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:

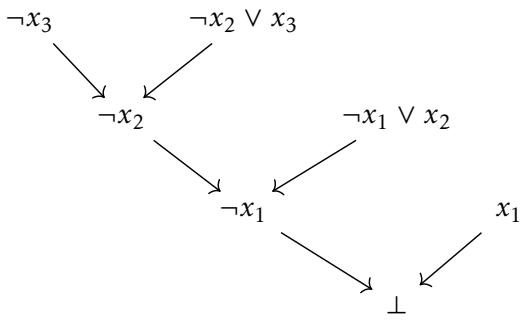
1. $$\frac{C \vee x \qquad D \vee \neg x}{C \vee D} \text{ (resolution)}$$
2. $$\frac{C}{C \vee D} \text{ (weakening)}$$

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

E.G. 2.1 ♠ *Examples* ♣

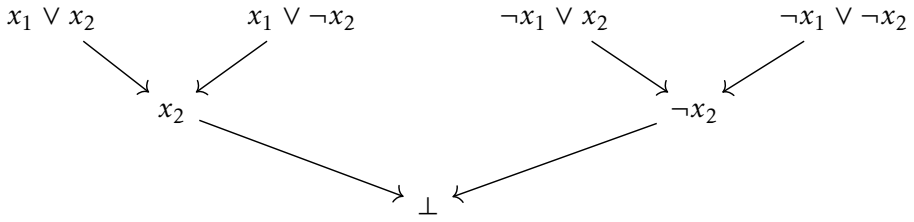
Eg 1: Let $F = x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$. Notice that this is not satisfiable. If F is true, we deduce the following

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.

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 on variables $\{x_1, ..., x_n\}$ with all possible width-n clauses on these variables with distinct literals. DEF 2.4

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. PROP 2.2

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). □

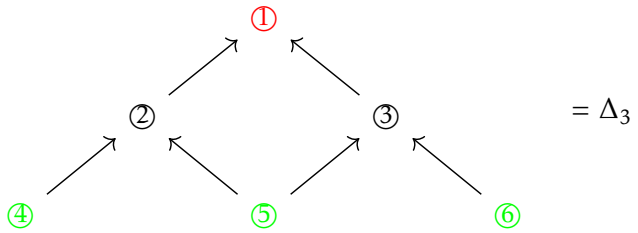
What is easy and what is hard for (tree) resolution, where "easy" means there exists a polynomial-size refutation $n^{O(1)}$, where $n = S(\Pi)$? We saw previously $x_1 \wedge (\neg x_1 \vee x_2) \wedge ... \wedge (\neg x_{n-1} \vee x_n) \wedge \neg x_n$, which has a proof of size $O(n)$, width 2, and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

DEF 2.6 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

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

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."

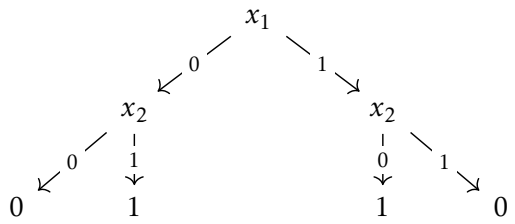
We assert that, for fixed $i = 1, ..., m \geq n+1$,

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

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 $2^{\Omega(n)}$.

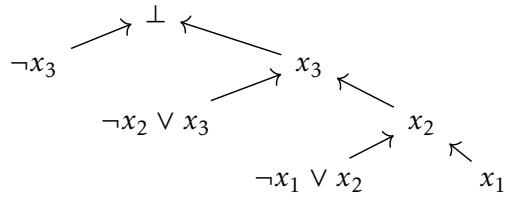
DEF 2.8 Let $f : \{0,1\}^n \to S$ be a function. A *decision tree* T computing f is a rooted, labeled, binary tree in which every leaf l is labeled with an element of S, every internal node is labeled with an input x_i, and the two outgoing edges from each node are labeled with 0 and 1.

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



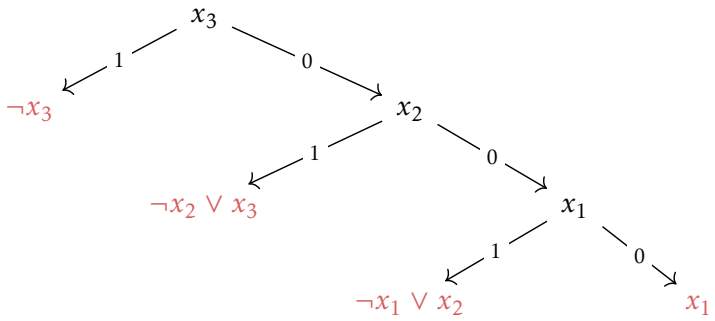
T computes f if, for each input $x \in \{0,1\}^n$, the leaf of T reached by the path consistent with x is labeled $f(x)$.

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: $x_{ij} \in \{0,1\}\ \forall i = 1, \ldots, n+1; j = 1, \ldots, n$
  for $i = 1, \ldots, n+1$ **do**
    for $j = 1, \ldots, n$ **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 $S_{\text{Res}}(\text{PHP}_n^{n+1}) = 2^{\Omega(n)}$.

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,

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

DEF 2.10

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

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

PROOF.

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

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

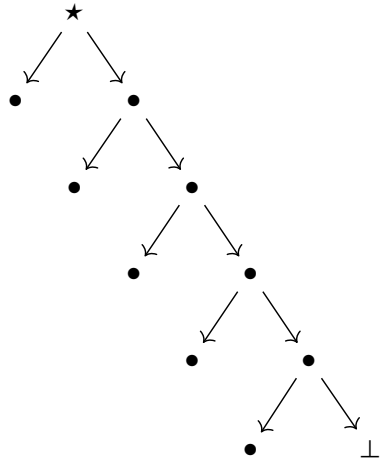
$$\mathbb{P}(\exists) \leq |V|\mathbb{P}(v) = |V|\left(\frac{3}{2}\right)^{\frac{-n}{4}} \implies |V| \geq \left(\frac{3}{2}\right)^{\frac{n}{4}}$$

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

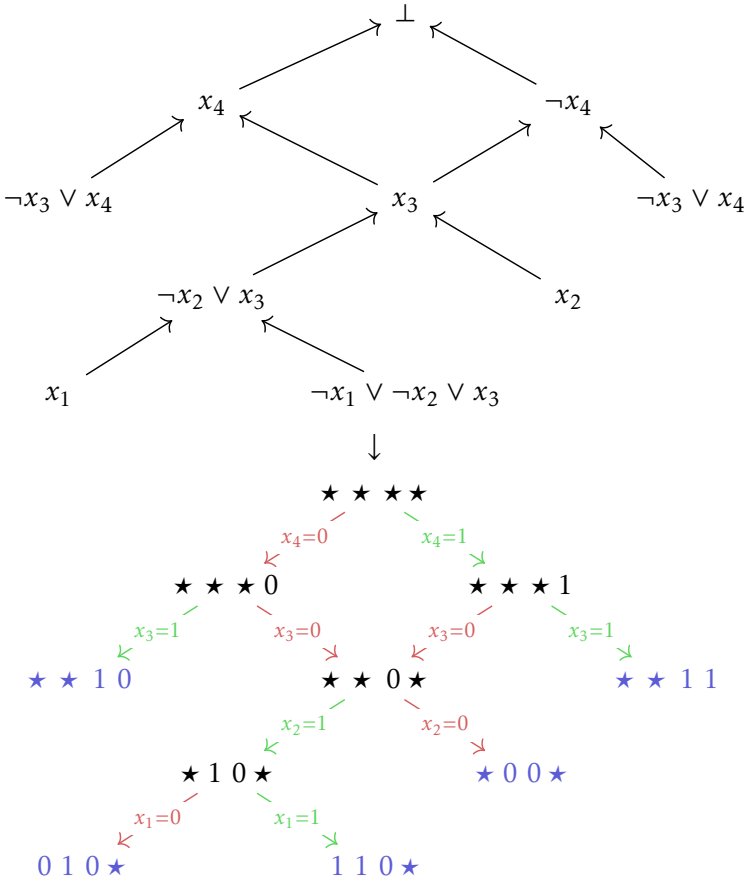
$$\sum_{i=0}^{w} 2^i \binom{n}{i} \leq \sum_{i=0}^{w} 2^i n^i = O((2n)^w) = O(n^{w+1})$$

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 $i = 1, \ldots, N$ **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:

Fact 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.

Fact 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 $\widetilde{\text{BPHP}}_n$.

As a 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

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

Now, let Π be a proof of $\widetilde{\text{BPHP}}_n$. We extract a proof of BPHP_n with width $\leq 3\log(|\Pi|)$. Sample $\rho \sim D$ and restrict *all* clauses in Π with ρ. Fact 1 above says that $\Pi \restriction \rho$ is a proof of BPHP'_n. If Π_t is the set of clauses in Π with width $\geq t$, then

$$\mathcal{P} = \mathbb{P}(\exists C \in \Pi_t : C \restriction \rho \neq 1) \leq |\Pi| \mathbb{P}[C \restriction \rho \neq 1] \leq |\Pi| \left(\frac{3}{4}\right)^{\frac{t}{2}}$$

Then the probability of interest $\mathcal{P} < 1 \iff |\Pi| < \left(\frac{4}{3}\right)^{\frac{t}{2}}$, which can accomplish for any sufficiently large t. But, heuristically, if $\mathcal{P} < 1$, then certainty there exists an assignment ρ for which $C \restriction \rho = 1 \; \forall C \in \Pi_t$. Hence, with this assignment, all $\geq t$ clauses in the proof of $\widetilde{\text{BPHP}}_n \restriction \rho$ disappear. But $\widetilde{\text{BPHP}}_n \restriction \rho$ is BPHP_n with some variables negated, and so we yield a width $< t$ proof of this. But we know that $w_{\text{Res}}(\text{BPHP}_n) \geq n$, so $t \geq n$. $\square$

2.8 Size-Width Bounds

$$S_{\text{TreeRes}}(F) \geq 2^{w_{\text{Res}}(F) - w(F)}$$

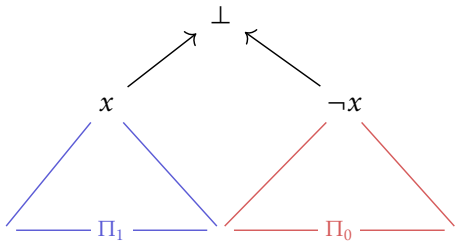
$$S_{\text{Res}}(F) \geq 2^{\Omega\left(\frac{(w_{\text{Res}}(F) - w(F))^2}{n}\right)}$$

These bounds imply that it is insufficient to bound $S_{(\text{Tree})\text{Res}}$ by w_{Res} alone: one needs $w(F)$ as well. To illisutrate the distinction, consider a refutation of

$$F = (x_1 \vee \cdots \vee x_n) \wedge \neg x_1 \wedge \cdots \wedge \neg x_n$$

$w(F) = w_{\text{Res}} = n$, but we have a relatively short proof (of size n).

We discuss the intuition behind a proof first. Consider a tree-like refutation of F, Π. As its penultimate step, we resolve a variable x with $\neg x$. Hence, we derive both x and $\neg x$ from sub-tree refutations Π_0 and Π_1, respectively.



The proof $\Pi_0 \restriction (x = 1)$ proves $F \restriction (x = 1)$, and similarly $\Pi_1 \restriction (x = 0)$ proves (resolution-refutationally) $F \restriction (x = 0)$. But what *is* $F \restriction (x = b)$? If $F = C_1 \wedge \cdots \wedge C_n$, then restricting by $(x = 1)$ will disappear all appearances of $\neg x$ in C_i, and disappear entire clauses C_j which contain x. Hence, $F \restriction (x = 1)$ is the same as F with all clauses resolved on by x.

Let x be derived from F by Π_1. Then, resolve x on all clauses in F used by Π_0. From here, we use $\Pi_0 \restriction (x = 1)$ to yield $\bot$. We continue this process to obtain (hopefully), an upper bound on $w_{\text{Res}} - w$ by the size of the proof.

PROOF. WLOG let $|\Pi| = 2^\ell$ be a tree-like proof of F. Then by Prop 2.8 and Prop 2.9, we conclude

$$w_{\text{Res}}(F) \leq \ell + w(F) \implies \ell \geq w_{\text{Res}}(F) - w(F) \implies |\Pi| \geq 2^{w_{\text{Res}}(F) - w(F)}$$

Analogous propositions in the dag-like setting will prove the second statement of the theorem in much the same way. □

DEF 2.15 $x^b := x$ if $b = 1$ and $\neg x$ if $b = 0$. We employ some helpful propositions:

PROP 2.8 Let F be an unsatisfiable CNF formula. Let x be a variable of F, $b \in \{0, 1\}$. If $w_{\text{Res}}(F \restriction x = b) \leq k - 1$ and $w_{\text{Res}}(F \restriction x = 1 - b) = k$, then $w_{\text{Res}} \leq k$.

PROOF. Let Π_b be the width-$k-1$ refutation of $F \restriction x = b$. Let Π_{1-b}, similarly, be the refutation of $F \restriction x = 1 - b$.

We weaken x^{1-b} into all clauses of Π_b to yield a proof of x^{1-b} of width k.

Resolve x^{1-b} with all clauses in F to obtain $F \restriction x = 1 - b$. Then we use $\Pi_{1-b} \restriction x = 1 - b$ to refute F. (See picture above). Then we yield a width k resolution proof. □

PROP 2.9 If $\ell, n > 0$ and Π is a tree-like refutation of F on variables $x_1, ..., x_n$ such that $|\Pi| \leq 2^\ell$, then $w_{\text{Res}}(F) \leq \ell + w(F)$.

PROOF. We do double-induction on ℓ and n. Consider the base cases:

$\ell = 0 \implies |\Pi| = 1$, so $\Pi = \bot \in F$.

$n = 0 \implies F = \bot$, so $1 \leq 0 + 1$.

Let $\ell, n > 0$. Let $|\Pi| \leq 2^\ell$. Let x be the last variable resolved in this proof. WLOG let $|\Pi_1| < \frac{|\Pi|}{2} \leq 2^{\ell-1}$. Restrict Π_1 by $x = 0$. Then we yield a refutation of $F \restriction x = 0$ in size $\leq 2^{\ell-1}$. By induction, we have that

$$w_{\text{Res}}(F \restriction x = 0) \leq \ell - 1 + w(F)$$

Similarly, restrict Π_0 by $x = 1$ to yield a refutation of $F \restriction x = 1$ in size $\leq 2^\ell$. While we cannot perform induction on ℓ, we can on n, since restricting by $x = 1$ is equivalent to resolving on x for all clauses. Again, then

$$w_{\text{Res}}(F \restriction x = 1) \leq \ell - w(F)$$

By Prop 2.8, then we conclude $w_{\text{Res}}(F) \leq \ell + w(F)$ □

III Frege Systems

Let's take a step back and recall what resolution allowed (and didn't allow) us to do. We were able to work with formulas which look like

$$F = C_1 \wedge \cdots \wedge C_n$$

for clauses C_i, and we could weaken or resolve on variables in these clauses, with the hope of reducing to $\bot$. The proof system is hence extremely tailored to proving refutations, but is ill-equipped to handle tautologies or deductions. In fact, it is pretty limited in its scope for refutation: we have $2^{\Omega(n)}$-size proofs of BPHP_n (meh), and can only work with conjunctions of disjunctions (also meh).

How can we improve? One thought is to more rules to resolution, but in fact

$$\text{Refutation} + \text{Sound Inference Rules} \equiv \text{Resolution}$$

so this won't work. As it turns out, resolution, though not the most powerful proof system, can simulate all inference rules.

Try this for $C, D \vDash E \implies C, D \vdash E$

DEF 3.1 An *inference rule* is any sound method to deduce new boolean formulas from old ones.

DEF 3.2 A *substitution* σ is a function mapping variables to formulas. It is hence a mapping of formulas to formulas, i.e. if F is a formula, then $\sigma(F)$ is the formula resulting from applying σ to the variables of F.

DEF 3.3 Let R be a set of finite inference rules on variables. Let $G_1, ..., G_m$ and F be boolean formulas. An *R-Frege proof of F* from $G_1, ..., G_m$ is a sequence of formulas $F_1, ..., F_s = F$ such that each $F_i = G_j$ for some j or there is a rule $A_1, ..., A_t \vdash B$ and a substitution σ such that $\forall \ell, \sigma(A_\ell) = F_k : k < i$ and $\sigma(B) = F_i$.

DEF 3.4 A set of inference rules is called *implicationally complete* if, whenever $F_1, ..., F_m \vDash G$, then there is a Frege proof of G from $F_1, ..., F_m$.

DEF 3.5 Let $U(F, \Pi)$ and $V(F, \Pi)$ be propositional proof systems, i.e. for all tautologies F there exists a proof Π such that $U(F, \Pi) = 1$ and U runs in polynomial time (and similarly for V). We say that U *p-simulates* V if there is a polynomial time algorithm f such that $V(F, \Pi) = 1 \implies U(F, f(\Pi)) = 1$.

If U and V p-simulate eachother, we sometimes write $U \equiv_p V$

3.1 Reckhow

Let R_1, R_2 be finite implicationally complete inference rules. Then R_1-Frege p-simulates R_2-Frege.

A language, like $\{\wedge, \vee, \neg\}$, is a collection of symbols which operate on boolean formulas. We are only concerned with informationally complete languages, i.e. languages which may express any boolean formula, and in this event we may translate between one and the other without trouble.

PROOF. Assume R_1 and R_2 have the same language (otherwise, we may translate). Let $\Pi = F_1, ..., F_s = F$ be an R_2-Frege proof of F. Consider any R_2-Frege inference rule, i.e. $A_1, ..., A_t \vdash B$. Then there is an R_1-Frege proof of B from $A_1, ..., A_t$ by completeness (Def 3.5), i.e.

$$A_1, ..., A_t, B_1, ..., B_\ell = B$$

where each B_i results from a rule of R_1 under some substitution σ. Hence, we may prove F in R_1-Frege, and in particular with size $O(\ell S(\Pi))$, where Π is the size of the proof in R_2-Frege, and ℓ is the maximal length of any R_1 derivation of an R_2 inference rule. □

DEF 3.6 A Frege proof is *tree-like* if each derived formula is used by at most one inference rule. Equivalently, the proof graph is a tree.

Can tree-like resolution p-simulate dag-like resolution? No! The other way is fine (any tree-like proof *is* a dag-like proof). However, not every dag-like proof may be translated to a tree-like proof while maintaining polynomially-bounded size increases. One can look to $d_{\text{Res}}(\text{PEB}_{\Delta_h}) = \Omega(h) = \Omega(\sqrt{\#\text{vars}})$ to conclude that $S_{\text{TreeRes}}(\text{PEB}_{\Delta_h} \circ \text{XOR}) \geq 2^{\Omega(\sqrt{n})}$. But we can also show $S_{\text{Res}}(\text{PEB}_{\Delta_h} \circ \text{XOR}) = n^{O(1)}$

A fact we haven't proved, but is good exersice

DEF 3.7 Shoenfeld Calculus

$$\frac{}{p \vee \neg p}\ \text{excluded middle} \qquad \frac{p}{p \vee q}\ \text{weakening} \qquad \frac{p \vee (q \vee r)}{(p \vee q) \vee r}\ \vee\ \text{associativity} \qquad \frac{p \vee p}{p}\ \vee\ \text{contracting} \qquad \frac{p \vee q \quad \neg p \vee r}{q \vee r}\ \text{cut}$$

Sometimes we add in and associativity and $p, q \vdash p \wedge q$.

DEF 3.8 For a Frege proof $\Pi = C_1, ..., C_\ell$, the *length* is the number of lines ℓ, the *size* is $\sum_{i=1}^{\ell} |C_i|$, and the *depth* is the max root-leaf path in a proof.

3.2

Let F be any Frege system, and let $\Pi = C_1, ..., C_\ell$ be any Frege proof starting from formulas $B_1, ..., B_t$ and with size s. Then there is a tree-like Frege proof Π' such that the length of Π' is $O(\ell \log(\ell))$, the size is $O(s\ell \log(\ell))$, and the depth is $O(\log(\ell))$.

PROOF.

WLOG assume the first lines $C_1 = B_1, ..., C_t = B_t$. Define $D_i := C_1 \wedge C_2 \wedge \cdots \wedge C_i$, bracketed as a binary tree, i.e. $((C_1) \wedge (C_2)) \wedge \cdots \wedge ((C_{i-1}) \wedge (C_i))$. Then $d(D_i) = O(\log(i))$. We show how to derive D_{i+1} from D_i. Let C_{i+1} be derived from C_{j_1} and C_{j_2}, where $j_1, j_2 \leq i$.

Claim: $D_i \vdash D_i \wedge C_k$ for any $k \leq i$ in tree-like Frege.

We use the law of excluded middle to conclude

$$\begin{aligned}
\neg(C_k \wedge D_i) \vee (C_k \wedge D_i) &\implies \neg C_k \vee \neg D_i \wedge (C_k \wedge D_i) \\
&\implies \neg C_k \vee \left(\bigvee_{a=1}^{i} \neg C_a \right) \vee (C_k \wedge D_j) \\
&\implies \bigvee_{a=1}^{i} \neg C_a \vee (C_k \wedge D_i) \implies C_k \wedge D_i
\end{aligned}$$

To then derive D_{i+1} from D_i, we first derive

$$D_i \vdash D_i \wedge C_{j_1} \wedge C_{j_2} \vdash D_i \wedge C_{i+1} \equiv_{\text{syn}} D_{i+1}$$

Hence, we prove $D_1, ..., D_\ell$, and extract C_ℓ from D_ℓ. The size of this proof is $O(|D_{i+1}| \log(i))$ □

There are no natural tautologies that are exponentially hard (or even conjectured to be) for Frege to prove. These systems (we will see Extended Frege soon, which is even stronger than Frege) can essentially formalize most standard mathematical arguments.

How can we make Frege *even stronger*?

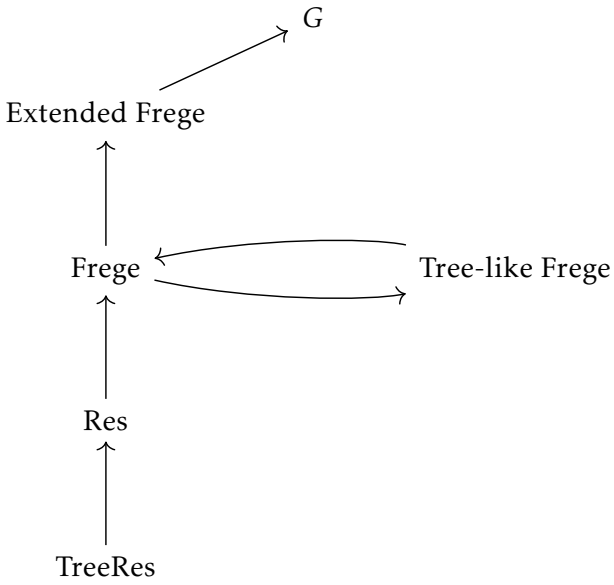
1. Use quantifiers, i.e.

   $$A(s) \vdash \forall y, A(y) \qquad A(B(t)) \vdash \exists z : A(z)$$

   We call this system G. As it turns out, tree-like, G restricted to 1 essential quantifier is equivalent to Extended Frege.

2. We can introduce definitions! This is exaclty what Extended Frege is. We allow, at any time, the introduction of lines $(q \vee \neg F) \wedge (\neg q \vee F)$, i.e. $q = F$, where q is a new formal symbol.

3. Allow substitutions, i.e. from $A(x) \vdash A(B(y))$, where B is another formula. However, this is sound only for proving tautologies A. This system is sometimes called substitution Frege.



3.3 thmstart

Substitution Frege $\equiv_p$ Extended Frege

EXTENDED FREGE

DEF 3.9 An Extended Frege proof of A from $B_1, ..., B_t$ is a sequence of formulas $C_1, ..., C_s = A$ in which each C_i is either deduced from earlier lines by Frege rules or is an instance of the extension rule, i.e.

$$C_i = (\neg q \vee F) \wedge (q \vee \neg F)$$

where q is a new atom such that

1. q doesn't appear in any of $B_1, ..., B_t$
2. q doesn't appear in F
3. q doesn't appear in $C_1, ..., C_{i-1}$

The extension rule is akin to "definitions"

We adopt the notation $q := F$ to mean the extension rule.

DEF 3.10 The length of an Extended Frege proof Π s is the number of formulas in it. The size is $\sum_{i=1}^{s} |C_i|$, where $|C_i|$ is the number of atoms and connectives in it.

In general, length and size do not have to be related.

3.4 Pigeons in Extended Frege

PHP_n^{n+1} has polynomial-size Extended Frege proofs.

In fact, the pigeonhole principle has polynomial-size *regular* proofs, but this is an endeavor to prove

PROOF.

A natural approach: remark that there are $n+1$ pigeons, each pigeon is mapped to a hole, and there are n holes. The injectivity axioms will imply that $n+1 < n$, hence a contradiction. This is how the statement is proven for regular Frege.

Now for the proof idea: suppose that $p : [n+1] \to [n]$ is an injection. We will use p to create a new injection $q : [n] \to [n-1]$. By induction, then, we generate an injection $[2] \to [1]$, which one can brute-force refute. Recall the boolean definition of PHP_n^{n+1}:

$$\text{PHP}_n^{n+1}(\vec{p}) := \bigwedge_{i=1}^{n+1} \underbrace{\left(\bigvee_{j=1}^{n} p_{ij} \right)}_{\text{each pigeon is mapped}} \wedge \underbrace{\bigwedge_{i \neq k, j} \left(\neg p_{ij} \vee \neg p_{kj} \right)}_{\text{no collisions}}$$

Conceptualize a truth assignment p as an injection. We define $q : [n] \to [n-1]$ as follows.

$$q(i) := \begin{cases} j & p(i) = j < n \\ p(n+1) & p(i) = n \end{cases} \quad \star$$

Now, to argue that $q(i)$ is an injection. It is clearly defined for each i by totality of the pigeon hole principle. Suppose q is not injective, and write $q(i) = q(k) = j$. We know that $p(h) = j$ or $p(h) = n$ for $h \in \{i, k\}$. If $p(i) = p(k)$, then we are done, so suppose $p(i) = j$ and $p(k) = n$. Then $p(n+1) = j$ since $q(k) = j$, which is a contradiction! To implement this proof in Extended Frege is tedious, so we will sketch the proof.

In particular, we'll assume we can use De Morgan, distributivity or $\wedge$ over $\vee$

We deduce $\text{PHP}_{n-1}^{n}(\vec{q})$ from $\text{PHP}_n^{n+1}(\vec{p})$, where we use the extension rule to implement the arguments above. Let

$$q_{ij} := p_{ij} \vee (p_{in} \wedge p_{n+1,j})$$

for $i = 1, \ldots, n$, $j = 1, \ldots, n-1$, as in $\star$, using the extension rule.

We need now to prove $\bigvee_{j=1}^{n-1} q_{ij} := Q$. To do so, we introduce the axiom $Q \vee \neg Q$. Then, by De Morgans, we have

$$Q \vee (\neg q_{i1} \wedge \neg q_{i2} \wedge \cdots \wedge \neg q_{i,n-1})$$

and the distributive rule to find

$$(Q \vee \neg q_{i1}) \wedge (Q \vee \neg q_{i2}) \wedge \cdots \wedge (Q \vee \neg q_{i,n-1})$$

extracting the $\wedge$'s yields

$$Q \vee \neg q_{ij} \quad \text{for each } j = 1, ..., n-1$$

Cutting this with the definition of q_{ij} yields

$$Q \vee \neg \Big(p_{ij} \vee (p_{in} \wedge p_{n+1,j}) \Big)$$

with more De Morgan and distributivity yields

$$(Q \vee \underset{\clubsuit}{\neg p_{ij}}) \wedge (Q \vee \neg p_{in} \vee \underset{\spadesuit}{\neg p_{n+1,j}})$$

Cutting $\clubsuit$ with $\bigvee_{j=1}^{n} p_{ij}$ for each $j = 1, ..., n-1$ gets us $Q \vee p_{in}$ for each i. And cutting this result with $\spadesuit$ will get us $Q \vee \neg p_{n+1,j} for j = 1, ..., n$. Cutting this with p_{n+1} totality deduces Q, as desired.

The collision algorithm is deduced similarly. We repeat this argument to deduce, inductively, PHP_1^2 □

Repeated uses of the extension axioms are *key* to a proof like the above.

3.5 Size and Length Contraction of Extended Frege Proofs

Let A be a tautology, and suppose there is an Extended Frege proof of A with length k. Then there exists an Extended Frege proof of A with length $O(k + |A|)$ and size $O(k + |A|^2)$, and in which each individual line has size $O(|A|)$.

PROOF.

Let $\Pi = D_1, ..., D_k$ be an Extended Frege proof of A, so in particular $D_k = A$. For any formula F, we let $\text{Ext}(F)$ be the following set of extension axioms. Namely, for each subformula E of F, create an atom q_E which represents this subformula, and add the extension axioms:

1. $E = \neg G$, include $q_E := \neg q_G$
2. $E = G \wedge H$, include $q_E := q_G \wedge q_H$
3. $E = G \vee H$, include $q_E = q_G \vee q_H$
4. $E = p$, include $q_E := p$

Now we argue that there is always a constant-size proof of the atom q_{D_i} from $q_{D_1}, ..., q_{D_{i-1}}$ and $\cup_{j<i}\text{Ext}(D_i)$. Each D_i was deduced from earlier lives by a Frege rule, deduced D_i from $D_{i_1}, ..., D_{i_c}$. Unpacking $D_{i_1}, ..., D_{i_c}$, we can derive q_{D_i} in $O(1)$ steps by applying the Frege rule.

Therefore, we can derive q_A in $O(k)$ steps, by simulating Π and paying a constant factor. Then, by exactly $O(|A|)$ uses of cut with the extension axioms in $\text{Ext}(A)$, we can derive A from q_A in $O(|A|)$ steps. Each line will require at most $O(|A|)$ symbols, so we observe a final proof size of $O(k + |A|^2)$ and length $O(k + |A|)$. □

Line count characterizes the complexity of Extended Frege proofs. In fact, one can show

$$L_{EF}(F) = \Theta(L_F(F))$$

We consider a restriction on Extended Frege and hearken back to the past chapter:

DEF 3.11 Given a clause $C = \ell_1 \vee \cdots \vee \ell_k$, an atom q, the extension rule $q := C$ is given by

$$\neg q \vee \ell_1 \vee \cdots \vee \ell_k \quad \text{and} \quad q \vee \neg\ell_1, q \vee \neg\ell_2, ..., q \vee \neg\ell_k$$

DEF 3.12 An *Extended Resolution* refutation of an unsatisfiable CNF $F = C_1 \wedge \cdots \wedge C_m$ is a sequence of the form $D_1, ..., D_\ell = \bot$, where each D_i is deduced from earlier lines by resolution or weakening; or is a clause from F; or is an instance of $q := C$ as above.

3.6 ER $\geq_p$ EF

Extended Resolution p-simulates Extended Frege.

DEF 3.13 A *De Morgan formula* is a formula where all negations are immediately above the leaves. The nomenclature comes from the fact that, using De Morgan's law, we may transform any formula F into a De Morgan formula F' where $F \equiv F'$ and $|F'| = O(|F|)$.

DEF 3.14 If F is a De Morgan formula, the *alternation depth* of F is the maximum number of alternations of $\wedge$ and $\vee$ on any root-leaf path in F.

DEF 3.15 A *depth d Frege proof* is one in which all lines have alternation depth at least d.

DEF 3.16 We denote by Frege_d the Frege system restricted to depth–d proofs.

PROP 3.1 Resolution $\equiv_p \text{Frege}_1$ and $\text{Frege} \equiv \text{Frege}_{O(\log(n))}$

In the space between 1 and $O(\log(n))$, we *do* have meaningful lower bounds: as recently as 2024, we know that there exist super-polynomial lower bounds on CNFs in $\text{Frege}_{O\left(\frac{\log(n)}{\log(\log(n))}\right)}$

Not to be confused with the definition of depth for a Frege proof, c.f. Def 3.8

Is Frege_i stronger than Frege_{i-1}? We don't know! We cannot even differentiate between Frege_2 and $\text{Frege}_{O(1)}$

DEF 3.17 A k-DNF formula is one of the form

$$\bigvee_i \bigwedge_j \ell_{ij}$$

where each $\bigwedge_j$ has at most k literals.

DEF 3.18 A Res(k) refutation of a CNF $F = C_1 \wedge \cdots \wedge C_m$ is a sequence of k-DNF formulas $D_1, ..., D_s = \bot$ such that each D_i is either a clause from F or it is deduced by one of the following

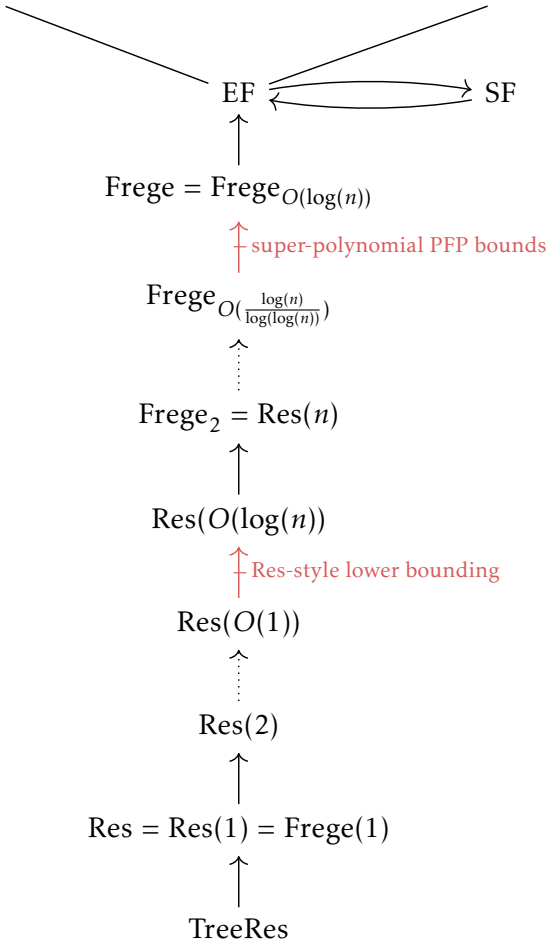
$$\frac{A \vee \bigwedge_{i=1}^{j} \ell_i \quad B \vee \neg\ell_1 \vee \neg\ell_2 \vee \cdots \vee \neg\ell_j}{A \vee B} \quad \text{cut rule}$$

$$\frac{A}{A \vee \ell_i} \quad \text{weakening}$$

$$\frac{A \vee \ell_1 \quad A \vee \ell_2 \quad \cdots \quad A \vee \ell_j}{A \vee (\bigwedge_{i=1}^{j} \ell_i)} \quad \text{and creation}$$

$$\frac{A \vee \bigwedge_{i=1}^{j} \ell_i}{A \vee \ell_1 \quad A \vee \ell_1 \quad \cdots \quad A \vee \ell_j} \quad \text{and extraction}$$

$\text{Frege}_2 = \text{Res}(\#\text{ of vars}) = \text{Res}(n)$ We update our proof strength graph: PROP 3.2



We may go even further if we'd like, but proof systems stronger than EF are hard to grab onto, and are eventually just logician's playthings. **The punchline:** very strong proof systems cannot escape first-order logic and set theory. If we believe NP ≠ coNP, then we must believe that there is a family of CNF formulas that require long proofs in ZFC (i.e. an arbitrarily strong proof system)

PEANO ARITHMETIC

Peano arithmetic captures reasoning about natural numbers. It is a first-order logical theory. It is defined with the symbols $L = \{0, S(\cdot), +, \times\}$, where $S(\cdot)$ is a function which takes $n \mapsto n + 1$. The axioms are, in short, We may say "PA"

A_1 $\{0, S(\cdot), +, \times\}$ all behave as we expect under the ring $\mathbb{N}(+, \times)$.

A_2 Induction: for all formulas φ,

$$[\varphi(0) \wedge \forall x : \varphi(x) \to \varphi(s(x))] \to \forall x, \varphi(x)$$

It turns out the PA is *extremely* powerful, and can formalize practically all theorems we know today.

Can we formalize reasoning about *polynomial-time* computation? In other words, is there a theory T s.t. T proves the existence of all and only the functions which are computable by polynomial-time algorithms. We discuss two such theories:

Cook, circa 1970
Buss, circa 1980

PV ("polynomially verifiable") and S_1^2, which is a subset of PA.

Before we continue, we introduce $|x| := \lfloor \log_2(x) \rfloor$, where $x \in \mathbb{N}$. (This is the "length" of x, in the sense of how many bits you need to represent it). We also have $x\#y := 2^{|x||y|}$ (Reads: x smash y).

DEF 3.20 The language of S_1^2 is given by $L_{S_1^2} = L \cup \{|\cdot|, \#\}$, with the axioms $A_1 \cup$ "a special induction," to be defined soon.

First, for some intuition. If $\varphi(x, y)$ is a sentence, where $\forall x \, \exists y : \varphi(x, y)$ holds, how can we *witness* the truth of this by an algorithm? Suppose we had an algorithm A which always halts s.t., given x, $\varphi(x, A(x))$ is true. An exceptionally dumb algorithm:

Require: x
  for $y = 0, \ldots$ **do**
    check $\varphi(x, y)$
    if $\varphi(x, y)$ true **then**
      output y

If, indeed, $\forall x \, \exists y : \varphi(x, y)$ holds, then this algorithm will work. But it is not polynomial. We can make it so by restricting $y = O(x^{O(1)})$, then we will do polynomial-time checks. And the checking operation itself must be polynomial-time.

DEF 3.21 A formula φ over $L_{S_1^2}$ is Σ_1^b if

$$\varphi(x) := \exists y \leq t(x) : \psi(x, y)$$

where $\psi(x, y)$ is "sharply bounded," i.e. every quantifier $\forall z \leq |x|$ or $\exists z \leq |x|$ in ψ is bounded by the length of x or y.

We return now to Def 3.20, and define "special induction" to be the induction principle only on Σ_1^b formulas.

3.7

If $S_2^1 \vdash \forall x \exists y \leq t(x) : \varphi(x, y)$, where φ is sharply bounded, then there is a polynomial-time algorithm A s.t. $\forall \, x \varphi(x, A(x))$ holds.

Convsely, we have

3.8

If A is a polynomial-time algorithm, then there is a term t s.t. $S_1^2 \vdash \forall x \exists y \leq t(x)$

This establishes a correspondence between polynomial-computable algorithms and provably total formulas in S_2^1. If $S_2^1 \vdash \forall x \forall y \leq t(x) \psi(x, y)$, we consider the "propositionalized translation," typically denoted $\|\psi\|^n$. Then $EF \vdash \|\psi\|^n$ efficiently.

DEF 3.22

Let $x_1, ..., x_n \in \{0, 1\}$ be propositional variables. A *boolean circuit* over these variables is a sequence

$$g_1, ..., g_s$$

where each $g_i \in \{x_i, 0, 1\}$ or $g_i = g_j \wedge g_k$ or $g_i = g_j \vee g_k$ or $g_i = \neg g_j$, where $j, k < i$.

For each of the symbols $+, \times, S, |\cdot|, \#$, there is an $O(n^2)$-size boolean circuit which computes them on n-bit inputs.