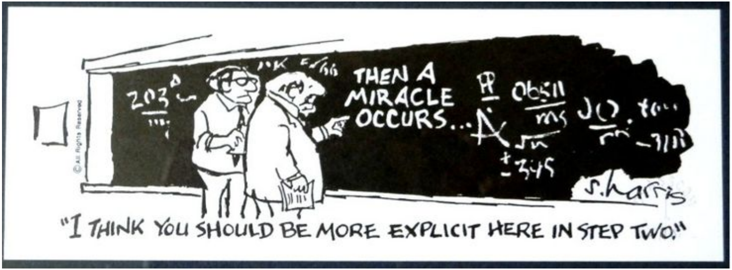
DD2442 Seminars on TCS: Proof Complexity
Lecture 1: Introduction, Overview, and Practicalities

Jakob Nordström

Theoretical Computer Science Group
KTH Royal Institute of Technology

Monday September 5, 2016

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, structural complexity, et cetera)
- How are these notions related?

Today's Lecture

- Brief (and therefore biased) introduction to proof complexity
- Even briefer discussion of connections to neighbouring areas such as computational complexity theory and SAT solving
- Some concrete examples of interesting methods of reasoning and interesting formulas to reason about
- Some "teasers" for what to expect in coming lectures
- Might go slightly fast, but
  - slides will be online to allow recap
  - most things won't be crucially needed for upcoming lectures

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

- $$\begin{aligned} 25957 &\equiv 1 \pmod{2} \\ 25957 &\equiv 1 \pmod{3} \\ 25957 &\equiv 2 \pmod{5} \\ &\vdots \\ 25957 &\equiv 19 \pmod{99} \end{aligned} \qquad \begin{aligned} 25957 &\equiv 0 \pmod{101} \\ 25957 &\equiv 1 \pmod{103} \\ &\vdots \\ 25957 &\equiv 0 \pmod{257} \\ &\vdots \end{aligned}$$
  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

- $$\begin{aligned} 25957 &\equiv 1 \pmod{2} & 25957 &\equiv 0 \pmod{101} \\ 25957 &\equiv 1 \pmod{3} & 25957 &\equiv 1 \pmod{103} \\ 25957 &\equiv 2 \pmod{5} & &\vdots \\ &\vdots & 25957 &\equiv 0 \pmod{257} \\ 25957 &\equiv 19 \pmod{99} & &\vdots \end{aligned}$$
  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) such that $\mathcal{P}(x, \pi) = 1$
- 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) such that $\mathcal{P}(x, \pi) = 1$
- 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) such that $\mathcal{P}(x, \pi) = 1$
- 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 $\textsc{Taut}$ 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 $\land$
- disjunction $\lor$
- 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 $\textsc{Prop}$ 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., $\alpha : Vars \to \{0, 1\}$

Propositional Logic: Semantics

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

Extend α from variables to formulas by:

- $\alpha(\neg F) = 1$ if $\alpha(F) = 0$
- $\alpha(F \vee G) = 1$ unless $\alpha(F) = \alpha(G) = 0$
- $\alpha(F \wedge G) = 1$ if $\alpha(F) = \alpha(G) = 1$
- $\alpha(F \rightarrow G) = 1$ 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., $\alpha : \mathit{Vars} \rightarrow \{0, 1\}$

Extend α from variables to formulas by:

- $\alpha(\neg F) = 1$ if $\alpha(F) = 0$
- $\alpha(F \vee G) = 1$ unless $\alpha(F) = \alpha(G) = 0$
- $\alpha(F \wedge G) = 1$ if $\alpha(F) = \alpha(G) = 1$
- $\alpha(F \rightarrow G) = 1$ 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	$(p \land (q \lor r)) \leftrightarrow ((p \land q) \lor (p \land r))$
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

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

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])

NP = 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.

□

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.

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

□

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.

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

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

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 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

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 $F \in \textsc{Taut}$ 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 $F \in \textsc{Taut}$ 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 mention a few 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 Stephen Cook's NP-completeness paper [Coo71]

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 Stephen Cook's NP-completeness paper [Coo71]

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 Stephen Cook's NP-completeness paper [Coo71]

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 20 years (see, e.g., [BS97, MS99, MMZ$^+$01, ES04, AS09, Bie10])
- 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

A truly fascinating area... (But I am severely biased)

- A lot of my own research is about investigating these questions
- But we will essentially ignore such connections in this course
- However, lots of good problems for, e.g., MSc theses

Proof Search Algorithms and Automatizability

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 Automatizability

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 $F \in \textsc{Taut}$ 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 (under reasonable complexity assumptions)

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

Fascinating area, but this course will not go into this at all

Transforming Tautologies to Unsatisfiable CNFs

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 CNFs

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 $\vee$ and $\rightarrow$ ($\wedge$ and $\leftrightarrow$ are analogous):

$$G \equiv H_1 \vee H_2 : \qquad Cl(G) := \left(\neg x_G \vee x_{H_1} \vee x_{H_2}\right) \wedge \left(x_G \vee \neg x_{H_1}\right) \wedge \left(x_G \vee \neg x_{H_2}\right)$$

$$G \equiv H_1 \rightarrow H_2 : \qquad Cl(G) := \left(\neg x_G \vee \neg x_{H_1} \vee x_{H_2}\right) \wedge \left(x_G \vee x_{H_1}\right) \wedge \left(x_G \vee \neg x_{H_2}\right)$$

- 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

A Concrete Example

$$(x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg x \lor \neg z)$$

A Concrete Example

$$(x \vee y) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z) \wedge (\neg x \vee \neg z)$$

- Variables should be set to true ($= 1$) or false ($= 0$)

A Concrete Example

$$(x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg x \lor \neg z)$$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \lor \neg y \lor z)$: means x or z should be true or y false

A Concrete Example

$$(x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg x \lor \neg z)$$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \lor \neg y \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously

A Concrete Example

$$(x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg x \lor \neg z)$$

- Variables should be set to true $(= 1)$ or false $(= 0)$
- Constraint $(x \lor \neg y \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously

Is there a truth value assignment satisfying all these conditions?
Or can we find efficient proof that some constraint must fail to hold?

Some Notation and Terminology

- Literal a: variable x or its negation $\overline{x}$ (rather than $\neg x$); let $\overline{\overline{x}} = x$
- Sometimes write $x^1 = x$ and $x^0 = \overline{x}$ (x^b satisfied by setting $x = b$)
- Clause $C = a_1 \vee \ldots \vee a_k$: set of literals
  At most k literals: k-clause
- CNF formula $F = C_1 \wedge \ldots \wedge C_m$: set of clauses
  k-CNF formula: CNF formula consisting of k-clauses
- $Vars(\cdot)$: set of variables in clause or formula
  $Lit(\cdot)$: set of literals in clause or formula
- $F \vDash D$: semantical implication, $\alpha(F)$ true $\Rightarrow \alpha(D)$ true
  for all truth value assignments α
- $[n] = \{1, 2, \ldots, n\}$

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

Let's look at some such proof systems that we will study 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:

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

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

Using Resolution to Refute CNF Formulas

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule

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

then $F \wedge D$ is also satisfiable

Using Resolution to Refute CNF Formulas

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule

$$\frac{B \lor x \quad C \lor \overline{x}}{B \lor C} \ ,$$

then $F \land D$ is also satisfiable

So if we can use the resolution rule to derive a contradiction from F, this shows that F is unsatisfiable

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:

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

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

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

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

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

Example Resolution Refutation

Recap of set-up:

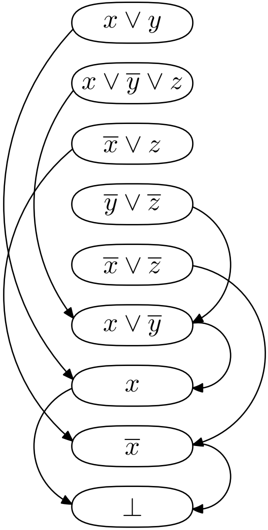
- Goal: refute **unsatisfiable** CNF
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

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

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

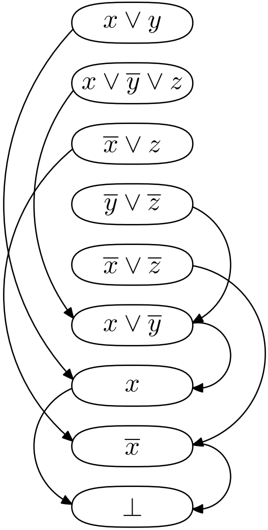
$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Refutation/proof ends when empty clause $\bot$ 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.	$\bot$	Res$(7, 8)$

Resolution Space

Space = amount of memory needed when performing refutation

Can be measured in different ways:

- line space (or 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.	$\bot$	Res$(7, 8)$

Resolution Space

Space = amount of memory needed when performing refutation

Can be measured in different ways:

- line space (or clause space)
- total space

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

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:

- line space (or clause space)
- total space

Line 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.	$x \lor y$	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

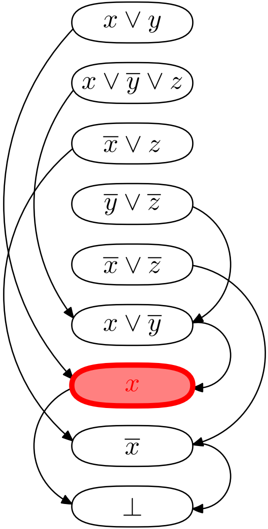
Space = amount of memory needed when performing refutation

Can be measured in different ways:

- line space (or clause space)
- total space

Line 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

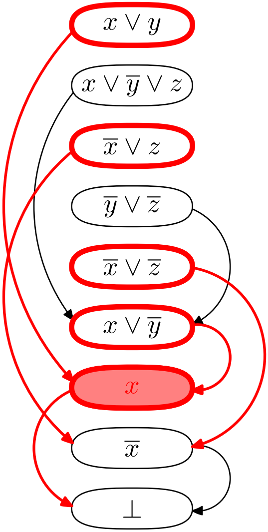
Space = amount of memory needed when performing refutation

Can be measured in different ways:

- line space (or clause space)
- total space

Line 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

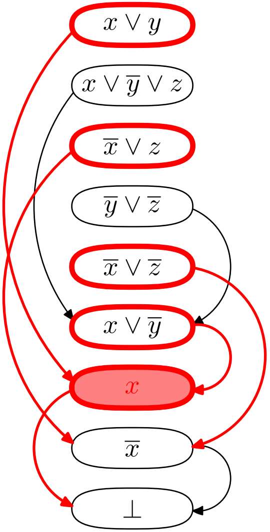
Space = amount of memory needed when performing refutation

Can be measured in different ways:

- line space (or clause space)
- total space

Line 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

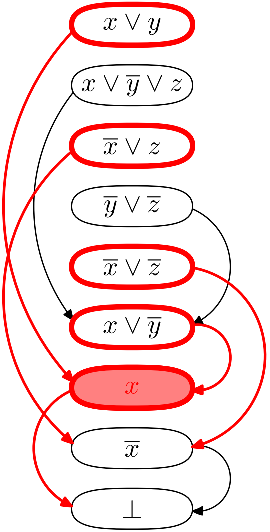
Can be measured in different ways:

- line space (or clause space)
- total space

Line 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 Length, Size, and Space

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

Length of refuting F $=$ length of shortest $\mathcal{P}$-refutation of F
Size of refuting F $=$ size of smallest $\mathcal{P}$-refutation of F
Line 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 Length, Size, and Space

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

Length of refuting F = length of shortest $\mathcal{P}$-refutation of F
Size of refuting F = size of smallest $\mathcal{P}$-refutation of F
Line 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 ($\approx$ SAT solver running time)
- space bounds ($\approx$ SAT solver memory usage)
- size-space trade-offs (because solvers aggressively minimize both)

Generalizing Resolution to k-DNF Formulas

Family of proof systems $\mathcal{R}(k)$ parameterized by $k \in \mathbb{N}^+$ [Kra01]
($\mathcal{R}(1)$ is resolution)

Generalizing Resolution to k-DNF Formulas

Family of proof systems $\mathcal{R}(k)$ parameterized by $k \in \mathbb{N}^+$ [Kra01]
($\mathcal{R}(1)$ is resolution)

Lines in k-DNF-resolution refutation are k-DNF formulas
i.e., disjunctions of conjunctions (terms) of size $\leq k$

Generalizing Resolution to k-DNF Formulas

Family of proof systems $\mathcal{R}(k)$ parameterized by $k \in \mathbb{N}^+$ [Kra01] ($\mathcal{R}(1)$ is resolution)

Lines in k-DNF-resolution refutation are k-DNF formulas i.e., disjunctions of conjunctions (terms) of size $\leq k$

Inference rules

Notation: G, H k-DNF formulas; T, T' k-terms; $a_1, \ldots, a_k$ literals:

k-cut $$\frac{(a_1 \wedge \cdots \wedge a_{k'}) \vee G \qquad \overline{a}_1 \vee \cdots \vee \overline{a}_{k'} \vee H}{G \vee H}, \ (k' \leq k)$$

$\wedge$-introduction $$\frac{G \vee T \qquad G \vee T'}{G \vee (T \wedge T')}$$, as long as $|T \cup T'| \leq k$

$\wedge$-elimination $$\frac{G \vee T}{G \vee T'}$$ for any $T' \subseteq T$

Weakening $$\frac{G}{G \vee H}$$ for any k-DNF formula H

k-DNF Resolution Measures

Length
\# derivation steps
(= # k-DNF formulas counted with repetitions)

Size
Total # literals in proof counted with repetitions

Line space (or formula space)
Max # k-DNF formulas in memory (analogue of clause space)

Total space
Max total # literals in memory counted with repetitions

Cutting Planes: Informal Description

- Geometric proof system introduced in [CCT87]

Cutting Planes: Informal Description

- Geometric proof system introduced in [CCT87]
- Translate clauses to linear inequalities for real variables in $[0, 1]$

Cutting Planes: Informal Description

- Geometric proof system introduced in [CCT87]
- Translate clauses to linear inequalities for real variables in $[0, 1]$
- For instance, $x \vee y \vee \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$, i.e., $x + y - z \geq 0$

Cutting Planes: Informal Description

- Geometric proof system introduced in [CCT87]
- Translate clauses to linear inequalities for real variables in $[0, 1]$
- For instance, $x \vee y \vee \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$, i.e., $x + y - z \geq 0$
- Manipulate linear inequalities to derive contradiction $0 \geq 1$

Cutting Planes: Inference Rules

Lines in cutting planes (CP) refutation: linear inequalities with integer coefficients

Cutting Planes: Inference Rules

Lines in cutting planes (CP) refutation: linear inequalities with integer coefficients

Inference rules

Variable axioms $\dfrac{}{x \geq 0}$ and $\dfrac{}{-x \geq -1}$ for all variables x

Addition $$\frac{\sum a_i x_i \geq A \qquad \sum b_i x_i \geq B}{\sum (a_i + b_i) x_i \geq A + B}$$

Multiplication $$\frac{\sum a_i x_i \geq A}{\sum c a_i x_i \geq cA}$$ for $c \in \mathbb{N}^+$

Division $$\frac{\sum c a_i x_i \geq A}{\sum a_i x_i \geq \lceil A/c \rceil}$$ for $c \in \mathbb{N}^+$

Cutting Planes: Inference Rules

Lines in cutting planes (CP) refutation: linear inequalities with integer coefficients

Inference rules

Variable axioms $\dfrac{}{x \geq 0}$ and $\dfrac{}{-x \geq -1}$ for all variables x

Addition $\dfrac{\sum a_i x_i \geq A \qquad \sum b_i x_i \geq B}{\sum (a_i + b_i) x_i \geq A + B}$

Multiplication $\dfrac{\sum a_i x_i \geq A}{\sum c a_i x_i \geq cA}$ for $c \in \mathbb{N}^+$

Division $\dfrac{\sum c a_i x_i \geq A}{\sum a_i x_i \geq \lceil A/c \rceil}$ for $c \in \mathbb{N}^+$

A CP refutation ends when the inequality $0 \geq 1$ has been derived

Cutting Planes Measures

Length
derivation steps

Size
symbols needed to represent proof (coefficients can be huge)

Line space
Max # linear inequalities in memory (analogue of clause space)

Total space
Max total # variables in memory counted with repetitions
$+ \log$ of coefficients

Polynomial Calculus: Using Algebra to Reason About CNFs

- Algrebraic system introduced in [CEI96] under the name of "Gröbner proof system"

Polynomial Calculus: Using Algebra to Reason About CNFs

- Algrebraic system introduced in [CEI96] under the name of "Gröbner proof system"
- Clauses are interpreted as multilinear polynomial equations over some field $\mathbb{F}$ (typically finite; $\mathrm{GF}(2)$ perhaps most interesting)

Polynomial Calculus: Using Algebra to Reason About CNFs

- Algrebraic system introduced in [CEI96] under the name of "Gröbner proof system"
- Clauses are interpreted as multilinear polynomial equations over some field $\mathbb{F}$ (typically finite; $\mathrm{GF}(2)$ perhaps most interesting)
- Here, natural to flip convention and think of 0 as true and 1 as false

Polynomial Calculus: Using Algebra to Reason About CNFs

- Algrebraic system introduced in [CEI96] under the name of "Gröbner proof system"
- Clauses are interpreted as multilinear polynomial equations over some field $\mathbb{F}$ (typically finite; $\mathrm{GF}(2)$ perhaps most interesting)
- Here, natural to flip convention and think of 0 as true and 1 as false
- For instance, clause $x \lor y \lor \overline{z}$ gets translated to $xy(1-z) = 0$ or $xy - xyz = 0$

Polynomial Calculus: Using Algebra to Reason About CNFs

- Algrebraic system introduced in [CEI96] under the name of "Gröbner proof system"
- Clauses are interpreted as multilinear polynomial equations over some field $\mathbb{F}$ (typically finite; $\mathrm{GF}(2)$ perhaps most interesting)
- Here, natural to flip convention and think of 0 as true and 1 as false
- For instance, clause $x \vee y \vee \overline{z}$ gets translated to $xy(1-z) = 0$ or $xy - xyz = 0$
- Derive contradiction by showing that there is no common root for the polynomial equations corresponding to all the clauses

Polynomial Calculus: Inference Rules

Lines in polynomial calculus (PC) refutation: multivariate polynomial equations $p = 0$, where $p \in \mathbb{F}[x, y, z, \ldots]$

Polynomial Calculus: Inference Rules

Lines in polynomial calculus (PC) refutation: multivariate polynomial equations $p = 0$, where $p \in \mathbb{F}[x, y, z, \ldots]$

Customary to omit "$= 0$" and only write p

Polynomial Calculus: Inference Rules

Lines in polynomial calculus (PC) refutation: multivariate polynomial equations $p = 0$, where $p \in \mathbb{F}[x, y, z, \ldots]$

Customary to omit "$= 0$" and only write p

Inference rules

Notation: $\alpha, \beta \in \mathbb{F}$; $p, q \in \mathbb{F}[x, y, z, \ldots]$; x is any variable:

Variable axioms $\dfrac{}{x^2 - x}$ for all variables x (forcing 0/1-solutions)

Linear combination $\dfrac{p \quad\quad q}{\alpha p + \beta q}$

Multiplication $\dfrac{p}{xp}$

Polynomial Calculus: Inference Rules

Lines in polynomial calculus (PC) refutation: multivariate polynomial equations $p = 0$, where $p \in \mathbb{F}[x, y, z, \ldots]$

Customary to omit "$= 0$" and only write p

Inference rules

Notation: $\alpha, \beta \in \mathbb{F}$; $p, q \in \mathbb{F}[x, y, z, \ldots]$; x is any variable:

Variable axioms $\dfrac{}{x^2 - x}$ for all variables x (forcing 0/1-solutions)

Linear combination $\dfrac{p \qquad q}{\alpha p + \beta q}$

Multiplication $\dfrac{p}{xp}$

A PC refutation ends when 1 has been derived (i.e., $1 = 0$)
(Note that multilinearity follows w.l.o.g. from $x^2 = x$ for all variables x)

Polynomial Calculus: Alternative View

Can also (equivalently) consider a PC refutation to be a calculation in the ideal generated by polynomials corresponding to clauses

Then a refutation concludes by proving that 1 is in this ideal, i.e., that the ideal is everything

Clearly implies that there is no common root

Less obvious: if there is no common root, then 1 is always in the ideal (requires some algebra)

Polynomial Calculus Measures

Length

\# derivation steps
(= # polynomial equations counted with repetitions)
Turns out to not make too much sense; exponentially large polynomials are too powerful

Size

Total # monomials in the refutation counted with repetitions
(Ignore linear factor here; in the same spirit as resolution)

Monomial space

Max # monomials in memory counted with repetitions
(Again an analogue of clause space; line space doesn't make sense for same reason as length)

Total space

Max total # variables in memory counted with repetitions

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 see some examples

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

- Resolution: All versions hard [Hak85] *(next lecture)*
- Polynomial calculus: Onto FPHP easy [Rii93]; other versions hard [Raz98, IPS99, MN15]
- Cutting planes: All versions easy [CCT87]

Tseitin Formulas

"Sum of degrees of vertices in graph is even" (handshaking lemma)

Variables = edges (in undirected graph of bounded degree)

Tseitin Formulas

"Sum of degrees of vertices in graph is even" (handshaking lemma)

Variables = edges (in undirected graph of bounded degree)

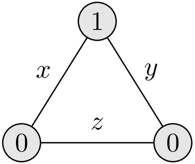
- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label

Tseitin Formulas

"Sum of degrees of vertices in graph is even" (handshaking lemma)

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



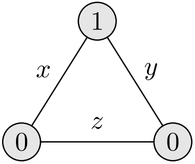
$$
\begin{array}{ll}
(x \lor y) & \land\ (\overline{x} \lor z) \\
\land\ (\overline{x} \lor \overline{y}) & \land\ (y \lor \overline{z}) \\
\land\ (x \lor \overline{z}) & \land\ (\overline{y} \lor z)
\end{array}
$$

Tseitin Formulas

"Sum of degrees of vertices in graph is even" (handshaking lemma)

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



$$
\begin{array}{ll}
(x \lor y) & \land\ (\overline{x} \lor z) \\
\land\ (\overline{x} \lor \overline{y}) & \land\ (y \lor \overline{z}) \\
\land\ (x \lor \overline{z}) & \land\ (\overline{y} \lor z)
\end{array}
$$

- Resolution: Hard for well-connected expander graphs [Urq87] *(in two lectures)*
- Polynomial calculus: Easy if field is $\mathrm{GF}(2)$
- Cutting planes: Believed hard; **big** open problem

Subset Cardinality Formulas

Variables = 1s in matrix with four 1s per row/column + extra 1
Row ⇒ majority of variables true; column ⇒ majority false

$$
\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}
$$

$$
\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}
$$

Subset Cardinality Formulas

Variables = 1s in matrix with four 1s per row/column **+ extra 1**
Row ⇒ majority of variables true; column ⇒ majority false

$$
\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}
$$

$$
\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}
$$

Subset Cardinality Formulas

Variables = 1s in matrix with four 1s per row/column **+ extra 1**
Row ⇒ majority of variables true; column ⇒ majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

Subset Cardinality Formulas

Variables = 1s in matrix with four 1s per row/column **+ extra 1**

Row ⇒ majority of variables true; column ⇒ majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

Subset Cardinality Formulas

Variables = 1s in matrix with four 1s per row/column **+ extra 1**

Row ⇒ majority of variables true; column ⇒ majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
&(x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ &(x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ &(x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ &(x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
&\vdots \\
\land\ &(\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ &(\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ &(\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ &(\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

- Resolution: Hard for expanding matrices (1s well spread out) [MN14]
- Polynomial calculus: Ditto [MN14]
- Cutting planes: Easy (not hard to show)

Random k-CNF Formulas

Δn randomly sampled k-clauses over n variables

Random k-CNF Formulas

Δn randomly sampled k-clauses over n variables

I.e., repeat Δn times:

- Pick randomly set of k out of n variables
- Choose randomly for each picked variable whether it should be negated or unnegated
- Add clause to formula under construction

Random k-CNF Formulas

Δn randomly sampled k-clauses over n variables

I.e., repeat Δn times:

- Pick randomly set of k out of n variables
- Choose randomly for each picked variable whether it should be negated or unnegated
- Add clause to formula under construction

Fact: $\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF asymptotically almost surely

Random k-CNF Formulas

Δn randomly sampled k-clauses over n variables

I.e., repeat Δn times:

- Pick randomly set of k out of n variables
- Choose randomly for each picked variable whether it should be negated or unnegated
- Add clause to formula under construction

Fact: $\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF asymptotically almost surely

- Resolution: Hard to refute asymptotically almost surely [CS88] *(later during the course)*
- Polynomial calculus: Ditto [AR03, BI10]
- Cutting planes: Believed hard; another **big** open problem

Main Focus of This Course

Study proof systems such as:

- Resolution
- Cutting planes
- k-DNF resolution
- Even stronger system known as bounded-depth Frege
- Perhaps also algebraic ideal proof system somewhat similar in flavour to polynomial calculus (but stronger, and harder to explain)

Main Focus of This Course

Study proof systems such as:

- Resolution
- Cutting planes
- k-DNF resolution
- Even stronger system known as bounded-depth Frege
- Perhaps also algebraic ideal proof system somewhat similar in flavour to polynomial calculus (but stronger, and harder to explain)

Main focus:

- Lower bounds (and some upper bounds) on proof size
- Hopefully also hardness of proof search
- (Will most likely **not** study space or size-space trade-offs, although those are also fun topics)

Practicalities

- Read course webpage `www.csc.kth.se/DD2442/semteo16` carefully — contains lots of useful information
- Sign up at `piazza.com/kth.se/fall2016/dd2442` ASAP to get course announcements and to ask questions
- Also need to register at KTH in whatever way appropriate for you
- Examination is by problem sets + scribed lecture notes
- Please note this is a research-level course, so edges can be a bit rough
  - Sometimes lectures a bit buggy (usually fixed quickly)
  - Sometimes problem sets a bit buggy! (Student bug reports appreciated)
  - Don't hesitate to ask at Piazza if anything is unclear!
- Course intended to be fun and interesting (and challenging)
  Need feed-back to make that happen — let me know what you think!

Examination: Problem Sets + Scribed Lecture Notes

Problem sets

- Solve individually or in groups of two
- Then peer evaluate solutions of other participant (individually)
- Plus discuss solutions on Piazza
- See `www.csc.kth.se/DD2442/semteo16/administration/#psets` for detailed description of set-up

Examination: Problem Sets + Scribed Lecture Notes

Problem sets

- Solve individually or in groups of two
- Then peer evaluate solutions of other participant (individually)
- Plus discuss solutions on Piazza
- See `www.csc.kth.se/DD2442/semteo16/administration/#psets` for detailed description of set-up

Scribed lecture notes

- Produce high-quality notes in LaTeX
- As the course progresses, you yourselves create the textbook that doesn't exist
- Good way to learn material in-depth
- Useful exercise to develop scientific/technical writing skills
- Detailed instructions + sign-up sheets will be posted soon (first lectures already covered)

Instructors and Assistants

Jakob Nordström

Main instructor

Responsible for all aspects of course

(So any complaints should be directed to me ☺)

Instructors and Assistants

Jakob Nordström
Main instructor
Responsible for all aspects of course
(So any complaints should be directed to me ☺)

Ilario Bonacina
Lectures
Grading of problem sets
Reviewing of scribe notes
Maybe informal office hours
(will vote on this later)

Instructors and Assistants

Jakob Nordström
Main instructor
Responsible for all aspects of course
(So any complaints should be directed to me ☺)

Marc Vinyals
Couple of guest lectures
Reviewing of scribe notes
Maybe informal office hours

Ilario Bonacina
Lectures
Grading of problem sets
Reviewing of scribe notes
Maybe informal office hours
(will vote on this later)

Susanna F. de Rezende
Reviewing of scribe notes
Maybe informal office hours

Instructors and Assistants

Jakob Nordström
Main instructor
Responsible for all aspects of course
(So any complaints should be directed to me ☺)

Marc Vinyals
Couple of guest lectures
Reviewing of scribe notes
Maybe informal office hours

Ilario Bonacina
Lectures
Grading of problem sets
Reviewing of scribe notes
Maybe informal office hours
(will vote on this later)

Susanna F. de Rezende
Reviewing of scribe notes
Maybe informal office hours

. . . And that concludes today's lecture. . .

Next time we will get down to business
and start proving theorems!

References I

[AR03] Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. *Proceedings of the Steklov Institute of Mathematics*, 242:18–35, 2003. Available at `http://people.cs.uchicago.edu/~razborov/files/misha.pdf`. Preliminary version in *FOCS '01*.

[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.

[BI10] Eli Ben-Sasson and Russell Impagliazzo. Random CNF's are hard for the polynomial calculus. *Computational Complexity*, 19:501–519, 2010. Preliminary version in *FOCS '99*.

[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.

References II

[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.

[CEI96] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 174–183, May 1996.

[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, 1971.

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

[CS88] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, October 1988.

References III

[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.

[IPS99] Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999.

[Kra01] Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1-3):123–140, 2001.

References IV

[Mil00] The Millennium Problems of the Clay Mathematics Institute, May 2000. See `http://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.

[MN14] Mladen Mikša and Jakob Nordström. Long proofs of (seemingly) simple formulas. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 121–137. Springer, July 2014.

[MN15] Mladen Mikša and Jakob Nordström. A generalized method for proving polynomial calculus degree lower bounds. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 467–487, June 2015.

[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*.

References V

[Raz98] Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, December 1998.

[Rii93] Søren Riis. *Independence in Bounded Arithmetic*. PhD thesis, University of Oxford, 1993.

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

[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.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.