**Computability and Complexity 2022/23** Mar 23, 2023

# Lecture 18

*Lecturer: Jakob Nordström* *Scribe: Jakob Nordström*

## 1 Brief Introduction and Outline of Today's Lecture

For the purposes of this lecture we can define *propositional proof complexity* as the study of how to certify unsatisfiability of formulas in conjunctive normal form (CNF). In what follows, let us write Unsat to denote the set/language of (syntactically well-formed) CNF formulas that do not have any satisfying assignments. Unless otherwise specified, $F$ will denote a CNF formula, which is usually assumed to be unsatisfiable.

**Definition 1.1.** A *proof system* for Unsat is a deterministic algorithm $\mathcal{P}(F, \pi)$ that runs in time polynomial in the size $|F| + |\pi|$ of the input and is such that:

- if $F \in$ Unsat, then there exists a *proof*, or *refutation*, $\pi$ such that $\mathcal{P}(F, \pi) = 1$;
- if $F \notin$ Unsat, then for any purported proof $\pi$ it holds that $\mathcal{P}(F, \pi) = 0$.

**Definition 1.2.** A proof system $\mathcal{P}$ is *polynomially bounded* if there exists a polynomial $p$ such that for all $F \in$ Unsat there is a proof $\pi$ such that $|\pi| \leq p(|F|)$ and $\mathcal{P}(F, \pi) = 1$.

It is widely believed that no polynomially bounded proof systems for Unsat exist, because this would imply $\mathsf{NP} = \mathsf{coNP}$. The converse is also true in that if $\mathsf{NP} = \mathsf{coNP}$, then there exists a polynomially bounded proof system for Unsat (with a standard $\mathsf{NP}$ verifier acting as the proof checker $\mathcal{P}$). Since $\mathsf{P} = \mathsf{NP}$ implies $\mathsf{NP} = \mathsf{coNP}$, we can conclude the following theorem.

**Theorem 1.3 ([CR79]).** *If there are no polynomially bounded proof systems for Unsat, then* $\mathsf{P} \neq \mathsf{NP}$.

One conceivable approach to prove $\mathsf{P} \neq \mathsf{NP}$ is to prove superpolynomial lower bounds for Unsat for stronger and stronger proof systems until we reach a deep enough understanding that makes it possible to prove lower bounds for completely general proof systems for Unsat. This is known in the proof complexity community as *Cook's program.*[1]

Cook's program has not been very successful if measured in terms of progress towards its (very ambitious) final goal, but it has generated a lot of beautiful (and sometimes mysterious) mathematics. Today we will see an example of this in the form of an exponential lower bound on proof length for the *resolution* proof system. Resolution was introduced in [Bla37] and started being studied more in earnest in the context of satisfiablity algorithms in [DP60, DLL62, Rob65]. This proof system is arguably the most well-studied proof system in proof complexity and is (still) the basis of state-of-the-art satisfiability algorithm using a paradigm known as *conflict-driven clause learning (CDCL)* [BS97, MS99, MMZ$^+$01].

Our plan for today is as follows:

- Define the resolution proof system.
- Show how short resolution proofs can be turned into small circuits via the *interpolation* technique.
- Use the lower bound for monotone circuits computing clique that we showed in the last couple of lectures to obtain a lower bound on resolution proof size for a related family of CNF formulas.

We will only be able to scratch the surface of proof complexity. Some more reading on this topic can be found in the survey article [BN21] or the book [Kra19].

[1] Actually, Cook's program was not proposed by Steve Cook, at least not according to Steve Cook, but this name is well established. Probably the first to refer to this approach towards proving $\mathsf{P} \neq \mathsf{NP}$ was Peter Clote in his PhD thesis.

## 2 Resolution

The resolution proof system starts with the clauses of an unsatisfiable CNF formula $F$ and iteratively derives new clauses until an explicit contradiction is reached. The formal definition is as follows.

**Definition 2.1 (Resolution proof system).** A *resolution refutation* $\pi$ of an unsatisfiable CNF formula $F$, which we will often denote $\pi : F \vdash \bot$, is a sequence of clauses $\pi = (D_1, D_2, \ldots, D_{L-1}, D_L)$ such that $D_L$ is the empty clause containing no literals, denoted $\bot$, and each clause $D_i$ is either

(a) an *axiom clause* $D_i \in F$, or

(b) a clause on the form $D_i = B \vee C$ derived from clauses $D_j = B \vee x$ and $D_k = C \vee \overline{x}$ for $j, k < i$ by the *resolution rule*

$$\frac{B \vee x \qquad C \vee \overline{x}}{B \vee C}\,, \tag{2.1}$$

where we say that $B \vee C$ is the *resolvent over* $x$ of $B \vee x$ and $C \vee \overline{x}$.

We view clauses as sets of literals, so that there is no repetition and order is immaterial. Without loss of generality, we will also assume that all clauses we encounter in resolution derivations are nontrivial in that they do not contain both $x$ and $\overline{x}$ for any variable $x$. It is not hard to show that any trivial clauses can always just be removed from a resolution derivation.

An important property for any proof system for UNSAT is that it should be *sound* (i.e., should never be able to refute a formula that in fact is satisfiable) and *(refutationally) complete* (i.e., should be able to refute any unsatisfiable formula). Resolution is sound and refutationally complete.

**Lemma 2.2.** *A CNF formula $F$ is unsatisfiable if and only if there exists a resolution refutation of $F$.*

*Proof sketch.* ($\Leftarrow$) Suppose there exists a satisfying assignment $\alpha$ to $F$, i.e., such that for every axiom clause in $F$ the assignment $\alpha$ satisfies some literal in it. By induction over $\pi = (D_1, D_2, \ldots, D_{L-1}, D_L)$ we conclude that $\alpha$ must satisfy some literal in every resolvent (this follows by a simple case analysis for the resolution rule (2.1)). But this is a contradiction since there is no literal to be satisfied in the empty clause $\bot$ (this is why the empty clause is just another way of denoting contradiction).

($\Rightarrow$) This direction is not hard, but requires an argument. It is left as an exercise. □

We can think of a resolution refutation as a list of clauses annotated with explanations for each clause how it was obtained. This is illustrated in Figure 1a for the example CNF formula

$$F = (x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{z} \vee w) \wedge (\overline{z} \vee \overline{w})\,. \tag{2.2}$$

To every such refutation $\pi$ we can also associate a directed acyclic graph (DAG) $G_\pi$ in the following way. Sources of the DAG are axioms, and the unique sink is the empty clause $\bot$.[2] Every node that is not a source has indegree two, and is the resolvent of its two predecessors. See Figure 1b for the proof DAG corresponding to the refutation in Figure 1a.

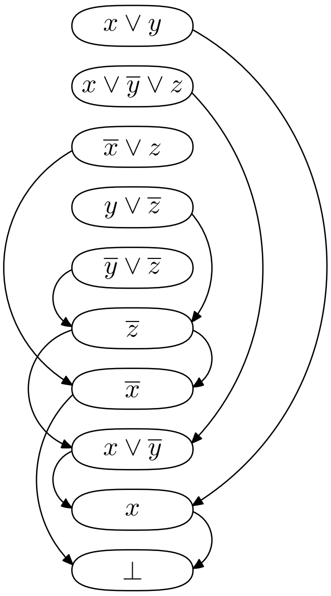
Given an unsatisfiable formula, the complexity measure we care most about is the *length* of refuting it in resolution. The length of a refutation $\pi$ is the number of clauses in it (so the length of the refutation in Figure 1a is 10), and is denoted $L(\pi)$. The length of refuting $F$ is the length of a shortest refutation of $F$, and is denoted $L_{\mathcal{R}}(F \vdash \bot)$. We will sometimes drop the subscript and write just $L(F \vdash \bot)$ if it is clear from context that we are looking at resolution refutations.

A more general measure that can be defined in any proof system is the *size* of a refutation, measured as the total number of symbols in it. When defining size in proof complexity, we tend to be somewhat relaxed, however, in that we do not care too much about a linear factor more or less—what we are really interested in is whether the size is polynomial or superpolynomial. Since every clause can have at most linear size, the length and size measures for resolution differ by at most a linear factor, and in fact in the

[2] Strictly speaking, the DAG could have several sinks, but if so this means that the resolution refutation contains redundant clauses that can be removed. Therefore, we can assume without loss of generality that the sink is unique.

| | | |
|---|---|---|
| 1. | $x \vee y$ | Axiom |
| 2. | $x \vee \overline{y} \vee z$ | Axiom |
| 3. | $\overline{x} \vee z$ | Axiom |
| 4. | $y \vee \overline{z}$ | Axiom |
| 5. | $\overline{y} \vee \overline{z}$ | Axiom |
| 6. | $\overline{z}$ | Res$(4, 5)$ |
| 7. | $\overline{x}$ | Res$(3, 6)$ |
| 8. | $x \vee \overline{y}$ | Res$(2, 6)$ |
| 9. | $x$ | Res$(1, 8)$ |
| 10. | $\bot$ | Res$(7, 9)$ |

**(a)** Resolution refutation as an annotated list.



**(b)** Resolution refutation as a DAG.

**Figure 1:** Resolution refutation for the CNF formula in (2.2).

literature the size of a resolution refutation is usually defined to simply be the length. There are other proof systems where the distinction between length and size is more important, however. In addition to length/size, other complexity measures of interest for resolution are, for instance, *clause space*, which is the number of clauses needed in memory while carrying out a refutation, and *width*, which is the size of the largest clause in the refutation, but in this lecture we will only care about length.

## 3 Circuits

Just to make sure we are on the same page, let us recall what we mean by a circuit.

**Definition 3.1 (Circuit).** A *circuit* is a directed acyclic graph (DAG). It has $n$ *sources*, which are nodes labelled by variable inputs, and a unique *sink* without outgoing edges. All non-source vertices are labelled by one of a fixed set of Boolean functions, and are often referred to as *gates*. Typically we require the *fan-in*, which is just another name for the *in-degree* of a vertex, to be at most 2. The standard gates are $\wedge$ (AND) with fan-in 2, $\vee$ (OR) with fan-in 2, and $\neg$ (NOT) with fan-in 1. A circuit $C$ can be thought of as a Boolean function $f_C : \{0,1\}^n \to \{0,1\}$, where the source vertices are the input values, every non-source vertex computes the function it is labelled with on the values provided by its immediate predecessors, and the output of the function is the value computed at the sink. The *size* of a circuit is the total number of vertices in the DAG.

For $x, y \in \{0,1\}^n$ we write $x \leq y$ if for all $i \in [n]$ it holds that $x_i \leq y_i$. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $x \leq y$ implies that $f(x) \leq f(y)$. Equivalently, a function $f$ is monotone if flipping an input bit from 0 to 1 can never flip $f$ from 1 to 0.

**Fact 3.2.** *A Boolean function $f : \{0,1\}^n \to \{0,1\}^n$ can be computed by a monotone circuit if and only if it is monotone.*

For a family of functions $\{f_n : \{0,1\}^n \to \{0,1\}\}_{n=1}^{\infty}$ we can study the sizes of the smallest circuits computing these functions. This field of research is known as *circuit complexity* and is another line of attack for proving P $\neq$ NP (by trying to show something stronger, namely that NP cannot be decided by polynomial-size circuits, or in computational complexity notation that NP $\subsetneq$ P/poly). Just as proof complexity, this approach has not been terribly successful at reaching its ultimate goal, but there have been many results for restricted subclasses of circuits such as monotone circuits.

## 4 Interpolation and Clique-Colouring Formulas

There are several different techniques for proving lower bounds in proof complexity. Today we want to talk about one of the most powerful ones, namely the *interpolation* method introduced by Krajíček [Kra94] and used by Pudlák [Pud97] to establish his celebrated lower bound for formulas talking about cliques in and colouring of graphs.

Pudlák focused on *cutting planes* [CCT87], a proof system that is exponentially stronger than resolution and much less well understood. In order to illustrate the interpolation method we will do a simpler version of his proof that applies to the resolution proof system. This will makes our lives a little bit easier, but will still illustrate the key ideas in this method. We now proceed to give a formal description of the clique-colouring formulas for which we want to prove lower bounds.

The *clique-colouring formulas* are unsatisfiable CNF formulas encoding the contradictory claim that there exist undirected graphs $G = (V, E)$ on $n = |V|$ vertices which have an $m$-clique but are also $(m-1)$-colourable. The encoding uses sets of Boolean variables $\mathbf{p} = \{p_{i,j} \mid 1 \leq i < j \leq n\}$, $\mathbf{q} = \{q_{k,i} \mid k \in [m],\ i \in [n]\}$, and $\mathbf{r} = \{r_{i,\ell} \mid i \in [n],\ \ell \in [m-1]\}$, which are intended to be interpreted as follows:

- $p_{i,j}$ indicates whether the edge $(i, j)$ is present in $G$ or not (where we enforce $i < j$ since the graph is undirected);
- $q_{k,i}$ indicates whether the vertex $i$ in $G$ is the $k$th member of the $m$-clique;
- $r_{i,\ell}$ indicates whether the vertex $i$ in $G$ has colour $\ell$.

The clique-colouring formula consists of the following clauses:

- for each $k \in [m]$, some vertex in $G$ is the $k$th member of the clique:

$$\bigvee_{i \in [n]} q_{k,i}\,, \tag{4.1a}$$

- for all $k, k' \in [m]$, $k \neq k'$, $i \in [n]$, clique vertices have unique member numbers:

$$\overline{q}_{k,i} \vee \overline{q}_{k',i}\,, \tag{4.1b}$$

- for all $k, k' \in [m]$, $k \neq k'$, $i, j \in [n]$, $i < j$, the clique members are connected by edges:

$$p_{i,j} \vee \overline{q}_{k,i} \vee \overline{q}_{k',j}\,, \tag{4.1c}$$

- for each $i \in [n]$, vertex $i$ gets assigned some colour:

$$\bigvee_{\ell \in [m]} r_{i,\ell}\,, \tag{4.1d}$$

- for all $\ell \in [m-1]$, $i, j \in [n]$, $i < j$, neighbouring vertices have distinct colours:

$$\overline{p}_{i,j} \vee \overline{r}_{i,\ell} \vee \overline{r}_{j,\ell}\,. \tag{4.1e}$$

We observe for later use that the clauses in the clique-colouring formula can be split into two parts sharing only the variables $\mathbf{p}$ encoding the edges of the graph. That is, it can be written as $A(\mathbf{p}, \mathbf{q}) \wedge B(\mathbf{p}, \mathbf{r})$ for $A(\mathbf{p}, \mathbf{q})$ being the conjunction of the clauses (4.1a)–(4.1c) and $B(\mathbf{p}, \mathbf{r})$ being the conjunction of the clauses (4.1d)–(4.1e), where the sets of variables $\mathbf{p}, \mathbf{q}, \mathbf{r}$ are disjoint.

Given a partial truth value assignment, or *restriction*, $\rho$ to the variables $\mathit{Vars}(F)$ of a CNF formula $F$, we write $F{\restriction}_\rho$ for the new formula obtained by assigning variable values according to $\rho$ and then simplifying $F$ by removing satisfied clauses and falsified literals. For example, for the formula $F = (x \lor y) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z})$ and restriction $\rho = \{z \mapsto 0\} = \{\overline{z}\}$ we obtain $F\!\restriction_\rho = (x \lor y) \land \overline{x}$.

Suppose we have an unsatisfiable CNF formula on the form $A(\mathbf{p}, \mathbf{q}) \land B(\mathbf{p}, \mathbf{r})$ for disjoint variable sets $\mathbf{p}, \mathbf{q}, \mathbf{r}$ as above (which could be a clique-colouring formula or some other formula). Notice that if we plug in some assignment $\rho$ to $\mathbf{p}$, we get the conjunction of two formulas $A(\mathbf{p}, \mathbf{q})\!\restriction_\rho = A'(\mathbf{q})$ and $B(\mathbf{p}, \mathbf{r})\!\restriction_\rho = B'(\mathbf{r})$ over disjoint sets of variables $\mathbf{q}$ and $\mathbf{r}$. Since the original formula was unsatisfiable, at least one of these restricted subformulas must be unsatisfiable.

We say that a Boolean circuit $I(\mathbf{p})$ is an *interpolant* for CNF formula $A(\mathbf{p}, \mathbf{q}) \land B(\mathbf{p}, \mathbf{r})$ with disjoint sets of variables $\mathbf{p}, \mathbf{q}, \mathbf{r}$ if for every assignment $\rho$ to the variables $\mathbf{p}$ it holds that $I(\rho) = 0$ implies that $A\!\restriction_\rho$ is unsatisfiable and $I(\rho) = 1$ implies that $B\!\restriction_\rho$ is unsatisfiable. In case both subformulas are unsatisfiable the interpolant is free to choose whichever subformula it likes best (but the function has to be well-defined, so it has to make a choice). Note that such an interpolant always exists by definition—we can define a function $I(\mathbf{p})$ that evaluates to $0$ whenever $A\!\restriction_\rho$ is unsatisfiable and takes the value $1$ otherwise, and this is a well-defined mathematical function that can be computed by some circuit—but the interpolating circuit might be quite large. We are interested in when the interpolant can be written as a small (i.e., polynomial-size) Boolean circuit. It turns out this is possible if the formula $A(\mathbf{p}, \mathbf{q}) \land B(\mathbf{p}, \mathbf{r})$ has a short resolution refutation! (This is not obvious, and indeed we will have to spend quite some time on proving this.) Flipping this implication around, in the other direction this means that *interpolants can be used to obtain proof complexity lower bounds from circuit complexity lower bounds.*

This suggests the following strategy for proving lower bounds on refutation length:

- Start with a formula $A(\mathbf{p}, \mathbf{q}) \land B(\mathbf{p}, \mathbf{r})$.
- Assume, towards contradiction, that the formula has a short resolution refutation.
- Deduce that there exist a small interpolating circuit.
- Appeal to a(n already known) circuit complexity lower bound saying that no such circuit can exist.
- Contradiction! Hence there cannot exist any short resolution refutation.

Proof systems for which this strategy works are said to have *feasible interpolation*. Resolution has feasible interpolation (as does cutting planes). We will use this fact to show that clique-colouring formulas are hard for resolution.

## 5 Statement of Clique-Colouring Formula Lower Bound

In order to establish lower bounds on the refutation length of clique-colouring formulas, we will need the following circuit complexity lower bound.

**Theorem 5.1 ([Raz85, AB87]).** *Let an undirected graph $G$ be represented by $\binom{n}{2}$ bits encoding its edges and non-edges. Then for $m = \Theta(\sqrt[4]{n})$ there is no* monotone *circuit of size $2^{o(\sqrt{m})}$ that can distinguish the following two cases:*

- *$G$ has an $m$-clique.*
- *$G$ is $(m-1)$-colourable.*

But what an interpolant $I(\mathbf{p})$ for the clique-colouring formula does is exactly to distinguish the two cases in Theorem 5.1. Since an interpolant determines which part of the formula is unsatisfiable for a given assignment to the variables $\mathbf{p}$ encoding the graph, it can separate the cases when $G$ has an $m$-clique and when it is $(m-1)$-colourable. What Theorem 5.1 says is that every monotone circuit computing such an interpolant must have size $\exp\bigl(\Omega\bigl(\sqrt[8]{n}\bigr)\bigr)$.

**Remark 5.2.** The monotonicity assumption in Theorem 5.1 is *very* important. We do not have such strong lower bounds for explicit functions for non-monotone circuits.

In what follows, we will focusing on constructing an interpolant $I(\mathbf{p})$ without caring too much about the (crucial) fact that we want it to be a monotone circuit. Let us define the ternary *selector* function sel by

$$\mathsf{sel}(x,y,z) = \begin{cases} y & \text{if } x = 0, \\ z & \text{if } x = 1. \end{cases} \tag{5.1}$$

We will build circuits with gates $\{\land, \lor, \mathsf{sel}\}$. It is not hard to see that sel can be implemented by a subcircuit over $\{\land, \lor, \neg\}$ of constant size, so using sel is just a convenient shorthand. A more serious concern is that sel is not a monotone function, but let us decide not to worry about this for now and take care of it at the end of the lecture.

We can now state the theorem that is the main goal of this lecture.

**Theorem 5.3 ([Pud97]).** *Suppose that $A(\mathbf{p},\mathbf{q}) \land B(\mathbf{p},\mathbf{r})$ is an unsatisfiable CNF formula over disjoint sets of variables $\mathbf{p}, \mathbf{q}, \mathbf{r}$, and that there is a resolution refutation $\pi : A \land B \vdash \bot$ in length $L$. Then the following holds:*

1. *There is an interpolating circuit $I(\mathbf{p})$ over gates $\{\land, \lor, \mathsf{sel}\}$ of size $\mathrm{O}(L)$;*
2. *From $\pi$ we can construct a resolution refutation*

   *(a) $\pi_A : A(\rho,\mathbf{q}) \vdash \bot$ if $I(\rho) = 0$, or*

   *(b) $\pi_B : B(\rho,\mathbf{r}) \vdash \bot$ if $I(\rho) = 1$,*

   *in both cases of length at most $L$;*
3. *If the $\mathbf{p}$-variables occur only positively in $A(\mathbf{p},\mathbf{q})$ or only negatively in $B(\mathbf{p},\mathbf{r})$, then sel-gates can be replaced by $\land$- and $\lor$-gates, yielding a monotone circuit of size $\mathrm{O}(L)$.*

Here is the plan for the proof:

- Fixing the $\mathbf{p}$-variables to take values as assigned by $\rho$, we will split the (restricted) clauses of $\pi$ into two derivations $\pi_A$ from $A(\rho,\mathbf{q})$ and $\pi_B$ from $B(\rho,\mathbf{r})$.
- At least one of $\pi_A$ and $\pi_B$ will be assigned the final empty clause and will thus be a resolution refutation of $A(\rho,\mathbf{q})$ or $B(\rho,\mathbf{r})$, respectively.
- We can build a circuit representing our choice of how to split the clauses in $\pi$ that figures out whether $\pi_A$ or $\pi_B$ gets assigned the final clause, and hence which of the formulas $A(\rho,\mathbf{q})$ and $B(\rho,\mathbf{r})$ is unsatisfiable.

# 6 Proof of Clique-Colouring Formula Lower Bound

Suppose that $A(\mathbf{p},\mathbf{q}) \land B(\mathbf{p},\mathbf{r})$ is an unsatisfiable CNF formula over disjoint sets of variables $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{r}$ (which is certainly the case for the clique-colouring formula). Let $\pi = (C_1, \ldots, C_L)$ be any resolution refutation of $A(\mathbf{p},\mathbf{q}) \land B(\mathbf{p},\mathbf{r})$ and let $\rho$ be any assignment to the $\mathbf{p}$-variables. Let us first prove part 2 of Theorem 5.3.

## 6.1 Extracting a Resolution Refutation of One of the Subformulas

We start by making a pair of key definitions. For a fixed restriction $\rho$ of the variables in $\mathbf{p}$, we define a $\mathbf{q}$-*clause* to be a clause $C$ over variables $\mathbf{q}$ that is in $A(\rho,\mathbf{q})$ or is derivable from $A(\rho,\mathbf{q})$. Similarly, an $\mathbf{r}$-*clause* is a clause $C$ over variables $\mathbf{r}$ which is a member of or is derivable from $B(\rho,\mathbf{r})$. We will let $1$ denote the trivial clause that is always satisfied, and we will consider $1$ to be derivable from anything so that it qualifies both as a $\mathbf{q}$-clause and as an $\mathbf{r}$-clause.

We will go trough the clauses in $\pi = (C_1, \ldots, C_L)$ in order and construct a sequence of clauses $\widetilde{\pi} = \big(\widetilde{C}_1, \ldots, \widetilde{C}_L\big)$ which will contain the two derivations $\pi_A$ from $A(\rho, \mathbf{q})$ and $\pi_B$ from $B(\rho, \mathbf{r})$ mentioned above in our proof plan for Theorem 5.3. More precisely, from each $C_i$ in $\pi$ we will obtain a clause $\widetilde{C}_i$ satisfying the following properties:

1. $\widetilde{C}_i$ is designated to be either a $\mathbf{q}$-clause or $\mathbf{r}$-clause as defined above, where we write $\mathit{type}\big(\widetilde{C}_i\big) = \mathbf{q}$ or $\mathit{type}\big(\widetilde{C}_i\big) = \mathbf{r}$ to denote the label chosen for $\widetilde{C}_i$.

2. $\widetilde{C}_i = 1$ only if $C_i{\restriction_\rho} = 1$, and if $\widetilde{C}_i \neq 1$ it holds that $\widetilde{C}_i \subseteq C_i \setminus \{a, \overline{a} \mid a \in \rho\}$.

3. With any $\widetilde{C}_i = 1$ we associate an axiom clause $E_i$ that is satisfied by an assignment $\rho(a) = 1$ for some literal $a \in C_i \cap E_i$, where we have $E_i \in A(\mathbf{p}, \mathbf{q})$ if $\mathit{type}\big(\widetilde{C}_i\big) = \mathbf{q}$ and $E_i \in B(\mathbf{p}, \mathbf{r})$ if $\mathit{type}\big(\widetilde{C}_i\big) = \mathbf{r}$.

The clauses $\widetilde{C}_i$ that are mainly of interest to us are nontrivial clauses, but in general we can expect to have a number of trivial clauses since the restriction $\rho$ might satisfy a large portion of the clauses in the formula. For such trivial clauses we can think of the clause $E_i$ in Property 3 as a *justification axiom* with *justification literal* $a \in C_i \cap E_i$ such that $\rho(a) = 1$ explaining why we chose the trivial clause 1 for $\widetilde{C}_i$. These justification axioms and literals will not be needed in our construction of the resolution refutation in part 2 of Theorem 5.3, but will play an important role later when we prove part 3 about monotone circuits in Section 6.3.

We construct $\widetilde{C}_i$ for each $C_i \in \pi$ by forward induction over $\pi$. Observe that this is sufficient to establish part 2 of Theorem 5.3. To see this, note that when we reach the final clause $C_L = \bot \in \pi$, by Property 2 we will have that $\widetilde{C}_L \subseteq C_L{\restriction_\rho} = \bot$, i.e., $\widetilde{C}_L = \bot$. Furthermore, $\widetilde{C}_L$ will be labelled as either a $\mathbf{q}$-clause or an $\mathbf{r}$-clause by Property 1, meaning that it is derived from $A(\rho, \mathbf{q})$ only or $B(\rho, \mathbf{r})$ only, respectively. It will be clear from the construction that follows below that the length of this derivation is at most $L$.

In the base case $C_i$ is an axiom. In this case simply let $\widetilde{C}_i = C_i{\restriction_\rho}$. If $C_i$ is part of $A(\mathbf{p}, \mathbf{q})$, we label $\widetilde{C}_i$ as a $\mathbf{q}$-clause, and if $C_i$ is from $B(\mathbf{p}, \mathbf{r})$, then $\widetilde{C}_i$ is an $\mathbf{r}$-clause. Properties 1 and 2 are clearly satisfied by definition. It is important to note that for many axiom clauses $C_i$ we might have $\widetilde{C}_i = 1$, but that does not violate Property 2. If $\widetilde{C}_i = C_i{\restriction_\rho} = 1$, then we let $E_i = C_i$ be the justification axiom, which obviously fulfils the conditions in Property 3.

For the inductive step, suppose $C_i = C \vee D$ was derived by applying the resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D} \tag{6.1}$$

to two previous clauses $C_j = C \vee x$ and $C_k = D \vee \overline{x}$ for $j, k < i$. By induction, we have already constructed $\widetilde{C}_j$ and $\widetilde{C}_k$ and we know their types as $\mathbf{q}$- or $\mathbf{r}$-clauses. Also, if $\widetilde{C}_j = 1$ and/or $\widetilde{C}_k = 1$, then we have justification axioms $E_j$ and/or $E_k$ that are satisfied by $\rho$. We make a case analysis over the variable $x$ resolved over depending on whether $x \in \mathbf{p}$, $x \in \mathbf{q}$, or $x \in \mathbf{r}$.

**Case 1** ($x \in \mathbf{p}$)**:** If $\rho(x) = 0$, then we set $\widetilde{C}_i = \widetilde{C}_j$ and let $\mathit{type}\big(\widetilde{C}_i\big) = \mathit{type}\big(\widetilde{C}_j\big)$. Observe that if $C_i{\restriction_\rho} \neq 1$, then we have $\widetilde{C}_i \subseteq C_j{\restriction_\rho} \subseteq C_i{\restriction_\rho}$. If $\widetilde{C}_i = \widetilde{C}_j = 1$, we copy the justification axiom also and let $E_i = E_j$. Note that any justification literal $a \in C_j \cap E_j$ (which cannot be $x$ since $\rho(x) = 0$) is present in $C_i$ as well. If instead $\rho(x) = 1$, then if $C_i{\restriction_\rho} \neq 1$ we have $\widetilde{C}_k \subseteq C_k{\restriction_\rho} \subseteq C_i{\restriction_\rho}$, so we set $\widetilde{C}_i = \widetilde{C}_k$ and let $\widetilde{C}_i$ inherit its type, and possibly its justification axiom $E_i$, from $\widetilde{C}_k$. This ensures that we maintain Properties 2 and 3. Property 1 holds since if $\rho(x) = 0$, then by the inductive hypothesis we have that $\widetilde{C}_i = \widetilde{C}_j$ is derivable from only $A(\rho, \mathbf{q})$ or only $B(\rho, \mathbf{r})$ depending on its type, and if $\rho(x) = 1$ then the same holds for $\widetilde{C}_i = \widetilde{C}_k$.

As noted above, it might well be that $\widetilde{C}_j = 1$ or $\widetilde{C}_k = 1$ (or both), but we do not care about this. The case analysis based on the value of $\rho(x)$ remains valid, and Properties 1 and 2 are preserved.

**Case 2** ($x \in \mathbf{q}$)**:** Here we divide the analysis into subcases depending on the types of $\widetilde{C}_j$ and $\widetilde{C}_k$.

- If exactly one of $\widetilde{C}_j$ or $\widetilde{C}_k$ is an $\mathbf{r}$-clause, set $\widetilde{C}_i$ to that $\mathbf{r}$-clause. If both $\widetilde{C}_j$ or $\widetilde{C}_k$ are $\mathbf{r}$-clauses, arbitrarily pick one of them (say, the one with smallest index). Label $\widetilde{C}_i$ as an $\mathbf{r}$-clause, and if $\widetilde{C}_i = 1$ let it inherit the justification clause from its chosen parent clause. Note that by our inductive hypothesis $\widetilde{C}_i$ constructed in this way will not contain any variable in $\mathbf{q}$. Since $x \in \mathbf{q}$ is the only variable that disappears in the resolution step, this means that we preserve Properties 2 and 3.
- Otherwise, if either $\widetilde{C}_j$ or $\widetilde{C}_k$ is a $\mathbf{q}$-clause not containing $x$ (which is true, for instance, if one if them is a trivial clause 1 labelled by type $\mathbf{q}$), let $\widetilde{C}_i$ equal that $\mathbf{q}$-clause and set $\mathit{type}\big(\widetilde{C}_i\big) = \mathbf{q}$. (Again, if both $\widetilde{C}_j$ or $\widetilde{C}_k$ qualify, just arbitrarily pick one of them.) Also, let $\widetilde{C}_i$ inherit the justification clause from its chosen parent if it is trivial.
- Otherwise, if either $\widetilde{C}_j$ or $\widetilde{C}_k$ is a $\mathbf{q}$-clause not containing $\overline{x}$, then let $\widetilde{C}_i$ equal that $\mathbf{q}$-clause (or choose a clause arbitrarily if both qualify), label $\widetilde{C}_i$ a $\mathbf{q}$-clause, and let it inherit the justification clause from its parent if needed.
- If none of the above cases apply, then we have two $\mathbf{q}$-clauses $\widetilde{C}_j = \widetilde{C}'_j \lor x$ and $\widetilde{C}_k = \widetilde{C}'_k \lor \overline{x}$ which are both nontrivial (i.e., distinct from 1). Set $\widetilde{C}_i$ to be the resolvent $\widetilde{C}'_j \lor \widetilde{C}'_k$ of these two clauses and let $\mathit{type}\big(\widetilde{C}_i\big) = \mathbf{q}$. (Note that this is the first time we actually used the resolution rule to construct $\widetilde{C}_i$.)

By construction, $\widetilde{C}_i$ will not contain $x$ and it can be verified that we will only have $\widetilde{C}_i = 1$ if $C_i\restriction_\rho = 1$ (using again that $x \in \mathbf{q}$ is the only variable that disappears in the resolution step, and recalling that $\rho$ does not assign values to any variables in $\mathbf{q}$). Hence, Property 2 holds. Property 3 holds since no variables in $\mathbf{p}$ disappear in the resolution step. For Property 1, just observe that if $\widetilde{C}_i$ gets classified as an $\mathbf{r}$-clause then no resolution step is involved, and if the end result is a $\mathbf{q}$-clause obtained by resolution, then both premises are $\mathbf{q}$-clauses derivable from $A(\rho, \mathbf{q})$ and so this holds also for their resolvent.

**Case 3** ($x \in \mathbf{r}$)**:** this case is analogous to the case $x \in \mathbf{q}$ but exchanging the roles of $\mathbf{r}$- and $\mathbf{q}$-variables. We leave the details to the reader.

As explained above, part 2 of Theorem 5.3 now follows by the induction principle.

## 6.2 Writing down the Interpolating Circuit

We proceed to prove part 1 of Theorem 5.3. We will use the construction in Section 6.1 with the labelling of the clauses $\widetilde{C}_i$ as $\mathbf{q}$- or $\mathbf{r}$-clauses to build the desired interpolant $I(\mathbf{p})$. Note that at the very end of the process we label the final clause $\widetilde{C}_L$ as either a $\mathbf{q}$-clause or an $\mathbf{r}$-clause, and this tells us whether $A(\rho, \mathbf{q})$ or $B(\rho, \mathbf{r})$ is unsatisfiable. All that we need to do is to build a circuit that performs the same kind of classification of the clauses in the refutation until we know what type is assigned to $\widetilde{C}_L$.

We will build this circuit $I(\mathbf{p})$ using gates $\{\lor, \land, \mathsf{sel}\}$ and constants $\{0, 1\}$.[3] As we construct the circuit we will associate the vertices $v_i$ in it with the clauses $C_i \in \pi$. We will maintain the invariant that if $\widetilde{C}_i$ is labelled as a $\mathbf{q}$-clause under some assignment $\rho$ of $\mathbf{p}$, then the value computed at $v_i$ in the circuit on input $\rho$ is 0, and if $\widetilde{C}_i$ is labelled as an $\mathbf{r}$-clause, then $v_i$ computes value 1. The output of the circuit, which is the type of $\widetilde{C}_L = \perp$, will then tell us that $A(\rho, \mathbf{q})$ is unsatisfiable if $I(\rho) = 0$ and that $B(\rho, \mathbf{r})$ is unsatisfiable if $I(\rho) = 1$. This is all that we need to show part 1 of the theorem.

More formally, as the blueprint for our circuit $I(\mathbf{p})$ we will take the DAG representation $G_\pi$ of the resolution refutation $\pi$. For every clause $C_i \in \pi$ we will label the vertex $v_i$ in $I(\mathbf{p})$ with a suitable gate taking suitable inputs. As in Section 6.1, we argue by forward induction over $\pi = (C_1, \ldots, C_L)$.

[3] We did not have constants in Definition 3.1, but if this troubles us we can get rid of them in a postprocessing step by propagating simplifications $0 \land x = 0$, $1 \land x = x$, $0 \lor x = x$, and $1 \lor x = 1$ through the circuit until all constants have been removed.

If $C_i$ is an axiom in $A(\mathbf{p},\mathbf{q})$ we fix $v_i$ to the constant $0$, otherwise if it belongs to $B(\mathbf{p},\mathbf{r})$ we fix $v_i$ to $1$. Remember that we want to maintain the invariant that $\mathbf{q}$-clauses correspond to vertices $v_i$ computing $0$ and $\mathbf{r}$-clauses correspond to vertices $v_i$ computing $1$. So far, so good.

If $C_i$ was derived by resolution from $C_j = C \lor x$ and $C_k = D \lor \overline{x}$ for $j, k < i$, we have the same kind of case analysis as in Section 6.1. Let us overload $\mathit{type}\bigl(\widetilde{C}_i\bigr)$ to denote the *binary value* of the type of the clause as defined above, so that

$$\mathit{type}\bigl(\widetilde{C}_i\bigr) = \begin{cases} 0 & \text{if } \widetilde{C}_i \text{ is a } \mathbf{q}\text{-clause,} \\ 1 & \text{if } \widetilde{C}_i \text{ is a } \mathbf{r}\text{-clause.} \end{cases} \tag{6.2}$$

We now choose the gate for $v_i$ as follows.

**Case 1** ($x \in \mathbf{p}$)**:** Mark vertex $v_i$ with the selector function sel taking as inputs $x$ and the outputs of $v_j$ and $v_k$ (which by our inductive hypothesis compute $\mathit{type}\bigl(\widetilde{C}_j\bigr)$ and $\mathit{type}\bigl(\widetilde{C}_k\bigr)$, respectively). It is straightforward to verify that the way the type of $\widetilde{C}_i$ is determined in Section 6.1 is by computing

$$\mathit{type}\bigl(\widetilde{C}_i\bigr) = \mathsf{sel}\bigl(x, \mathit{type}\bigl(\widetilde{C}_j\bigr), \mathit{type}\bigl(\widetilde{C}_k\bigr)\bigr) = \mathsf{sel}(x, v_j, v_k)\,. \tag{6.3}$$

**Case 2** ($x \in \mathbf{q}$)**:** Mark vertex $v_i$ with an OR-gate $\lor$ taking $v_j$ and $v_k$ as inputs. If at least one of $\widetilde{C}_j$ or $\widetilde{C}_k$ has been classified as an $\mathbf{r}$-clause, which by our inductive hypothesis means that either $v_j$ or $v_k$ computes the value $1$, then $\widetilde{C}_i$ gets classified as an $\mathbf{r}$-clause, and otherwise it becomes a $\mathbf{q}$-clause. This is just another way of saying that $\mathit{type}\bigl(\widetilde{C}_i\bigr) = \mathit{type}\bigl(\widetilde{C}_j\bigr) \lor \mathit{type}\bigl(\widetilde{C}_k\bigr)$, which is exactly the values that $v_i$ now computes.

**Case 3** ($x \in \mathbf{r}$)**:** Mark vertex $v_i$ with an AND-gate $\land$ taking inputs $v_j$ and $v_k$. This is (anti-)symmetric to the case when $x \in \mathbf{q}$, and as in Section 6.1 we leave the details of this case to the reader.

## 6.3 Removing Selector Gates

We have constructed an interpolating circuit and have proven parts 1 and 2 of our main theorem for today. To establish part 3 we need to prove that if the $\mathbf{p}$-variables occur only positively in $A(\mathbf{p},\mathbf{q})$ or only negatively in $B(\mathbf{p},\mathbf{r})$, then we can change the circuit above slightly by replacing the sel-gates with small monotone subcircuits. This will give us a monotone interpolating circuit.

Let as assume that the $\mathbf{p}$-variables appear only positively in $A(\mathbf{p},\mathbf{q})$. In this case we choose to replace all occurrences of

$$\mathsf{sel}(x, a, b) = (x \lor a) \land (\overline{x} \lor b) \tag{6.4}$$

by the function

$$(x \lor a) \land b\,. \tag{6.5}$$

What this means is that we replace sel-gates in (6.3), the only case where a non-monotone gate could be introduced in $I(\mathbf{p})$, with the computation

$$\mathit{type}\bigl(\widetilde{C}_i\bigr) = \bigl(x \lor \mathit{type}\bigl(\widetilde{C}_j\bigr)\bigr) \land \mathit{type}\bigl(\widetilde{C}_k\bigr)\,. \tag{6.6}$$

A closer study of the functions in (6.4) and (6.5) reveals that the only differ in that the monotone function in (6.5) returns $0$ instead of $1$ on input $(x, a, b) = (0, 1, 0)$.

When does this happen in our proof? This is when we are in the first case in our case analysis, i.e., when we have $C_i = C \lor D$ derived from $C_j = C \lor x$ and $C_k = D \lor \overline{x}$ for $x \in \mathbf{p}$ such that $\rho(x) = 0$. Moreover, $\widetilde{C}_j$ has been classified as an $\mathbf{r}$-clause and $\widetilde{C}_k$ as a $\mathbf{q}$-clause under the current restriction $\rho$. According to our original case analysis we should have set $\widetilde{C}_i = \widetilde{C}_j$ and classified $\widetilde{C}_i$ as an $\mathbf{r}$-clause. Instead, according to (6.5) we set $\widetilde{C}_i = \widetilde{C}_k$, which is a $\mathbf{q}$-clause.

Why is this a problem? The clause $\widetilde{C}_k$ does not contain $x \in \mathbf{p}$ by construction, and so if $\widetilde{C}_k$ is nontrivial Property 2 holds since $x \in \mathbf{p}$ is the only variable that disappears in the resolution step. It is also easy to see that Property 1 always holds by the inductive hypothesis when we copy a clause. What is more problematic, though, is if we have a resolvent $C_i$ such that

$$C_i{\restriction_\rho} = (C \lor D){\restriction_\rho} \neq 1 \tag{6.7}$$

but have chosen $\widetilde{C}_k = 1$ since

$$C_k{\restriction_\rho} = (D \lor \overline{x}){\restriction_\rho} = 1\,. \tag{6.8}$$

If this is the case, then setting $\widetilde{C}_i = \widetilde{C}_k$ violates Property 2 in the definition of $\mathbf{q}$- and $\mathbf{r}$-clauses, since now $\widetilde{C}_i = 1$ but $C_i{\restriction_\rho} \neq 1$. Should this happen, the whole proof crashes and burns. Not good.

Let us analyse this worrying scenario more closely. If $\widetilde{C}_k = 1$, then by Property 3 in our construction there is also a justification axiom clause $E_k$ satisfied by an assignment $\rho(a) = 1$ for some literal $a \in C_k \cap E_k$, where in addition $E_k \in A(\mathbf{p}, \mathbf{q})$ since $\widetilde{C}_k$ is a $\mathbf{q}$-clause. In view of (6.7) and (6.8), the justification literal $a$ must satisfy $a \in (D \lor \overline{x}) \setminus (C \lor D)$, i.e., we must have $a = \overline{x}$ satisfied by the assignment $\rho(x) = 0$. But this means that the axiom clause $E_k \in A(\mathbf{p}, \mathbf{q})$ would need to contain the literal $\overline{x}$, and by assumption variables $x \in \mathbf{p}$ appear only positively in $A(\mathbf{p}, \mathbf{q})$. So we do not need to worry—this problematic scenario never arises. Although the computation for $v_i$ in the monotone version of the circuit seems to make a mistake for the input $(x, a, b) = (0, 1, 0)$, the previously constructed clause $\widetilde{C}_k$ is guaranteed to be nice enough for the invariants to be preserved when we set $\widetilde{C}_i = \widetilde{C}_k$ and $\mathit{type}\bigl(\widetilde{C}_i\bigr) = \mathit{type}\bigl(\widetilde{C}_k\bigr)$.

An analogous hack works if $\mathbf{p}$-variables instead appear only negatively in $B(\mathbf{p}, \mathbf{r})$. We leave the details to the reader. This concludes our proof of part 3 of Theorem 5.3. If we apply this theorem to the formulas in (4.1a)–(4.1e) and combine it with Theorem 5.1, then we can deduce that for the clique-colouring formulas for graphs over $n$ vertices with $m = \Theta\bigl(\sqrt[4]{n}\bigr)$ it holds that resolution needs refutations of length $\exp\bigl(\Omega\bigl(n^{\delta}\bigr)\bigr)$ for $\delta = 1/8$.

## References

[AB87] Noga Alon and Ravi B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, March 1987.

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 233–350. IOS Press, 2nd edition, February 2021. Available at `http://www.jakobnordstrom.se/publications/`.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979. Preliminary version in *STOC '74*.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Kra94] Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic*, 59(1):73–86, 1994.

[Kra19] Jan Krajíček. *Proof Complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, March 2019.

[MMZ$^{+}$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[Pud97] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, September 1997.

[Raz85] Alexander A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Soviet Mathematics Doklady*, 31(2):354–357, 1985. English translation of a paper in *Doklady Akademii Nauk SSSR*.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.