PROPOSITIONAL PROOF COMPLEXITY

COMP 532

NICHOLAS HAYEK

Lectures by Prof. Robert Robere

CONTENTS

I Introduction

MOTIVATIONS

A *proof* may be...

1. A logical argument, deducing statements from assumptions by deduction rules.
2. A certificate of knowledge about something.
3. A counterexample.
4. Affirmative example.
5. An algorithm.
6. An infinite list of affirmative examples (if it can be verified... i.e. have a finite representation, i.e. countable).
7. Social consensus and trust?

A proof should always be verifiable and finitely represented.

♠ *Examples* ♣ E.G. 1.1

The following are theorems:

Eg 1: Finite Simple Groups classifies all finite simple groups into a small set of categories. The full proof of this took about 100 years, and is about 10,000 pages. Why do we trust it? Eh. It probably is OK. The monster group!

Eg 2: 4 Color Theorem states that all planar graphs are 4-colorable. Proof is computer-assisted. Lots of case analysis.

Eg 3: Boolean Pythagorean Triples asks whether it is possible to 2-color (i.e. choose a subset of) the natural numbers such that no pythagorean triple is monochromatic. This happens to be false. The proof was 200T long. $\{1, ..., 7824\}$ does have a coloring, but $\{1, ..., 7825\}$ does not.

We don't just want to *know* if something is true, but come to an *understanding* about why a claim is true.

1.1 Gödel Completeness

Every tautology in first order logic has a finite, deductive proof.

This says nothing, however, about the length of the proof. Quantitative bounds suggest something doubly exponential of the number of symbols in the tautology.

1.2 Gödel Incompleteness

There exists a first order statement φ_G in the language of arithmetic which is true, but for which no peano arithmetic proof exists.

The Main Question

Is there a propositional proof system P such that every tautology has a "short" proof in P? We need to define the underlined terms.

A *proposition proof system* P is an algorithm that takes two strings F and Π as input. Then, $\forall F$ encoding a boolean formula, F is a tautology $\iff \exists$ a proof Π such that $P(F, \Pi) = 1$. Furthermore, P is a polynomial-time, computable algorithm.

We think of P as the verification of the proof Π for F.

A proof system P is *polynomially bounded* if, for every tautology F, there is a proof Π of F of length $|\Pi| = |F|^n$.

The Main Question, Restated

Is there a polynomially bounded proof system?

1.3 Cook-Reckhow

The Main Question $\iff$ NP = coNP.

The Cook-Reckhow "Program" says: assume proof systems are not polynomially bounded, and build up a toolkit to attack NP vs coNP.

PROPOSITIONAL LOGIC DEFINITIONS

Some syntax:

DEF 1.1 We have *atoms*, 0 and 1, which mean false and true, as well as variables $a, b, ..., z$.

We have *connectives* $\neg$ (not) as well as $\vee$ (or) and $\wedge$ (and).

We have *formulas*. Any atom is a formula. If F is a formula, then so is $\neg F$. And if F, G are formulas, then $(F \wedge G), (F \vee G)$ are formulas.

Some semantics:

DEF 1.2 A *truth assignment* τ is a mapping $\{\text{atoms}\} \mapsto \{0, 1\}$ such that $\tau(0) = 0$ and $\tau(1) = 1$. Let F be a formula:

1. If $F = G \vee H$, then $\tau(F) = 1$ if $\tau(G) = 1$ or $\tau(H) = 1$, and 0 otherwise.
2. If $F = G \wedge H$, then $\tau(F) = 1$ if $\tau(G) = 1$ and $\tau(H) = 1$, and 0 otherwise.
3. If $F = \neg G$, then $\tau(F) = 1 \iff \tau(G) = 0$.

Syntactic equality is denoted by $=_{\text{syn}}$. For example, $(F \vee G) \vee H \neq_{\text{syn}} F \vee (G \vee H)$, even though they are *semantically* the same.

We say that τ *satisfies* F if $\tau(F) = 1$, and *falsifies* it otherwise. We say that F is *satisfiable* if there is a satisfying assignment τ. We say that F is a *tautology* if every assignment satisfies it. DEF 1.3

The *size* of a formula F is the number of atoms and connectives in it, and denoted $|F|$. DEF 1.4

A *literal* is either a variable or a negation of an variable. DEF 1.5

A *clause* is a disjunction of literals *without* repeated literals. For example, $x \vee \neg y$ and $\neg x$. $\perp$ denotes the *empty clause* (False). DEF 1.6

The *width* of a clause is the number of literals in it. DEF 1.7

A disjunction is an "or" of literals. A conjunction is an "and" of literals. DEF 1.8

A formula F is in *CNF* (conjunctive normal form) if it is a conjunction of clauses. DEF 1.9

A formula F is in *DNF* (disjunctive normal form) if it is an or of ands of literals (i.e. a disjunction of conjunctions of literals). DEF 1.10

If a formula F depends on variables $x_1, ..., x_n$, we write $F(x_1, ..., x_n)$. DEF 1.11

If F and G are formulas, we write $F \vDash G$ if, for every truth assignment τ, $\tau(F) = 1 \implies \tau(G) = 1$. DEF 1.12

If $F(x_1, ..., x_n)$ is a formula, the *truth function* is the function $F : \{0,1\}^n \to \{0,1\}$ such that $F(\vec{x}) = 1 \iff$ the truth assignment τ corresponding to $\vec{x}$ has $\tau(F) = 1$. DEF 1.13

A *propositional proof system* is a polynomial-time algorithm P which takes F and Π as inputs, such that F encodes a tautology $\iff \exists$ a proof Π such that $P(F, \Pi) = 1$ (i.e. accepts). DEF 1.14

Equivalently, we could have said "$\neg F$ a contradiction" instead of "F a tautology."

II Resolution

Let $C \vee x$ and $D \vee \neg x$ be clauses which are both true. Let C, D be clauses. Then either C is true or D is true. In better notation:

$$\frac{C \vee x \Downarrow 1 \qquad D \vee \neg x \Downarrow 1}{C \vee D \Downarrow 1} \qquad \text{(we omit } \Downarrow 1\text{)}$$

DEF 2.1 Let $F(x_1, ..., x_n)$ be a CNF formula and let C be a clause. A *resolution proof* of C from F is a sequence of clauses $D_1, ..., D_m = C$. Each clause D_i is either a clause from F or is deduced from earlier clauses in the sequence by one of two rules:

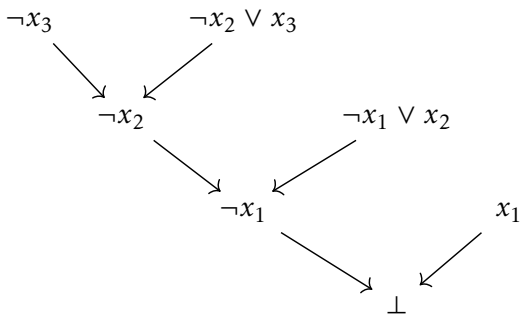
1. $$\frac{C \vee x \qquad D \vee \neg x}{C \vee D} \text{ (resolution)}$$
2. $$\frac{C}{C \vee D} \text{ (weakening)}$$

The proof is called a *refutation* of F if $C = \bot$.

E.G. 2.1 ♠ *Examples* ♣

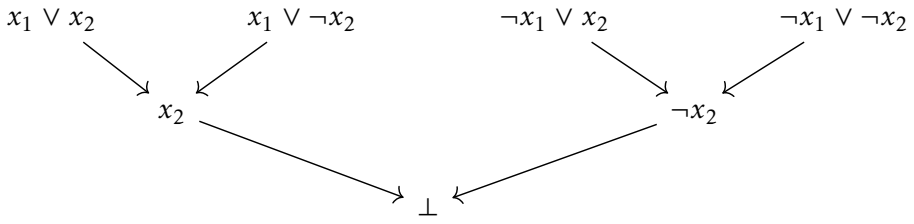
Eg 1: Let $F = x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$. Notice that this is not satisfiable. If F is true, we deduce the following

1. $\neg x_3 (\vee \bot)$
2. $\neg x_2 \vee x_3$
3. $\neg x_2$ (by resolution on 1 and 2)
4. $\neg x_1 \vee x_2$
5. $\neg x_1$ (by resolution on 3 and 4)
6. x_1
7. $\bot$ (by resolution, we can see $\frac{x_1 \quad \neg x_1}{\bot}$)



Eg 2: Let $\text{IND}_n = F_n(x_1, ..., x_n) = x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge ... \wedge (\neg x_{n-1} \vee x_n) \wedge \neg x_n$. The proof (refutation) of this would have size $2n = O(n)$, depth $n = O(n)$, and width 2, using resolution as above.

Eg 3: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2) := \text{CT}_2$



Let Π be a resolution refutation (as in Example 1.2). Then $S(\Pi)$, the *size*, denotes the number of clauses in Π. $w(\Pi)$, the *width*, denotes the maximum width of a clause in Π. Lastly, $d(\Pi)$, the *depth*, denotes the length of the longest path in the graph associated with Π from a clause of F to the final clause. DEF 2.2

For Π, S_{Res}, w_{Res}, d_{Res} denote the minimum of these parameters over *all* resolution refutations. DEF 2.3

Resolution is a propositional proof system. PROP 2.1

PROOF. Let F be a CNF formula and let $\Pi = (C_1, ..., C_m)$ be a (possible) resolution refutation of F. $P(F, \Pi)$ checks, for each $i = 1, ..., m$, that $C_i \in F$ or is deduced from earlier clauses by resolution/weakening. If not, output 0. If $C_m = \bot$, then output 1. Otherwise, output 0. We need to check the following:

1. (Completeness) For all unsatisfiable F, there is Π such that $P(F, \Pi) = 1$.
2. (Soundness) If there is a Π such that $P(F, \Pi) = 1$, then F is unsatisfiable.
3. P is polynomial-time.

We tackle (2) first. In shorthand, $F \vdash_{\text{Res}} \bot \implies F \vDash \bot$. Some case analysis:

1. $\bot \in F$. Then we are done, since F is a CNF.
2. $\bot \notin F$. We prove that the resolution rules are sound (i.e. preserve truth assignments), and the result follows by induction. Suppose τ satisfies $C \vee x$ and $D \vee \neg x$. Then τ must set one of x or $\neg x$ to 0, so τ must satisfy C or D, respectively.

□

For $n \geq 1$, CT_n, the *complete tautology*, denotes the unsatisfiable CNF formula DEF 2.4

on variables $\{x_1, ..., x_n\}$ with all possible width-n clauses on these variables with distinct literals.

PROP 2.2 Resolution is (refutationally) complete. In other words, if F is an unsatisfiable CNF, then there is a resolution refutation of F. The refutation Π has $S(\Pi) = O(2^n)$, $d(\Pi) = n + 1$, $w(\Pi) = n$, where n is the number of variables in F.

PROOF. We'll do this naively by trying all truth assignments, and vertifying that F always evaluates to 0. However, we need to write this in the language of resolution proofs.

CT_n has 2^n clauses, and its refutation has size $O(2^n)$, depth n, and width n.

To show this last, we proceed by induction. For $n = 1$, we have $x_1 \wedge \neg x_1 \implies \bot$. Let CT_{n-1} have a refutation Π. Let Π_1 be obtained by adding x_n to all clauses in Π. Let Π_0 be obtained by adding $\neg x_n$ to all clauses in Π. Then Π_1 is a proof of x_n from $\mathrm{CT}_{n-1} \cup \{x_n\}$ (an abuse of notation, but forgive it). Similarly, Π_0 is a proof of $\neg x_n$. Then $\mathrm{CT}_n = (\mathrm{CT}_{n-1} \cup \{x_n\}) \cup (\mathrm{CT}_{n-1} \cup \{\neg x_n\})$, and we resolve $x_n, \neg x_n \implies \bot$. The size, depth, and widths follow easily by induction.

Returning to the original question, let $F = C_1 \wedge ... \wedge C_m$ be a conjunction of clauses. For every clause C of CT_n, there is a clause C' in F such that $C' \subseteq C$. Suppose toward contradiction. Let $C \in \mathrm{CT}_n$ be such a clause. Then F is satisfiable, since the assignment which falsifies C is going to satisfy all clauses of F. For further verification, note that every clause C' in F is not contained in C, so C' must have a literal L occurring with the opposite sign as in C.

Now, to refute F, we derive from F each clause of CT_n using one weakening step. This increases the length of the CT_n refutation by 2^n and increases the depth by 1. □

DEF 2.5 A resolution proof is said to be *tree-like* or *dag-like* if its graph is a tree or dag, respectively. As before, S_{TreeRes}, w_{TreeRes}, d_{TreeRes} of Π denote the minimal S, w, d over all resolution refutations of Π.

A dag is a directed graph with no directed cycles.

PROP 2.3 $S_{\mathrm{TreeRes}} \gneq S_{\mathrm{Res}}$, $w_{\mathrm{TreeRes}} = w_{\mathrm{Res}}$, and $d_{\mathrm{TreeRes}} = d_{\mathrm{Res}}$.

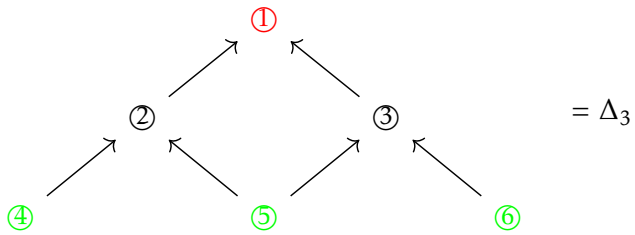
PROOF. Every tree-like proof of Π is also a resolution proof, so automatically $S_{\mathrm{TreeRes}} \geq S_{\mathrm{Res}}$, $w_{\mathrm{TreeRes}} \geq w_{\mathrm{Res}}$, and $d_{\mathrm{TreeRes}} \geq d_{\mathrm{Res}}$. We can turn a resolution proof into a tree-like proof by re-computing all necessary clauses when they need to be reused. This tree-like proof will have the same width, so $w_{\mathrm{TreeRes}} \leq w_{\mathrm{Res}} \implies w_{\mathrm{TreeRes}} = w_{\mathrm{Res}}$. The depth will also remain constant (when a clause that has already been computed is needed at level d, its computation will take $\leq d - 1$ levels; hence, we can compute it in parallel without increasing depth). □

What is easy and what is hard for (tree) resolution, where "easy" means there exists a polynomial-size refutation $n^{O(1)}$, where $n = S(\Pi)$? We saw previously $x_1 \wedge (\neg x_1 \vee x_2) \wedge ... \wedge (\neg x_{n-1} \vee x_n) \wedge \neg x_n$, which has a proof of size $O(n)$, width 2, and depth $O(n)$ (though one can get this down to $O(\log(n))$).

Let Δ_h denote the pyramid graph of height h. PEB_{Δ_h} is a CNF with variables x_u for every $u \in V(\Delta_h)$ and its clauses are as follows: $\neg x_u$ if u is the root and DEF 2.6

$$x_u \vee \bigvee_{v \in \delta^-(u)} \neg x_v \quad \forall u \in V(\Delta_h)$$

otherwise.



$= \Delta_3$

PEB_{Δ_h} may be refuted by resolution. PROP 2.4

PROOF. We prove by induction on the level i that all notes are true. For $i = 1$ it is given. Let u be any node in level i with predecessors v, w. Then x_v and x_w are true, by induction, and hence resolve

$$\frac{x_u \vee \neg x_v \vee \neg x_w \quad x_v}{x_u \vee \neg x_w} \rightarrow \frac{x_u \vee \neg x_w \quad x_w}{x_u}$$

Hence, for $i = h$, we find that $x_h, \neg x_h \implies \bot$. By examining the proof, each literal x_u may be derived using at most 2 inference rules, plus one more for the $\bot$ contradiction. Hence, the length is $\leq 2N + 1 = O(N)$, where $N = \frac{h(h+1)}{2}$, the number of total nodes. The depth of this proof is $O(h)$. The width of the widest clause is 3, so the width is 3.

The proof above is *not* tree-like, since we have repeated use of nodes. However, there *is* a tree-like proof for PEB of length $O(N)$, depth $O(N)$, and width $O(h)$. □

Any resolution refutation Π of PEB_{Δ_h} requires $d(\Pi)w(\Pi) \geq n$. PROP 2.5

PROOF. Out of the scope of this course. □

Let G be a tree-like resolution with a unique sink node r. The formula PEB_G is defined with variables x_u as before, with clauses $\neg x_r$ for r and DEF 2.7

$$x_u \vee \bigvee_{v \in \delta^-(u)} \neg x_v \quad \forall u \in V(G)$$

otherwise.

PIGEONHOLE PRINCIPLE

Recall the pigeonhole principle: if $n+1$ pigeons must fit inside n holes, at least one hole must contain at least 2 pigeons.

For $m > n$, let PHP_n^m be the following formula: we use variables x_{ij} for $i = 1, ..., m$ and $j = 1, ..., n$, the truth of each meaning "pigeon i goes to hole j."

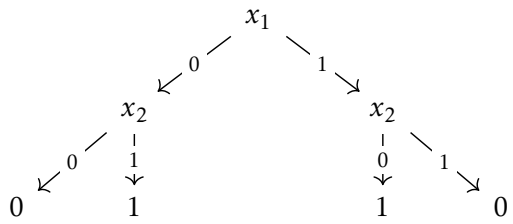
We assert that, for fixed $i = 1, ..., m \geq n+1$,

$$\bigvee_{j=1}^{n} x_{ij} \qquad \text{i.e. each pigeon has a hole}$$

We also assert that $\neg x_{ij} \vee \neg x_{kj} \forall k \neq i \forall j$, i.e. each pigeonhole contains at most one pigeon. It has been shown that any refutation of PHP_n^{n+1} requires size $2^{\Omega(n)}$.

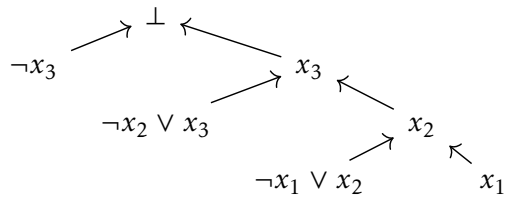
DEF 2.8 Let $f : \{0,1\}^n \to S$ be a function. A *decision tree* T computing f is a rooted, labeled, binary tree in which every leaf l is labeled with an element of S, every internal node is labeled with an input x_i, and the two outgoing edges from each node are labeled with 0 and 1.

For example, we have the following for x_1 XOR x_2, denoted $x_1 \oplus x_2$.



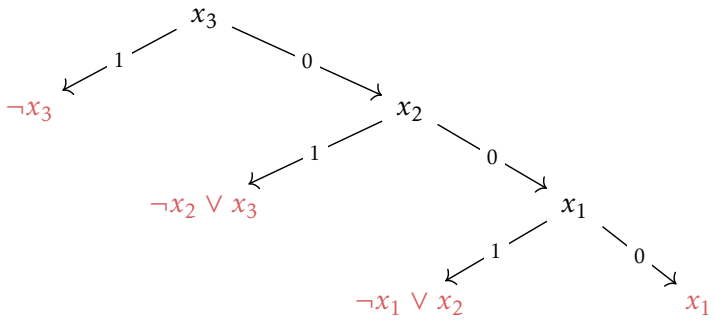
T computes f if, for each input $x \in \{0,1\}^n$, the leaf of T reached by the path consistent with x is labeled $f(x)$.

Recall $x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$ and consider its resolution tree:



We can generate an associated decision tree based on the variable being resolved

in each step:



Each leaf gives us the falsified clause given a truth assignment!

If $F = C_1 \wedge ... \wedge C_m$ is an unsatisfiable CNF on variables $x_1, ..., x_n$, define DEF 2.9

$$S(F) \subseteq \{0,1\}^n \times [m]$$

by $(\vec{x}, i) \in S(F) \iff C_i(\vec{x}) = 0$. Since F is unsatisfiable, $\forall \vec{x}, \exists i : (x, i) \in S(F)$. A property sometimes called "totality"

2.1 Tree-Like Resolution by Decision Trees

Any size s, depth d treelike resolution refutation for F implies a size $\leq s$, depth $\leq d$ decision tree solving $S(F)$ (and vice-versa).

A decision tree algorithm "solving" $S(F)$ does the following: given an input $\vec{x} \subseteq \{0,1\}^n$ for F on n variables, output a clause in F that is falsified by $\vec{x}$.

PROOF. Let $F(x_1, ..., x_n) = C_1 \wedge ... \wedge C_m$ be an unsatisfiable CNF, and let Π be a treelike resolution refutation. Let $s = S(\Pi)$ and $d = d(\Pi)$. Given this proof, we'd like a solution to the problem $S(F)$, i.e. given a truth assignment $\vec{x}$, which clause C_i is falsified by it? The following algorithm solves this problem:

Require: $x \in \{0,1\}^n, \rho = \emptyset, \Pi$
Ensure: ρ, a truth assignment and u, a clause in F falsified by ρ

```
u ← ⊥ ∈ Π                                   ▷ We expect the invariant ρ(u) = 0
while u not a leaf do
    if u derived from weakening then
        u ← child of u
    else if u derived from resolution on x_i then
        u ← child falsified by ρ
        b ← query x_i
        ρ ← ρ ∪ {x_i = b}
    else
        Something went wrong!
output ρ
```

The depth of the tree is the maximum number of queries it needs to make on any input. We make $\leq d$ queries on any input, since we make one query for each resolution along the path.

Each state of the algorithm corresponds to a unique clause in the proof. Hence, the size of the tree is at most the size of the proof.

Let T be a decision tree solving $S(F)$. Let ℓ be a path in the tree to a node π.

If π is a node, we abuse notation and use π to denote the unique assignment along the path from the root to π. If π' is a leaf, there is a clause of F which is falsified by ℓ. (Generated by the algorithm above).

If ρ is a partial truth assignment, then denote by C_ρ is the maximal clause falsified by ρ (by width). For example, if $\rho = \{x_1 = 1, x_3 = 0, x_7 = 1\}$, then $C_\rho = \neg x_1 \vee x_3 \vee \neg x_7$.

For every node π in the decision tree, we'll show how to derive C_π from F by induction.

Base Case: π is a leaf. To derive C_π, we can weaken some clause in F (specially, any clause falsified by π, generated by the algorithm).

Induction: If π is an internal node querying x with children π_0, π_1, $C_{\pi_0} = C_\pi \vee x$ and $C_{\pi_1} = C_\pi \vee \neg x$. These can be resolved to get C_π. □

2.2 PHP_n^{n+1} Tree-Like Proof

There is a tree-like resolution proof of PHP_n^{n+1} of size $2^{O(n \log(n))}$.

PROOF.

We give a decision tree solving $S(\text{PHP}_n^{n+1})$ given an assignment of pigeons to holes. We would like to find either (a) a hole with two pigeons, or (b) a pigeon not mapped to a hole.

Require: $x_{ij} \in \{0,1\}\ \forall i = 1, ..., n+1; j = 1, ..., n$
  for $i = 1, ..., n+1$ **do**
    for $j = 1, ..., n$ **do**
      QUERY(x_{ij})
      if $x_{ij} = 1$ **then**
        Break
    if pigeon i not mapped **then**
      output $\bigvee_{j=1}^{n} x_{ij}$
    else if pigeon i collides with pigeon $k < i$ in hole j **then**
      output $\neg x_{ij} \vee \neg x_{kj}$

□

The depth of this tree, generated by the algorithm, is $O(n^2)$. To prove its size, we have $S(T_{k+1}) \leq nS(T_k)+k+1$ and $S(T_2) = O(1)$, where T_{k+1} is the tree resolution for PHP_k^{n+1}. Then, $S(T_n) = 2^{O(n\log(n))}$. It has been shown that $S_{\text{Res}}(\text{PHP}_n^{n+1}) = 2^{\Omega(n)}$.

DEF 2.10

Let $x \in \{0,1\}^k$, $\langle x \rangle = \sum_{i=1}^{k} 2^{k-i} x_i$.The *bit pigeonhole principle* BPHP_n (for simplicity, $n = 2^k$) is defined on variables $\vec{x_1}, ..., \vec{x_{n+1}}$, where $\vec{x_i} = x_{i1} \cdots x_{ik}$. In this context, $\vec{x_i}$ encodes the i^{th} pigeon that is held in hole $\langle \vec{x_i} \rangle$. We want to assert the following:

$$\forall i \neq j \in \{1, ..., n+1\}, \ell \in \{0, ..., 2^k - 1\}, [\langle \vec{x_i} \rangle \neq \ell] \vee [\langle \vec{x_j} \rangle \neq \ell]$$

i.e. we assert that any given hole contains at most 1 pigeon. As a logical formula,

$$\bigwedge_{\substack{i \neq j \in [n+1] \\ \ell \in [0, n-1]}} [\vec{x_i} \neq \ell] \vee [\vec{x_j} \neq \ell] \quad = \quad \text{BPHP}_n$$

For example, if $x = 1010$, then $\langle x \rangle = 10$. In this case, to assert $\langle x \rangle = 10$, we have

$$= \iff x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4$$

$$\neq \iff \neg x_1 \vee x_2 \vee \neg x_3 \vee x_4$$

2.3 Tree-Like Depth of BPHP_n

Let $n = 2^k$. A tree-like resolution refutation of BPHP_n requires depth $\Omega(n)$.

PROOF.

Let T be a decision tree solving $S(\text{BPHP}_n)$. We show that T must make $\geq n$ queries on some input $x \in \{0,1\}^{(n+1)k}$. Each query in T is to a variable x_{ij} for $i \in [n+1], j \in [k]$. The *idea* is to move pigeons in as efficiently as possible, only running into a problem at the very end.

Let ρ be a partial assignment to the variables of BPHP_n. Initially, $\rho = \varnothing$. Let $P(\rho) = \{i \in [n+1] : \exists j : x_{ij} \in \text{dom}(\rho)\}$, where $\text{dom}(\rho)$ denotes the set of pigeons fixed by ρ (the domain of M).

We want to maintain the following invariant on the algorithm below: there is always some matching M of $P(\rho)$ to holes consistent with ρ. Before execution, the invariant is true, as $\rho = \varnothing$ has a matching $M = \varnothing$ that is consistent.

In an algorithm, consider the next variable x_{ij} queried by T. Before it is queried, we have a partial assignment ρ and matching M which is consistent.

Case 1: x_{ik} was already queried for some $k \neq j$. Hence $\{x_{ik} = b\} \subseteq \rho$. In this case, pigeon i is already mapped to some hole $h \in \{0,1\}^k$. We respond with $h_j \in \{0,1\}$ that is consistent. Hence, $\rho = \rho \cup \{x_{ij} = h_j\}$, and $M = M$ is maintained. Our invariant is maintained.

Case 2: x_{ij} is the first bit of $\vec{x_i}$ to be queried. As long as $P(\rho) < n$, there is an available hole h that does not contain any pigeon in M. Hence, $M = M \cup \{(i, h)\}$ and $\rho = \rho \cup \{x_{ij} = h_j\}$. Again, our invariant is maintained.

We can maintain this algorithm at least n times, while $|M| \leq n$ □

Note that the depth bound $d_{\text{Res}}(\text{PHP}_n^{n+1}) \geq n$ is pretty trivial, since each clause is $\bigvee_{j=1}^{n} x_{ij}$, which has n variables.

We just proved that a tree-like proof of BPHP_n has a ($\geq n$) long path. Can we extend this idea to a *size* lower bound?

A matching from $[n+1] \to [n]$ is encoded by $M : [n+1] \to [n] \cup \{\star\}$, where $\star$ denotes a partial function. Then $\text{dom}(M) = \{i : M(i) \neq \star\} = M^{-1}([n])$.

We say that x_{ij} is *forced* by M if $M(i) \neq \star$, *or* $M(i) = *$ and all holes consistent with $x_{ij} = b$ are filled by other pigeons in M. We'll arrange such that the second condition never happens.

2.4 Tree-Like Size of BPHP_n

Let $n = 2^k$. Any tree-like resolution refutation of BPHP_n has size at least

$$\left(\frac{3}{2}\right)^{\frac{n}{4}} = 2^{\Omega(n)}$$

This proof is sometimes called the "bottleneck counting method."

PROOF. We sample a random node at depth $\Omega(n)$, and prove that the probability of sampling any particular node is small (say $\frac{1}{c}$). Since we always output some node, but the probability of sampling that node is small, there must be many nodes (say c).

Require: $M : M(i) = \star \; \forall i$

Require: $\rho = \varnothing$

Require: v = root of the decision tree T for $S(\text{BPHP}_n)$

while $|\text{dom}(M)| < \frac{n}{4}$ **do**

  $x_{ij} \leftarrow$ variable queried by v

  if i is forced by M **then**

    $x_{ij} \leftarrow$ the forced value b

    $\rho \leftarrow \rho \cup \{x_{ij} = b\}$

  else

    pick a random avalible hole h

    $M(i) \leftarrow h$

    $\rho \leftarrow \rho \cup \{x_j = h_j\}$

    update v

output v

Let V be the set of all nodes than can be outputted by the algorithm. We'll

show that the probability $\mathbb{P}(v) \leq \left(\frac{3}{2}\right)^{\frac{-n}{4}}$ for $v \in V$. The probability $\mathbb{P}(\exists)$ that there *is* a node $v \in V$ is 1. Hence

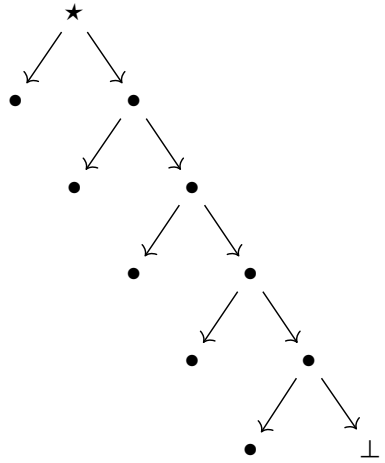
$$\mathbb{P}(\exists) \leq |V|\mathbb{P}(v) = |V|\left(\frac{3}{2}\right)^{\frac{-n}{4}} \implies |V| \geq \left(\frac{3}{2}\right)^{\frac{n}{4}}$$

To show this, let $v \in V$, and consider the unique path from the root to v in the decision tree. There are $\leq \frac{n}{4}$ random assignments of pigeons to holes along this path, and all other variables are forced.

Let $x_{i_1 j_1}, x_{i_2 j_2}, ..., x_{i_l, j_l}$ be these unforced variables, and let $b_1, ..., b_l$ be the bits assigned to these.

Each particular matching occurs with probability $\frac{1}{n(n-1)\cdots(n-l+1)}$, but many of these matchings are consistent with the bit pattern. □

A *deterministic adversary* is an algorithm that constructs a "bad input" for a tree T, which helps prove depth bounds by generating a long path. (See the following example): DEF 2.11



However, for well-filled trees, a *randomized adversary* allows us to prove size bounds (e.g. with probability p we select node n of depth at least $\geq d$).

$S_{\text{TreeRes}}(F) \leq O(2^{d_{\text{Res}}(F)})$ PROP 2.6

Binary trees of depth d have $O(2^d)$ nodes. □ PROOF.

$S_{\text{Res}}(F) = O(n^{w_{\text{Res}}+1})$ PROP 2.7

There are $\binom{n}{w}2^w$ ways of constructing a width w clause. Then, if $w = w_{\text{Res}}(F)$, PROOF.

there are at most

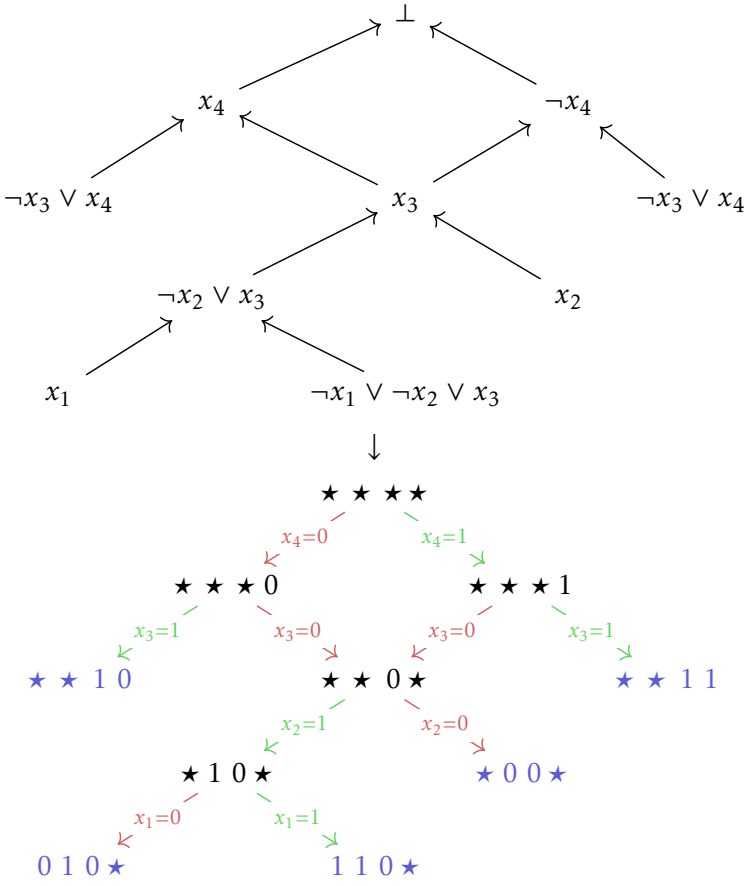
$$\sum_{i=0}^{w} 2^i \binom{n}{i} \leq \sum_{i=0}^{w} 2^i n^i = O((2n)^w) = O(n^{w+1})$$

clauses in a proof of F. □

DEF 2.12 A *Prover-Adversary Game* is a "decision tree with forgetting." In algorithmic-speak, we have

Require: $C = C_1 \wedge \cdots \wedge C_m$, $\rho : \{0,1\}^n \to \{0,1,\star\}$
  while ρ doesn't falsify F **do**
    prover picks a variable x_i s.t. $\rho(x_i) = \star$
    adversary responds with $b \in \{0,1\}$
    $\rho \leftarrow \rho \cup \{x_i = b\}$
    prover chooses $S \subseteq \rho^{-1}(\{0,1\})$
    for all $x_i \in S$ **do**
      $\rho(x_i) \leftarrow \star$

DEF 2.13 The *width* of a prover strategy is the maximum number of bits that the prover needs to remember before the game ends. (Against any delayer).



2.5 Prover Strategy Completeness

If there is a resolution refutation of F of size s, depth d, and width w, then there is a prover strategy that wins against any adversary with the (at most) the same parameters.

Conversely, if there is a prover strategy with parameters s, d, w, then there is a resolution proof of F with (at most) twice the same parameters.

PROOF.

($\Longrightarrow$) We simulate the decision tree argument fromThm 2.1, but forget any variable that is not needed to falsify a clause.

($\Longleftarrow$) We work bottom-up from a prover strategy by induction. Let C_p be the widest clause falsified by $\rho : \vec{x} \to \{0, 1, \star\}$. If ρ is a state in a prover strategy, we can derive C_ρ from F in resolution. Hence, $\rho = \varnothing$, for which $C_\rho = \bot$, may be derived from F in resolution.

Base case: a game-terminating assignment ρ falsifies a clause C in F. But we forget all variables not required to falsify C, so $C = C_\rho$.

Let ρ have predecessors σ_1, σ_2. Assume C_{σ_1} and C_{σ_2} can be derived in resolution. Then, ρ generates its predecessors by querying a variable x and forgetting a set S of assignments. WLOG let $x \in C_{\sigma_1}, \neg x \in C_{\sigma_2}$. Then, we resolve C_{σ_1} and C_{σ_2} to yield $C \subseteq C_\rho$. We then obtain C_ρ by weakening. □

2.6 Width of BPHP$_n$

$$w_{\text{Res}}(\text{BPHP}_n) \geq n$$

PROOF.

This adds to the theorems Thm 2.3 and Thm 2.4 in showing lower bounds on proofs of BPHP$_n$, except now we allow for dag-like (i.e. forgetting) proofs.

Fix a prover strategy P with width $d \leq n - 1$. We desire a delayer strategy which will force the game to never end. Recall that, in the bit pigeonhole principle, the game ends when it witnesses a collision of pigeons, i.e.

$$\bigwedge_{\substack{i \neq j \in [n+1] \\ \ell \in [0, n-1]}} [\vec{x_i} \neq \ell] \vee [\vec{x_j} \neq \ell] \quad = \quad \text{BPHP}_n$$

Let $P(\rho)$ denote the number of distinct pigeons in the fixed bits of a truth assignment ρ. We wish to maintain ρ such that if $P(\rho) \leq d \leq n - 1$, then there exists ρ^* extending ρ such that ρ^* encodes a valid matching of d pigeons to distinct holes. This is true initially, where $\rho = \varnothing$.

Require: ρ which satisfies invariant.
Ensure: A response to a delayer ρ which satisfies invariant.
  prover picks $x_{ij} \in \rho^{-1}(\star)$
  if i not assigned by ρ^* **then**
    Select a hole to assign pigeon i to that does not contain a pigeon in ρ^*.
    $\rho \leftarrow \rho \cup j^{th}$ bit of this hole.
  else
    Output consistent with ρ^*'s assignment, i.e.
    $\rho \leftarrow \rho \cup \rho^*(x_{ij})$

Hence, the prover strategy must have width $\geq n$. □

DEF 2.14 $\widetilde{\text{BPHP}}_n := \text{BPHP}_n \circ \text{XOR}(x, y)$, where

$$\text{XOR}(x, y) = (\overline{x} \vee \overline{y}) \wedge (x \vee y) = (\overline{x} \wedge y) \vee (x \wedge \overline{y})$$

For example, if $F = z_1 \vee \overline{z_2}$, then

$$F \circ \text{XOR}(x, y) = [(x_1 \vee y_1) \wedge (\overline{x_1} \vee \overline{y_1})] \vee \neg[(x_2 \vee y_2) \wedge (\overline{x_2} \vee \overline{y_2})]$$

2.7 Size of BPHP_n

$$S_{\text{Res}}(\text{BPHP}_n) \geq 2^{\Omega(n)}$$

PROOF. Let $n = 2^k - 1$ and $N = nk$. Consider the following distribution D on a partial assignment ρ:

  for $i = 1, ..., N$ **do**
    Flip a coin
    if heads **then**
      $x_i \leftarrow 0$ or 1 with probability $\mathbb{P} = \frac{1}{2}$
    if tails **then**
      $y_i \leftarrow 0$ or 1 with probability $\mathbb{P} = \frac{1}{2}$

Let $\rho \in D$. We observe the following facts:

1. $\widetilde{\text{BPHP}}_n \restriction \rho$, i.e. restricted to the truth assignment, is BPHP_n with some variables negated, say BPHP'_n. Note that $w_{\text{Res}}(\text{BPHP}'_n) = w_{\text{Res}}(\text{BPHP}_n)$ also.

2. $\mathbb{P}[C \restriction \rho \neq 1] \leq \left(\frac{3}{4}\right)^{\frac{t}{2}}$, where C is a width $\geq t$ clause over the variables of $\widetilde{\text{BPHP}}_n$.

PROOF.

Let $C = \ell_1 \vee \cdots \vee \ell_s : s \geq t$. Then $\ell_i = x_i, y_i, \overline{x_i}$, or $\overline{y_i}$. Hence, $\mathbb{P}[\ell_i \restriction \rho = 1] = \frac{1}{4}$, so $\mathbb{P}(\ell_i \restriction \rho \neq 1) = \frac{3}{4}$. Since we sample ℓ_i independently, and would like $\ell_i \neq 1 \; \forall i = 1, ..., s$, we have

$$\mathbb{P}[C \restriction \rho \neq 1] = \prod_{i=1}^{s} \mathbb{P}(\ell_i \restriction \rho \neq 1) \leq \left(\frac{3}{4}\right)^{\frac{s}{2}} \leq \left(\frac{3}{4}\right)^{\frac{t}{2}} \qquad \square$$

$\square$

sadf

asdf