

$$R(y) + S(z) = \sum_{j < i} a_j U_j^{(1)}(y) + \sum_{k < i} b_k U_k^{(2)}(z) \geq \sum_{j < i} a_j t_j^{(1)} + \sum_{k < i} b_k t_k^{(2)} \geq t.$$

Since the dual can be constructed and solved efficiently, we were able to split $T_3(y, z)$ into two parts, as desired, i.e.:

- $\sum_{j < i} a_j U_j^{(1)}(y) \geq \sum_{j < i} a_j t_j^{(1)}$, which can be derived from $P_{<i}^{(1)}$,
- $\sum_{k < i} b_k U_k^{(2)}(z) \geq \sum_{k < i} b_k t_k^{(2)}$, which can be derived from $P_{<i}^{(2)}$.

We list them as $U_{2i}^{(1)}(y) \geq t_{2i}^{(1)}$ and $U_{2i}^{(2)}(y) \geq t_{2i}^{(2)}$, respectively. This completes the induction.

$\square$

By the same token as for the Cutting Planes (see Section 7.2.1), this gives *conditional* size lower bounds for the system LS_* modulo (say) cryptographic assumptions. More technically, we need a pair of disjoint NP-sets

that can not be separated by any set in P/poly. However, since we do not know if the monotone version of the feasible interpolation theorem holds here or not (the last step in the above argument does require linear programming), unconditional size lower bounds are currently out of reach even for the weaker system LS. In fact, I would not be very much surprised if the CLIQUE-COLORING contradiction *would* have had short LS-proofs (a natural separator might be provided by Lovász's θ-function).

A.3   Elements of Space Complexity

<div align="center">
Scribe: Yuan Li, University of Chicago.
Date: February 27, 2014
</div>

Space complexity is about the *space* needed by proofs. Informally, space complexity deals with problems of the following kind: if there are only 6 white boards in the classroom, is there a proof (of some theorem) which can be presented on those white boards (such that every step follows from some previous ones which have not been erased)?

Let's make the definition precise for resolution proof system.

Definition A.1. (syntactical resolution derivation) Let τ be an unsatisfiable CNF. A *configuration* is a set of clauses (that can also be naturally viewed as a CNF). A *resolution proof* for τ is a sequence of configurations $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_t$ such that $\mathcal{M}_0 = \emptyset$, $\mathcal{M}_t = \{\bot\}$, and for all $i \in [t]$, $\mathcal{M}_i$ is obtained from $\mathcal{M}_{i-1}$ by one of the following rules:

- Axiom Download: $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ for some clause $C \in \tau$.

- Memory Erasing: $\mathcal{M}_i = \mathcal{M}_{i-1} - \mathcal{M}'$ for some $\mathcal{M}' \subseteq \mathcal{M}_{i-1}$.

- Inference: $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ for some clause C obtained by a single application of the resultion rule to two clauses in $\mathcal{M}_{i-1}$ or weakening rule to one clause in $\mathcal{M}_{i-1}$.

Definition A.2. (clause space) The *clause space* of a set of configurations $\pi = \{\mathcal{M}_0, \ldots, \mathcal{M}_t\}$ is

$$CSpace(\pi) := \max\{|\mathcal{M}_i| : i \in [t]\},$$

where $|\mathcal{M}_i|$ denotes the number of clauses in $\mathcal{M}_i$. The *resolution clause space* of an unsatisfiable CNF τ is

$$CSpace(\tau) := \min\{CSpace(\pi)\},$$

where the minimum is taken over all refutations of τ.

Other than the number of cluases, we can measure the space by the number of variables (literals).

Definition A.3. (total space) The *total space* of a configuration $\mathcal{M}$ is $\sum_{C \in \mathcal{M}} |C|$, where $|C|$ is the number of literals in C. Let

$$VSpace_R(\pi) := \max\{\sum_{C \in \mathcal{M}_i} |C| : i \in [t]\},$$

and

$$VSpace_R(\tau) := \min\{VSpace_R(\pi)\},$$

where, again, the minimum is taken over all refutations of τ.

Theorem 33. If τ is a contradictiory set of clauses over n variables, then $CSpace(\tau) \leq n + 1$.

Proof. (Sketch) Consider a complete binary tree of depth $n + 1$. The root is 1 with two children x_1 and $\overline{x_1}$; x_1 has children $x_1 \vee x_2$ and $x_1 \vee \overline{x_2}$; $\overline{x_1}$ has children $\overline{x_1} \vee x_2$ and $\overline{x_1} \vee \overline{x_2}$; $\cdots$ To prove τ is unsatisfiable, perform DFS (depth-first search) on this tree, which requires (clause) space $n + 1$. □

Now, let us prove a space lower bound for Pigeon-Hole Principle (PHP). Denote by PHP_n^{n+1} the following CNF, as in Section 6.2. Let $i \in [n + 1]$ denote pigeon; $j \in [n]$ denote hole. For every $i \in [n + 1], j \in [n]$, let $x_{ij} \in \{0, 1\}$ be the indicator variable that ith pigeon flies to jth hole. The CNF PHP_n^{n+1} consists of the following clauses:

- ($\mathcal{P}$) $\overline{x_{i_1 j}} \vee \overline{x_{i_2 j}}$ for every $i_1 \neq i_2 \in [n + 1]$, $j \in [n]$.

- ($\mathcal{R}$) $\vee_{j=1}^n x_{ij}$ for $i \in [n + 1]$.

For convenience, denote the set of the clauses of the first type by $\mathcal{P}$, the second by $\mathcal{R}$.

Let Γ be the set of matching restrictions[1]. A *matching restriction* is defined as follows: take a partial matching between $[n + 1]$ and $[n]$, say $(i_1, j_1), \ldots, (i_\ell, j_\ell)$, $\ell \leq n$, assign $x_{i_1 j_1}, \ldots, x_{i_\ell j_\ell}$ to 1, and everything else to $\{0, *\}$ arbitrarily. Define the *size* of a matching restriction ρ to be the number of assigned variables, that is, $|\rho^{-1}(\{0, 1\})|$.

We will prove:

---

[1] The definition of matching restriction here is slightly different from that in Section 6.2, in which everything else touching the vertices of the matching must be set to 0 other than $\{0, *\}$.

Theorem 34.
$$CSpace(PHP_n^{n+1}) > n.$$

(Notice that the number of variables in PHP_n^{n+1} is $n(n+1)$.)

The main idea of the proof is the following: assume there is a small size proof $\pi = \{\mathcal{M}_0, \ldots, \mathcal{M}_t\}$, then for every i, we inductively prove that there exists a restriction $\rho_i \in \Gamma$ which satisfies every clause in $\mathcal{M}_i$. Since $\mathcal{M}_t = \{\bot\}$ is unsatisfiable, it will be a contradiction.

To help the reader better understand its proof, we point out that only the following properties about PHP_n^{n+1} are used:

(a) For every $\rho \in \Gamma$ and $\rho' \subseteq \rho$, we have $\rho' \in \Gamma$. In words, Γ is downward closed.

(b) For every $\rho \in \Gamma$ and $C \in \mathcal{P}$, restriction ρ and clause C are consistent.

(c) For every $\rho \in \Gamma$ with size $< n$, and $C \in \mathcal{R}$, there exists some $\rho' \in \Gamma$ such that ρ' satisfies C and $\rho \subseteq \rho'$.

Since the proof of the lower bound only depends on the above properties, the lower bound of $CSpace(PHP_n^{n+1})$ can be generalized to other CNFs satisfying certain criterion. Interested readers may refer to the paper "Space complexity in propositional calculus" by Aleknovich, Ben-Sasson, Razborov, and Wigderson (2001).

Proof. (of Theorem 34) Assume for contradiction that there is a proof $\pi = \{\mathcal{M}_0 = \emptyset, \mathcal{M}_1, \ldots, \mathcal{M}_t = \{\bot\}\}$ of PHP_n^{n+1} such that $|\mathcal{M}_i| \leq n$ for every i.

We prove by induction that

$$\text{for every } i \in \{0, \ldots, t\}, \text{ there exists } \rho \in \Gamma \text{ such that } \rho \models \mathcal{M}_i, \qquad (8)$$

where $\rho \models \mathcal{M}_i$ means ρ satisfies all clauses in $\mathcal{M}_i$.

For the basis $\mathcal{M}_0 = \emptyset$, there is nothing to prove. For the induction step, assume that for $\mathcal{M}_{i-1}$, there exists $\rho' \in \Gamma$ such that $\rho' \models \mathcal{M}_{i-1}$. By Definition A.1, there are 3 cases. Memory Erasing and Inference are trivial; simply let $\rho = \rho'$ and (8) will be satisfied. Axiom Download is the only nontrivial case, where $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$, $C \in \mathcal{P} \cup \mathcal{R}$. For the case $C \in \mathcal{P}$, by property (b), let $\rho = \rho'$, and (8) will be satisfied.

For the case $C \in \mathcal{R}$, we claim that there exists some $\rho'' \subseteq \rho'$ of size $\leq |\mathcal{M}_{i-1}|$ such that $\rho'' \models \mathcal{M}_{i-1}$. Let us prove the claim first. For each clause $D \in \mathcal{M}_{i-1}$, we know $D|_{\rho'} = 1$, which implies that restriction ρ' sets at least one literal in D to 1. For each clause $D \in \mathcal{M}_{i-1}$, pick one such

literal arbitrarily, set it to 1 and add to ρ'' (initially, ρ'' is empty). It's easy to see that ρ'' satisfies all clauses in $\mathcal{M}_{i-1}$, and has size $\leq |\mathcal{M}_{i-1}|$. By our assumption, $|\mathcal{M}_i| \leq n$ and thus $|\mathcal{M}_{i-1}| \leq n-1$, which implies that the size of ρ'' is $\leq n-1$. By property (a), $\rho'' \in \Gamma$. By property (c), there exists some $\rho \supseteq \rho''$ satisfying clause C, which completes our induction step of (8).

Since $\mathcal{M}_t = \{\bot\}$ is unsatisfiable, (8) can not hold, and thus our assumption $|\mathcal{M}_i| \leq n$ for all i is false. $\qquad\square$

Let us end this lecture by listing a few open problems; more details and more problems resulted from this mini-course can be found at
`http://people.cs.uchicago.edu/~razborov/teaching/index.html`

- Prove nontrivial ($\omega(n)$) lower bounds for total space[2].

- Prove trade-off results between size and space. If τ has refutation proof of size s, is it true that there must exist size $s^O(1)$ and $O(n)$ space proof? Beame, Beck and Impagliazzo (2012) proved that, there exists CNF that has proof of space and size each roughly $s^{\log_2 s}$ for which any proof using space s and size t requires

$$t \geq (s^{0.58 \log_2 s}/s)^{\Omega(\log \log s / \log \log \log s)}.$$

  Beck, Nordström and Tang (2012) proved similar results for Polynomial Calculus. Prove stronger trade-off results.

- Recall that PCR (Polynomial Calculus and Resolution) is the proof system that mixes algebra with Boolean world, that is, Polynomial Calculus with resolution rule. Are there any relationships between degree and PCR clause space?

- Prove space lower bounds for Cutting Plane system.

---

[2]Added in proof. This was solved by Bonacina, Galesi and Thapen (see `http://eccc.hpi-web.de/report/2014/038/`) while these notes were being prepared.