6.841 Advanced Complexity Theory Apr 27, 2009

Lecture 21

Lecturer: Jakob Nordström *Scribe: Rishi Gupta*

Today and Thursday we will give an (extremely) selective overview of **Proof Complexity**. We will cover some major concepts, give an overview of the area, prove a classic result, and develop some nice tools along the way.

Introduction to Proof Systems

Proof Complexity is about figuring out how hard it is to prove things. But what is a proof? I claim 25957 is a product of two primes. What would we accept as proof of this fact? Some possibilities:

- "Left to the reader". In a sense, every statement is its own proof.
- Proof by exhaustion: compute 25957 mod 2 and every odd number less than 25957, and see that exactly 2 of them are 0.
- $25957 = 101 \cdot 257$, check that 101 and 257 are prime.

Observation by Cook: A proof should be *efficiently* verifiable. In particular, a *proof system* $\mathcal{P}$ for a language L is a deterministic function $\mathcal{P}(x, \pi) \in \{0, 1\}$, polynomial-time in x and π, where

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \text{there exists } \pi \text{ such that } \mathcal{P}(x,\pi) = 1 \text{ (a proof exists)} \\ x \notin L \quad &\Rightarrow \quad \text{for all } \pi, \mathcal{P}(x,\pi) = 0 \text{ (there are no fake proofs)} \end{aligned}$$

Note that there is no restriction on the length of π.

Propositional Proof Complexity

We're interested in proving *tautologies*, namely (boolean) formulas that are true no matter how you assign values to the variables. For instance, $(x \vee \bar{x}) \in$ TAUT. Why are they interesting?

- For a while, people thought this would help with the P vs NP question.
- It provides a framework for quantifying/understanding limits of different kinds of mathematical reasoning. Proof systems have varying strengths, like the complexity classes we're familiar with.
- Commercial SAT-solvers are absurdly quick, and routinely solve formulas that have millions of variables. In fact, good SAT-solvers are now roughly $\Theta(n)$. This subfield is called automated theorem proving.

A proof system has complexity g if every $x \in \mathcal{P}$ has a proof of size $g(|x|)$. If g is polynomial we say $\mathcal{P}$ is a polynomially-bounded [propositional] proof system, or a pbpps for short.

Observation by Cook and Reckhow (1979): NP = coNP if and only if there exists a polynomially-bounded propositional proof system for TAUT (easy exercise).
Corollary: If there is no pbpps for TAUT, P $\neq$ NP, since P is closed under complementation.

Proof systems are sort of like circuits; we study stronger and stronger systems, and prove lower bounds on things. As with circuits, it's hard to look at the general case, so we restrict as much as reasonable, but we still haven't been able to say much.

Examples of Proof Systems

Notational convenience: We switch from proving tautologies to refuting unsatisfiable formulas. Note that many papers in the field use the word tautology to refer to both something that is always true and something that is never true. We use capital letters to denote clauses, and lowercase letters to denote literals (variables).

Also, WLOG we can require all formulas to be in CNF. This follows from Cook's theorem (everything in NP is reducible to SAT), but we can do it with only linear blowup as well. Roughly, the steps are:

(a) Introduce new variables for every subformula.

(b) Add constraints showing values propogate correctly.

(c) Negate the formula (in line with the notational convenience noted above)

As an example, after steps (a) and (b), $F = G \Rightarrow H$ would turn into $(\bar{x_F} \vee \bar{x_G} \vee \bar{x_H}) \wedge (x_F \vee x_G) \wedge (x_F \vee \bar{x_H})$. Ands and ors and so on can similarly be encoded.

Some classic proof systems are:

- Truth table: Simply a table of all the 2^n possible assignments, and whether each assignment satisfies the formula or not.
- Resolution: We write new clauses using the rule $(C \vee x) \wedge (D \vee \bar{x}) \Rightarrow (C \vee D)$. If we eventually imply the empty clause, the formula was unsatisfiable. An example will be presented later. [add in bar notation]
- Cutting Planes: Replace $(x \vee y \vee \bar{z})$ with linear constraints like $x + y + (1 - z) \geq 1$, $0 \leq x \leq 1$, etc. as we've seen before. We then write everything in the form $\sum c_i x_i \geq a$, where c_i, a are constants. Addition and multiplication by scalars work as expected, and then special power is obtained by the rule for division: if $d | c_i$ for each i, then $\sum c_i x_i \geq a \Rightarrow \sum c_i/d x_i \geq \lceil a/d \rceil$.
- Frege system: Like resolution, but with clauses: $(A \vee C) \wedge (B \vee \bar{C}) \Rightarrow (A \vee B)$. We call this the cut rule. The Frege system is one of the strongest widely studied proof systems.

Note that we need all of these to be sound and complete. Soundness is clear for all of them, though completeness is slightly non-apparent for all but truth tables. We'll show soundness for Resolution later on.

We say $\mathcal{P}_1$ simulates $\mathcal{P}_2$ if there is a poly-time function f that maps proofs in $\mathcal{P}_2$ to proofs in $\mathcal{P}_1$. P-equivalence (P as in polynomial) and $\leq_P$ are defined in the obvious way.

Comparison of Proof Systems

We look at some statements that can be nicely encoded as Satisfiability questions.

- **Graph Tautology (GT_n)** We want to show that every transitive DAG has a source. We set variables x_{ij} as 1 or 0 if edge (i, j) does or does not exist. Unsatisfiability can then represented as:

$$\bigwedge_{i \neq j \neq k \neq i} \bar{x_{ij}} \vee \bar{x_{jk}} \vee x_{ik} \bigwedge_{i \neq j} \bar{x_{ij}} \vee \bar{x_{ji}} \bigwedge_{j} \bigvee_{i \neq j} x_{ij}.$$

  I'm pretty sure I copied down from the board correctly, but I feel like the first big And is incorrect .. I'll check up on this. It's not too hard to see that this is the correct idea, though.

- **Pigeonhole Principle (PHP$_n^m$)** We want to show that for m pigeons and n holes, with $m > n$, there are 2 pigeons that share a hole. Unsatisfiability can be encoded as:

$$\bigwedge_{i \in [m]} \bigvee_{j \in [n]} x_i j \wedge \bigwedge_{i_1, i_2 \in [m], j \in [n]} (\bar{x_{i_1 j}} \vee \bar{x_{i_2 j}}).$$

  The first big And says every pigeon has a hole, and the second And says that no two pigeons share a hole.

- **Random k-CNF Formula** Note that these will be unsatisfiable with high probability.

We look at what happens as n goes to infinity.

	Poly-bounded	GT_n	PHP_n^{n+1}	Random k
TT	no	exponential	exponential	exponential
Res	no[1] for next time!	$O(n^3)$	exponential[2]	exponential
CP	no	$O(n^3)$	poly(n)[3]	open[4]
Frege	big open question[5]	poly(n)	poly(n)[6]	open

1) Proved in 1985, and the result we'll prove on Thursday. 2) Breakthrough result. 3) Good exercise. 4) Probably more tractable than the other open problems. 5) Similar to big open questions in circuit theory. 6) Minor surprise when it was shown.

It turns out we can also show that TT $\leq_P$ Res $\leq_P$ CP $\leq_P$ Frege. Res $\leq_P$ CP is a nice exercise to work out.

Lower bound on PHP with Res

For the rest of class we will set up the machinery to prove the lower bound on PHP with Res. We won't do it the way Hawkin originally did it, since it was relatively painful.

To clear up some notation: A literal is a variable or the negation of a variable. A clause is a disjuncton of literals, like $(x \vee y \vee z)$, and we often think of clauses as sets of literals. A CNF formula is a conjunction of clauses, like $A \wedge B$, and we often think of formulas as sets of clauses. We let vars(F) denote the variables of a formula F. We write $F \models D$ if any assignment that satisfies F satisfies D. We denote $\{1, 2, \ldots, n\}$ by $[n]$, and set 1 to be true, and 0 to be false. WLOG, we assume any formula F is in CNF form.

A *Resolution derivation* $\pi : F \vdash D$ is a sequence of clauses $\pi = (D_1, D_2, \ldots, D_L)$, where each D_i is either in F or the result of the resolution rule applied to clauses appearing earlier in the list. In the former case we call D_i an *axiom*. A *Resolution refutation* of F is a $\pi : F \vdash 0$ (the empty clause).

Example: A resolution refutation of $(\bar{x} \vee y) \wedge (x \vee z) \wedge (\bar{z} \vee w) \wedge (x \vee \bar{w}) \wedge (\bar{x} \vee \bar{y})$ is

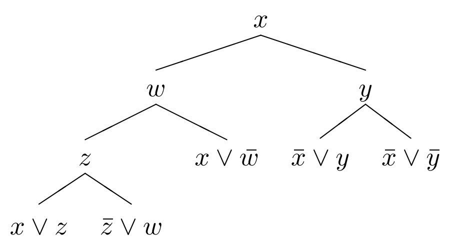
$\bar{x} \vee y$	axiom
$x \vee z$	axiom
$\bar{z} \vee w$	axiom
$x \vee \bar{w}$	axiom
$\bar{x} \vee \bar{y}$	axiom
$x \vee w$	Res(2,3)
x	Res(4,6)
$\bar{x}$	Res(1,5)
0	Res(7,8)

We define the *length* $L(\pi)$ to be the number of clauses in π. For instance, in this case $L = 9$. We define $L(F \vdash 0)$ to be the length of the shortest resolution refutation of F.

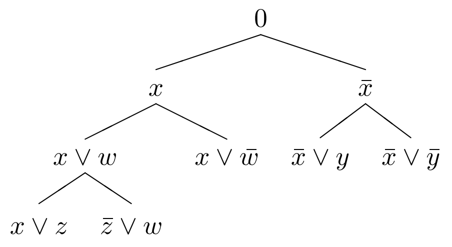
For technical convenience, we also add in a rule called *weakening*, which takes C to $(C \vee D)$. This is not actually needed, and weakening can always be eliminated from any refutation (easy exercise).

The soundness of Res is obvious, but we give a proof sketch of completeness:

We first build a search tree to find a satisfying assignment for F. At each vertex, we check a variable, and stop each branch when it directly contradicts an axiom. Note that if F is unsatisfiable, every branch must eventually end this way, in depth n. A possible search tree for the example above would be (pretend there are 0s on all the left edges and 1s on the right):



From here, we can build the resolution refutation from the bottom up, as follows:



$\triangleright$

We let $L_T(F \vdash 0)$ be the length of the shortest tree-like refutation of F. Note that $L_T \leq$ max number of nodes in the tree $= 2^{n+1} - 1$.

We define a *restriction* or partial truth assignment ρ on a clause or formula in the obvious way; restricting our example above with $\bar{w}$ would leave $(\bar{x} \vee y) \wedge (x \vee z) \wedge (\bar{z}) \wedge (\bar{x} \vee \bar{y})$. If π is a refutation of F then $\pi_{\restriction\rho}$ is a refutation of $F_{\restriction\rho}$ (simple exercse in induction, weakening comes in handy).

We define the width of π to be the length of the longest clause; in the example above it would be 2. We let $W(F \vdash 0)$ be smallest width of any refutation of F.

On Thursday we'll use the machinery we just developed to show that PHP with Res is exponential. The general line of attack will be to show that if the width of a formula is large, so is its length, and then we'll show that the width of PHP_n^{n+1} is large.