Chapter 6

Proof Systems

6.1 Introduction

We have introduced proof systems briefly in chapter 2 (see page 11). A *proof system* (PS) for a language L is a polytime relation V such that $x \in L \iff \exists p V(x,p)$. Here V is called the *verifier* and p is the encoding of a proof that x is in L. *Soundness* means that $\exists p V(x,p) \Rightarrow x \in L$, and *completeness* means that $x \in L \Rightarrow \exists p V(x,p)$.

The *complexity* of V is $f_V : \mathbb{N} \longrightarrow \mathbb{N}$, where

$$f_V(n) = \max_{x \in L, |x|=n} \min_{p,\ V(x,p)} |p|,$$

and V is *polynomially bounded (polybounded)* iff f_V is bounded by some polynomial, i.e., there exists a polynomial q such that $f_V(n) \leq q(n)$.

As was mentioned in chapter 2, the canonical example of a language with a polybounded PS is SAT. However, the proof systems in which we are most interested are those for TAUT and UNSAT, where UNSAT is the complement of SAT. Thus, we make the following definition: a *propositional proof system* (PPS) is a proof system for TAUT or UNSAT.

Theorem 6.1.1 (Cook-Reckhow) *A polybounded propositional proof system (PPS) exists iff* NP = co-NP.

PROOF: ($\Rightarrow$) Suppose that NP = co-NP, and so TAUT is in NP. Let M be the nondeterministic polybounded TM that decides TAUT. Let $V_M(\phi, p)$ be true iff $p = \langle C_1, C_2, \ldots, C_m \rangle$ encodes an accepting computation of M on ϕ.

($\Leftarrow$) Suppose that there exists a polybounded PPS, V. Let M be the following nondeterministic TM: on input ϕ, M guesses a p and verifies $V(\phi, p)$. Thus M decides TAUT, and so TAUT is in NP. □

We say that a propositional proof system V is *automatizable*, if there exists an algorithm A, such that given a ϕ in TAUT as input, A outputs a proof p of ϕ in time polynomial in the length of the shortest proof of ϕ. (If $\phi \notin$ TAUT, then A outputs anything.)

Lemma 6.1.2 *There exists a polybounded automatizable PPS iff* $\mathsf{P} = \mathsf{NP}$.

PROOF: ($\Rightarrow$) Suppose that there exists a polybounded automatizable PPS. Given any ϕ in TAUT we know that it has a proof of polynomial size, and that we can find a proof of ϕ in time polynomial in its length. As TAUT is co-NP-complete, we get that $\mathsf{P} = \mathsf{co\text{-}NP}$ and, what follows, $\mathsf{P} = \mathsf{NP}$.

($\Leftarrow$) Let us now assume that $\mathsf{P} = \mathsf{NP}$. Then SAT can be decided by some deterministic polytime Turing machine M. Take any formula ϕ and notice, that it is a tautology iff $\neg\phi$ is not satisfiable. Consider the computation of M on $\neg\phi$. M finishes its work in polynomial time, thus it can use only polynomial space. Therefore the encoding of the computation has polynomial size. But this encoding is a proof that $\phi \in$ TAUT (we can verify it in polynomial time, simply checking that it is a correct computation of M on ϕ), and it can be produced in polynomial time (as M produces it). Thus we have found a polybounded automatizable PPS. □

Let BF be the class of languages recognizable by a family of polynomial size Boolean Formulas. That is, $L \in \mathsf{BF}$ iff there exists a family of Boolean formulas $\Phi = \{\phi_n\}$ such that $\phi_n = \phi_n(x_1, x_2, \ldots, x_n)$, i.e., ϕ_n has n variables, and there is a fixed polynomial p such that $|\phi_n| \leq p(n)$, and

$$a \in L \iff \phi_{|a|}(a_1, a_2, \ldots, a_{|a|}) \text{ is true.}$$

This is analogous to the definition of a language being recognizable by a family of circuits given on page 55.

Theorem 6.1.3 (Spira) $\mathsf{NC}^1 = \mathsf{BF}$.

PROOF: Consider an NC^1 circuit; it can be represented by a Boolean formula, by transforming (efficiently) each circuit in the family to be tree-like (a tree-like circuit is just a Boolean formula). To this end, we traverse the circuit from its output towards the inputs, repeating sub-circuits that have been used multiple times. But as the depth of the circuit is bounded by $O(\log n)$, and each gate had at most 2 inputs, the total size of the resulting formula will be bounded by $2^{O(\log n)} = n^{O(1)}$.

We now show by induction that any Boolean formula having no more than n logical connectives (from among $\{\wedge, \vee, \neg\}$) can be represented by a (fan-in 2) circuit of depth not greater than $4\log(n+1)$.

For $n = 0$ (a single variable) it is obviously true.

Fix any Boolean function ϕ with n connectives. The tree representing this formula has n internal nodes. Denote by $s(T)$ the size (number of nodes) of subtree T. Consider the subtree T of smallest size larger than $n/3$. It must be that $n/3 < s(T) \leq 2n/3$ (otherwise one of the subtrees of T would be smaller, while still larger than $n/3$).

Let $\phi = (T \wedge \phi(1/T)) \vee (\neg T \wedge \phi(0/T))$. The formula T has size less than $2n/3$. When we replace T in ϕ by 0 or 1, we will get a formula of size bounded by $2n/3$ (because the size of T was at least $n/3$). Both these formulas can be represented (according to inductive assumption) by circuits of depth bounded by

$$4\log(2n/3+1) \leq 4(\log(2/3)+\log(n+1)) \leq 4\log(n+1) - 2.$$

Thus, we can represent our original formula ϕ by a circuit of depth bounded by

$$4\log(n+1) - 2 + 2 = 4\log(n+1).$$

Our circuit has logarithmic depth and constant fan-in, so its size will be polynomial. □

6.2 Resolution

A *literal* is a variable x, or its negation, $\bar{x}$. A *clause* is a set of literals, $\{l_1, l_2, \ldots, l_k\}$. A truth assignment τ *satisfies* a clause C, written $\tau \vDash C$, if it makes at least one literal in C true. A (finite) set of clauses $\mathcal{S}$ is *satisfiable* if there exists a truth assignment τ that satisfies all the clauses in $\mathcal{S}$.

For example, $\mathcal{S} = \{\{x, \bar{y}, \bar{z}\}, \{\bar{x}\}, \{y, z\}\}$ is satisfiable, and the truth assignment τ given by $\tau(x) = \tau(y) = 0, \tau(z) = 1$ satisfies it. On the other hand, $\mathcal{S} = \{\{x\}, \{\bar{x}, y\}, \{\bar{y}, z\}, \{\bar{z}\}\}$ is unsatisfiable.

The *Pigeonhole Principle*, PHP^n_{n-1}, $n > 1$, is the set of clauses given by:

1. $\{P_{i1}, P_{i2}, \ldots, P_{i(n-1))}\}$, $1 \leq i \leq n$
2. $\{\bar{P}_{ik}, \bar{P}_{jk}\}$, $1 \leq i < j \leq n, 1 \leq k \leq (n-1)$

Note that PHP^n_{n-1} is a set of $n + \binom{n}{2} \cdot (n-1) = O(n^3)$ clauses, and PHP^n_{n-1} is unsatisfiable for every n, because if it were satisfiable, a τ that satisfies it would effectively give us an injective relation on $[n] \times [n-1]$, which is not possible.

Resolution is a propositional proof system for the language of unsatisfiable set of clauses. Given two clauses $C \cup \{x\}, D \cup \{\bar{x}\}$, the *resolution rule*

permits us to conclude $C \cup D$ from them. Here $C \cup \{x\}$ indicates that there are no x's in C; however, it may be the case that $\bar{x} \in C$ (and $D \cup \{\bar{x}\}$ indicates that $\bar{x}$ is not in D, but x may be present in D). Note that the resolution rule is *sound*: if τ satisfies $C \cup \{x\}, D \cup \{\bar{x}\}$, then τ must also satisfy $C \cup D$.

A *resolution refutation* (RR) of an unsatisfiable set of clauses $\mathcal{S}$ is a sequence of clauses $C_1, C_2, \ldots, C_n$, where:

1. each C_i is either in $\mathcal{S}$, or follows by the resolution rule from some C_j, C_k, $j, k < i$,

2. $C_n = \{\}$, i.e., it is the empty clause.

The *size* of a RR is the number of clauses in it.

For example, consider the pigeonhole principle for $n = 2$, that is,

$$\mathrm{PHP}_1^2 = \{\{P_{11}\}, \{P_{21}\}, \{\bar{P}_{11}, \bar{P}_{21}\}\}.$$

Its RR is:

$$\{P_{11}\}, \{P_{21}\}, \{\bar{P}_{11}, \bar{P}_{21}\}, \{\bar{P}_{21}\}, \{\},$$

and its size is 5.

Resolution is both *sound* and *complete*. To see that it is sound, note that the resolution rule is sound, so it can be shown inductively that if there is a τ that satisfies $\mathcal{S}$, then τ must also satisfy every C_i in the refutation, and in particular $C_n = \{\}$, which is not possible (because to satisfy a clause, τ must make at least one literal in the clause true, and the empty clause $\{\}$ has no literals). So if we can derive $\{\}$ from $\mathcal{S}$, we know that $\mathcal{S}$ cannot be satisfiable[1].

To show completeness, we use the Davis-Putnam algorithm. The first version, known as DPLL, gives us *tree-like* refutations; the second version, known as DP, gives us *dag-like* refutations.

A tree-like refutation is one where each clause in the proof, except for the original clauses, is used at most once as a premise of a rule. A dag-like refutation can be transformed into a tree-like one by re-deriving a clause each time it is used as a premise; of course, this may give rise to an exponential blow up in size.

[1] On the other hand, the convention is to make an empty set of clauses satisfiable, in fact by any assignment, because no assignment can falsify a clause in an empty set of clauses.

6.2.1 DPLL and DP

The most prominent PPS for UNSAT is DPLL (Davis-Putnam-Logemann-Loveland).

Algorithm 6.2.1 (DPLL)
On input $\mathcal{S}$ (a set of clauses):
We pick an $x \in \text{var}(\mathcal{S})$, and branch out on $x = 0$ and $x = 1$. On each branch we continue selecting new variables, until one of two things happen:

1. a leaf corresponds to a (partial) truth assignment falsifying some clause, in which case we label it with such a clause, or
2. on the path from that leaf to the root we examined all the variables, in which case we get a (full) truth assignment satisfying the original set of clauses.

If at the end each leaf is labeled with a clause, we know $\mathcal{S}$ is unsatisfiable.

Lemma 6.2.2 *DPLL is correct, that is, given any set of clauses $\mathcal{S}$, it outputs a refutation if $\mathcal{S}$ is unsatisfiable, and a truth assignment if it is satisfiable.*

PROOF: Observe that $\mathcal{S}$ is unsatisfiable iff both $\mathcal{S}|_{x=0}$ and $\mathcal{S}|_{x=1}$ are unsatisfiable, for any $x \in \text{Var}(\mathcal{S})$. $\mathcal{S}|_{x=t}$ denotes $\mathcal{S}$ where each instance of x has been replaced by the truth value $t \in \{0, 1\}$, and the result simplified. The simplification is obvious: if a literal l is made true by setting $x = t$, any clause containing it is eliminated altogether; if it is made false, the literal is eliminated from all the clauses containing it. So, an inductive proof on the number of variables in $\mathcal{S}$, shows the correctness of DPLL. □

We can now prove the completeness of Resolution.

Lemma 6.2.3 *A DPLL refutation yields a RR of the same size (in terms of the number of nodes).*

PROOF: We can transform a DPLL refutation into a RR. We imagine the tree "upside-down", with the leaves at top, and the root at the bottom, and the leaves are already labeled with clauses (each leaf is labelled my some clause falsified by the partial truth assignment implicit in the branch ending at the leaf—if several clauses are falsified by this truth assignment, we pick any one of them).

If an internal node is labeled with variable x, the idea is to now label it with the clause resulting from resolving on its two parents nodes (to which

we already, inductively, assigned clauses) on x. The resulting refutation is tree-like.

More formally, suppose that n is a node with parents n_1, n_2 above it that have clauses C_{n_1}, C_{n_2}, respectively, attached to them. If n was labeled with variable x (in the DPLL tree), it is now labeled with C_n, which is obtained as follows: (i) if $x \in C_{n_1}, \bar{x} \in C_{n_2}$, then C_n is the result of resolving C_{n_1}, C_{n_2} on x. (ii) If neither contains x or $\bar{x}$, let C_n be C_{n_1} (we could have made it C_{n_2}; the point is, it does not matter). It is not possible for both to contain x or for both to contain $\bar{x}$, as the next invariant shows.

The clause C_n is falsified by the (partial) truth assignment given by the path from the root to node n. Thus, if C_{n_1}, C_{n_2} both had x, or both had $\bar{x}$, then one of them would be satisfied by the partial truth assignment. From the construction given in the above paragraph it is easy to show this invariant by induction. □

For example, consider the set of clauses $\{\{x\}, \{\bar{x}, y\}, \{\bar{y}, z\}, \{\bar{z}\}\}$. In figure 6.2.1 we show a DPLL to resolution transformation; in brackets we have the corresponding resolution clauses.

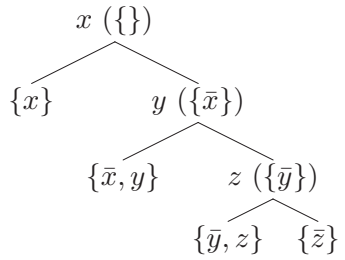

Figure 6.1: From DPLL to resolution.

So tree-like completeness follows from the correctness of DPLL. We can also go efficiently in the reverse direction; from tree-like Resolution to DPLL. However, there is one technical difficulty: a variable in a RR might be resolved on more than once on the same path!

We say that a RR is *regular* if on every path from the leaves to the root ($\{\}$) each variable is resolved on at most once.

Theorem 6.2.4 *A tree-like RR can be transformed into a regular tree-like RR efficiently.*

PROOF: Suppose that we have a path in the refutation labeled by clauses $C_1, C_2, \ldots, C_k$, $k > 2$, and there is a literal l in C_1 and in C_k, but l is not in any C_j, $1 < j < k$.

(Note that C_i is a premise for C_{i+1}; that's what being a path means here; also C_1 is *not necessarily* a leaf and C_k is *necessarily not* $\{\}$!)

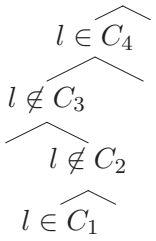

Figure 6.2: Transforming tree-like Resolution to regular tree-like Resolution

Simply discard the first resolution on l. We now have to modify all the C_i's all the way down to $\{\}$. Here is how to do this. In Figure 6.2 discard the right-parent-subtree of C_2, and let $C_2' := C_1$. Since we discarded the right-parent-subtree of C_2, some literals will no longer be in the C_i''s as we make our way to $\{\}$.

Say l_D is such a literal, and in the original refutation we resolve on some C_i ($i > 2$) and some E to get rid of it. Since l_D is no longer there, we simply discard E, and the subtree rooted at E, and let $C_i' := C_{i-1}'$. We follow through all the way to $\{\}$, making such adjustments.

Claim 6.2.5 *For all $2 \leq i \leq k$, $C_i' \subseteq C_i \cup \{l\}$, and for $i > k$, $C_i' \subseteq C_i$.*

This finishes the proof. □

Corollary 6.2.6 *A minimal tree-like RR is regular.*

Algorithm 6.2.7 (DP)
On input $\mathcal{S}$:

1. While ($\mathcal{S} \neq \emptyset$ and $\{\} \notin \mathcal{S}$)
2. Choose any $x \in \mathrm{Var}(\mathcal{S})$
3. $\mathcal{S} \leftarrow \mathcal{S} - \{C \in \mathcal{S} : \{x, \bar{x}\} \subseteq C\}$
4. $T \leftarrow \{C \cup D : C \cup \{x\}, D \cup \{\bar{x}\} \in \mathcal{S}\}$
5. $\mathcal{S} \leftarrow T \cup \{C \in \mathcal{S} : \{x, \bar{x}\} \cap C = \emptyset\}$
6. If $\mathcal{S} = \emptyset$ then return "satisfiable"
7. else return "unsatisfiable"

Notice that the DP algorithm produces a dag-like RR, unlike DPLL which produces a tree-like RR.

Claim 6.2.8 *DP is correct.*

PROOF: If the input $\mathcal{S}$ is an unsatisfiable set of clauses, then the set of clauses resulting from each iteration remains unsatisfiable (this is the *loop invariant*).

Let $\mathcal{S}'$ be the result of one iteration on $\mathcal{S}$. Suppose $\sigma \models \mathcal{S}'$. Then, one of the following two holds:

1. $\sigma \models C$ for all C such that $C \cup \{x\} \in \mathcal{S}$, or
2. $\sigma \models D$ for all D such that $D \cup \{\bar{x}\} \in \mathcal{S}$

Otherwise, $\sigma \not\models C_0$ and $\sigma \not\models D_0$, so $\sigma \not\models C_0 \cup D_0$, contradicting that $\sigma \models \mathcal{S}'$.

Therefore, if 1., extend σ to $\hat{\sigma}$ such that $\hat{\sigma}(x) = 0$, and if 2., $\hat{\sigma}(x) = 1$. Clearly, $\hat{\sigma} \models \mathcal{S}$; we continue inductively, and show that the original set of clauses $\mathcal{S}$ must be satisfiable. □

The following result indicates that resolution is a general PPS; that is, it can be applied to any Boolean tautology (i.e., not only those in CNF).

Lemma 6.2.9 *A general Boolean formula ϕ (not necessarily in CNF), can be transformed in polynomial time into a formula in CNF ϕ' such that, ϕ is satisfiable iff ϕ' is satisfiable.*

PROOF: Introduce a new variable q_A for any subformula A of ϕ. Let $\mathcal{C}(A)$ be a set of clauses with the property that $\mathcal{C}(A) \cup \{q_A\}$ is satisfiable iff A is satisfiable, and, furthermore, $\mathcal{C}(A)$ holds iff the truth values associated with the variables representing all the subformulas of A are computed correctly in terms of the input values. We define $\mathcal{C}(A)$ by structural induction on A: if $A = x$, then $\mathcal{C}(A) := \{\{q_A, \bar{x}\}, \{\bar{q}_A, x\}\}$.

If $A = \neg B$, then $\mathcal{C}(A) := \{\{q_A, q_B\}, \{\bar{q}_A, \bar{q}_B\}\} \cup \mathcal{C}(B)$.

If $A = B_1 \vee B_2$, then $\mathcal{C}(A) := \{\{\bar{q}_A, q_{B_1}, q_{B_2}\}, \{q_A, \bar{q}_{B_1}\}, \{q_A, \bar{q}_{B_2}\}\} \cup \mathcal{C}(B_1) \cup \mathcal{C}(B_2)$. Similarly for conjunction. Also need to show inductively that the correctness condition holds. Finally, note that $|\mathcal{C}(A)| = O(|A|)$. □

Suppose that we managed to perform the above transformation *without* introducing any new variables; what would be the consequence of that for propositional proof systems? If we could translate an arbitrary Boolean formula into an equivalent CNF *without* adding new variables and with at most a polynomial increase in size, we would effectively have a polybounded PPS (and consequently NP = co-NP). The reason is that a formula in CNF is a tautology iff each clause contains some variable x and its negation; this can be checked in linear time in the length of the formula.

Extended Resolution (ER) is define as the usual Resolution PPS, together with the extension rule which allows us to abbreviate formulas by

new variable names. That is, we allow the introduction of new variables v_{new}, and a definition $v_{\text{new}} \equiv \alpha$; now we can refer to α by the new name v_{new}.

A clausal form for $v_{\text{new}} \equiv \alpha$ can be obtained by translating the formula to the equivalent formula $(\neg v_{\text{new}} \vee \alpha) \wedge (v_{\text{new}} \vee \neg\alpha)$.

Theorem 6.2.10 (Cook) *ER proves PHP_n^{n+1} in polynomial size.*

PROOF: This result was shown in [Coo76]. The idea is to argue by contradiction: assume that PHP_n^{n+1} holds, and then deduce PHP_{n-1}^n from it. Repeating the process arrive at PHP_1^2, which is easy to refute. Each step of the induction requires polynomially many formulas, so the whole refutation is polynomial in size.

Introduce variables, $B_{i,j}^n$ and let $B_{i,j}^n \equiv P_{i,j}$ for $i = 1, \ldots, n+1$ and $j = 1, \ldots, n$. Then, for $k = n-1, \ldots, 1$, introduce variables $B_{i,j}^k$, defined as follows

$$B_{i,j}^k \equiv B_{i,j}^{k+1} \vee (B_{i,k+1}^{k+1} \wedge B_{k+2,j}^{k+1})$$

for $1 \leq i \leq k+1, 1 \leq j \leq k$. From $\{\{B_{i,1}^{k+1}, \ldots, B_{i,k+1}^{k+1}\}, \{\neg B_{i,l}^{k+1}, \neg B_{j,l}^{k+1}\}\}$ deduce $\{\{B_{i,1}^k, \ldots, B_{i,k}^k\}, \{\neg B_{i,l}^k, \neg B_{j,l}^k\}\}$.

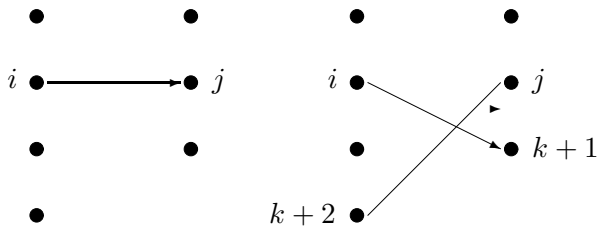

Figure 6.3: Definition of $B_{i,j}^k$.

Refute the resulting set of clauses $\{\{B_{1,1}^1\}, \{B_{2,1}^1\}, \{\neg B_{1,1}^1, \neg B_{2,1}^1\}\}$. □

Lemma 6.2.11 *Let 2SAT be the language of satisfiable formulas in CNF, where each clause has exactly 2 literals. Then, 2SAT in in* P.

PROOF: When we resolve two clauses with at most 2 literals each, we obtain a clause with at most two literals: therefore, there are $\binom{2n}{2} = O(n^2)$ possible clauses to be obtained by resolving on the initial clause. Thus, we can decide 2SAT with great alacrity by computing all the possible resolvents (stopping to accept when no new clauses are being created), and rejecting if we ever produce the empty clause. □

6.3 A lower bound for resolution

This section consists in the proof of the following theorem.

Theorem 6.3.1 (Haken) *Any RR of* PHP^n_{n-1} *requires at least* $2^{\frac{n}{20}}$ *clauses.*

A truth assignment (ta) σ is *i-critical* if σ leaves pigeon i out, and maps the remaining $(n-1)$ pigeons to holes bijectively. See figure 6.4 for an example.

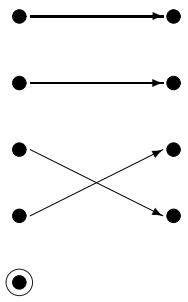

Figure 6.4: A 5-critical truth assignment.

Two clauses C_1, C_2 are *equivalent wrt critical ta's* if $\sigma \vDash C_1 \iff \sigma \vDash C_2$, for all critical σ. An inference $C_1, C_2 \vdash C_3$ is *sound wrt critical ta's* if whenever a critical σ satisfies C_1, C_2, it also satisfies C_3. Note that this "inference" is not necessarily an application of the resolution rule; C_1, C_2, C_3 are any three clauses with the property that if a critical ta satisfies C_1, C_2 it also satisfies C_3.

Consider a resolution refutation $\mathcal{R}$ of PHP^n_{n-1}, and in every clause of $\mathcal{R}$ replace $\bar{P}_{ik}$ by the literals $\{P_{lk} | l \neq i\}$, obtaining thus $\hat{\mathcal{R}}$. All clauses in $\hat{\mathcal{R}}$ are *monotone*, equivalent to the corresponding original clauses wrt critical ta's, and all "inferences" in $\hat{\mathcal{R}}$ are sound wrt critical ta's. *But $\hat{\mathcal{R}}$ is not a resolution refutation per se* !

Define a clause to be *large* if it contains at least $n^2/10$-many variables, and let $S =$ *the number of large clauses in* $\hat{\mathcal{R}}$.

Claim 6.3.2 *There is a P_{ij} contained in at least $S/10$-many large clauses of $\hat{\mathcal{R}}$.*

Proof: There are $< n^2$ variables, and suppose that each of them is in less than $S/10$ many large clauses. Now what is the largest collection of clauses

we can form where each clause is required to be large and we have $< S/10$ "copies" of each of the n^2 variables?

$$< \frac{n^2 \cdot S/10}{n^2/10} = S.$$

Contradiction, since we *do* have S large clauses by assumption. □

We are now going to derive a contradiction from $S < 2^{n/20}$. Choose such a P_{ij}, set it to 1, and set to 0 all P_{il} and $P_{l'j}$, where $l \neq j$ and $l' \neq i$. Apply this restriction to $\hat{\mathcal{R}}$, to obtain a monotone refutation $\hat{\mathcal{R}}'$ of PHP^{n-1}_{n-2}. (Note the commutative diagram in figure 6.3.)

$$\begin{array}{ccc} \mathcal{R} \text{ for } \mathrm{PHP}^n_{n-1} & \xrightarrow{\bar{P}_{ik} \rightsquigarrow \{P_{lk} | l \neq i\}} & \hat{\mathcal{R}} \text{ for } \mathrm{PHP}^n_{n-1} \\ \downarrow & & \downarrow {\scriptstyle P_{ij}=1, \forall l \neq j, l' \neq i, P_{il}=0, P_{l'j}=0} \\ \mathcal{R}' \text{ for } \mathrm{PHP}^{n-1}_{n-2} & \longrightarrow & \hat{\mathcal{R}}' \text{ for } \mathrm{PHP}^{n-1}_{n-2} \end{array}$$

Figure 6.5: Commutative diagram.

The number of large clauses in $\hat{\mathcal{R}}'$ for PHP^{n-1}_{n-2} is at most $9S/10$. Repeat this $\log_{10/9} S$ many times until we knock out all large clauses. So we end up with a monotone RR of $\mathrm{PHP}^{n'}_{n'-1}$ where

$$n' \geq n - \log_{10/9} S > n(1 - (\log_{10/9} 2)/20) > 0.671n, \tag{6.1}$$

and where there are no large (i.e., of size $\geq n^2/10$) clauses[2].

Lemma 6.3.3 *Any $\hat{\mathcal{R}}$ for PHP^n_{n-1} must have a clause with at least $2n^2/9$ literals (which by definition are all positive variables!).*

From lemma 6.3.3 we get a contradiction, since by (6.1) we have that $n' > 0.671n$, and so $2(n')^2/9 > n^2/10$.

PROOF:[of lemma 6.3.3] For each C in $\hat{\mathcal{R}}$, let $\mathrm{Complex}(C)$ be the minimum number of clauses in PHP^n_{n-1} that imply C on all critical ta's. Note that only "pigeon" clauses $\{P_{i1}, P_{i2}, \ldots, P_{i(n-1)}\}$ are included in a minimal set

[2] Actually, after $\log_{10/9} S$ many times you may still have one more large clause left, so really $n' > 0.671n - 1$. But the argument can still be made to work by taking n sufficiently large. More precisely, we are interested in $S \cdot (9/10)^i = 1$, so taking logarithms of both sides we obtain that $i = \frac{\log S}{\log(10/9)} = \log_{10/9} S$. Now, $1 - \frac{\log_{10/9}(2)}{20} = 0.6710578$. If we take this constant, we see that for n sufficiently large (say more than a million), it is clear that $0.6710578n - 1 > 0.671n$.

implying C (as we restrict ourselves to critical ta's and clauses $\{\bar{P}_{ik}, \bar{P}_{jk}\}$ are always satisfied by critical ta's). The complexity of "pigeon" clauses is 1, and of the empty clause $\{\}$ it is n.

If $C_1, C_2 \vdash C_3$, then $\text{Complex}(C_3) \leq \text{Complex}(C_1) + \text{Complex}(C_2)$.

Claim 6.3.4 *$\exists C$, $n/3 < Complex(C) \leq 2n/3$.*

PROOF: Take C to be a clause of complexity greater than $n/3$. Since $\text{Complex}(\{\}) = n$, such a clause exists. If both parent clauses are of complexity at most $n/3$, we are done. Otherwise, pick the parent clause of complexity greater than $n/3$, and repeat. This process must end, since the input clauses are of complexity 0 or 1. □

Let P be a minimal subset of "pigeon" clauses that implies C, and let $m = |P| = \text{Complex}(C)$.

Claim 6.3.5 *$|C| \geq (n-m)m$.*

Note that $(n-m)m \geq 2n^2/9$, since for $0 < n/3 \leq m \leq 2n/3$, as a function of m, it takes its minimum when at the extremes, i.e., when $m = n/3$ or $m = 2n/3$.

PROOF:[of claim 6.3.5] For any $\{P_{i1}, \ldots, P_{i(n-1)}\} \in P$, consider an i-critical α such that $\alpha \not\models C$. (If every i-critical α satisfied C, we would not need the clause $\{P_{i1}, \ldots, P_{i(n-1)}\}$ in P.)

For each $\{P_{j1}, \ldots, P_{j(n-1)}\} \notin P$ (remember $\text{Complex}(C) \leq 2n/3$), let α' be α except $\alpha'(i) = \alpha(j)$, and α' is j-critical. We have that $\alpha' \models C$: since $\{P_{j1}, \ldots, P_{j(n-1)}\} \notin P$, and α' is j-critical, α' satisfies P, so it must satisfy C. But α and α' are the same on all variables, except on P_{jl} and P_{il}. Since $\alpha \not\models C$ but $\alpha' \models C$, it follows that $P_{il} \in C$.

Using the same α on all $j \notin P$ we get $(n-m)$ *distinct* variables

$$P_{il_1}, P_{il_2}, \ldots, P_{il_{n-m}}$$

in C. Repeating the whole argument for each $i \in P$, gives us $(n-m)m$ variables in C. □ □

6.4 Automatizability and interpolation

If C is a clause, let the *width* of C, $w(C)$ denote the number of literals in C. Extend this definition to a set of clauses $\mathcal{S}$ in the obvious way: $w(\mathcal{S})$ is the largest width of all the clauses in $\mathcal{S}$. Also, if P is a resolution refutation, let $w(P)$ be the largest width of all the clauses in P. Finally, let $sw(\mathcal{S})$ be the smallest width of any resolution refutation of $\mathcal{S}$.

Lemma 6.4.1 *Any tree-like resolution refutation of $\mathcal{S}$ of size s can be converted to one of width bounded above by $(\lceil \log s \rceil + w(\mathcal{S}))$.*

PROOF: We prove it by induction on s. If $s = 1$, $\mathcal{S} = \{\{\}\}$, so $\mathcal{S}$ is itself a resolution refutation of width $0 = \lceil \log 1 \rceil + w(\mathcal{S})$.

Now consider a tree-like resolution refutation P of $\mathcal{S}$ of size $s > 1$. Suppose that x is the last variable to be resolved, i.e., the last step of P is

$$\dfrac{\dfrac{\vdots}{\{x\}} \qquad \dfrac{\vdots}{\{\bar{x}\}}}{\{\}}. \tag{6.2}$$

The dots denote the left and right subtrees, respectively. We want to show that we can transform P to have width $w = \lceil \log(s) \rceil + w(\mathcal{S})$.

Note that one of the subtrees of (6.2) has to be of size $< s/2$. Assume that the subtree rooted at $\{\bar{x}\}$ is of size $< s/2$, and the other subtree, the one rooted at $\{x\}$ is of size $< s$. (The opposite case is symmetric.)

Let $\mathcal{S}|_{x=t}$ be the set of clauses $\mathcal{S}$, with x set to $t \in \{0, 1\}$, and the natural simplifications done (if the assignment makes a literal false, eliminate it from all the clauses that have it, and if it makes a literal true, eliminate all the clauses that have it).

Thus, $\mathcal{S}|_{x=1}$ has the refutation $P|_{x=1}$ given by the right subtree of (6.2). By the induction hypothesis, $P|_{x=1}$ can be transformed to P' of width

$$\lceil \log(s/2) \rceil + w(\mathcal{S}|_{x=1}) \leq \overbrace{\underbrace{\lceil \log(s) \rceil + w(\mathcal{S})}_{=w} - 1}^{=(*)}. \tag{6.3}$$

Now transform P' as follows: to every input clause that originally contained $\bar{x}$, add $\bar{x}$ back in and propagate it down the proof. This way, we obtain a derivation of $\{\bar{x}\}$ of width at most $1 + (*) = w$, from a subset of the original clauses (note that the clauses that contained x were eliminated, and clauses that contain neither $\bar{x}$ nor x have not changed). How do we know that adding $\bar{x}$ to those input clauses ensures that $\bar{x}$ appears in the conclusion? Since $\bar{x}$ did appear in the conclusion of the original refutation P, there must have been a path from the conclusion of P (i.e., from $\{\bar{x}\}$) to an input clause containing $\bar{x}$; this path remains in P'.

Using the left subtree of (6.2), we obtain a refutation P'' of $\{\}$ from $\mathcal{S}|_{x=0}$ of width at most w (by induction hypothesis; recall that the left subtree of (6.2) has size $< s$). But we can obtain $\mathcal{S}|_{x=0}$ from $\mathcal{S}$ using P' to eliminate all the x's from the clauses of $\mathcal{S}$ which have x, and simply ignoring the clauses that have $\bar{x}$. This way, we get a refutation of $\mathcal{S}$ of width w. □

Corollary 6.4.2 *Any tree-like resolution refutation of $\mathcal{S}$ requires size*

$$\Omega(2^{sw(\mathcal{S})-w(\mathcal{S})}).$$

PROOF: Follows directly from $sw(\mathcal{S}) \leq \lceil \log(s) \rceil + w(\mathcal{S})$. □

Let $f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ be a function. We say that a propositional refutation system V is $f(n,s)$-*automatizable* iff there exists an algorithm A_V which on input $\mathcal{S}$, $|\mathcal{S}| = n$, outputs a refutation P of $\mathcal{S}$ in time at most $f(n,s)$ where s is the size of the shortest refutation of $\mathcal{S}$. (Note that we use $\mathcal{S}$, since we are thinking of clauses, but this definition is more general.)

Lemma 6.4.3 *For k-CNF formulas, tree resolution is $s^{O(\log n)}$-automatizable.*

Exercise 6.4.4 *Prove lemma 6.4.3*

Let $\alpha(\vec{p},\vec{q}) \wedge \beta(\vec{p},\vec{r})$ be an unsatisfiable CNF formula. A *Craig interpolant* (just *interpolant* from now on), is a function C such that given a value assignment $\vec{p}_0$ to $\vec{p}$:

$$C(\vec{p}_0) = \begin{cases} 0 & \alpha(\vec{p}_0,\vec{q}) \quad \text{is unsatisfiable} \\ 1 & \beta(\vec{p}_0,\vec{r}) \quad \text{is unsatisfiable} \end{cases}.$$

Lemma 6.4.5 *If for every unsatisfiable formula $\alpha(\vec{p},\vec{q}) \wedge \beta(\vec{p},\vec{r})$ there exists a polytime interpolant (i.e., there is a polytime algorithm computing C), then* $\mathsf{NP} \cap \mathsf{co\text{-}NP} \subseteq \mathsf{P/poly}$[3].

PROOF: Let L be a language in $\mathsf{NP} \cap \mathsf{co\text{-}NP}$. For inputs of a given length n, let $\alpha(\vec{p},\vec{q})$ code the statement "$\vec{q}$ is a witness that $\vec{p}$ is in L", and let $\beta(\vec{p},\vec{r})$ code the statement "$\vec{r}$ is a witness that $\vec{p}$ is *not* in L". (Such formulas exist since $\mathsf{NP} \cap \mathsf{co\text{-}NP} = \Sigma_1^p \cap \Pi_1^p$.)

Since for any pair (α,β) we have a polytime interpolant C, we also have a polysize circuit family $S = \{S_i\}$, where S_i implements C on inputs of length i, and S decides L: on input w, $|w| = n$, compute $S_n(w)$, and accept iff $S_n(w) = 1$, i.e., iff $\alpha(w,\vec{q})$ is satisfiable. A different circuit interpolant exists for different input lengths; we do not know how to generate the interpolants, but we know they are of polynomial size. □

Let $f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ be a function. We say that a propositional refutation system V has $f(n,s)$-*interpolation* iff given $\alpha(\vec{p},\vec{q}) \wedge \beta(\vec{p},\vec{r})$ (of size n) with minimum refutation size s, there exists a circuit of size at most $f(n,s)$

[3]Note that the most prominent problem to be in $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ is factoring; see [Pap94, §10.3.].

computing the interpolant C for $\alpha(\vec{p},\vec{q}) \wedge \beta(\vec{p},\vec{r})$. We say that V has *feasible interpolation* if f is bounded by a polynomial (in n, s), and *monotone interpolation* if whenever $\vec{p}$ occur only positively in α (or, dually, only negatively in β) the circuit computing the interpolant is monotone (i.e., it has only `And` and `Or` gates).

Lemma 6.4.6 *If a polybounded refutation system V has feasible interpolation, then* $\mathsf{NP} \subseteq \mathsf{P/poly}$.

Exercise 6.4.7 *Prove lemma 6.4.6*

Recall that a *clique* of size m is a subset of m vertices, which is fully connected (i.e., there is an edge between every pair of vertices), and a *co-clique* of size m is a partition of the vertices of the graph into m sets, so that all edges are between sets, and no edges within any single set.

Theorem 6.4.8 (Razborov) *There exists an ε such that for sufficiently large n, and $m = \frac{n}{10}$, any monotone circuit which outputs a 1 on all m-cliques, and a 0 on all $(m-1)$-co-cliques, requires size 2^{n^ε}.*

Lemma 6.4.9 *If V has monotone feasible interpolation, then V is not polynomially bounded.*

PROOF: Consider the formula $\alpha(\vec{p},\vec{q}) \wedge \beta(\vec{p},\vec{r})$, where $\vec{p}$ encodes an undirected graph G over n vertices; let (p_{ij}) be the adjacency matrix of a graph, $i, j \in [n]$. We show how to construct $\alpha(\vec{p},\vec{q})$ which asserts that $\vec{p}$ has a clique of size m, and $\beta(\vec{p},\vec{r})$ which asserts that $\vec{p}$ has a co-clique of size $(m-1)$.

We use $\vec{q}$ to describe a clique of size m. Let q_{ij}, for $i \in [n], j \in [m]$ assert that vertex i is the j-th vertex of the clique. We can state that $\vec{p}$ has an m-clique with the following clauses:

1. For each $j \in [m]$ we have the clause $\{q_{1j}, \ldots, q_{nj}\}$ asserting that some vertex is the j-th vertex of the clique.

2. For each pair $j \neq j'$ in $[m] \times [m]$, and for each $i \in [n]$, we have the clause $\{\bar{q}_{ij}, \bar{q}_{ij'}\}$ asserting that no vertex is placed in the clique twice.

3. For each pair $j \neq j'$ in $[m] \times [m]$, and for each $1 \leq i < i' \leq n$, we have $\{p_{ii'}, \bar{q}_{ij}, \bar{q}_{i'j'}\}$, asserting that if two vertices are in the clique, they must be connected by an edge.

All these clauses taken together (i.e., their conjunction) make up α. Note that $\vec{p}$ occur in α only positively.

Now let r_{ij} state that vertex $i \in [n]$ is in the j-th group of the partition, $j \in [m-1]$. The following clauses assert that we have a $(m-1)$-co-clique:

1. For $i \in [n]$ we have clauses $\{r_{i1}, \ldots, r_{i(m-1)}\}$, asserting that every vertex belongs to some group,

2. For each pair $i \neq i'$ in $[n] \times [n]$ and $1 \leq j \leq (m-1)$ we have clauses $\{\bar{r}_{ij}, \bar{r}_{i'j}, \bar{p}_{ii'}\}$, asserting that any two vertices in the same group are not connected by an edge.

The conjunction of these clauses makes up β, and $\vec{p}$ occur in β only negatively.

A feasible monotone interpolant for $\{\alpha \wedge \beta\}_n$, where $m = \frac{n}{10}$, contradicts Razborov's theorem, unless V is not polynomially bounded. □

Lemma 6.4.10 *Resolution has monotone feasible interpolation.*

PROOF: Suppose that P is a resolution refutation of $\alpha(\vec{p}, \vec{q}) \cup \beta(\vec{p}, \vec{r})$, where the p_i's occur only positively in α (the dual case, only negatively in β, is analogous).

Let σ be a truth value assignment to $\vec{p}$. We show how to transform P to obtain a refutation $P|_\sigma$ and at the same time construct a feasible monotone interpolant $\mathbf{I}(\sigma)$. If C was a clause in the original refutation, it will be denoted C' after applying the procedure, and $\mathbf{I}_C(\sigma)$ will be the interpolant associated with clause C.

We say that a clause C' is an α-clause if it contains variables from among $\vec{p}, \vec{q}$ only, and if it only contains variables from $\vec{p}$, it is an α-clause if all its ancestors are α-clauses. Similarly, we define a β-clause symmetrically, with $\vec{r}$ and negations of $\vec{p}$.

We let $\mathbf{I}_C(\sigma)$ be 0 if C' is an α-clause, and 1 if it is a β-clause. Initially, every clause C in α and β stays the same, i.e., $C' = C$, all clauses in α are declared to be α-clauses, and all clauses in β are declared to be β-clauses. We now describe the rest of the procedure.

Suppose a clause C in P is obtained by:

$$\frac{C_1 \cup \{x\} \qquad C_2 \cup \{\bar{x}\}}{C}. \tag{6.4}$$

Assume inductively that we have already transformed the premises, and we have obtained $(C_1 \cup \{x\})', (C_2 \cup \{\bar{x}\})'$, and we have also declared their sides.

The task is to define C' and $\mathbf{I}_C(\sigma)$. We do it separately for x being a variable in one of the three groups: p_i, q_i, r_i.

In the case when $x = p_i$, let C' be $(C_1 \cup \{p_i\})'$ if $p_i = 0$, and $(C_2 \cup \{\bar{p}_i\})'$ otherwise. One might be tempted to define

$$\mathbf{I}_C := (p_i \vee \mathbf{I}_{(C_1 \cup \{p_i\})}) \wedge (\bar{p}_i \vee \mathbf{I}_{(C_2 \cup \{\bar{p}_i\})}). \tag{6.5}$$

But we have to be *"Unmoved, cold, and to temptation slow,"* (Shakespeare, Sonnet XCIV); (6.5) would work if we were not constructing a monotone circuit for the interpolant, but we are, so $\bar{p}_i$ is not allowed. To fix this, keep in mind that one of the goals of the definition of C' is to ensure that it is either an α-clause or a β-clause. Consider the truth table for $\mathbf{I}_C$, given in figure 6.6, when defined the wrong way as in (6.5).

p_i	$\mathbf{I}_{(C_1 \cup \{p_i\})}$	$\mathbf{I}_{(C_2 \cup \{\bar{p}_i\})}$	$\mathbf{I}_C$
0	0	0	0
1	0	0	0
0	1	0	$\boxed{1}$
1	1	0	0
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	1

Figure 6.6: Truth table for $\mathbf{I}_C$.

Note the two "problem rows," where I_C goes from 1 to 0, despite the fact that p_i changes from 0 to 1. This is the only case where monotonicity is spoiled. But, using the values of $\mathbf{I}$ on the premises, we can make this problem vanish. The point is that we can turn the 1 in the box into a 0, while defining C' consistently. (Note hat a "symmetric" answer is possible: turn the 0 below the boxed 1 into a 1.)

Let $\mathbf{I}_C := (p_i \vee \mathbf{I}_{(C_1 \cup \{p_i\})}) \wedge \mathbf{I}_{(C_2 \cup \{\bar{p}_i\})}$, and we let C' be $(C_1 \cup \{p_i\})'$ or $(C_2 \cup \{\bar{p}_i\})'$ as defined the wrong way, except that if $(C_2 \cup \{\bar{p}_i\})'$ is an α-clause, we let C' be equal to it, regardless of the value of p_i. This works, because if $(C_2 \cup \{\bar{p}_i\})'$ is an α-clause, then by definition it cannot have $\bar{p}_i$ in it (it must have disappeared along the way).

For $x = q_i$, let C' be the result of resolving $(C_1 \cup \{q_i\})'$ and $(C_2 \cup \{\bar{q}_i\})'$ on q_i if possible (meaning that one still has q_i and the other $\bar{q}_i$), and taking C' to be the β-clause, if one of them is (if they both are, say, the left one),

and if neither is a β-clause, take the one without the q_i-literal (say, the left one, if they both miss the q_i-literal). Let $\mathbf{I}_C := \mathbf{I}_{C_1 \cup \{q_i\}} \vee \mathbf{I}_{C_2 \cup \{\bar{q}_i\}}$. The case of $x = r_i$ is the dual case to the q_i, and so the interpolant is defined with a conjunction.

We now show that if C' is an α-clause, then $\alpha|_\sigma \vDash C'|_\sigma$, and same for β-clauses. This can be done with an inductive argument on the depth of the clause.

Basis Case: the claim is clearly true for the input clauses. Suppose that we are deriving as in (6.4), with $x = q_i$. Suppose that $\hat{\sigma} \supseteq \sigma$ is a truth assignment to all the variables, extending the truth assignment σ to the $\vec{p}$. Suppose both premises are α-clauses, and $\hat{\sigma}$ satisfies them. If C' is the result of resolution on q_i, then by soundness of the resolution rule it is satisfied by $\hat{\sigma}$.

Exercise 6.4.11 *Show what to do in the case when x is p_i and r_i.*

Define the interpolant for P to be $\mathbf{I}_{\{\}}$, i.e., the value of the interpolant at the final empty clause. It is monotone because we only use $\{\wedge, \vee, 0, 1\}$ and variables (no negations). It is feasible since the number of connectives is linear in the size of the proof P. It works correctly because it establishes that $\{\}' = \{\}$ is an α- or β-clause, and so $\{\}$ is logically implied by either α or β clauses, and so α or β must be unsatisfiable. □

Lemmas 6.4.9 and 6.4.10 give us the following corollary, which is an alternative proof of a lower bound for resolution. Note that our original lower bound, given in section 6.3, is much simpler; the lower bound just given requires the machinery of interpolation, plus Razborov's lower bound for monotone circuits computing CLIQUE.

Corollary 6.4.12 *Resolution is not polynomially bounded.*

6.5 Answers to selected exercises

Exercise 6.4.4. If $w(\mathcal{S}) = k$, i.e., it is constant, we know from lemma 6.4.1 that a tree-like resolution refutation of size s can be transformed to be of width $O(\log(s))$. Given input size n, there can be at most n-many variables, and hence we can form at most $2^{O(\log(s))}\binom{n}{O(\log(s))} = n^{O(\log(s))} = s^{O(\log(n))}$-many clauses of width at most $O(\log(s))$. Now, starting with $l = 1$, being careless with space, we use breadth-first search to generate all tree-like resolution refutations with leaves labeled by clauses in $\mathcal{S}$ and with clauses of

width at most l. When we can no longer generate new clauses, we increase l by 1.

Exercise 6.4.7. Suppose that V has feasible interpolation and it is also polybounded. Consider the following two formulas: $\alpha(\vec{p}, \vec{q})$ asserting "$\vec{q}$ is a satisfying truth assignment for the propositional formula encoded by $\vec{p}$", and $\beta(\vec{p}, \vec{r})$ asserting "$\vec{r}$ encodes a V-refutation of $\vec{p}$". Obviously $\alpha \wedge \beta$ is not satisfiable. A polysize interpolant for these formulas would give us a polysize circuit for satisfiability, and hence NP would be contained in P/poly.

6.6 Notes

Section 6.3 is based on [BP96]. Theorem 6.4.8 (i.e., Razborov's CLIQUE theorem) is presented in [Pap94, §14.4] (albeit, with a slightly different statement). Section 6.4 is based on the scribe notes of the lectures (lectures #6 and #7) of Toniann Pitassi, Propositional Proof Complexity (CS2429), Fall 2002, available on the web at

`http://www.cs.toronto.edu/~toni/Courses/Proofcomplexity/CS2429.html`