Proof Complexity as a Computational Lens: Lecture 2

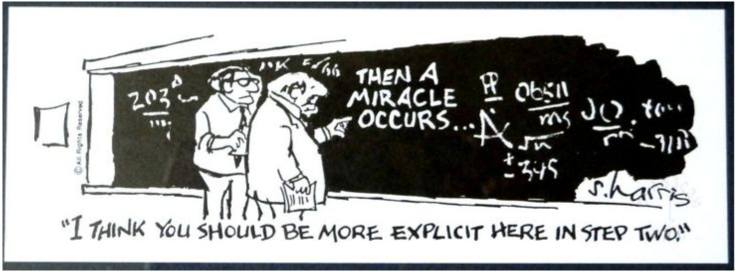
Theory Basics, Resolution, and the Pigeonhole Principle

Jakob Nordström

University of Copenhagen and Lund University

November 4, 2025

What is a Proof?



The Subject Matter of This Course

- What is a proof?
- Which (logical) statements have efficient proofs?
- How can we find such proofs? (Is it even possible?)
- What are good methods of reasoning about logical statements?
- What are natural notions of "efficiency" of proofs? (size, complexity, et cetera)
- How are these notions related?

Today's Lecture

- More "theory-oriented" introduction to proof complexity
- Some "teasers" for what to expect in coming lectures
- Recap of resolution proof system
- Proof that resolution cannot reason efficiently about the pigeonhole principle (on the board)
- Introductory slides might go slightly fast, but
  - everything will be online to allow recap
  - we will repeat everything more carefully when we need it later

So What Is a Proof?

Claim: 25957 is the product of two primes

True or false? What kind of proof would convince us?

So What Is a Proof?

Claim: 25957 is the product of two primes

True or false? What kind of proof would convince us?

- "I told you so. Just factor and check it yourself!"
  Not much of a proof

So What Is a Proof?

Claim: 25957 is the product of two primes

True or false? What kind of proof would convince us?

- "I told you so. Just factor and check it yourself!"
  Not much of a proof

- $$\begin{array}{ll} 25957 \equiv 1 \pmod{2} & 25957 \equiv 0 \pmod{101} \\ 25957 \equiv 1 \pmod{3} & 25957 \equiv 1 \pmod{103} \\ 25957 \equiv 2 \pmod{5} & \vdots \\ \vdots & 25957 \equiv 0 \pmod{257} \\ 25957 \equiv 19 \pmod{99} & \vdots \end{array}$$
  OK, but maybe even a bit of overkill

So What Is a Proof?

Claim: 25957 is the product of two primes

True or false? What kind of proof would convince us?

- "I told you so. Just factor and check it yourself!"
  Not much of a proof
- $$\begin{array}{llll} 25957 \equiv 1 & \pmod{2} & 25957 \equiv 0 & \pmod{101} \\ 25957 \equiv 1 & \pmod{3} & 25957 \equiv 1 & \pmod{103} \\ 25957 \equiv 2 & \pmod{5} & \vdots & \\ \vdots & & 25957 \equiv 0 & \pmod{257} \\ 25957 \equiv 19 & \pmod{99} & \vdots & \end{array}$$
  OK, but maybe even a bit of overkill
- "$25957 = 101 \cdot 257$; check yourself that these are primes"

Key demand: A proof should be efficiently verifiable

Proof system

Proof system for a language L (adapted from Cook & Reckhow [CR79]):

Deterministic algorithm $\mathcal{P}(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there is a string π (a proof) for which $\mathcal{P}(x, \pi) = 1$
- for all $x \notin L$ it holds for all strings π that $\mathcal{P}(x, \pi) = 0$

Proof system

Proof system for a language L (adapted from Cook & Reckhow [CR79]):

Deterministic algorithm $\mathcal{P}(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there is a string π (a proof) for which $\mathcal{P}(x, \pi) = 1$
- for all $x \notin L$ it holds for all strings π that $\mathcal{P}(x, \pi) = 0$

Think of $\mathcal{P}$ as "proof checker"
Note that proof π can be very large compared to x
Only have to achieve polynomial time in $|x| + |\pi|$

Proof system

Proof system for a language L (adapted from Cook & Reckhow [CR79]):

Deterministic algorithm $\mathcal{P}(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there is a string π (a proof) for which $\mathcal{P}(x, \pi) = 1$
- for all $x \notin L$ it holds for all strings π that $\mathcal{P}(x, \pi) = 0$

Think of $\mathcal{P}$ as "proof checker"
Note that proof π can be very large compared to x
Only have to achieve polynomial time in $|x| + |\pi|$

Propositional proof system: proof system for the language TAUT of all valid propositional logic formulas (or tautologies)

Propositional Logic: Syntax

Set $Vars$ of Boolean variables ranging over $\{0, 1\}$ (false and true)

Propositional Logic: Syntax

Set $Vars$ of Boolean variables ranging over $\{0, 1\}$ (false and true)

Logical connectives:

- negation $\neg$
- conjunction $\wedge$
- disjunction $\vee$
- implication $\rightarrow$
- equivalence $\leftrightarrow$

Propositional Logic: Syntax

Set Vars of Boolean variables ranging over $\{0, 1\}$ (false and true)

Logical connectives:

- negation $\neg$
- conjunction $\wedge$
- disjunction $\vee$
- implication $\rightarrow$
- equivalence $\leftrightarrow$

Set PROP of propositional logic formulas is smallest set X such that

- $x \in X$ for all propositional logic variables $x \in \mathit{Vars}$
- if $F, G \in X$ then $(F \wedge G), (F \vee G), (F \rightarrow G), (F \leftrightarrow G) \in X$
- if $F \in X$ then $(\neg F) \in X$

Propositional Logic: Semantics

Let α denote a truth value assignment, i.e., $\alpha : \mathit{Vars} \rightarrow \{0, 1\}$

Propositional Logic: Semantics

Let α denote a truth value assignment, i.e., $\alpha : \mathit{Vars} \to \{0, 1\}$

Extend α from variables to formulas by:

- $\alpha(\neg F) = 1$ if $\alpha(F) = 0$
- $\alpha(F \lor G) = 1$ unless $\alpha(F) = \alpha(G) = 0$
- $\alpha(F \land G) = 1$ if $\alpha(F) = \alpha(G) = 1$
- $\alpha(F \to G) = 1$ unless $\alpha(F) = 1$ and $\alpha(G) = 0$
- $\alpha(F \leftrightarrow G) = 1$ if $\alpha(F) = \alpha(G)$

Propositional Logic: Semantics

Let α denote a truth value assignment, i.e., $\alpha : \mathit{Vars} \to \{0,1\}$

Extend α from variables to formulas by:

- $\alpha(\neg F) = 1$ if $\alpha(F) = 0$
- $\alpha(F \lor G) = 1$ unless $\alpha(F) = \alpha(G) = 0$
- $\alpha(F \land G) = 1$ if $\alpha(F) = \alpha(G) = 1$
- $\alpha(F \to G) = 1$ unless $\alpha(F) = 1$ and $\alpha(G) = 0$
- $\alpha(F \leftrightarrow G) = 1$ if $\alpha(F) = \alpha(G)$

We say that F is

- satisfiable if there is an assignment α with $\alpha(F) = 1$
- valid or tautological if all assignments satisfy F
- falsifiable if there is an assignment α with $\alpha(F) = 0$
- unsatisfiable or contradictory if all assignments falsify F

Example Propositional Proof System

Example (Truth table)

p	q	r	$(p \land (q \lor r)) \leftrightarrow ((p \land q) \lor (p \land r))$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Example Propositional Proof System

Example (Truth table)

p	q	r	$(p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r))$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Certainly polynomial-time checkable measured in “proof” size
Why does this not make us happy?

Proof System Complexity

Complexity $cplx(\mathcal{P})$ of a proof system $\mathcal{P}$:

Smallest $g : \mathbb{N} \rightarrow \mathbb{N}$ such that $x \in L$ if and only if there is a proof π of size $|\pi| \leq g(|x|)$ such that $\mathcal{P}(x, \pi) = 1$

Proof System Complexity

Complexity $cplx(\mathcal{P})$ of a proof system $\mathcal{P}$:

Smallest $g : \mathbb{N} \to \mathbb{N}$ such that $x \in L$ if and only if there is a proof π of size $|\pi| \leq g(|x|)$ such that $\mathcal{P}(x, \pi) = 1$

If a proof system is of polynomial complexity, it is said to be polynomially bounded or p-bounded

Proof System Complexity

Complexity $cplx(\mathcal{P})$ of a proof system $\mathcal{P}$:

Smallest $g : \mathbb{N} \to \mathbb{N}$ such that $x \in L$ if and only if there is a proof π of size $|\pi| \leq g(|x|)$ such that $\mathcal{P}(x, \pi) = 1$

If a proof system is of polynomial complexity, it is said to be polynomially bounded or p-bounded

Example (Truth table continued)

Truth table is a propositional proof system, but of exponential complexity!

Proof systems and P vs. NP

Theorem (Cook & Reckhow [CR79])

$\mathrm{NP} = \mathrm{coNP}$ *if and only if there exists a polynomially bounded propositional proof system*

Proof systems and P vs. NP

Theorem (Cook & Reckhow [CR79])

NP $=$ coNP *if and only if there exists a polynomially bounded propositional proof system*

Proof sketch.

NP is *exactly* the set of languages with p-bounded proof systems. $\square$

Proof systems and P vs. NP

Theorem (Cook & Reckhow [CR79])

NP $=$ coNP *if and only if there exists a polynomially bounded propositional proof system*

Proof sketch.

NP is *exactly* the set of languages with p-bounded proof systems.

($\Rightarrow$) $\mathrm{TAUT} \in$ coNP since F is *not* a tautology iff $\neg F \in \textsc{Sat}$.
If NP $=$ coNP, then $\mathrm{TAUT} \in$ NP has a p-bounded proof system by definition.

$\square$

Proof systems and P vs. NP

Theorem (Cook & Reckhow [CR79])

NP $=$ coNP *if and only if there exists a polynomially bounded propositional proof system*

Proof sketch.

NP is *exactly* the set of languages with p-bounded proof systems.

($\Rightarrow$) $\mathrm{TAUT} \in$ coNP since F is *not* a tautology iff $\neg F \in \textsc{Sat}$.
If NP $=$ coNP, then $\mathrm{TAUT} \in$ NP has a p-bounded proof system by definition.

($\Leftarrow$) Suppose there exists a p-bounded proof system. Then $\mathrm{TAUT} \in$ NP, and since TAUT is complete for coNP it follows that NP $=$ coNP. $\square$

Polynomial Simulation

The conventional wisdom is that NP $\neq$ coNP
Seems that proof of this is light-years away
(Would imply P $\neq$ NP as a corollary)

Polynomial Simulation

The conventional wisdom is that $\mathrm{NP} \neq \mathrm{coNP}$
Seems that proof of this is light-years away
(Would imply $\mathrm{P} \neq \mathrm{NP}$ as a corollary)

Reason 1 for proof complexity: approach this distant goal by studying successively stronger proof systems and relating their strengths

Polynomial Simulation

The conventional wisdom is that NP $\neq$ coNP
Seems that proof of this is light-years away
(Would imply P $\neq$ NP as a corollary)

Reason 1 for proof complexity: approach this distant goal by studying successively stronger proof systems and relating their strengths

Definition (p-simulation)

$\mathcal{P}_1$ polynomially simulates, or p-simulates, $\mathcal{P}_2$ if there exists a polynomial-time computable function f such that for all $F \in \mathrm{TAUT}$ it holds that $\mathcal{P}_2(F, \pi) = 1$ iff $\mathcal{P}_1(F, f(\pi)) = 1$

Polynomial Simulation

The conventional wisdom is that NP $\neq$ coNP
Seems that proof of this is light-years away
(Would imply P $\neq$ NP as a corollary)

Reason 1 for proof complexity: approach this distant goal by studying successively stronger proof systems and relating their strengths

Definition (p-simulation)

$\mathcal{P}_1$ polynomially simulates, or p-simulates, $\mathcal{P}_2$ if there exists a polynomial-time computable function f such that for all $F \in \mathrm{TAUT}$ it holds that $\mathcal{P}_2(F, \pi) = 1$ iff $\mathcal{P}_1(F, f(\pi)) = 1$

Weak p-simulation: $cplx(\mathcal{P}_1) = (cplx(\mathcal{P}_2))^{\mathcal{O}(1)}$ but we do not know explicit translation function f from $\mathcal{P}_2$-proofs to $\mathcal{P}_1$-proofs

Polynomial Equivalence

Definition (p-equivalence)

Two propositional proof systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are polynomially equivalent, or p-equivalent, if each proof system p-simulates the other

Polynomial Equivalence

Definition (p-equivalence)

Two propositional proof systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are polynomially equivalent, or p-equivalent, if each proof system p-simulates the other

If $\mathcal{P}_1$ p-simulates $\mathcal{P}_2$ but $\mathcal{P}_2$ does not (even weakly) p-simulate $\mathcal{P}_1$, then $\mathcal{P}_1$ is strictly stronger than $\mathcal{P}_2$

Polynomial Equivalence

Definition (p-equivalence)

Two propositional proof systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are polynomially equivalent, or p-equivalent, if each proof system p-simulates the other

If $\mathcal{P}_1$ p-simulates $\mathcal{P}_2$ but $\mathcal{P}_2$ does not (even weakly) p-simulate $\mathcal{P}_1$, then $\mathcal{P}_1$ is strictly stronger than $\mathcal{P}_2$

Lots of results proven relating strength of different proof systems
Will see some examples in this course

A Fundamental Theoretical Problem. . .

The constructive version of the problem:

Problem

Given a propositional logic formula F, can we decide efficiently whether is it true no matter how we assign values to its variables?

A Fundamental Theoretical Problem...

The constructive version of the problem:

Problem

Given a propositional logic formula F, can we decide efficiently whether is it true no matter how we assign values to its variables?

TAUT: Fundamental problem in theoretical computer science ever since the discovery of NP-completeness [Coo71, Lev73]

A Fundamental Theoretical Problem...

The constructive version of the problem:

Problem

Given a propositional logic formula F, can we decide efficiently whether is it true no matter how we assign values to its variables?

TAUT: Fundamental problem in theoretical computer science ever since the discovery of NP-completeness [Coo71, Lev73]

And significance realized much earlier — cf. Gödel's famous letter to von Neumann in 1956 (`rjlipton.wordpress.com/the-gdel-letter`)

A Fundamental Theoretical Problem...

The constructive version of the problem:

Problem

Given a propositional logic formula F, can we decide efficiently whether is it true no matter how we assign values to its variables?

TAUT: Fundamental problem in theoretical computer science ever since the discovery of NP-completeness [Coo71, Lev73]

And significance realized much earlier — cf. Gödel's famous letter to von Neumann in 1956 (`rjlipton.wordpress.com/the-gdel-letter`)

These days recognized as one of the main challenges for all of mathematics — one of the million dollar "Millennium Problems" of the Clay Mathematics Institute [Mil00]

. . . with Huge Practical Implications

- All known algorithms run in exponential time in worst case
- But enormous progress on applied computer programs last 30 years (see, e.g., [BS97, MS99, MMZ$^+$01, ES04, AS09, Bie10] or [BHvMW21] for more comprehensive references)
- These so-called SAT solvers are routinely deployed to solve large-scale real-world problems with 100 000s or even 1 000 000s of variables
- Used in, e.g., hardware verification, software testing, software package management, artificial intelligence, cryptography, bioinformatics, operations research, railway signalling systems, et cetera (and even in pure mathematics)
- But we also know small example formulas with only hundreds of variables that trip up even state-of-the-art SAT solvers

Automated Theorem Proving or SAT Solving

Reason 2 for proof complexity: understand proof systems used for solving formulas occurring in "real-world applications"

Automated Theorem Proving or SAT Solving

Reason 2 for proof complexity: understand proof systems used for solving formulas occurring in "real-world applications"

Approach:

- Study proof systems used by SAT solvers
- Model actual methods of reasoning used by SAT solvers as "refinements" (subsystems) of these systems
- Prove upper and lower bounds in these systems
- Try to explain or predict theoretically what happens in practice

Automated Theorem Proving or SAT Solving

Reason 2 for proof complexity: understand proof systems used for solving formulas occurring in "real-world applications"

Approach:

- Study proof systems used by SAT solvers
- Model actual methods of reasoning used by SAT solvers as "refinements" (subsystems) of these systems
- Prove upper and lower bounds in these systems
- Try to explain or predict theoretically what happens in practice

Interesting and (arguably) important questions
But messy reality is hard to model with clean mathematics...

Proof Search Algorithms and Automatability

Proof search algorithm $A_{\mathcal{P}}$ for propositional proof system $\mathcal{P}$:
Deterministic algorithm with

- input: formula F
- output: $\mathcal{P}$-proof π of F or report that F is falsifiable

Proof Search Algorithms and Automatability

Proof search algorithm $A_{\mathcal{P}}$ for propositional proof system $\mathcal{P}$:
Deterministic algorithm with

- input: formula F
- output: $\mathcal{P}$-proof π of F or report that F is falsifiable

Definition (Automatizability)

$\mathcal{P}$ is automatizable if there exists a proof search algorithm $A_{\mathcal{P}}$ such that if $F \in \mathrm{TAUT}$ then $A_{\mathcal{P}}$ on input F outputs a $\mathcal{P}$-proof of F in time polynomial in size of F plus size of a smallest $\mathcal{P}$-proof of F

Short Proofs Seem Hard to Find (at Least in Theory)

Example (Truth table continued)

Truth table is (trivially) an automatizable propositional proof system (but the proofs we find are of exponential size, so this is not very exciting)

Short Proofs Seem Hard to Find (at Least in Theory)

Example (Truth table continued)

Truth table is (trivially) an automatizable propositional proof system (but the proofs we find are of exponential size, so this is not very exciting)

We want proof systems that are both

- strong (i.e., have short proofs for all tautologies) and
- automatizable (i.e., we can find these short proofs efficiently)

Seems that this is not possible unless P = NP [AM20]

But can find proof search algorithms that work really well "in practice"

Potential and Limitations of Mathematical Reasoning

Reason 3 for proof complexity: understand how deep / hard various mathematical truths are

- Look at logic encoding of various mathematical theorems (e.g., combinatorial principles such as pigeonhole principle, least number principle, handshaking lemma, et cetera)
- Determine how strong proof systems are needed to provide efficient proofs
- Tells us how powerful mathematical tools are needed for establishing such statements

Potential and Limitations of Mathematical Reasoning

Reason 3 for proof complexity: understand how deep / hard various mathematical truths are

- Look at logic encoding of various mathematical theorems (e.g., combinatorial principles such as pigeonhole principle, least number principle, handshaking lemma, et cetera)
- Determine how strong proof systems are needed to provide efficient proofs
- Tells us how powerful mathematical tools are needed for establishing such statements

Fascinating questions that are systematically explored in bounded arithmetic
Some of the results we will cover are tangentially related, but this is not our main focus

Transforming Tautologies to Unsatisfiable CNF Formulas

Any propositional logic formula F can be converted to formula F' in conjunctive normal form (CNF) such that

- F' only linearly larger than F
- F' unsatisfiable if and only if ("iff") F tautology

Transforming Tautologies to Unsatisfiable CNF Formulas

Any propositional logic formula F can be converted to formula F' in conjunctive normal form (CNF) such that

- F' only linearly larger than F
- F' unsatisfiable if and only if ("iff") F tautology

Approach by Tseitin [Tse68]:

- Introduce new variable x_G for each subformula $G \doteq H_1 \circ H_2$ in F, $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$
- Translate G to set of disjunctive clauses $Cl(G)$ which enforces that truth value of x_G is computed correctly given x_{H_1} and x_{H_2}

Sketch of Transformation

Two examples for $\lor$ and $\rightarrow$ ($\land$ and $\leftrightarrow$ are analogous):

$$G \equiv H_1 \lor H_2 : \qquad Cl(G) := \begin{aligned}&(\neg x_G \lor x_{H_1} \lor x_{H_2})\\ \land\ &(x_G \lor \neg x_{H_1})\\ \land\ &(x_G \lor \neg x_{H_2})\end{aligned}$$

$$G \equiv H_1 \rightarrow H_2 : \qquad Cl(G) := \begin{aligned}&(\neg x_G \lor \neg x_{H_1} \lor x_{H_2})\\ \land\ &(x_G \lor x_{H_1})\\ \land\ &(x_G \lor \neg x_{H_2})\end{aligned}$$

- Finally, add clause $\neg x_F$

Proof Systems for Refuting Unsatisfiable CNFs

- Easy to verify that constructed CNF formula F' is unsatisfiable iff F is a tautology
- So any sound and complete proof system which produces refutations of formulas in CNF can be used as a propositional proof system
- From now on and for the rest of this course, we will focus exclusively on proof systems for refuting CNF formulas

Proof Systems for Refuting Unsatisfiable CNFs

- Easy to verify that constructed CNF formula F' is unsatisfiable iff F is a tautology
- So any sound and complete proof system which produces refutations of formulas in CNF can be used as a propositional proof system
- From now on and for the rest of this course, we will focus exclusively on proof systems for refuting CNF formulas

Warning:

- Because of this duality, proof complexity terminology is slightly schizophrenic
- Unsatisfiable formulas sometimes referred to as "tautologies" in the literature
- We won't go quite that far...
- But throughout the course "proof" and "refutation" will be synonyms

Sequential Proof Systems

Proof system could be any polynomial-time computable predicate...
But often natural to view proof as sequence of derivation steps

Sequential Proof Systems

Proof system could be any polynomial-time computable predicate. . .
But often natural to view proof as sequence of derivation steps

More formally, a proof system $\mathcal{P}$ is sequential if a proof π in $\mathcal{P}$ is a

- sequence of lines $\pi = \{L_1, \ldots, L_\tau\}$
- of some prescribed syntactic form (depending on the proof system in question)
- where each line is derived from previous lines by one of a finite set of allowed inference rules

Sequential Proof Systems

Proof system could be any polynomial-time computable predicate...
But often natural to view proof as sequence of derivation steps

More formally, a proof system $\mathcal{P}$ is sequential if a proof π in $\mathcal{P}$ is a

- sequence of lines $\pi = \{L_1, \ldots, L_\tau\}$
- of some prescribed syntactic form (depending on the proof system in question)
- where each line is derived from previous lines by one of a finite set of allowed inference rules

We will mostly study sequential proof systems in this course

The Resolution Proof System

Resolution:

- Most well-studied proof system in all of proof complexity
- Originally described by Blake [Bla37]
- Used in the context of SAT solving [DP60, DLL62, Rob65]
- Still the basis of state-of-the-art SAT solvers

The Resolution Proof System

Resolution:

- Most well-studied proof system in all of proof complexity
- Originally described by Blake [Bla37]
- Used in the context of SAT solving [DP60, DLL62, Rob65]
- Still the basis of state-of-the-art SAT solvers

Lines in refutation are disjunctive clauses

The Resolution Proof System

Resolution:

- Most well-studied proof system in all of proof complexity
- Originally described by Blake [Bla37]
- Used in the context of SAT solving [DP60, DLL62, Rob65]
- Still the basis of state-of-the-art SAT solvers

Lines in refutation are disjunctive clauses

Just one inference rule, the resolution rule:

$$\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$$

$B \vee C$ is the resolvent of $B \vee x$ and $C \vee \overline{x}$

Soundness and Completeness of Resolution

Resolution derivation π from CNF formula F:

- Start with clauses in F
- Interatively derive new clauses by resolution rule and add
- Final clause in π is $A \Leftrightarrow \pi$ is derivation of A (notation: $\pi : F \vdash A$)

Soundness and Completeness of Resolution

Resolution derivation π from CNF formula F:

- Start with clauses in F
- Interatively derive new clauses by resolution rule and add
- Final clause in π is $A \Leftrightarrow \pi$ is derivation of A (notation: $\pi : F \vdash A$)

Resolution is:

Sound If there is a resolution derivation $\pi : F \vdash A$ then $F \vDash A$ (easy to show)

Complete If $F \vDash A$ then there is a resolution derivation $\pi : F \vdash A'$ for some $A' \subseteq A$ (not hard to prove, but we will skip this)

Soundness and Completeness of Resolution

Resolution derivation π from CNF formula F:

- Start with clauses in F
- Interatively derive new clauses by resolution rule and add
- Final clause in π is $A \Leftrightarrow \pi$ is derivation of A (notation: $\pi : F \vdash A$)

Resolution is:

Sound If there is a resolution derivation $\pi : F \vdash A$ then $F \vDash A$ (easy to show)

Complete If $F \vDash A$ then there is a resolution derivation $\pi : F \vdash A'$ for some $A' \subseteq A$ (not hard to prove, but we will skip this)

In particular:

$$F \text{ is unsatisfiable}$$
$$\Updownarrow$$
$$\exists \text{ resolution refutation of } F = \text{derivation of unsatisfiable empty clause } \bot$$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

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

- Refutation/proof ends when empty clause $\perp$ derived

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

1. $x \lor y$
2. $x \lor \overline{y} \lor z$
3. $\overline{x} \lor z$
4. $\overline{y} \lor \overline{z}$
5. $\overline{x} \lor \overline{z}$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

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

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\bot$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$\boldsymbol{x \lor \overline{y} \lor z}$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\boldsymbol{\overline{y} \lor \overline{z}}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$\boldsymbol{x \lor \overline{y}}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$\boldsymbol{x \lor \overline{y}}$	Res$(2,4)$
7.	x	Res$(1,6)$
8.	$\overline{x}$	Res$(3,5)$
9.	$\perp$	Res$(7,8)$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$\boldsymbol{x \lor y}$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$\boldsymbol{x \lor \overline{y}}$	Res$(2,4)$
7.	x	Res$(1,6)$
8.	$\overline{x}$	Res$(3,5)$
9.	$\perp$	Res$(7,8)$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\bot$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2,4)$
7.	$\boldsymbol{x}$	Res$(1,6)$
8.	$\overline{x}$	Res$(3,5)$
9.	$\perp$	Res$(7,8)$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

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

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\boldsymbol{\overline{x} \lor z}$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\boldsymbol{\overline{x} \lor \overline{z}}$	Axiom
6.	$x \lor \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\boldsymbol{\overline{x}}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2,4)$
7.	x	Res$(1,6)$
8.	$\overline{x}$	Res$(3,5)$
9.	$\perp$	Res$(7,8)$

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

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

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\bot$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Example Resolution Refutation

Recap of set-up:

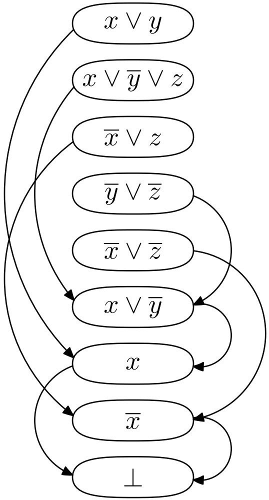
- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph



Example Resolution Refutation

Recap of set-up:

- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

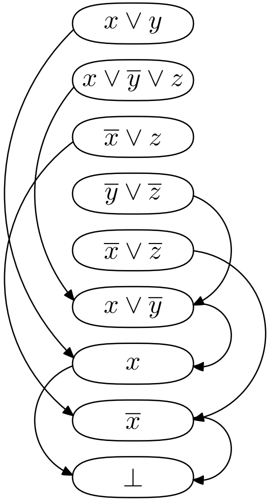
$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

Tree-like resolution if DAG is tree



Resolution Length and Size

Length $=$ # clauses in resolution refutation (9 in our example)

Resolution Length and Size

Length $=$ # clauses in resolution refutation (9 in our example)

Size $=$ total # literals in refutation, strictly speaking

Resolution Length and Size

Length $=$ # clauses in resolution refutation (9 in our example)

Size $=$ total # literals in refutation, strictly speaking

In practice, ignore linear factor and set size $=$ length for resolution

Resolution Length and Size

Length $=$ # clauses in resolution refutation (9 in our example)

Size $=$ total # literals in refutation, strictly speaking

In practice, ignore linear factor and set size $=$ length for resolution

Proof size/length is the most fundamental measure in proof complexity
Main complexity measure of interest in this course

Resolution Space

Space = amount of memory needed when performing refutation

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Resolution Space

Space = amount of memory needed when performing refutation

Can be measured in different ways:

- clause space
- total space

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Resolution Space

Space = amount of memory needed when performing refutation

Can be measured in different ways:

- clause space
- total space

Clause space at step t: $\#$ clauses at steps $\leq t$ used at steps $\geq t$

Total space at step t: Count also literals

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

Resolution Space

Space = amount of memory needed when performing refutation

Can be measured in different ways:

- clause space
- total space

Clause space at step t: # clauses at steps $\leq t$ used at steps $\geq t$
Total space at step t: Count also literals

Example: Line space at step 7

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

Resolution Space

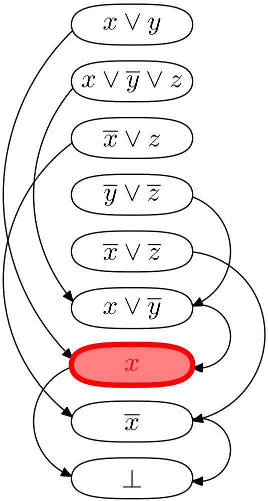
Space = amount of memory needed when performing refutation

Can be measured in different ways:

- clause space
- total space

Clause space at step t: $\#$ clauses at steps $\leq t$ used at steps $\geq t$
Total space at step t: Count also literals

Example: Line space at step 7



Resolution Space

Space = amount of memory needed when performing refutation

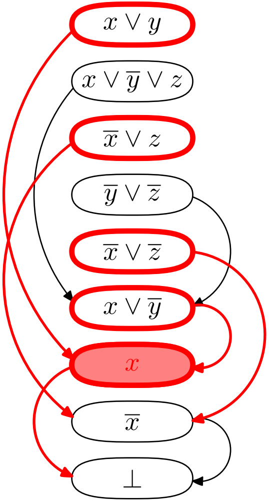
Can be measured in different ways:

- clause space
- total space

Clause space at step t: $\#$ clauses at steps $\leq t$ used at steps $\geq t$

Total space at step t: Count also literals

Example: Line space at step 7 is 5



Resolution Space

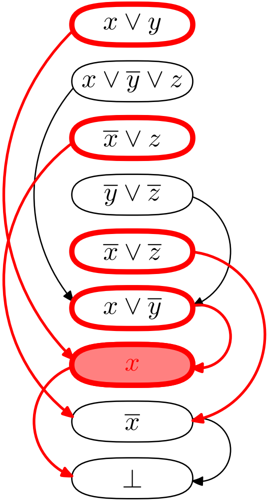
Space = amount of memory needed when performing refutation

Can be measured in different ways:

- clause space
- total space

Clause space at step t: $\#$ clauses at steps $\leq t$ used at steps $\geq t$
Total space at step t: Count also literals

Example: Line space at step 7 is 5
Total space at step 7 is 9



Resolution Space

Space = amount of memory needed when performing refutation

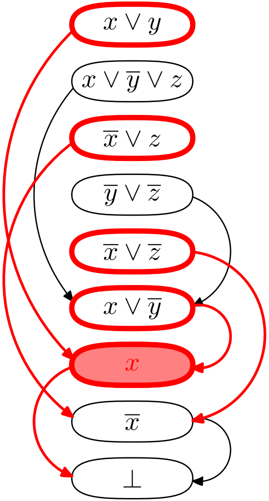
Can be measured in different ways:

- clause space
- total space

Clause space at step t: $\#$ clauses at steps $\leq t$ used at steps $\geq t$
Total space at step t: Count also literals

Example: Line space at step 7 is 5
Total space at step 7 is 9

Space of refutation: Max over all steps



Refutation Size and Space

For any unsatisfiable CNF formula F and any proof system $\mathcal{P}$:

Size of refuting F = size of smallest $\mathcal{P}$-refutation of F

Clause space of refuting F = max # lines in memory in most space-efficient $\mathcal{P}$-refutation of F

Total space of refuting F = max # literals in memory in most space-efficient $\mathcal{P}$-refutation of F

Refutation Size and Space

For any unsatisfiable CNF formula F and any proof system $\mathcal{P}$:

Size of refuting F = size of smallest $\mathcal{P}$-refutation of F

Clause space of refuting F = max # lines in memory in most space-efficient $\mathcal{P}$-refutation of F

Total space of refuting F = max # literals in memory in most space-efficient $\mathcal{P}$-refutation of F

Interesting to study:

- size bounds (≈ SAT solver running time)
- space bounds (≈ SAT solver memory usage)
- size-space trade-offs (because solvers aggressively minimize both)

How to Prove Size/Length Lower Bounds

- Find suitable family of unsatisfiable CNF formulas with size scaling polynomially
- Show that smallest possible refutations in proof system $\mathcal{P}$ of these formulas scale superpolynomially or even exponentially
- How to prove this? Have to establish that no short proofs exist, even totally crazy ones!
- In order to do so, need to understand formulas really well
- So the formulas we know how to prove lower bounds for are mostly formulas that look very easy to humans
- A bit of a paradox... Let's now turn to the most famous formula family

Pigeonhole Principle (PHP) Formulas

"$n + 1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon i goes into hole j", $i \in [n+1]$, $j \in [n]$

Pigeonhole Principle (PHP) Formulas

"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon i goes into hole j", $i \in [n+1]$, $j \in [n]$

$p_{i,1} \lor p_{i,2} \lor \cdots \lor p_{i,n}$	[every pigeon i gets a hole]
$\overline{p}_{i,j} \lor \overline{p}_{i',j}$	[no hole j gets two pigeons $i \neq i'$]

Can also add "functionality" and/or "onto" axioms

$\overline{p}_{i,j} \lor \overline{p}_{i,j'}$	[no pigeon i gets two holes $j \neq j'$]
$p_{1,j} \lor p_{2,j} \lor \cdots \lor p_{n+1,j}$	[every hole j gets a pigeon]

Pigeonhole Principle (PHP) Formulas

"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon i goes into hole j", $i \in [n+1]$, $j \in [n]$

$p_{i,1} \lor p_{i,2} \lor \cdots \lor p_{i,n}$	[every pigeon i gets a hole]
$\overline{p}_{i,j} \lor \overline{p}_{i',j}$	[no hole j gets two pigeons $i \neq i'$]

Can also add "functionality" and/or "onto" axioms

$\overline{p}_{i,j} \lor \overline{p}_{i,j'}$	[no pigeon i gets two holes $j \neq j'$]
$p_{1,j} \lor p_{2,j} \lor \cdots \lor p_{n+1,j}$	[every hole j gets a pigeon]

All versions are hard for resolution [Hak85]

We will give a proof for the simplest PHP version following the exposition in [Pud00]

References I

[AM20] Albert Atserias and Moritz Müller. Automating resolution is NP-hard. *Journal of the ACM*, 67(5):31:1–31:17, October 2020. Preliminary version in *FOCS '19*.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 399–404, July 2009.

[BHvMW21] 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*. IOS Press, 2nd edition, February 2021.

[Bie10] Armin Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical Report 10/1, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, August 2010.

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

[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.

References II

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, May 1971.

[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.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

References III

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. In Russian. Available at `http://mi.mathnet.ru/ppi914`.

[Mil00] The Millennium Problems of the Clay Mathematics Institute, May 2000. See `https://www.claymath.org/millennium-problems`.

[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*.

[Pud00] Pavel Pudlák. Proofs as games. *American Mathematical Monthly*, pages 541–550, 2000.

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

References IV

[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York-London, 1968.