Welcome to Proof Complexity + Applications!

Toni Pitassi toni@cs.columbia.edu

Lectures : Thurs 1:10 - 4

(usually we will end at 3:30)

See webpage for up-to-date info, syllabus, lecture notes, etc. www.cs.columbia.edu/~toni + follow Teaching Link

* No class next week (Jun 30).

Topics we will cover

1. Resolution UBs, LBs
2. CPs UBs, LBs, Feasible Interp + Automatizability
3. Frege Systems, Bded depth Frege, EF
4. Semi-Algebraic Pf Systems (Sherali Adams, Poly Calculus/Nsatz, SOS)
5. Applications / connections

Random + Semi Random CSPs + LDC's

TFNP

LBs for monotone circuit models, extended formulations

SAT solving / Thm Proving

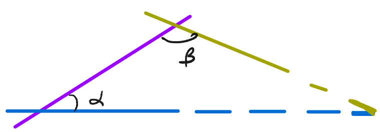
INTRO TO PROOF COMPLEXITY

A proof is an efficiently verifiable certificate of something

Example 1 Euclidean geometry (300 BC "Elements")

Euclid's Postulates

1. A straight line segment connects any 2 points
2. A straight line segment can be extended indefinitely in a straight line
3. given any straight line segment, a circle can be drawn having the segment as radius and one endpt as center
4. All right angles are congruent
5. If sum of $\alpha+\beta < 180$ then the 2 lines (blue + yellow) eventually meet (on same side as α, β angles)



Example 2 Peano Arithmetic

PA = system of First Order Logic, with function symbols $+$, $\cdot$, S
predicate symbols $=$, $\leq$, $\geq$

Plus Logical relations + quantifiers : $\vee$, $\wedge$, $\neg$, $\forall$, $\exists$

Logical Axioms / Rules + Arithmetic Axioms + Induction

Our focus will be Propositional, & Algebraic Proofs

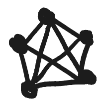
No quantifiers !

Domain of variables : usually finite (Boolean Finite group)

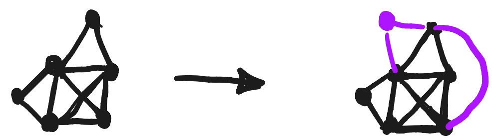
I. Graph non colorability & Hajos Calculus

How to prove a graph is Not 4-colorable?

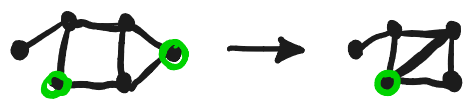
Hajos Calculus:

0. (Axiom) 

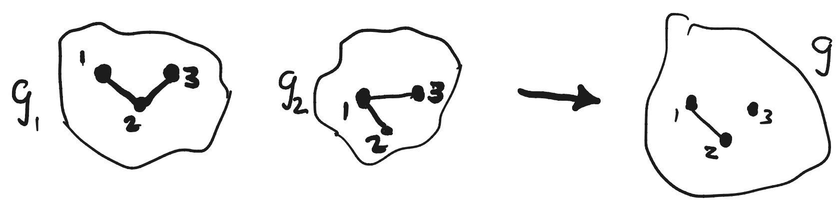
1. Add vertices, edges:



2. Contract Nonadjacent vertices



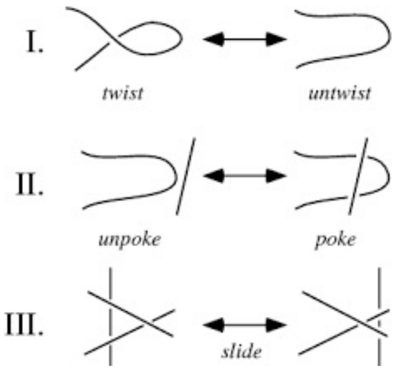
3. "Cut"



II. Knot Theory & Reidemeister Moves

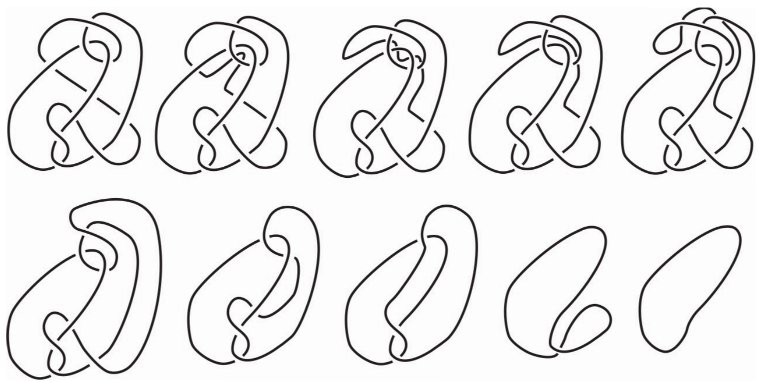
How to prove topological equivalence of 2 objects?

Reidemeister Moves:



II. Knot Theory & Reidemeister Moves

Example: The culprit knot is an "unknot"



III. Unsatisfiability & Propositional Proofs

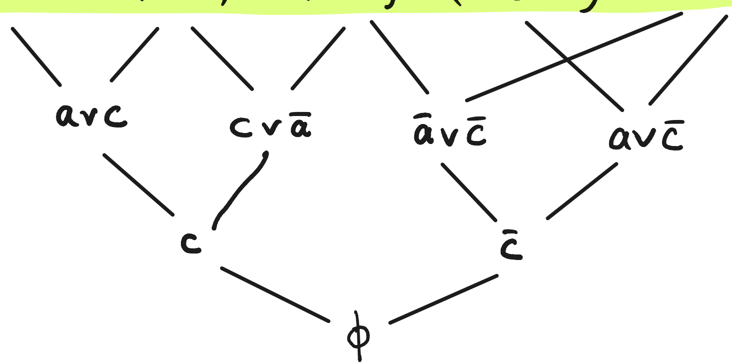
How to prove unsatisfiability of a CNF formula $f = C_1 \wedge C_2 \wedge \dots \wedge C_m$?

III. Unsatisfiability & Propositional Proofs

How to prove unsatisfiability of a CNF formula $f = C_1 \wedge C_2 \wedge \dots \wedge C_m$?

RESOLUTION REFUTATION:

$$f = (a \vee b)\ (\bar{b} \vee c)\ (\bar{a} \vee b)\ (a \vee b \vee \bar{c})\ (\bar{b} \vee \bar{c})$$



IV. Solvability of polynomials & Hilbert's Nullstellensatz

How to prove a system of polynomial equations is unsolvable?

$$\mathcal{P} = \{ p_1(x_1..x_n) = 0, \ldots, p_m(x_1..x_n) = 0 \}$$

Hilbert's Nullstellensatz A Nsatz (NS) proof of unsolvability of $\mathcal{P}$ (over algebraically closed field) is a set of poly's $Q = \{q_1, \ldots, q_m\}$ such that $\sum_{i=1}^{m} p_i(\vec{x})\, q_i(\vec{x}) = 1$

IV. Hilbert's Nullstellensatz

How to prove unsatisfiability of a CNF formula $f = C_1 \wedge C_2 \wedge \dots \wedge C_m$?

Nullstellensatz Refutation:

CNF $f = (x_1 \vee x_2 \vee x_3)\ (x_2 \vee \bar{x}_3)\ (\bar{x}_1)\ (\bar{x}_2)$

$$P(f) = \{ \underbrace{(1-x_1)(1-x_2)(1-x_3)}_{P_1} = 0,\ \underbrace{(1-x_2)x_3}_{P_2} = 0,\ \underbrace{x_1}_{P_3} = 0,\ \underbrace{x_2}_{P_4} = 0 \}$$

Convert each clause to an equivalent polynomial equation

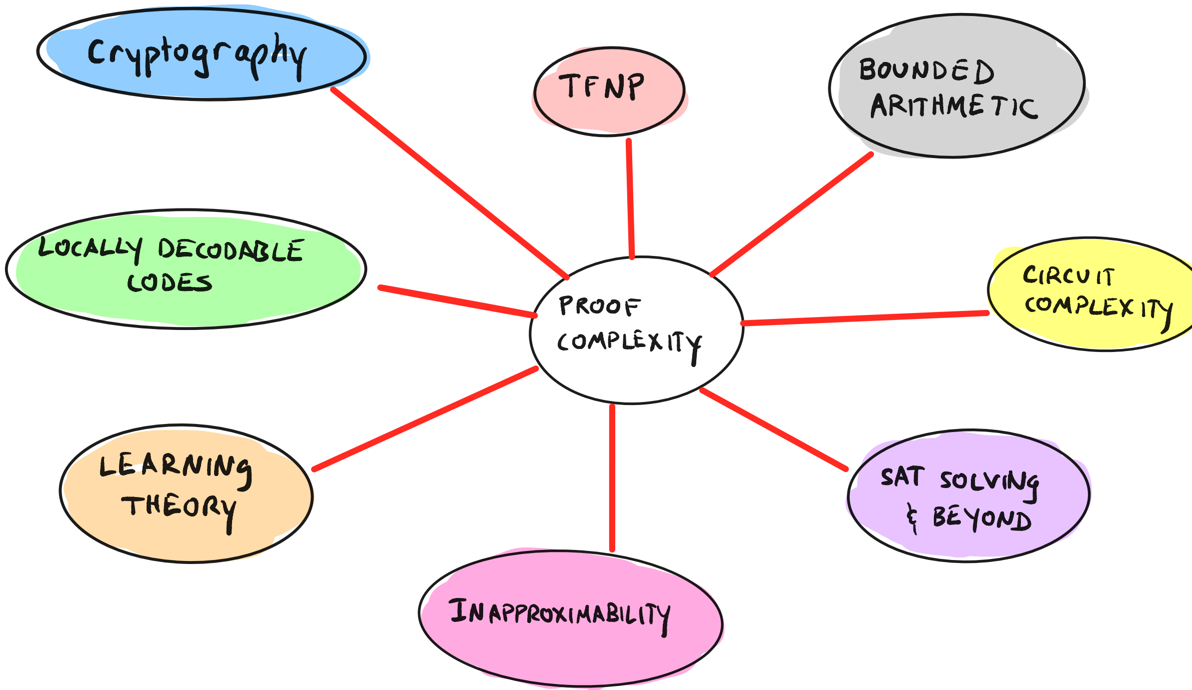
NS Refutation: $\underbrace{1}_{q_1} \cdot P_1 + \underbrace{(1-x_1)}_{q_2} P_2 + \underbrace{(1-x_2)}_{q_3} P_3 + \underbrace{1}_{q_4} \cdot P_4 = 1$

BIRTH OF PROPOSITIONAL PROOF COMPLEXITY

- Godel Letter to von Neumann 1956
- Tseitin, "On complexity of proof in prepositional calculus" 1968
- Cook, "The complexity of Theorem proving procedures" 1971
- Cook-Reckhow "The Relative efficiency of prop. pf systems" 1979

- Haken "Intractability of Resolution" 1986
- Urquhart "Hard examples for Resolution" 1987
- Chvatal, Szemeredi "Many hard examples.." 1988
- Ajtai "The complexity of the PHP" 1988

THE SURPRISING RISE & APPLICATIONS OF PROOF COMPLEXITY



PROPOSITIONAL PROOFS – DEFINITION

Define $\mathrm{TAUT} \subseteq \{0,1\}^*$ as

$$\mathrm{TAUT} := \{\text{encodings of all propositional tautologies}\}$$

under some "reasonable" encoding scheme.

<u>Defn</u> A propositional proof system is a polynomial-time algorithm V with 2 inputs: $x, p \in \{0,1\}^*$ such that:

$$\forall x \in \{0,1\}^* \quad x \in \mathrm{TAUT} \iff \exists p \in \{0,1\}^* \; V(x,p) \text{ accepts}$$

p is a "proof" that $x \in \mathrm{TAUT}$

$\Leftarrow$ direction is soundness

$\Rightarrow$ direction is completeness

PROPOSITIONAL PROOFS – DEFINITION

Define TAUT $\subseteq \{0,1\}^*$ as

$$\text{TAUT} := \{\text{encodings of all propositional tautologies}\}$$

<u>Defn</u> A propositional proof system is a polynomial-time algorithm V with 2 inputs: $x, p \in \{0,1\}^*$ such that:

$$\forall x \in \{0,1\}^* \quad x \in \text{TAUT} \iff \exists p \in \{0,1\}^* \; V(x,p) \text{ accepts}$$

Notes

1. We often assume TAUT is all DNF tautologies
2. We could also define a proof system as a refutation system for UNSAT (CNF) formulas; it is easy to go from one to the other

MAIN QUESTION IN PROPOSITIONAL PROOF COMPLEXITY

Q: Is there a propositional proof system V such that every propositional tautology has a short proof in V?

Defn A propositional proof system (pps) V is polynomially bounded if: $\forall x \in TAUT\ \exists p,\ |p| = poly(|x|)$ and $V(x, p)$ accepts

Cook and Reckhow proved:

There exists a polynomial-bounded proof system if and only if $NP = coNP$

MAIN QUESTIONS IN PROOF COMPLEXITY

given a particular proof system P:

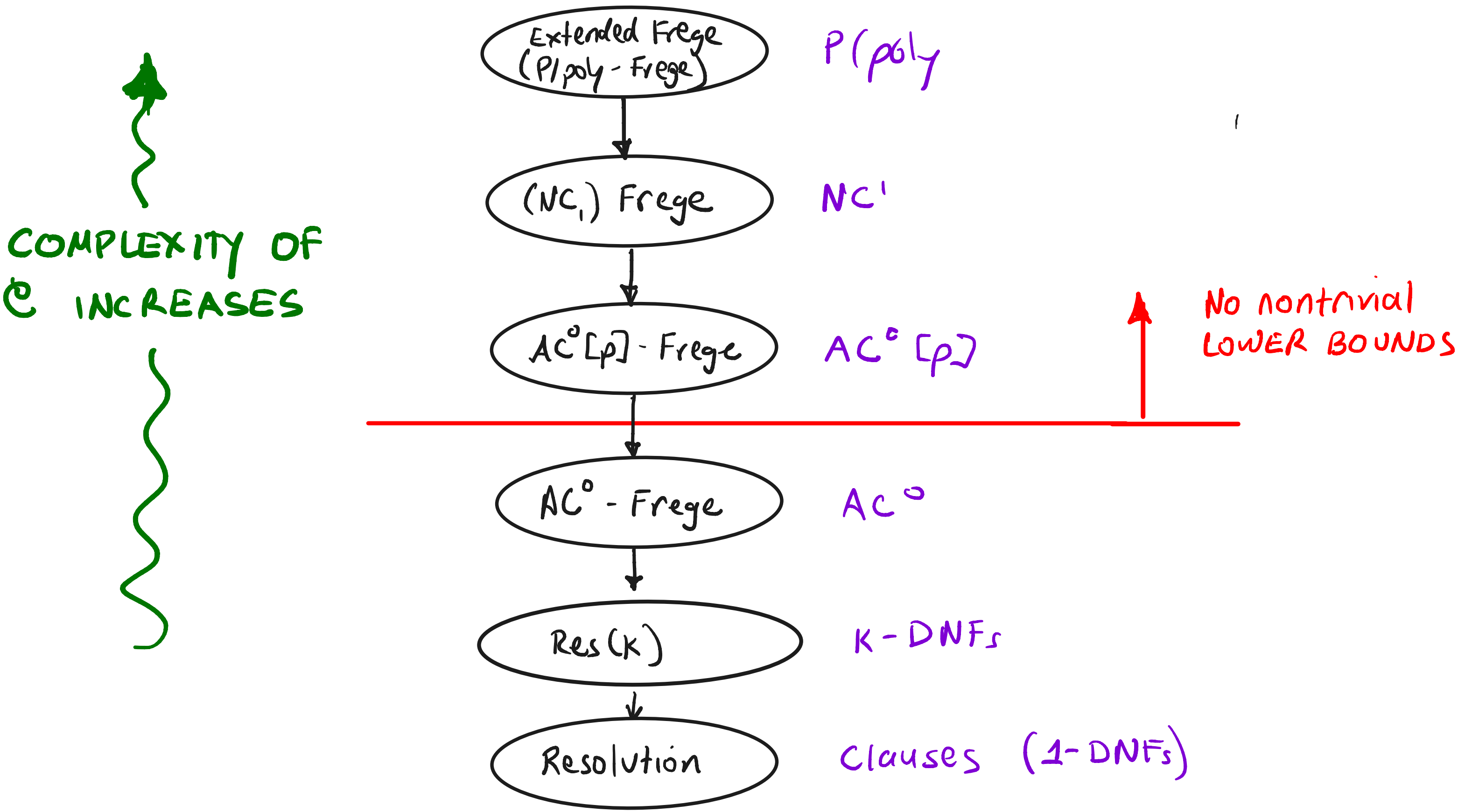
- Characterize which formulas have poly size refutations
  prove unconditional superpolynomial lower bounds
  even conditional lower bounds open

- Automatizability: how hard is it to find P-refutations

- Relate P to a natural class of algorithms $\mathcal{A}(P)$
  use lower bounds to prove limitations on exact + approximate
  $\mathcal{A}(P)$ algorithms for natural problems

- Compare proof strength of P to other proof systems

HIERARCHY OF $\mathcal{C}$-FREGE SYSTEMS



BOOLEAN (FREGE) PROOF SYSTEMS

Lines represent Boolean functions in some circuit class

Examples:

Proof System	Circuit class
Resolution	Clauses (depth-1 AC^0)
AC^0-Frege	AC^0
Frege	NC^1
Extended Frege	P/poly
Cutting Planes	Threshold formulas

Comparing Proof Systems

Proof System A p-simulates B if for all DNF tautologies f (CNF UNSAT formulas), for every y such that $B(y) = f$, $\exists y'$, $|y'| = \text{poly}|y|$ such that $A(y') = f$

A and B are p-equivalent iff A p-simulates B and B p-simulates A

POTENTIALLY HARD CNF FORMULAS?

① Pigeonhole Principle

$$PHP_n^{n+1} : \bigwedge_{i=1}^{n+1} (P_{i,1} \vee P_{i,2} \vee \dots \vee P_{i,n}) \wedge \bigwedge_{\substack{i_1, i_2 \leq n+1 \\ j \leq n}} (\overline{P_{i_1 j}} \vee \overline{P_{i_2 j}})$$

$n = 9$ holes

$n+1 = 10$ pigeons

② Tseitin mod p principle

② Random Formulas

③ Existence of pseudo-random generators / Circuit Lower Bounds

RESOLUTION

Refutation Proof system for UNSAT CNF formulas

One rule: Resolution Rule : $(A \vee x), (B \vee \bar{x}) \longrightarrow (A \vee B)$

A Resolution refutation of $f = C_1 \wedge \dots \wedge C_m$ is a sequence of clauses
(or a dag where every vertex of dag is labelled with a clause)
each clause derived from 2 previous clauses by resolution rule.
Last clause $= \phi$ (the empty clause)

RESOLUTION

Refutation Proof system for UNSAT CNF formulas

One rule: Resolution Rule : $(A \vee x), (B \vee \bar{x}) \longrightarrow (A \vee B)$

A Resolution refutation of $f = C_1 \wedge \cdots \wedge C_m$ is a sequence of clauses (or a dag where every vertex of dag is labelled with a clause) each clause derived from 2 previous clauses by resolution rule. Last clause = ϕ (the empty clause)

Example $f = x_1 \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge \bar{x}_3$

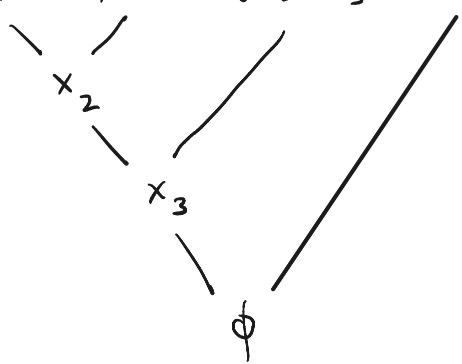
RESOLUTION

Refutation Proof system for UNSAT CNF formulas

One rule: Resolution Rule : $(A \vee x), (B \vee \bar{x}) \longrightarrow (A \vee B)$

A Resolution refutation of $f = C_1 \wedge \dots \wedge C_m$ is a sequence of clauses (or a dag where every vertex of dag is labelled with a clause) each clause derived from 2 previous clauses by resolution rule. Last clause $= \phi$ (the empty clause)

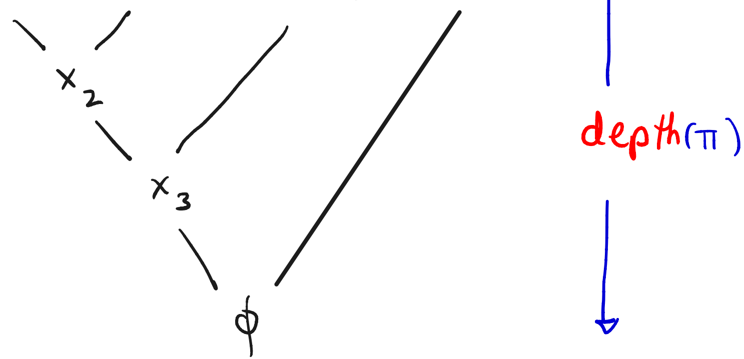
Example $f = x_1 \quad (\bar{x}_1 \vee x_2) \quad (\bar{x}_2 \vee x_3) \quad \bar{x}_3$



RESOLUTION

Example $f = x_1 \quad (\bar{x}_1 \vee x_2) \quad (\bar{x}_2 \vee x_3) \quad \bar{x}_3$

Π :



size(f) = total number of clauses in refutation

width(f) = $\max_{\text{clauses } C \in \Pi} (|C|)$ ← number of literals in C

Π is tree-like iff every derived clause is used once
(iff dag of Π is a tree, assuming clauses of f can be repeated)

Resolution soundness

Soundness: If there is a Res refutation of $f = C_1 \wedge \ldots \wedge C_m$ then f is unsatisfiable

Proof: ① Show the resolution rule is sound:

$$(A \vee x) \wedge (B \vee \bar{x}) \text{ satisfiable} \Rightarrow (A \vee x) \wedge (B \vee \bar{x}) \wedge (A \vee B) \text{ is satisfiable}$$

②
- Let π be a Res refutation of f.
- Assume for contradiction that f is satisfiable
- Then by ①, $\forall j \leq \text{size}(\pi)$, the conjunction of the 1st j clauses in π are satisfiable.
- Since last line of $\pi = \phi$ this is a contradiction

Proof Systems & Find-Falsified Clause Search Problem

Defn Let $f = C_1 \wedge \ldots \wedge C_m$ be UNSAT kCNF over $x_1, \ldots, x_n$

$\text{Search}_f : \{0,1\}^n \to [m]$ takes a truth assignment α as input

$\text{Search}_f(\alpha)$ should output some $i \in [m]$ such that $C_i(\alpha) = 0$

* since f UNSAT, Search_f is a total search problem

For many weak proof systems, we can associate a "query" model such that proofs in the proof system correspond to algorithms for solving Search_f in the query model.

Ex. 1 Tree Resolution $\approx$ Decision trees

we will use this to give a single proof of completeness for Resolution

Ex 2 Dag-Like Resolution $\approx$ PLS

(special type of Branching Program)

Resolution Completeness

Completeness: If $f = C_1 \wedge \dots \wedge C_m$ is unsatisfiable then there is a (tree-like) resolution refutation of f.

Proof

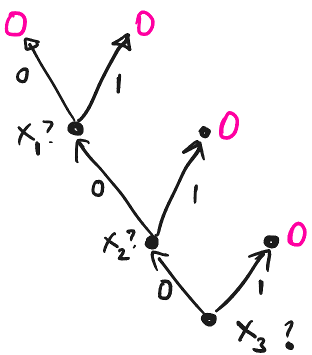
I. Make a decision tree that solves Search_f

II. Show that any decision tree for unsat f can be converted to a Res refutation of f.
(actually they are equivalent)

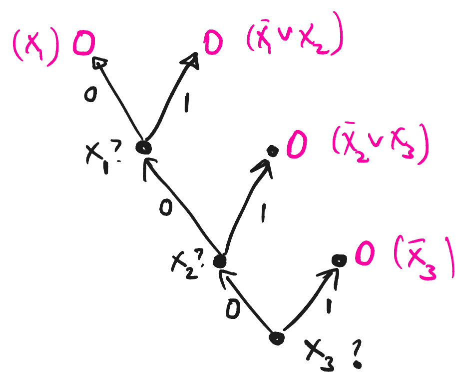
Resolution soundness & completeness

Completeness: If $f = C_1 \wedge \dots \wedge C_m$ is unsatisfiable then there is a (tree-like) resolution refutation of f.

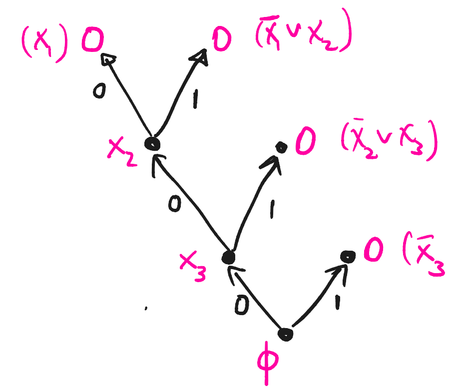
Example: $f = (x_1)\,(\bar{x}_1 \vee x_2)\,(\bar{x}_2 \vee x_3)\,(\bar{x}_3)$



decision tree for f



label leaves with a falsified clause



label intermediate vertices with a clause

Resolution soundness & Completeness

Completeness: If $f = C_1 \wedge \dots \wedge C_m$ is unsatisfiable then there is a (tree-like) resolution refutation of f.

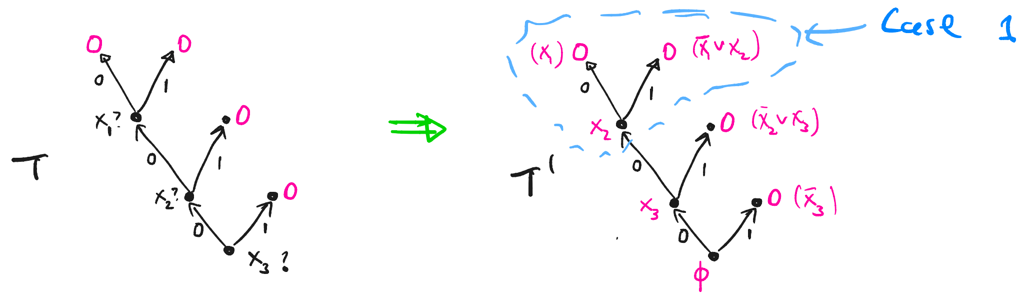
Lemma (Slightly Harder direction of Tree-Res $\approx$ DecTree complexity of Search_f)

A dec tree for Search_f is pruned if $\forall$ vertices v in the tree, if the path p_v from root to v falsifies a clause of f, then v is a leaf.

Let T be a pruned decision tree for Search_f. Then T can be converted into a (tree-like) RES refutation of f, T', of size $\leq$ size(T)

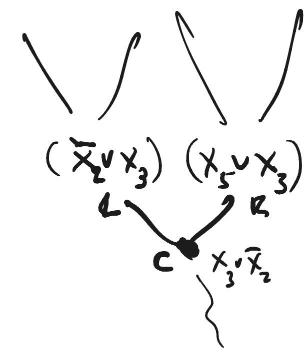
Resolution soundness & completeness

Proof of Lemma



Prove the relabelled vertices of T form a RES refutation of f.
Attempt to show the clauses (labelling vertices) can be derived from parent clauses by the Res rule.

<u>Case 1</u> The variable x queried in T occurs in both parents.
Then we can apply Res rule, resolving on x



<u>Case 2</u>: The variable x queried occurs in one parent, say the right parent, R
(Note x must occur in at least one parent since T is a pruned tree.
then we can remove entire derivation above C.

Ex. 2 $\boxed{\text{Prover - Delayer Definition of Dag-Like (general) Res}}$

Prover : claims f unsat ; Delayer : claims f is sat

Prover & delayer share a state $\rho \in \{0,1,*\}^n$. Initially $\rho = *^n$

Repeat :

1. Prover chooses a variable x_i that is currently unset
2. Delayer responds with a value $b \in \{0,1\}$. Update current $\rho = \rho \cup x_i = b$
3. Prover picks subset S of fixed-vars of ρ and updates ρ by setting all vars $x \in S$ to $*$ (prover "forgets" their values)

Abort whenever they reach a state ρ st. ρ falsifies some clause of f.

Size (of prover strategy) = # distinct states ρ required by prover across all delayer choices

width $= \max_{\text{states } \rho} (\text{\# fixed vars of } \rho)$

depth = max # rounds of strategy across all delayer choices

Ex 2 Prover-Delayer Definition of Resolution

Prover : claims f UNSAT ; Delayer : claims f is sat

Prover & delayer share a state $\rho \in \{0,1,*\}^n$. Initially $\rho = *^n$

Repeat :

1. Prover chooses a variable x_i that is currently unset
2. Delayer responds with a value $b \in \{0,1\}$. Update current $\rho = \rho \cup x_i = b$
3. Prover picks subset S of fixed-vars of ρ and updates ρ by setting all vars $x \in S$ to $*$ (prover "forgets" their values)

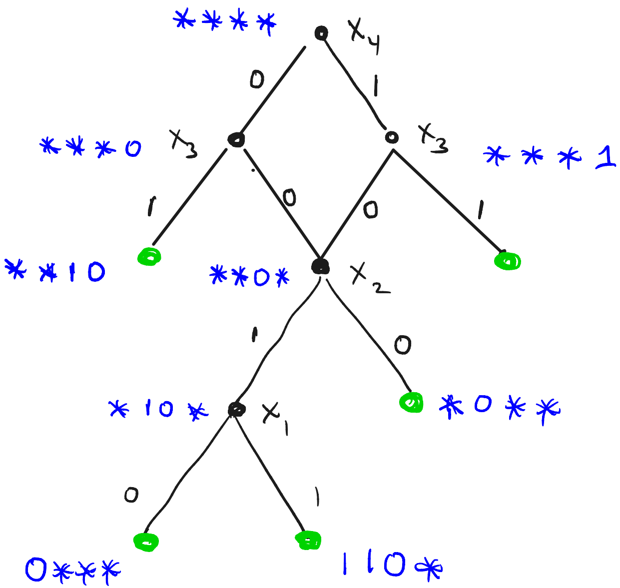
Abort whenever they reach a state ρ st. ρ falsifies some clause of f.

Theorem For any CNF F, F has a size s, depth d, width w Res refutation iff F has a size s, depth d, width w Prover-Delayer DAG.

(We'll see later Res pfs can also be characterized by BlackBox PLS)

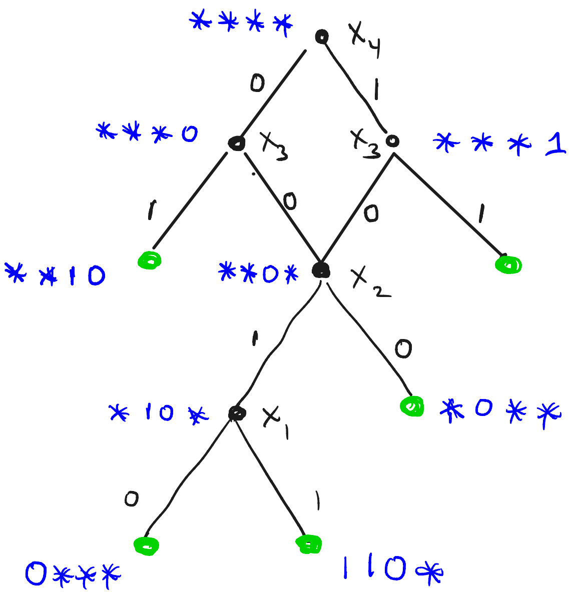
Ex 2 $\boxed{\text{Prover-Delayer Example}}$

$$f = x_1 \wedge x_2 \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_4)$$

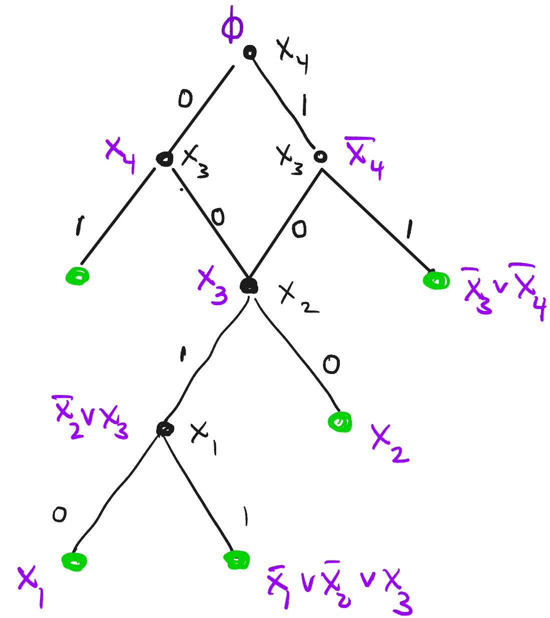


Ex 2 $\boxed{\text{Prover-Delayer Example}}$

$$f = x_1 \wedge x_2 \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_4)$$



Prover-Delayer game



Res Refutation

Prover-Delayer DAGs are Branching Programs for $Search_f$

$f = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$

Claim Prover-Delayer DAGs are Branching Programs for $Search_f$

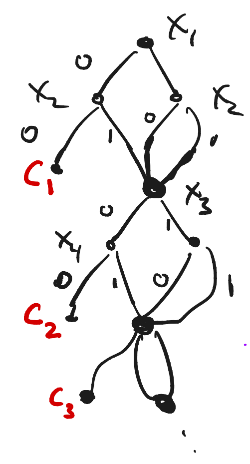
but importantly Not all Branching Programs solving $Search_f$ are Prover-Delayer DAGs

Example: For any UNSAT f, there is a small-size Branching Program solving $Search_f$:

Query all vars in Clause 1. If all false DONE

Else erase memory and Query all vars in clause 2.

⋮



Resolution Lower Bounds

Methods

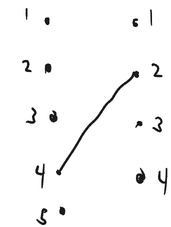
1. Width LBs $\rightarrow$ Size LBs via restriction argument
   or general size-width tradeoff

   Width LBs: via expansion of clause-variable graph of F

2. Feasible Interpolation

Res LBs for PHP

PHP^{n+1}_n : Variables: $P_{i,j}$ $i \in [n+1]$, $j \in [n]$



Clauses:

(1) $\forall i \in [n+1]$: $(P_{i1} \vee P_{i2} \vee \ldots \vee P_{in})$

one-to-one (2) $\forall i \neq i' \in [n+1]$, $j \in [n]$: $(\bar{P}_{ij} \vee \bar{P}_{i'j})$

UNSat

functional (3) $\forall i \in [n+1]$, $j \neq j' \in [n]$: $(\bar{P}_{ij} \vee \bar{P}_{ij'})$

onto (4) $\forall j \in [n]$: $(P_{1j} \vee P_{2j} \vee \ldots \vee P_{n+1,j})$

Standard version : Clauses of type (1) + (2)

Functional : (1), (2), (3)

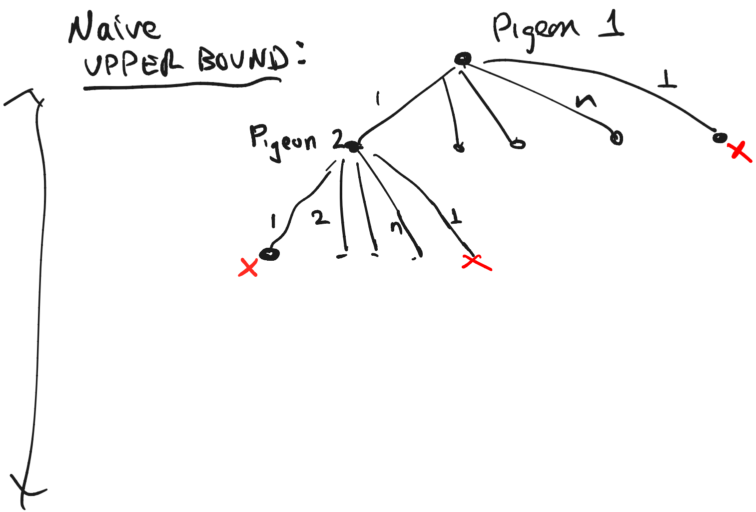
Functional + Onto : (1), (2), (3), (4)

Res Lower Bounds for PHP : Warmup Tree-Resolution

Show any decision tree for search_{PHP} requires size $2^{\Omega(n)}$

Q: Is this tight for tree-like Resolution?

Naive UPPER BOUND:



ht $O(n)$

fanout $O(n)$ so $n^n \sim 2^{n \lg n}$

Exercise:

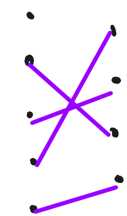
Show Res DAG (Prover Delayer) can solve search in size $2^{O(n)}$

Res Lower Bounds for PHP: Warmup Tree-Resolution

Show any decision tree for Search_{PHP} requires $2^{\Omega(n\lg n)}$ size

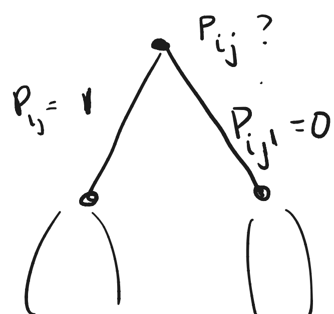
* actually $2^{\Omega(n^2)}$ which is tight for tree like

A truth assignment to $\{P_{ij}\}$ is a critical truth ass (cta) if it maps n pigeons bijectively to n holes, + remaining pigeon is unmapped



Lower Bound We will prove by induction on n: any decision tree for $\text{Search}_{PHP_n^{n+1}}$ that is correct on all critical truth ass's has size $\exp(\Omega(n))$:

By induction, 2 subproblems with n pigeons, $n-1$ holes.

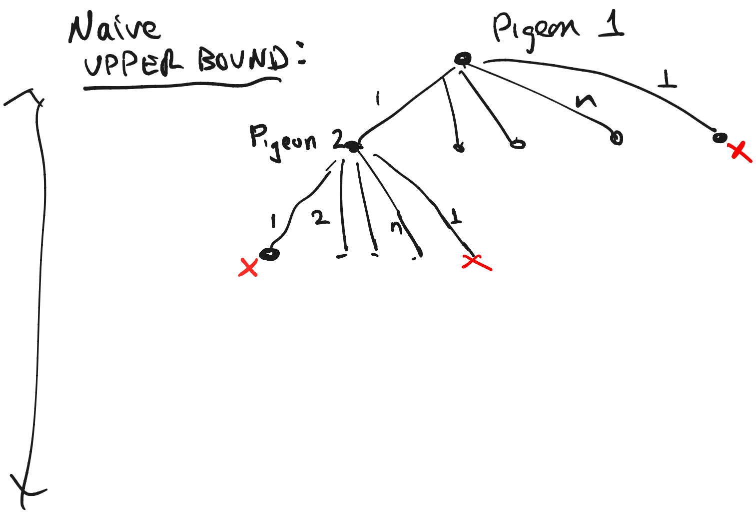


Res Lower Bounds for PHP : Warmup Tree-Resolution

Show any decision tree for search_{PHP} requires size $2^{\Omega(n)}$

Q: Is this tight for tree-like Resolution?

Naive
UPPER BOUND:



ht $O(n)$

fanout $O(n)$ so $n^n \sim 2^{n \lg n}$

Exercise:

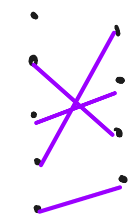
Show Res DAG (Prover Delayer) can solve search in size $2^{O(n)}$

Res Lower Bounds for PHP: Warmup Tree-Resolution

Show any decision tree for search_{PHP} requires $2^{\Omega(n \lg n)}$ size

* actually $2^{\Omega(n^2)}$ which is tight for tree like

A truth assignment to $\{P_{ij}\}$ is a critical truth ass (cta) if it maps n pigeons bijectively to n holes, + remaining pigeon is unmapped



<u>Lower Bound</u> Prove by induction on n: any decision tree for $\text{Search}_{PHP_n^{n+1}}$ that is correct on all critical truth ass's has size $\exp(\Omega(n))$:

By induction, 2 subproblems with n pigeons, $n-1$ holes.