COMS E6998: Proof Complexity and Applications (Spring'25) January 23, 2025

Lecture 1: Introduction & Resolution Proof System

Instructor: *Toniann Pitassi* Scribes: *Tianqi Yang*

1 Introduction to the course

Proof complexity, as the name suggests, is the study of the complexity of proving logical statements within different proof systems. While it is widely known in proof theory that many (propositional) proof systems are both complete and sound, meaning that any true statement has a proof, it is not clear whether a *short* proof always exists. A major question in proof complexity is whether every tautology has a short proof in certain proof systems and whether we can efficiently find such a proof.

In this course, we will show the classical upper and lower bounds in proof complexity and discuss some open problems in the field. We will then explore the connections between proof complexity and various topics in theoretical computer science, such as SAT solvers and the TFNP classes.

Introduction to proof complexity. Many different proof systems are of interest in proof complexity, including resolution and Frege systems. We will begin by introducing different proof systems and examining their relationships. We will then talk about the two most fundamental questions in proof complexity. The first question concerns the upper and lower bounds on proof length in certain proof systems.

In a given proof system, how long does a proof of a given true statement have to be?

The second question is the automatizability of the proof systems.

How easy is it to find a (short) proof in a given proof system?

We will mainly focus on *Boolean* formulas and proof systems in this course, where a true statement is a propositional tautology in classical logic. However, we will also talk about algebraic and semi-algebraic proof systems in the course.

Lower and upper bounds for common proof systems. As partial answers to the first question above, we will show lower and upper bounds for common proof systems in proof complexity. This is analogous to proving algorithmic lower and upper bounds in computational complexity. We will prove the classical results and present some open problems.

Applications and connections to other topics in TCS. In the second half of the course, we will explore connections between proof complexity and various areas of theoretical computer science. Topics include, among others, random and semi-random CSP problems, a recent surprising connection to lower bounds for locally decodable codes, SAT solving and automated theorem proving, the relationship between proof systems and TFNP subclasses, and various intriguing relationships between proof complexity bounds and algorithmic upper and lower bounds. See, e.g., [PT16; FKP19; RGR22] for surveys on related topics.

2 Proof systems

We begin by introducing the concept of proof systems. In this course, we will primarily focus on proof systems for Boolean formulas, but we will also discuss algebraic proof systems for equalities and inequalities. We will now introduce basic notations and properties that will be useful later.

2.1 Propositional proof system

In proof complexity, *propositional proof systems* refer to proof systems for proving that a Boolean formula is a tautology in classical propositional logic. Informally, the "input" to a propositional proof system $\mathcal{P}$ is a Boolean formula, and a proof in $\mathcal{P}$ provides an efficient way to verify that the input formula is always true. We will give a formal definition in Section 3. The *completeness* of $\mathcal{P}$ ensures that every tautology has a proof in $\mathcal{P}$, while the *soundness* of $\mathcal{P}$ guarantees that any Boolean formula with a proof in $\mathcal{P}$ is indeed a tautology.

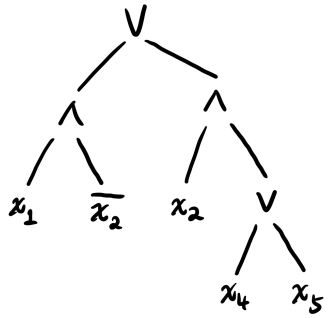


Figure 1: An example of Boolean formulas: $(x_1 \wedge \overline{x_2}) \vee (x_2 \wedge (x_4 \vee x_5))$

Notations. Let φ be a propositional formula and Γ a set of propositional formulas. In this note, we use the notation $\Gamma \models \varphi$ to denote that φ is a semantic consequence of Γ, meaning that φ is implied by Γ in classical logic. In particular, when $\Gamma = \varnothing$, $\models \varphi$ is equivalent to φ being a tautology.

For any proof system $\mathcal{P}$, we use the notation $\Gamma \vdash_{\mathcal{P}} \varphi$ to denote that φ is a syntactic consequence of Γ in $\mathcal{P}$, meaning that that φ can be derived from Γ in the proof system $\mathcal{P}$. In particular, $\vdash_{\mathcal{P}} \varphi$ means that there is a proof of φ in $\mathcal{P}$. The soundness and completeness of a proof system $\mathcal{P}$ is equivalent to saying that for any formula φ,

$$\models \varphi \quad \iff \quad \vdash_{\mathcal{P}} \varphi.$$

Let φ be a propositional formula over variables $x_1, \ldots, x_n$. Let α be a truth assignment (or interpretation) to the variables $x_1, \ldots, x_n$. We use the notation $\alpha \models \varphi$ to mean φ evaluates to true under the assignment α.

Refutations. For some proof systems (such as resolution, which we will discuss in Section 5), it is more convenient to view them as *refutation* systems. In such a system $\mathcal{P}$, instead of showing that φ is a *tautology*, a refutation of φ in $\mathcal{P}$ shows that φ is *unsatisfiable*. Since proving that φ is a tautology is equivalent to proving that $\neg\varphi$ is unsatisfiable, we will use these two perspectives interchangeably throughout the course. For such refutation systems, we will use the notation $\varphi \vdash_{\mathcal{P}} \bot$ to emphasize that the unsatisfiable formula φ has a refutation in $\mathcal{P}$.

CNF. Recall that a variable x or its negation $\bar{x}$ is called a *literal*. A *clause* is a disjunction of literals. A formula φ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, i.e.,

$$\varphi = C_1 \wedge C_2 \wedge \dots C_m \tag{1}$$

where $C_1, \dots, C_m$ are all clauses. The *width* of a clause is the number of literals it contains, and the *width* of a CNF formula is the maximum width among its clauses. We say that a formula is in k-CNF if it is a CNF of width at most k. One may similarly define *disjunctive normal form* (DNF) consisting of formulas that are disjunctions of conjunctions of literals.

By the Cook-Levin Theorem, any formula φ can be transformed into a CNF (even 3-CNF) ψ such that φ is unsatisfiable if and only if ψ is unsatisfiable. Therefore, for refutation systems, we often assume that the input formula φ is a CNF, and we want to find a refutation $\varphi \vdash_{\mathcal{P}} \perp$ if φ is unsatisfiable.

2.2 Algebraic and semi-algebraic proof systems

We will also talk about algebraic and semi-algebraic proof systems. Instead of refuting CNF formulas $f = C_1 \wedge \dots C_m$ over Boolean variables $x_i \in \{0, 1\}$, *algebraic proof systems* (*semi-algebraic proof systems*, resp.) start with a system P of polynomial equalities (inequalities, resp.) over some field or ring. A refutation of P in an algebraic proof system or semi-algebraic proof system is a "proof" that P has no solution over the underlying field or ring.

By mapping $\{\top, \perp\}$ to $\{1, 0\}$, any Boolean formula φ can be "algebrized" into a system P of polynomial equalities such that φ is unsatisfiable if and only if P has no solution. So any algebraic proof system or semi-algebraic proof system can also be viewed as refutation systems for the classical propositional logic. In this course, we will mainly talk about algebraic proof systems for such Boolean formulas.

Example 1. *For example, the CNF formula*

$$\varphi = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3 \vee x_4)$$

can be written as the following set of polynomial equations over reals:

$$\begin{cases} (1 - x_1) \cdot x_2 = 0 \\ x_1 \cdot (1 - x_3) \cdot (1 - x_4) = 0 \\ x_i^2 - x_i = 0, \qquad\qquad i = 1, 2, 3, 4 \end{cases},$$

where the last set of equations ensures that each x_i takes values in $\{0, 1\}$.

2.3 Other proof systems

There are many other types of proof systems. In general, any way of proving statements to others that can be easily verified can be regarded as a proof. Consider the following Hajós calculus system (see, e.g., [PU95; IST10]) when $k = 4$. It consists of one axiom, K_5, and three deduction rules:

(Vertex/Edge Introduction) Add an arbitrary number of vertices and edges.

(Contraction rule) Contract two non-adjacent vertices and remove duplicating edges.

(Edge Elimination rule) Let G_1 and G_2 be graphs that are identical except G_1 contains the edges $\{v_1, v_2\}$ and $\{v_1, v_3\}$, while G_2 contains the edges $\{v_1, v_2\}$ and $\{v_2, v_3\}$ for three vertices v_1, v_2, v_3. Let G_3 be the graph that is identical to G_1 and G_2 except it only contains the edge $\{v_1, v_2\}$ among v_1, v_2, v_3. Then G_3 is derivable from G_1 and G_2.

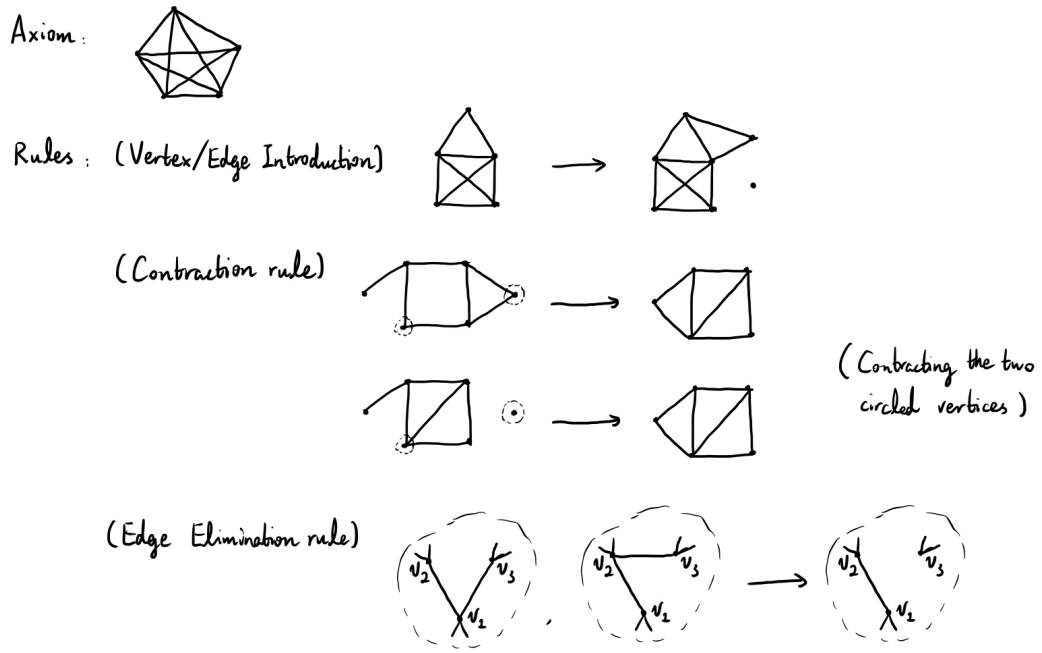


Figure 2: Examples of the Hajós calculus system ($k = 4$)

Figure 2 demonstrates examples of the rules. This system can be viewed as a Hilbert-style proof system proving that a graph is non-4-colorable. It is *sound and complete* in the sense that a graph is non-4-colorable if and only if it can be derived in the by the rules.

3 Cook-Reckhow propositional proof systems

We now present the formal definition of propositional proof systems as introduced by Cook and Reckhow [CR79]. We take an arbitrary typical binary encoding of propositional formulas and define the set $TAUT \subseteq \{0,1\}^*$ as the set of encodings of all tautologies, i.e.,

$$TAUT \triangleq \{\text{encoding of } \varphi \mid \varphi \text{ is a tautology}\}.$$

Definition 2 (Propositional proof system)**.** *A* propositional proof system *(pps) is a polynomial-time algorithm* $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ *with* 2 *inputs* (x, p) *such that for all* $x \in \{0,1\}^*$*,*

$$x \in TAUT \iff \exists p \in \{0,1\}^* \ \ s.t.\ V(x,p) = 1. \tag{2}$$

In the above definition, one may think of x as the encoding of a Boolean formula φ, and p as the encoding of a proof of $\vdash \varphi$ in the proof system. The $\Longleftarrow$ and $\Longrightarrow$ of Equation (2) correspond to the soundness and completeness of the proof system, respectively.

Note that in the above definition the running time of V is bounded by $\mathsf{poly}(|x|, |p|)$. However, Equation (2) does not impose a bound on the length of the proof p when $x \in TAUT$. So when the proof p is very long, say $|p| = 2^{|x|}$, then the running time of V can be super-polynomial in the length of the formula x.

Example 3 (Truth table)**.** *A (trivial) example of a propositional proof system is the truth table system, where the proof p consists of the truth table of the formula encoded by x. The algorithm V can simply verify whether p is indeed the truth table, and whether the truth table is all-*1*. Although the length of p is exponential in the number of variables in x, the running time of V is always bounded by* $\mathsf{poly}(|x|, |p|)$*. So this satisfies Definition 2.*

Remark 4. *The requirement that the algorithm V runs in polynomial time reflects the philosophical idea that in a well-designed proof system, one should be able to efficiently verify whether a proof is correct. In addition, we want the definition to have connections to other fields in complexity theory. Indeed, Theorem 6 shows the tight connections between propositional proof systems and fundamental questions in structural complexity.*

In Equation (2), the length of the proof p can be arbitrarily long. So it is natural to ask whether such there always exist a short proof p, or there are cases where the proofs have to be long. The following question is a central question in proof complexity:

> *Is there a propositional proof system V such that every propositional tautology has a short proof in V.*

Ideally, as complexity theorists, we want to ask whether there exists a pps such that every tautology has a polynomial-size proof. This leads to the following definition.

Definition 5 (Polynomially-bounded pps)**.** *A propositional proof system V is* polynomially-bounded *if for all $x \in \{0,1\}^*$,*

$$x \in TAUT \iff \exists p \in \{0,1\}^{\mathsf{poly}(|x|)} \ \ s.t.\ V(x,p) = 1.$$

Cook and Reckhow [CR79] proved that such proof systems cannot exist unless the polynomial hierarchy collapses.

Theorem 6 ([CR79])**.** *There exists a polynomially-bounded proof system if and only if* $\mathsf{NP} = \mathsf{coNP}$.

The proof of this theorem is straightforward: if a polynomially-bounded propositional proof system exists, then one can design a non-deterministic polynomial-time algorithm for UNSAT by guessing a refutation p. Conversely, any NP algorithm for UNSAT can be interpreted as a polynomially-bounded proof system.

Notably, Theorem 6 suggests a potential approach to proving $\mathsf{P} \neq \mathsf{NP}$ by ruling out the existence of polynomially-bounded propositional proof systems.

4 The landscape of proof complexity

Analogous to structural complexity theory, where we study the relationships between different complexity classes, an important question in proof complexity is understanding the relationships between different propositional proof systems. We first define what does "one proof system is stronger than the other" means in proof complexity.

Definition 7 (p-simulation)**.** *A proof system $\mathcal{P}$* p-simulates *another proof system $\mathcal{Q}$ if, for any propositional tautology φ, whenever φ has a proof of size s in $\mathcal{Q}$, it also has a proof of size* $\mathsf{poly}(s)$ *in $\mathcal{P}$.*

Definition 8 (p-equivalence)**.** *Two proof systems $\mathcal{P}$ and $\mathcal{Q}$ are* p-equivalent *if $\mathcal{P}$ p-simulates $\mathcal{Q}$ and $\mathcal{Q}$ p-simulates $\mathcal{P}$.*

We now introduce a basic hierarchy of propositional proof systems in Figure 3, where an arrow from $\mathcal{P}$ to $\mathcal{Q}$ indicates that $\mathcal{P}$ p-simulates $\mathcal{Q}$. Hopefully, we will gradually enrich and expand this graph throughout this course.

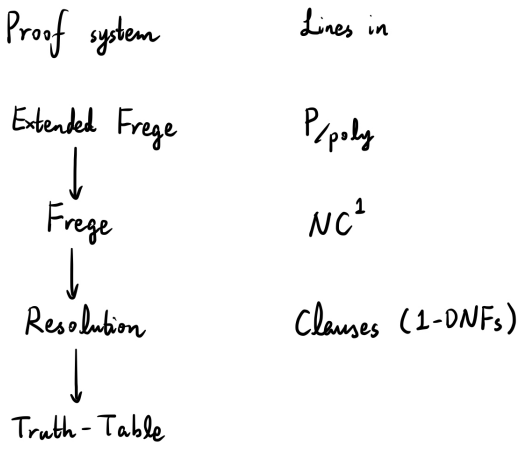


Figure 3: Hierarchy of proof systems

In Figure 3, Truth-Table represents the truth table proof system described in Example 3. Frege denotes the standard Hilbert-style proof system with an arbitrary finite set of axioms and deduction rules.[1]. The Extended Frege system, EF, is an extension of Frege that allows extension variables. The Resolution system, Res, will be discussed in detail in Section 5.

Conceptually, Res, Frege, and EF can all be regarded as Frege-like systems where proof lines are constrained to different circuit classes. We will see that Res, Frege, and EF correspond to Frege systems where proof lines are restricted to 1-DNF (single OR functions), NC^1, and $\mathsf{P}_{/\mathsf{poly}}$, respectively.

5 Resolution system

We now formally introduce our first propositional proof system: resolution. Resolution is a simple yet widely used proof system both in theory and in practice.

Resolution, denoted by Res, is a proof system for refuting unsatisfiable CNF formulas. For any propositional tautology, one must first convert it to an equivalent CNF using the Cook-Levin Theorem and then negate it to obtain an unsatisfiable formula. It has only one rule: the resolution rule

$$\frac{A \vee x \qquad B \vee \bar{x}}{A \vee B} \quad \text{(Resolution rule)}$$

[1]Cook and Reckhow [CR79] proved that Frege-like systems with any sets of axioms and rules are p-equivalent to each other, as long as they are sound and complete.

A resolution refutation of a CNF formula $\varphi = C_1 \wedge C_2 \wedge \dots C_m$ is represented as a directed acyclic graph (DAG), where each vertex has in-degree 0 or 2 and is labeled with a clause (a disjunction of literals). Each vertex with in-degree 0 is labeled with one of the clauses $C_1, \dots, C_m$ in φ. Each vertex with in-degree 2 is labeled with a clause derived from the two preceding clauses using the resolution rule. The proof ends with the empty clause $\varnothing$.

Example 9. *Consider the unsatisfiable formula*

$$\varphi = x_1 \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge \overline{x_3}.$$

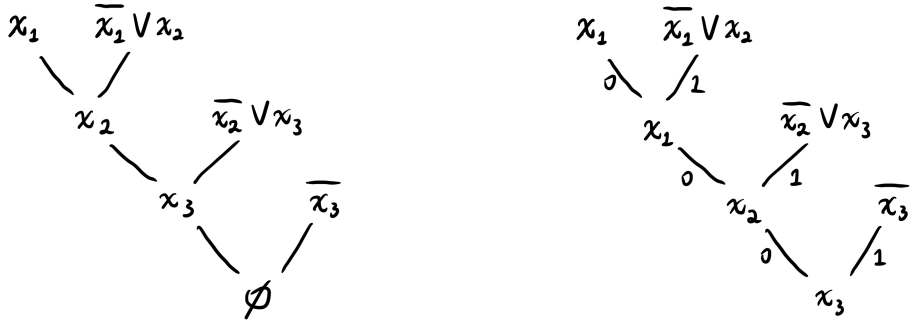
For resolution, it is often more convenient to represent φ as a set of clauses: $\{x_1, \overline{x_1} \vee x_2, \overline{x_2} \vee x_3, \overline{x_3}\}$. The tree in Figure 4a is a valid resolution refutation for $\varphi \vdash_{\mathsf{Res}} \bot$. It can be formally written as the following proof tree:

$$\dfrac{\dfrac{\dfrac{x_1 \qquad \overline{x_1} \vee x_2}{x_2} \qquad \overline{x_2} \vee x_3}{x_3} \qquad \overline{x_3}}{\bot}$$

Tree-like resolution, denoted by $\mathsf{TreeRes}$, is a restricted form of resolution in which the proof DAG is a tree. The proof in the above example is indeed a tree-like resolution. It is clear that resolution p-simulates tree-like resolution. Next week, we will show that the converse does not hold.

The *size* of a resolution proof is the number of clauses it contains. The *depth* of a tree-like resolution is the height of the proof tree. The *width* of a resolution proof is the maximum width of any clause in the proof. For instance, the proof in Example 9 has size 7, depth 4, and width 2.

Alternatively, a resolution proof can be represented as a sequence of clauses, where each clause is either one of the original clauses in φ or derived from two preceding clauses using resolution. The final clause in the sequence is the empty clause $\varnothing$. In this view, tree-like resolution corresponds to proofs where each line is used in the resolution rules at most once.



(a) Resolution proof tree of $\varphi \vdash_{\mathsf{Res}} \bot$ (b) Decision tree solving Search_φ

Figure 4: Resolution proof tree and the corresponding decision tree of Example 9

5.1 Soundness and completeness of resolution

In this section, we prove that resolution is both sound and complete. We begin with soundness, which is typically the easier direction.

We begin with the soundness of the resolution rule.

Lemma 10 (Soundness of the resolution rule)**.** *For any interpretation α, if $\alpha \models A \vee x$ and $\alpha \models B \vee \bar{x}$, then $\alpha \models A \vee B$.*

Proof. This follows directly from a case analysis on the truth value of x:

- If $\alpha(x) = \top$, then $\alpha \models B$, implying $\alpha \models A \vee B$.
- If $\alpha(x) = \bot$, then $\alpha \models A$, implying $\alpha \models A \vee B$.

□

We now show the soundness of resolution.

Theorem 11 (Resolution is sound)**.** $\varphi \vdash_{\mathsf{Res}} \bot \implies \models \neg\varphi$.

Proof. Suppose that Π is a resolution refutation of φ. Let $C_1, \ldots, C_m, D_1, \ldots, D_{m'}$ be the sequence of clauses in Π (in topological order), where $C_1, \ldots, C_m$ are initial clauses from φ and $D_1, \ldots, D_{m'}$ are derived clauses. By the syntax of resolution, $D_{m'} = \varnothing$ is the empty clause. Assume, for the sake of contradiction, that φ is satisfied by some assignment α. We prove by induction over i that $\alpha \models C_1 \wedge \cdots \wedge C_m \wedge D_1 \wedge \cdots \wedge D_i$.

The initial case $i = 0$ is trivial since $C_1 \wedge \cdots \wedge C_m$ is the formula φ.

Suppose that $\alpha \models C_1 \wedge \cdots \wedge C_m \wedge D_1 \wedge \cdots \wedge D_{i-1}$ for some $1 \leq i \leq m'$. The clause D_i is derived from two previous clauses via the resolution rule, so $\alpha \models D_i$ by Lemma 10.

However, $D_{m'}$ is the empty clause, so plugging in $i = m'$ gives $\alpha \models \bot$. Contradiction. □

For completeness, we show that even tree-like resolution is complete. The completeness of resolution then follows directly by definition.

Theorem 12 (Tree-like resolution is complete)**.** $\models \neg\varphi \implies \varphi \vdash_{\mathsf{Res}} \bot$.

The idea is to relate tree-like resolution proofs to decision trees solving search problems in TFNP. Since decision tree is a complete computational model, it follows that tree-like resolution must also be complete.

Consider the following search problem. Let $f = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$ be an unsatisfiable formula. The problem Search_f takes as input an assignment $\alpha \in \{0,1\}^n$ and outputs any index i for which $\alpha \not\models C_i$.

We consider decision trees solving Search_f, where each internal node is labeled with a variable x_i and each leaf is labeled with a clause. Figure 4b gives an example of a decision tree solving Search_φ for the φ in Example 9.

The key claim is stated below.

Claim 13. *Let f be an unsatisfiable CNF formula. If there is a decision tree solving Search_f, then there is a tree-like resolution proof refuting f.*

In fact, with some caveats, one can show that the optimal decision tree solving Search_f has exactly the same shape as the optimal resolution proof refuting f, as suggested by the similarity between Figure 4a and Figure 4b.

We first prove that tree-like resolution is complete from Claim 13.

Proof of Theorem 12. Let f be any unsatisfiable CNF formula. By definition, Search_f must be a total search problem. There is always a canonical decision tree solving the total search problem Search_f by enumerating over all possible inputs. By Claim 13, this transforms to a tree-like resolution proof refuting f. □

We now prove Claim 13.

Proof of Claim 13. Let T be a decision tree of the minimum size solving Search_f. We now relabel the nodes in T with clauses so that they form a tree-like resolution refutation of f. The relabeling is simple: each leaf is labeled with its corresponding decision tree output, and each internal node that queries the variable x_i is labeled with the clause obtained by applying the resolution rule with x_i resolved. We need to show that this indeed gives a valid tree-like resolution proof. We do so from leaves to the root and maintain the following property for any node v in T:

(*) *The clause C labeled at v is falsified by the partial assignment induced by the path from root to v.*

All leaves satisfy (*) by the definition of T solving Search_f. We now consider the internal nodes in a bottom-up order. Let v be an internal node of the decision tree T querying the variable x_i. Let v_0 be the child going from the edge $x_i = 0$, T_0 be the subtree rooted at v_0, and C_0 be the clause labeled at v_0. Similarly let v_1, T_1, and C_1 be the child, subtree, and clause from $x_i = 1$.

We claim that x_i occurs in both C_0 and C_1. To see this, assume that C_0 does not contain the variable x_i. The first observation is that by minimality, T does not query any variable twice along any path from the root to a leaf. This means that x_i will not be queried again in the subtree T_0, so x_i will not occur in any of the clauses in T_0. In particular, if we consider any leaf w in T_0, the decision tree outcome C_w at w, which is the same as the clause labeled at w, does not contain x_i. Let σ_w be the partial assignment induced by the path from root to w, C_w must also be falsified by the assignment $\sigma_w \setminus \{x_i \leftarrow 0\}$. Therefore, we can remove the query to x_i and replace the whole subtree at v with T_0. This gives us a smaller decision tree solving Search_f, contradicting the minimality of T. An analogous argument can be applied to the case when C_1 does not contain x_i.

By (*), x_i must occur positively in C_0 and negatively in C_1. Suppose that $C_0 = x_i \vee A_0$ and $C_1 = \overline{x_i} \vee A_1$. By definition we label v with the clause $A_0 \vee A_1$. Let σ be the partial assignment induced by the path from the root to v. Again by (*) at v_0, $\sigma \cup \{x_i \leftarrow 0\}$ falsifies C_0, which implies that σ falsifies A_0. Similarly, by (*) at v_1, σ falsifies A_1. Hence, σ falsifies $A_0 \vee A_1$, completing the induction step.

Applying (*) at the root, since the partial assignment at the root is empty, the only clause falsified by the empty assignment is the empty clause. This completes the proof. □

6 Pigeon-hole principle

To prepare for proving strong resolution lower bounds, we first introduce hard formulas. In proof complexity, not many hard candidate formulas are commonly used. Today we will define the pigeon-hole principle, which is one of the most classical hard formula in proof complexity. We will show that the pigeon-hole principle requires an exponential-size resolution proof next week.

The pigeon-hole principle $\neg\mathsf{PHP}_n^{n+1}$, or more generally $\neg\mathsf{PHP}_n^m$ for any $m > n$, is an unsatisfiable formula consisting of nm variables and $m + nm^2$ clauses. The variables P_{ij}, for $i \in [m]$ and $j \in [n]$, semantically indicate that the i-th pigeon is assigned to the j-th hole. It has two kinds of clauses:

(Pigeon clauses) For each $i \in [m]$, $\neg\mathsf{PHP}_n^m$ contains the clause

$$\bigvee_{j \in [n]} P_{ij}.$$

(Hole clauses) For each $j \in [n]$ and distinct indices $i_1, i_2 \in [m]$ with $i_1 \neq i_2$, $\neg\mathsf{PHP}_n^m$ contains the clause

$$\overline{P_{i_1 j}} \vee \overline{P_{i_2 j}}.$$

Intuitively, $\neg\mathsf{PHP}_n^m$ requires counting, making it difficult to prove in weak proof systems that lack the ability to express counting arguments. Next week, we will formalize this idea and show that $\neg\mathsf{PHP}_n^{n+1}$ requires a resolution proof of size $2^{\Omega(n)}$.

References

[CR79] Stephen A. Cook and Robert A. Reckhow. "The Relative Efficiency of Propositional Proof Systems". In: *J. Symb. Log.* 44.1 (1979), pp. 36–50. DOI: 10.2307/2273702. URL: https://doi.org/10.2307/2273702 (cit. on pp. 4, 5, 6).

[FKP19] Noah Fleming, Pravesh Kothari, and Toniann Pitassi. "Semialgebraic Proofs and Efficient Algorithm Design". In: *Found. Trends Theor. Comput. Sci.* 14.1-2 (2019), pp. 1–221. DOI: 10.1561/0400000086. URL: https://doi.org/10.1561/0400000086 (cit. on p. 1).

[IST10] Kazuo Iwama, Kazuhisa Seto, and Suguru Tamaki. "The complexity of the Hajós calculus for planar graphs". In: *Theor. Comput. Sci.* 411.7-9 (2010), pp. 1182–1191. DOI: 10.1016/J.TCS.2009.12.011. URL: https://doi.org/10.1016/j.tcs.2009.12.011 (cit. on p. 3).

[PT16] Toniann Pitassi and Iddo Tzameret. "Algebraic proof complexity: progress, frontiers and challenges". In: *ACM SIGLOG News* 3.3 (2016), pp. 21–43. DOI: 10.1145/2984450.2984455. URL: https://doi.org/10.1145/2984450.2984455 (cit. on p. 1).

[PU95] Toniann Pitassi and Alasdair Urquhart. "The Complexity of the Hajos Calculus". In: *SIAM J. Discret. Math.* 8.3 (1995), pp. 464–483. DOI: 10.1137/S089548019224024X. URL: https://doi.org/10.1137/S089548019224024X (cit. on p. 3).

[RGR22] Susanna F. de Rezende, Mika Göös, and Robert Robere. "Guest Column: Proofs, Circuits, and Communication". In: *SIGACT News* 53.1 (2022), pp. 59–82. DOI: 10.1145/3532737.3532746. URL: https://doi.org/10.1145/3532737.3532746 (cit. on p. 1).