On the Interplay Between Proof Complexity and SAT Solving

Jakob Nordström

KTH Royal Institute of Technology
Stockholm, Sweden

SAT/SMT/AR Summer School 2016
Lisbon, Portugal
June 24, 2016

The Satisfiability Problem (SAT)

$$(x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z})$$

The Satisfiability Problem (SAT)

$$(x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z})$$

- Variables should be set to true or false

The Satisfiability Problem (SAT)

$$(x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z})$$

- Variables should be set to true or false
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false

The Satisfiability Problem (SAT)

$$(x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z})$$

- Variables should be set to true or false
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously

The Satisfiability Problem (SAT)

$$(x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z})$$

- Variables should be set to true or false
- Constraint $(x \vee \overline{y} \vee z)$: means x or z should be true or y false
- $\wedge$ means all constraints should hold simultaneously

Is there a truth value assignment satisfying all these conditions?
Or is it always the case that some constraint must fail to hold?

The Satisfiability Problem (SAT)

$$(x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z})$$

- Variables should be set to true or false
- Constraint $(x \vee \overline{y} \vee z)$: means x or z should be true or y false
- $\wedge$ means all constraints should hold simultaneously

Is there a truth value assignment satisfying all these conditions?
Or is it always the case that some constraint must fail to hold?

Want to use computers to solve the SAT problem efficiently

Computational Complexity Theory and SAT Solving

Complexity theory

- Satisfiability of formulas in propositional logic (SAT) foundational problem
- SAT proven NP-complete by Stephen Cook in 1971
- Hence most likely totally intractable
- Just remains to prove this — one of the million-dollar "Millennium Problems"

Computational Complexity Theory and SAT Solving

Complexity theory

- Satisfiability of formulas in propositional logic (SAT) foundational problem
- SAT proven NP-complete by Stephen Cook in 1971
- Hence most likely totally intractable
- Just remains to prove this — one of the million-dollar "Millennium Problems"

Applied SAT solving

- Dramatic performance increase last 15–20 years
- State-of-the-art SAT solvers can deal with real-world formulas containing millions of variables
- But best solvers still based on methods from early 1960s
- Also, tiny formulas known that are totally beyond reach

SAT Solving Through the Lens of Proof Complexity

- How can state-of-the-art SAT solvers decide satisfiability of such huge formulas?

SAT Solving Through the Lens of Proof Complexity

- How can state-of-the-art SAT solvers decide satisfiability of such huge formulas?
- Why do they work so well? And why do they sometimes miserably fail?

SAT Solving Through the Lens of Proof Complexity

- How can state-of-the-art SAT solvers decide satisfiability of such huge formulas?
- Why do they work so well? And why do they sometimes miserably fail?
- State-of-the-art SAT solving techniques:
  - Conflict-driven clause learning (CDCL)
  - Algebraic reasoning (Gröbner bases / Gaussian elimination)
  - Geometric reasoning (cardinality / pseudo-Boolean constraints)
  - Sometimes even some extended resolution

SAT Solving Through the Lens of Proof Complexity

- How can state-of-the-art SAT solvers decide satisfiability of such huge formulas?
- Why do they work so well? And why do they sometimes miserably fail?
- State-of-the-art SAT solving techniques:
  - Conflict-driven clause learning (CDCL)
  - Algebraic reasoning (Gröbner bases / Gaussian elimination)
  - Geometric reasoning (cardinality / pseudo-Boolean constraints)
  - Sometimes even some extended resolution
- How can we analyze the power of these methods?
  Pretty much only approach for rigorous analysis: use **proof complexity** to study underlying methods of reasoning

Outline of This Presentation

This talk: overview of (or crash course in) proof complexity

Focus on connections with current approaches to SAT solving:

- Conflict-driven clause learning — resolution
- Algebraic Gröbner basis computations — polynomial calculus
- Geometric pseudo-Boolean solvers — cutting planes
- Might also mention extended resolution, but if so very briefly

Survey (some of) what is known about these proof systems

Show theoretical "benchmark formulas" used to understand potential and limitations of methods of reasoning

Outline of This Presentation

This talk: overview of (or crash course in) proof complexity

Focus on connections with current approaches to SAT solving:

- Conflict-driven clause learning — resolution
- Algebraic Gröbner basis computations — polynomial calculus
- Geometric pseudo-Boolean solvers — cutting planes
- Might also mention extended resolution, but if so very briefly

Survey (some of) what is known about these proof systems

Show theoretical "benchmark formulas" used to understand potential and limitations of methods of reasoning

Caveats:

- By necessity, selective and somewhat subjective coverage
- Won't do too much name-dropping — full references at end of slides

Some More Caveats and Clarifications

Only basic propositional logic proof search

- No SMT or first-order logic or anything in this talk
- No discussion of preprocessing techniques

Some More Caveats and Clarifications

Only basic propositional logic proof search

- No SMT or first-order logic or anything in this talk
- No discussion of preprocessing techniques

Limitations of proof complexity

- Asking for rigorous analysis is asking a lot...
- In addition, proof complexity considers optimal algorithms (so restrict focus to unsatisfiable formulas)
- Still can prove some highly nontrivial theorems
- Separate question how to interpret these theoretical theorems

Some More Caveats and Clarifications

Only basic propositional logic proof search

- No SMT or first-order logic or anything in this talk
- No discussion of preprocessing techniques

Limitations of proof complexity

- Asking for rigorous analysis is asking a lot...
- In addition, proof complexity considers optimal algorithms (so restrict focus to unsatisfiable formulas)
- Still can prove some highly nontrivial theorems
- Separate question how to interpret these theoretical theorems

Why theory benchmarks?

- See what SAT solvers can do (sometimes very neat things)
- See what SAT solvers cannot do (provably hard instances)
- See what SAT solvers "should be able" to do (formulas easy for proof system but hard for corresponding SAT solvers)

Outline

1. Resolution
   - Preliminaries
   - Length, Width and Space
   - Resolution Trade-offs
2. Connections Between Resolution and CDCL
   - Resolution and SAT Solving
   - Complexity Measures and CDCL
   - Research Questions and Future Directions
3. Stronger Proof Systems than Resolution
   - Polynomial Calculus
   - Cutting Planes
   - And Beyond...

Some Notation and Terminology

- Literal a: variable x or its negation $\overline{x}$
- Clause $C = a_1 \lor \cdots \lor a_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- CNF formula $F = C_1 \land \cdots \land C_m$: conjunction of clauses
- k-CNF formula: CNF formula with clauses of size $\leq k$
  (where k is some constant)
- Mostly assume formulas k-CNFs (for simplicity of exposition)
  Conversion to 3-CNF (most often) doesn't change much
- N denotes size of formula (# literals, which is $\approx$ # clauses)
- $\mathcal{O}(f(N))$ grows at most as quickly as $f(N)$ asymptotically
  $\Omega(g(N))$ grows at least as quickly as $g(N)$ asymptotically

The Resolution Proof System

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

The Resolution Proof System

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

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

The Resolution Proof System

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 $\bot$ 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.	$\bot$	Res$(7,8)$

The Resolution Proof System

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$(7, 8)$

The Resolution Proof System

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

The Resolution Proof System

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$(7,8)$

The Resolution Proof System

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$(7,8)$

The Resolution Proof System

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 $\bot$ derived

Can represent refutation as

- annotated list or
- directed acyclic graph

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

The Resolution Proof System

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$(7, 8)$

The Resolution Proof System

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$(7, 8)$

The Resolution Proof System

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 $\bot$ 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.	$\bot$	Res$(7, 8)$

The Resolution Proof System

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 $\bot$ 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.	$\boldsymbol{\overline{x}}$	Res$(3, 5)$
9.	$\bot$	Res$(7, 8)$

The Resolution Proof System

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$(7,8)$

The Resolution Proof System

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$(7, 8)$

The Resolution Proof System

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 $\bot$ 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.	$\bot$	Res$(7,8)$

The Resolution Proof System

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

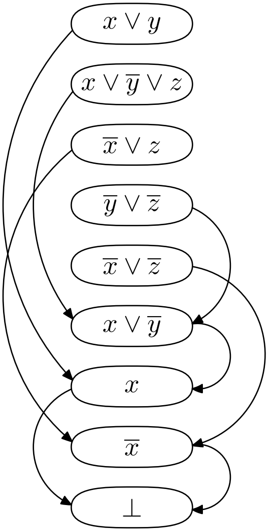
Derive new clauses by resolution rule

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

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

Can represent refutation as

- annotated list or
- directed acyclic graph



The Resolution Proof System

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

Derive new clauses by resolution rule

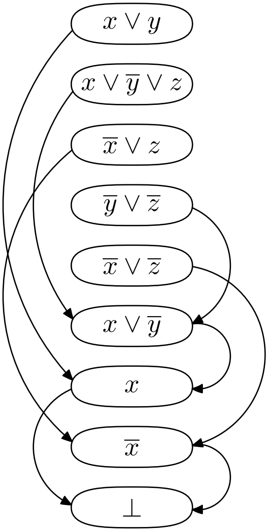
$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

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 Size/Length

Size/length = # clauses in refutation
(9 in our example)

Length of refuting F = length of shortest refutation of F

Most fundamental measure in proof complexity

Lower bound on CDCL running time
(can extract resolution proof from execution trace)

Never worse than $\exp(\mathcal{O}(N))$

Matching $\exp(\Omega(N))$ lower bounds known

Examples of Hard Formulas w.r.t. Resolution Length (1/4)

Pigeonhole principle (PHP) [Hak85]
"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon i goes into hole j"

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

Can also add "functionality" and "onto" axioms

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

Examples of Hard Formulas w.r.t. Resolution Length (1/4)

Pigeonhole principle (PHP) [Hak85]
"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon i goes into hole j"

$p_{i,1} \lor p_{i,2} \lor \cdots \lor p_{i,n}$	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 "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

Even onto functional PHP formula is hard for resolution
"Resolution cannot count"

Examples of Hard Formulas w.r.t. Resolution Length (1/4)

Pigeonhole principle (PHP) [Hak85]
"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon i goes into hole j"

$p_{i,1} \lor p_{i,2} \lor \cdots \lor p_{i,n}$	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 "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

Even onto functional PHP formula is hard for resolution
"Resolution cannot count"

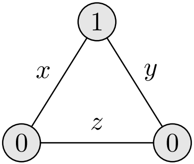
But only length lower bound $\exp\left(\Omega\left(\sqrt[3]{N}\right)\right)$ in terms of formula size

Examples of Hard Formulas w.r.t. Resolution Length (2/4)

Tseitin formulas [Urq87]
"Sum of degrees of vertices in graph is even"

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



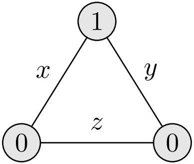
$$
\begin{array}{ll}
(x \vee y) & \wedge\ (\overline{x} \vee z) \\
\wedge\ (\overline{x} \vee \overline{y}) & \wedge\ (y \vee \overline{z}) \\
\wedge\ (x \vee \overline{z}) & \wedge\ (\overline{y} \vee z)
\end{array}
$$

Examples of Hard Formulas w.r.t. Resolution Length (2/4)

Tseitin formulas [Urq87]
"Sum of degrees of vertices in graph is even"

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



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

Requires length $\exp(\Omega(N))$ on well-connected so-called expanders
"Resolution cannot count $\mathrm{mod}\ 2$**"**

Examples of Hard Formulas w.r.t. Resolution Length (3/4)

Zero-one designs [Spe10, VS10, MN14]

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

Each row requires majority of variables true

Each column requires majority of variables false

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

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

Examples of Hard Formulas w.r.t. Resolution Length (3/4)

Zero-one designs [Spe10, VS10, MN14]

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

Each row requires majority of variables true

Each column requires majority of variables false

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

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

Examples of Hard Formulas w.r.t. Resolution Length (3/4)

Zero-one designs [Spe10, VS10, MN14]

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

Each row requires majority of variables true

Each column requires majority of variables false

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

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

Examples of Hard Formulas w.r.t. Resolution Length (3/4)

Zero-one designs [Spe10, VS10, MN14]

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

Each row requires majority of variables true

Each column requires majority of variables false

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

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

Examples of Hard Formulas w.r.t. Resolution Length (3/4)

Zero-one designs [Spe10, VS10, MN14]

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

Each row requires majority of variables true

Each column requires majority of variables false

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

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

Lower bound $\exp(\Omega(N))$ on expanding matrices (well spread-out)

Examples of Hard Formulas w.r.t. Resolution Length (4/4)

Random k-CNF formulas [CS88]
Δn randomly sampled k-clauses over n variables

($\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF almost surely)

Again lower bound $\exp(\Omega(N))$

Examples of Hard Formulas w.r.t. Resolution Length (4/4)

Random k-CNF formulas [CS88]
Δn randomly sampled k-clauses over n variables

($\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF almost surely)

Again lower bound $\exp(\Omega(N))$

And more...

- k-colourability [BCMM05]
- Independent sets and vertex covers [BIS07]
- Et cetera...

Resolution Width

Width $=$ size of largest clause in refutation (always $\leq N$)
(3 in our example)

Width of refuting F $=$ width of shortest refutation of F

Resolution Width

Width $=$ size of largest clause in refutation (always $\leq N$)
(3 in our example)

Width of refuting F $=$ width of shortest refutation of F

Width upper bound $\Rightarrow$ length upper bound

Proof: at most $(2 \cdot \#\text{variables})^{\text{width}}$ distinct clauses
(This simple counting argument is essentially tight [ALN16])

Resolution Width

Width = size of largest clause in refutation (always $\leq N$)
(3 in our example)

Width of refuting F = width of shortest refutation of F

Width upper bound $\Rightarrow$ length upper bound

Proof: at most $(2 \cdot \#\text{variables})^{\text{width}}$ distinct clauses
(This simple counting argument is essentially tight [ALN16])

Width lower bound $\Rightarrow$ length lower bound

Much less obvious...

Width Lower Bounds Imply Length Lower Bounds

Theorem ([BW01])

$$\textit{length} \geq \exp\left(\Omega\left(\frac{(\textit{width})^2}{(\textit{formula size } N)}\right)\right)$$

Width Lower Bounds Imply Length Lower Bounds

Theorem ([BW01])

$$\textit{length} \geq \exp\left(\Omega\left(\frac{(\textit{width})^2}{(\textit{formula size } N)}\right)\right)$$

Yields superpolynomial length bounds for width $\omega\left(\sqrt{N \log N}\right)$
Almost all known lower bounds on length derivable via width

Width Lower Bounds Imply Length Lower Bounds

Theorem ([BW01])

$$\textit{length} \geq \exp\left(\Omega\left(\frac{(\textit{width})^2}{(\textit{formula size } N)}\right)\right)$$

Yields superpolynomial length bounds for width $\omega\left(\sqrt{N \log N}\right)$
Almost all known lower bounds on length derivable via width

For tree-like resolution have length $\geq 2^{\text{width}}$ [BW01]

General resolution: width up to $\mathcal{O}\left(\sqrt{N \log N}\right)$ implies no length lower bounds — possible to tighten analysis? **No!**

Optimality of the Length-Width Lower Bound

Ordering principles [Stå96, BG01]
"Every finite ordered set $\{e_1, \ldots, e_n\}$ has minimal element"

Variables $x_{i,j} =$ "$e_i < e_j$"

$\overline{x}_{i,j} \lor \overline{x}_{j,i}$	anti-symmetry; not both $e_i < e_j$ and $e_j < e_i$
$\overline{x}_{i,j} \lor \overline{x}_{j,k} \lor x_{i,k}$	transitivity; $e_i < e_j$ and $e_j < e_k$ implies $e_i < e_k$
$\bigvee_{1 \leq i \leq n,\, i \neq j} x_{i,j}$	e_j is not a minimal element

Can also add "total order" axioms

$x_{i,j} \lor x_{j,i}$	totality; either $e_i < e_j$ or $e_j < e_i$

Optimality of the Length-Width Lower Bound

Ordering principles [Stå96, BG01]
"Every finite ordered set $\{e_1, \ldots, e_n\}$ has minimal element"

Variables $x_{i,j} =$ "$e_i < e_j$"

$\overline{x}_{i,j} \lor \overline{x}_{j,i}$	anti-symmetry; not both $e_i < e_j$ and $e_j < e_i$
$\overline{x}_{i,j} \lor \overline{x}_{j,k} \lor x_{i,k}$	transitivity; $e_i < e_j$ and $e_j < e_k$ implies $e_i < e_k$
$\bigvee_{1 \leq i \leq n,\, i \neq j} x_{i,j}$	e_j is not a minimal element

Can also add "total order" axioms

$x_{i,j} \lor x_{j,i}$	totality; either $e_i < e_j$ or $e_j < e_i$

Refutable in resolution in length $\mathcal{O}(N)$
Requires resolution width $\Omega\left(\sqrt[3]{N}\right)$ (for 3-CNF version)

Resolution Space

Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — focus here on most common measure clause space

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

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 = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — focus here on most common measure clause space

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

Example: 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

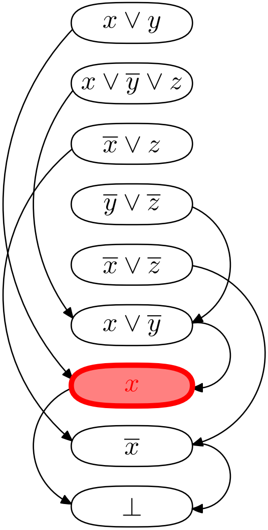
Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — focus here on most common measure clause space

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

Example: Space at step 7 ...



Resolution Space

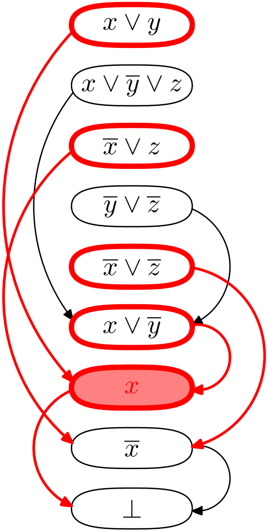
Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — focus here on most common measure clause space

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

Example: Space at step 7 is 5



Bounds on Resolution Space

Space always at most $N + \mathcal{O}(1)$ (!) [ET01]

Lower bounds subsequently proven for

- Pigeonhole principle [ABRW02, ET01]
- Tseitin formulas [ABRW02, ET01]
- Random k-CNFs [BG03]

Bounds on Resolution Space

Space always at most $N + \mathcal{O}(1)$ (!) [ET01]

Lower bounds subsequently proven for

- Pigeonhole principle [ABRW02, ET01]
- Tseitin formulas [ABRW02, ET01]
- Random k-CNFs [BG03]

Results always exactly matching width lower bounds
And proofs of very similar flavour. . .
Just a coincidence?

Space vs. Width

Theorem ([AD08])

$$\mathit{space} \geq \mathit{width} + \mathcal{O}(1)$$

Space vs. Width

Theorem ([AD08])

$$\mathit{space} \geq \mathit{width} + \mathcal{O}(1)$$

Width lower bound $\Rightarrow$ length **and space** lower bounds!
(Not a trivial claim, since space counts clauses)

Space vs. Width

Theorem ([AD08])

$$space \geq width + \mathcal{O}(1)$$

Width lower bound $\Rightarrow$ length **and space** lower bounds!
(Not a trivial claim, since space counts clauses)

Are space and width asymptotically always the same? **No!**

Space vs. Width

Theorem ([AD08])

$$\textit{space} \geq \textit{width} + \mathcal{O}(1)$$

Width lower bound $\Rightarrow$ length **and space** lower bounds!
(Not a trivial claim, since space counts clauses)

Are space and width asymptotically always the same? **No!**

Pebbling formulas [Nor09, NH13, BN08]

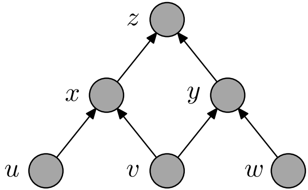
- Can be refuted in width $\mathcal{O}(1)$
- May require space $\Omega(N/\log N)$

A bit more involved to describe than previous benchmarks. . .

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \vee \overline{v} \vee x$
5. $\overline{v} \vee \overline{w} \vee y$
6. $\overline{x} \vee \overline{y} \vee z$
7. $\overline{z}$

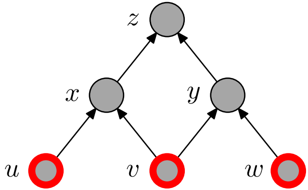


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$

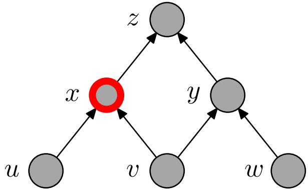


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$

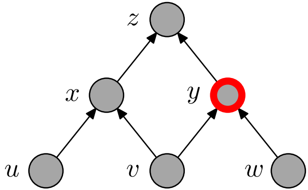


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$

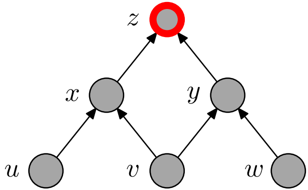


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$

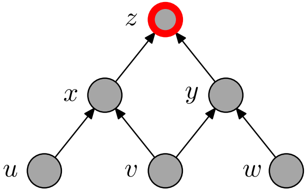


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$

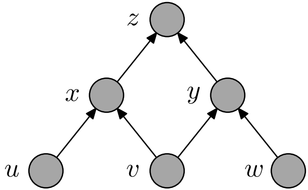


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$



- sources are true
- truth propagates upwards
- but sink is false

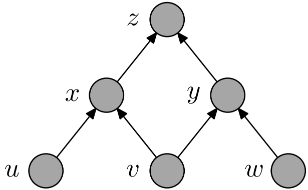
Extensive literature on pebbling space and time-space trade-offs from 1970s and 80s

Have been useful in proof complexity before in various contexts

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$



- sources are true
- truth propagates upwards
- but sink is false

Extensive literature on pebbling space and time-space trade-offs from 1970s and 80s

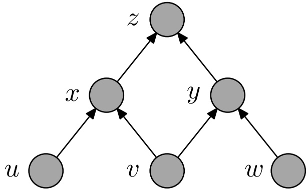
Have been useful in proof complexity before in various contexts

Hope that pebbling properties of DAG somehow carry over to resolution refutations of pebbling formulas.

Pebbling Formulas: Vanilla Version

CNF formulas encoding so-called pebble games on DAGs

1. u
2. v
3. w
4. $\overline{u} \lor \overline{v} \lor x$
5. $\overline{v} \lor \overline{w} \lor y$
6. $\overline{x} \lor \overline{y} \lor z$
7. $\overline{z}$



- sources are true
- truth propagates upwards
- but sink is false

Extensive literature on pebbling space and time-space trade-offs from 1970s and 80s

Have been useful in proof complexity before in various contexts

Hope that pebbling properties of DAG somehow carry over to resolution refutations of pebbling formulas. **Except...**

Substituted Pebbling Formulas

Won't work — formulas are supereasy (solved by unit propagation)

Substituted Pebbling Formulas

Won't work — formulas are supereasy (solved by unit propagation)

Make formula harder by substituting $x_1 \oplus x_2$ for every variable x (also works for other Boolean functions with "right" properties):

$$\overline{x} \vee y$$

$$\Downarrow$$

$$\neg(x_1 \oplus x_2) \vee (y_1 \oplus y_2)$$

$$\Downarrow$$

$$\begin{aligned} & (x_1 \vee \overline{x}_2 \vee y_1 \vee y_2) \\ \wedge\ & (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee y_1 \vee y_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2) \end{aligned}$$

Substituted Pebbling Formulas

Won't work — formulas are supereasy (solved by unit propagation)

Make formula harder by substituting $x_1 \oplus x_2$ for every variable x (also works for other Boolean functions with "right" properties):

$$\overline{x} \vee y$$

$$\Downarrow$$

$$\neg(x_1 \oplus x_2) \vee (y_1 \oplus y_2)$$

$$\Downarrow$$

$$\begin{aligned} & (x_1 \vee \overline{x}_2 \vee y_1 \vee y_2) \\ \wedge\ & (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee y_1 \vee y_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2) \end{aligned}$$

Now CNF formula inherits pebbling graph properties!

Trade-offs Between Complexity Measures?

Given a formula easy w.r.t. these complexity measures, can refutations be optimized for two or more measures simultaneously?

Trade-offs Between Complexity Measures?

Given a formula easy w.r.t. these complexity measures, can refutations be optimized for two or more measures simultaneously?

- Space vs. width: **No!** [Ben09]
  Vanilla pebbling formulas for right choice of graphs

Trade-offs Between Complexity Measures?

Given a formula easy w.r.t. these complexity measures, can refutations be optimized for two or more measures simultaneously?

- Space vs. width: **No!** [Ben09]
  Vanilla pebbling formulas for right choice of graphs
- Length vs. width: **No!** [Tha14]
  Nifty formulas that would take a bit too long to describe...

Trade-offs Between Complexity Measures?

Given a formula easy w.r.t. these complexity measures, can refutations be optimized for two or more measures simultaneously?

- Space vs. width: **No!** [Ben09]
  Vanilla pebbling formulas for right choice of graphs
- Length vs. width: **No!** [Tha14]
  Nifty formulas that would take a bit too long to describe...
- Length vs. space: Arguably most interesting case
  Length ≈ running time
  Space ≈ memory consumption
  SAT solvers aggressively try to minimize both

Length-Space Trade-offs

Theorem ([BN11, BBI12, BNT13])

There are formulas for which

- *exist refutations in short length*
- *exist refutations in small space*
- *optimization of one measure* causes *dramatic blow-up for other measure*

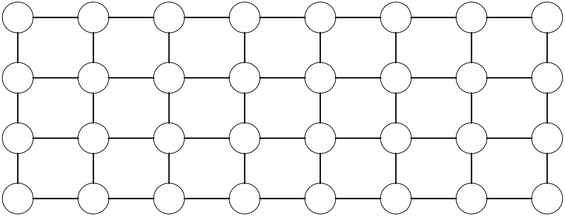
Holds for

- Substituted pebbling formulas on the right graphs
- Tseitin formulas on long, narrow rectangular grids

So no meaningful simultaneous optimization possible in worst case

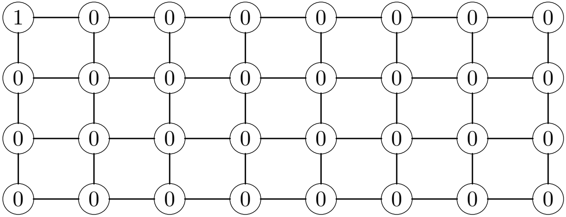
A Closer Look at Tseitin Formulas on Long, Narrow Grids

- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$



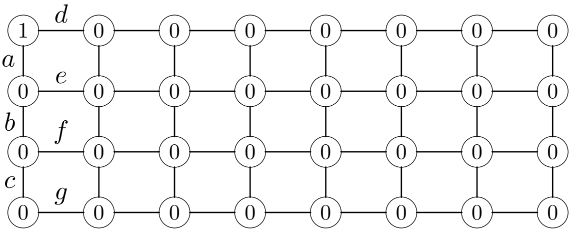
A Closer Look at Tseitin Formulas on Long, Narrow Grids

- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd



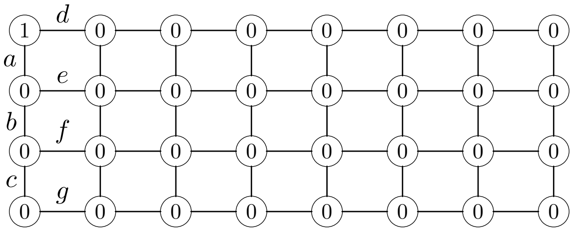
A Closer Look at Tseitin Formulas on Long, Narrow Grids

- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges



A Closer Look at Tseitin Formulas on Long, Narrow Grids

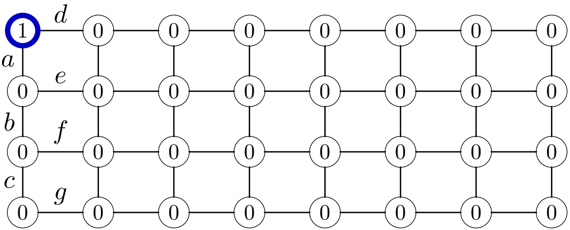
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges
- We have clauses encoding constraints "vertex label = parity of incident edges"



A Closer Look at Tseitin Formulas on Long, Narrow Grids

- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges
- We have clauses encoding constraints
  "vertex label = parity of incident edges"

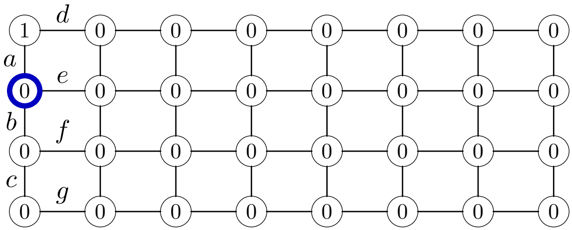
$$(a \vee d) \wedge (\overline{a} \vee \overline{d})$$



A Closer Look at Tseitin Formulas on Long, Narrow Grids

- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges
- We have clauses encoding constraints
  "vertex label = parity of incident edges"

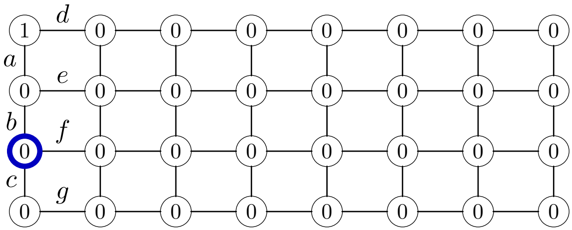
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e})
\end{aligned}
$$



A Closer Look at Tseitin Formulas on Long, Narrow Grids

- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges
- We have clauses encoding constraints
  "vertex label = parity of incident edges"

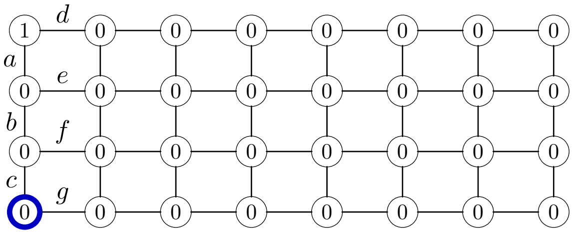
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e}) \\
\land\ & (b \lor c \lor \overline{f}) \\
\land\ & (b \lor \overline{c} \lor f) \\
\land\ & (\overline{b} \lor c \lor f) \\
\land\ & (\overline{b} \lor \overline{c} \lor \overline{f})
\end{aligned}
$$



A Closer Look at Tseitin Formulas on Long, Narrow Grids

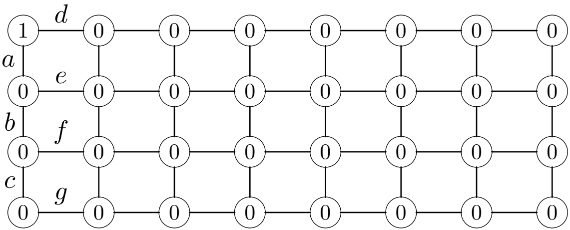
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges
- We have clauses encoding constraints
  "vertex label = parity of incident edges"

$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e}) \\
\land\ & (b \lor c \lor \overline{f}) \\
\land\ & (b \lor \overline{c} \lor f) \\
\land\ & (\overline{b} \lor c \lor f) \\
\land\ & (\overline{b} \lor \overline{c} \lor \overline{f}) \\
\land\ & (c \lor \overline{g}) \\
\land\ & (\overline{c} \lor g)
\end{aligned}
$$



A Closer Look at Tseitin Formulas on Long, Narrow Grids

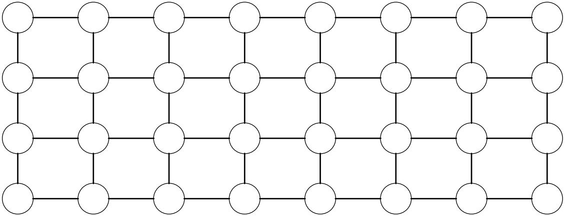
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Recall that variables = edges
- We have clauses encoding constraints "vertex label = parity of incident edges"
- Unsatisfiable — every edge counted twice, so total sum can't be odd



$$
\begin{aligned}
& (a \vee d) \\
\wedge\ & (\overline{a} \vee \overline{d}) \\
\wedge\ & (a \vee b \vee \overline{e}) \\
\wedge\ & (a \vee \overline{b} \vee e) \\
\wedge\ & (\overline{a} \vee b \vee e) \\
\wedge\ & (\overline{a} \vee \overline{b} \vee \overline{e}) \\
\wedge\ & (b \vee c \vee \overline{f}) \\
\wedge\ & (b \vee \overline{c} \vee f) \\
\wedge\ & (\overline{b} \vee c \vee f) \\
\wedge\ & (\overline{b} \vee \overline{c} \vee \overline{f}) \\
\wedge\ & (c \vee \overline{g}) \\
\wedge\ & (\overline{c} \vee g) \\
& \vdots
\end{aligned}
$$

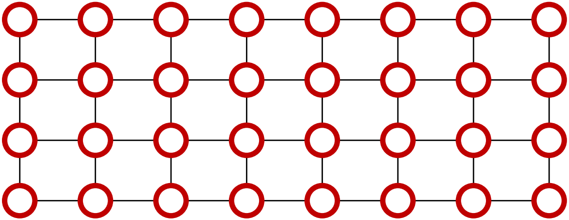
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges



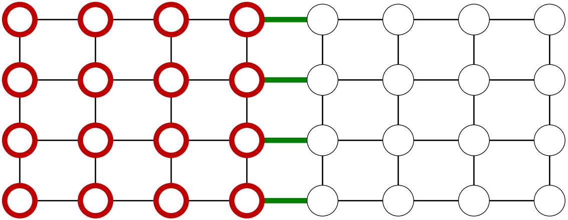
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component



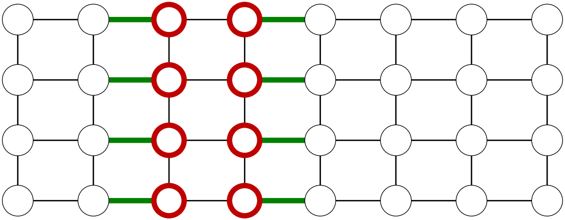
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component
- Disconnect into two pieces by querying edges; then recurse



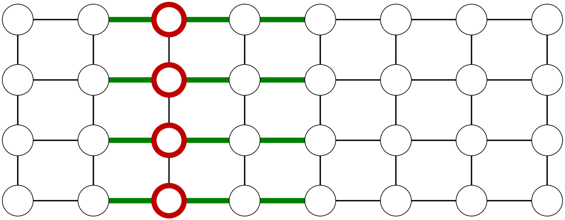
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component
- Disconnect into two pieces by querying edges; then recurse



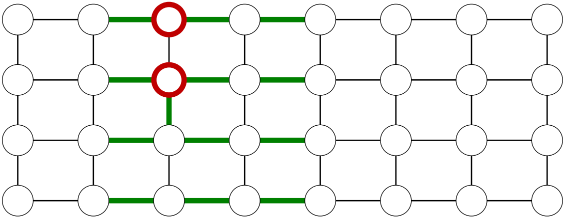
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component
- Disconnect into two pieces by querying edges; then recurse



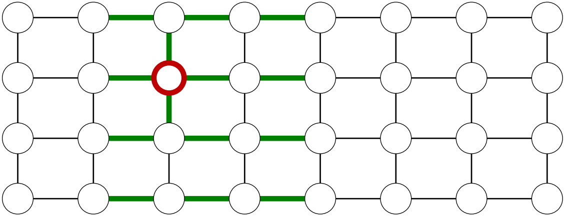
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component
- Disconnect into two pieces by querying edges; then recurse



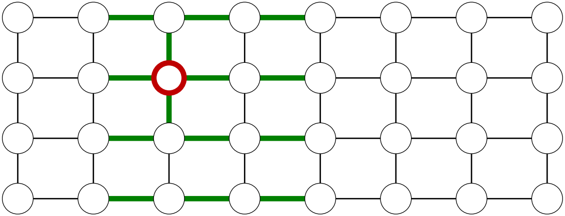
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component
- Disconnect into two pieces by querying edges; then recurse
- Violated vertex found after $w \log m$ queries



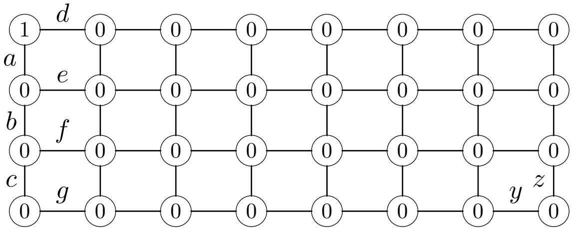
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges
- Identify odd-charge component
- Disconnect into two pieces by querying edges; then recurse
- Violated vertex found after $w \log m$ queries
- Height of tree = proof space = $w \log m$
  (very space-efficient, but proof size exponential in space)



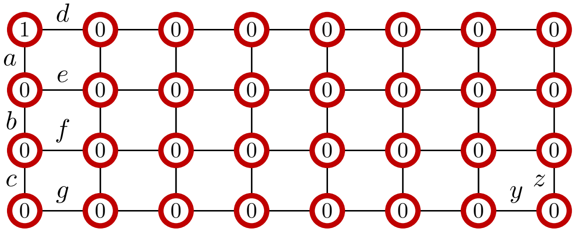
Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$



Small-Size “Dynamic Programming” Proof

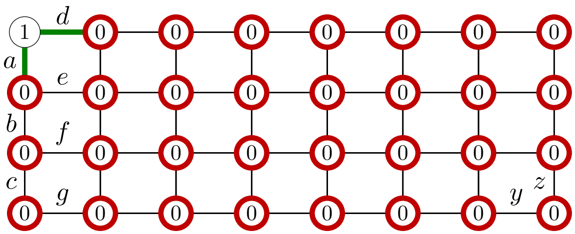
- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses

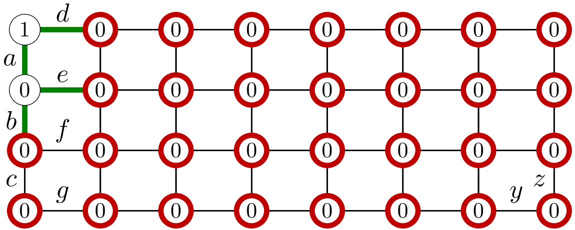
$a + d = 1$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

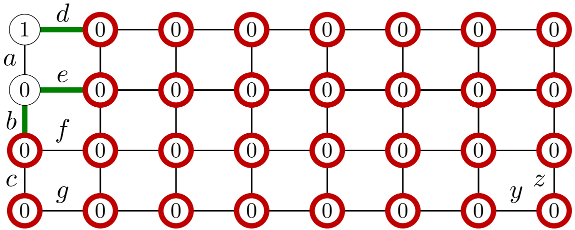
$$a + d = 1$$
$$a + b + e = 0$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

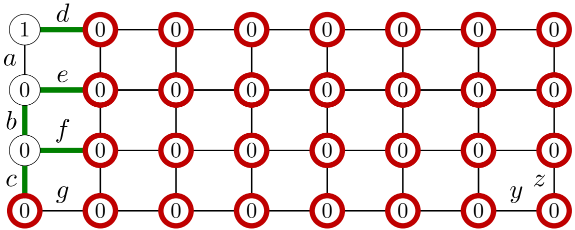
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow 2^w$ clauses

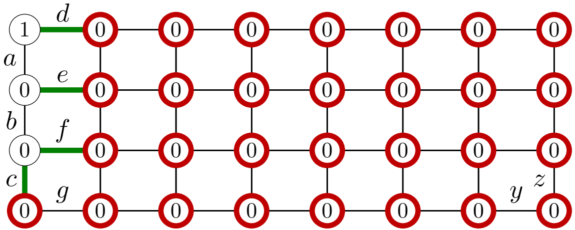
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow 2^w$ clauses

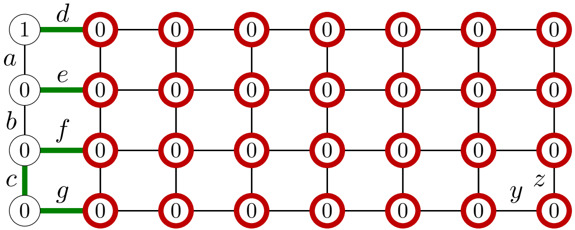
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

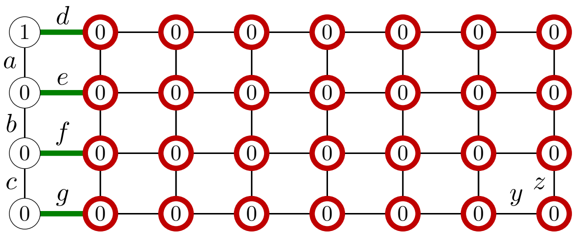
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$



Small-Size “Dynamic Programming” Proof

- View constraints as linear equations $\mod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

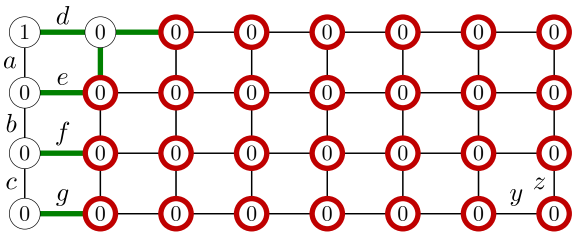
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

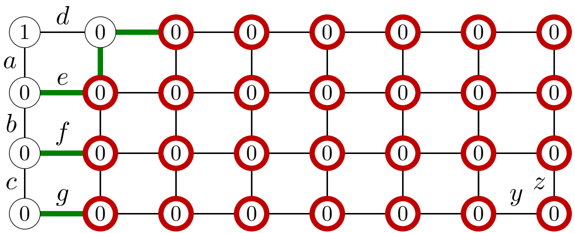
$$a+d=1$$
$$a+b+e=0$$
$$b+d+e=1$$
$$b+c+f=0$$
$$c+d+e+f=1$$
$$c+g=0$$
$$d+e+f+g=1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow 2^w$ clauses

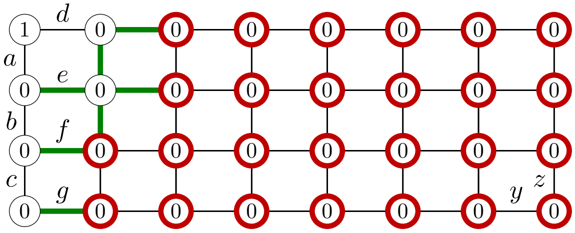
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses

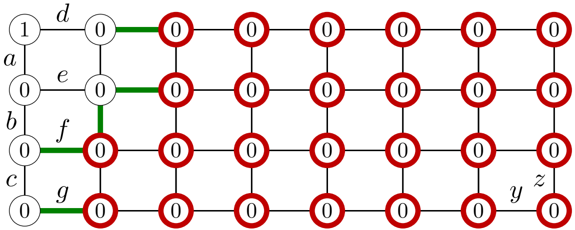
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

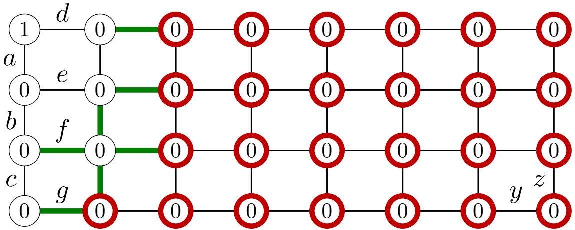
$$
\begin{aligned}
a+d &= 1 \\
a+b+e &= 0 \\
b+d+e &= 1 \\
b+c+f &= 0 \\
c+d+e+f &= 1 \\
c+g &= 0 \\
d+e+f+g &= 1 \\
&\vdots
\end{aligned}
$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

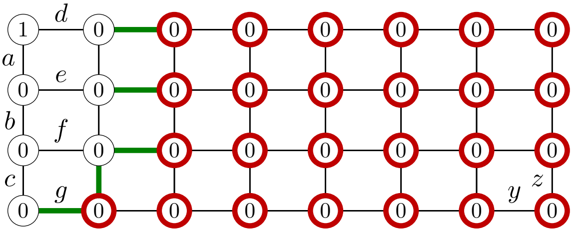
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

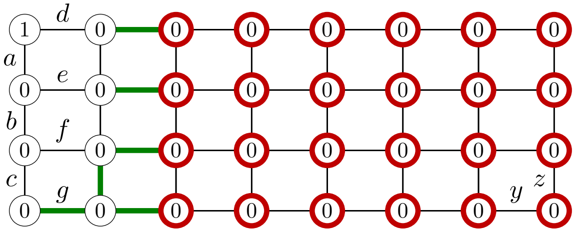
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

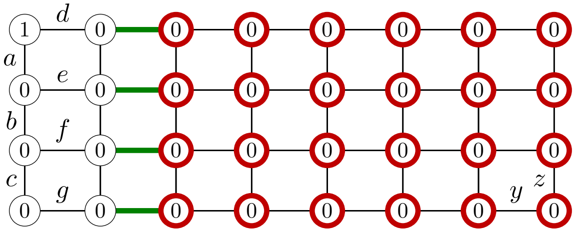
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

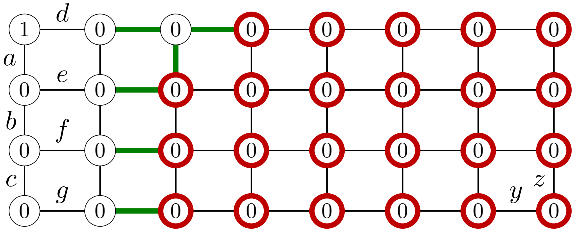
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses

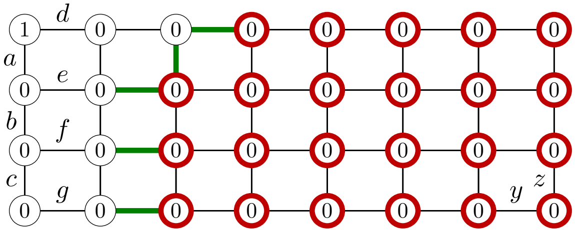
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

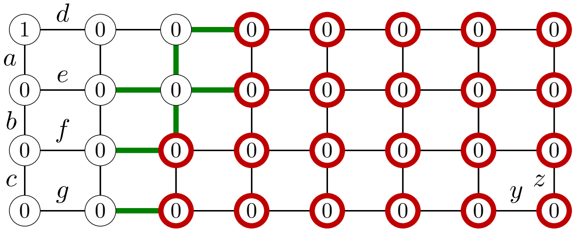
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size “Dynamic Programming” Proof

- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

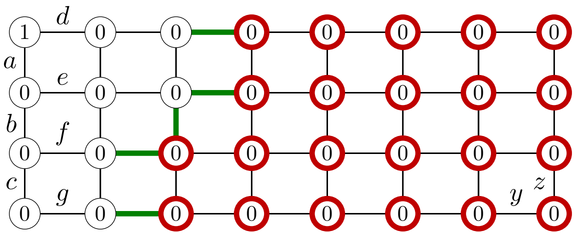
$$
\begin{aligned}
a + d &= 1 \\
a + b + e &= 0 \\
b + d + e &= 1 \\
b + c + f &= 0 \\
c + d + e + f &= 1 \\
c + g &= 0 \\
d + e + f + g &= 1 \\
&\vdots
\end{aligned}
$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

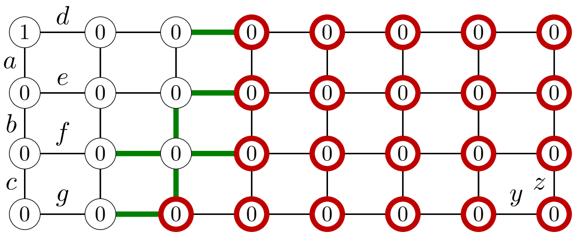
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

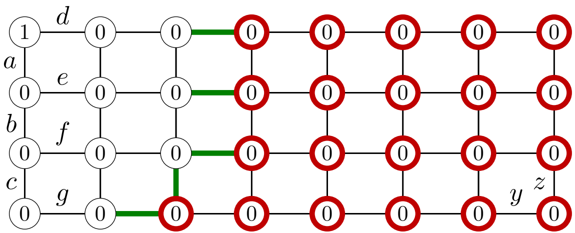
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

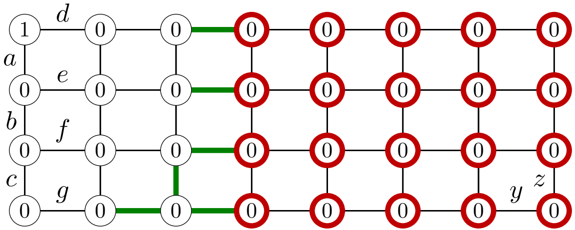
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses

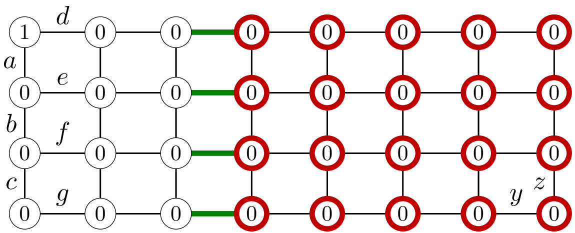
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size “Dynamic Programming” Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses

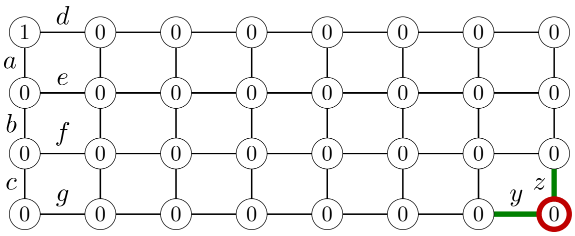
$$
\begin{aligned}
a + d &= 1 \\
a + b + e &= 0 \\
b + d + e &= 1 \\
b + c + f &= 0 \\
c + d + e + f &= 1 \\
c + g &= 0 \\
d + e + f + g &= 1 \\
&\vdots
\end{aligned}
$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses

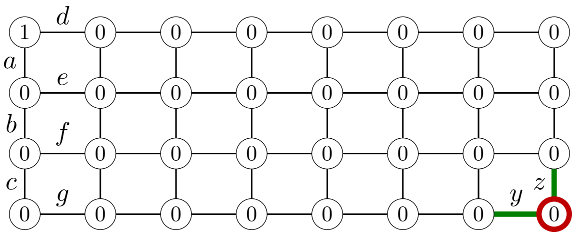
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$
$$y + z = 1$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mathrm{mod}\ 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow$ 2^w clauses

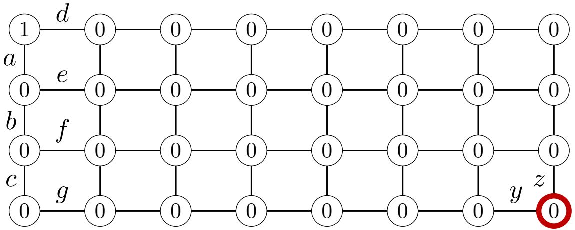
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$
$$y + z = 1$$
$$y + z = 0$$



Small-Size "Dynamic Programming" Proof

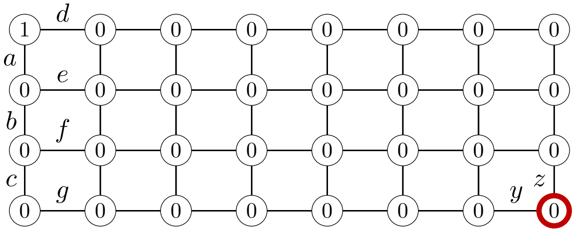
- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses
- Total of mw summations

$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$
$$y + z = 1$$
$$y + z = 0$$
$$0 = 1$$



Small-Size "Dynamic Programming" Proof

- View constraints as linear equations $\mod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w + 1$ variables $\Rightarrow$ 2^w clauses
- Total of mw summations
- Small proof size $\mathcal{O}(mw2^w) = poly(m)$
  However, space $\approx$ size — superlinear!



$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$
$$y + z = 1$$
$$y + z = 0$$
$$0 = 1$$

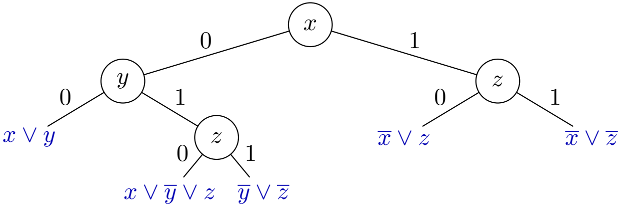
Resolution-Based SAT Solvers

- Resolution used for SAT algorithms already in 1960s
- Basis of best modern SAT solvers still DPLL method [DP60, DLL62]
- Addition of conflict-driven clause learning (CDCL) [BS97, MS99] exponential increase in reasoning power
- Plus lots of smart engineering and heuristics to make it fly in practice [MMZ$^+$01]
- Today there are highly successful CDCL SAT solvers such as, e.g., MiniSat [ES04], Glucose [AS09], and Lingeling [Bie10]

A Very Simplified Description of DPLL

Visualize execution of DPLL algorithm as search tree

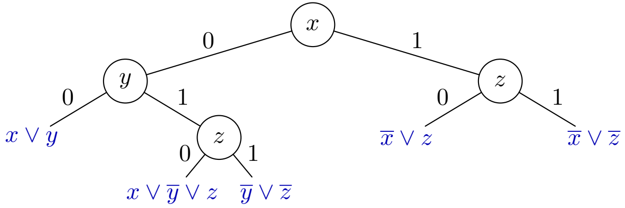
- Branch on variable assignments in internal nodes
- Stop in leaves when falsfied clause found



A Very Simplified Description of DPLL

Visualize execution of DPLL algorithm as search tree

- Branch on variable assignments in internal nodes
- Stop in leaves when falsfied clause found



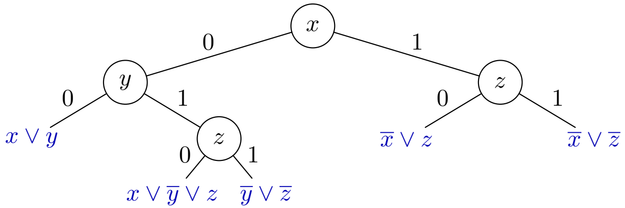
Many more ingredients in modern SAT solvers, for instance:

- Choice of branching variables crucial
- In leaf, compute & add reason for failure (clause learning)
- Restart every once in a while (but save computed info)

DPLL and Resolution

A DPLL execution is essentially a resolution proof

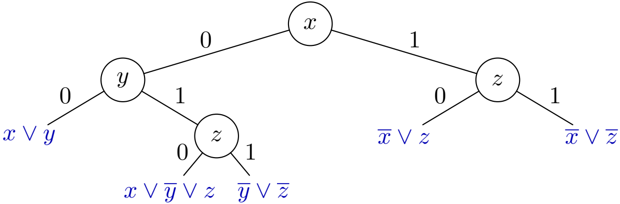
Look at our example again:



DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

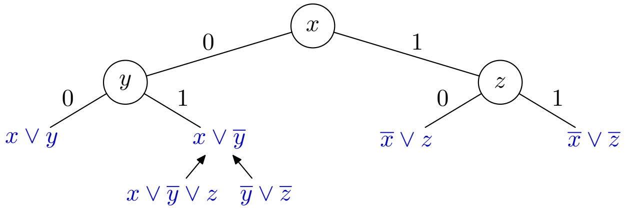


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

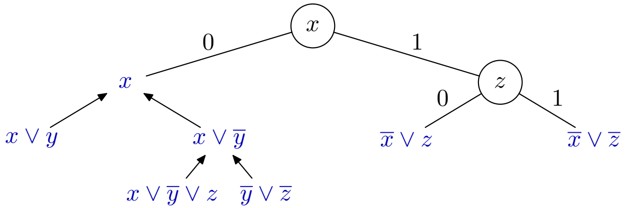


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

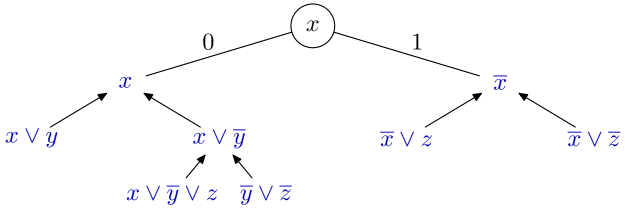


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

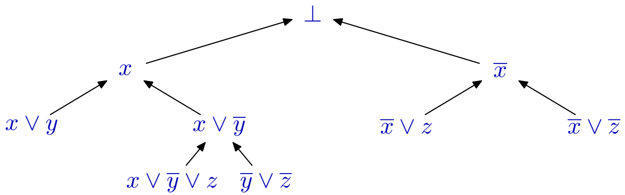


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

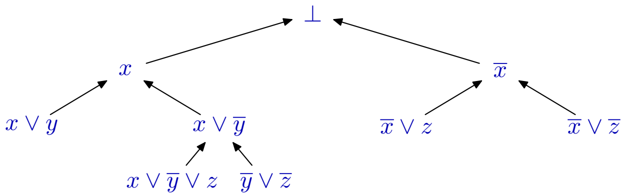


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:



and apply resolution rule bottom-up

Holds also for clause learning — makes tree into a DAG

Complexity Measures for Resolution: Summary

Recall that N = size of formula

Length

\# clauses in refutation — at most $\exp(N)$

Width

Size of largest clause in refutation — at most N

Space

Max # clauses one needs to remember when "verifying correctness of refutation" — at most N (!)

Proof Complexity Measures and CDCL Proof Search

Recall $\log(\text{length}) \lesssim \text{width} \lesssim \text{space}$

Proof Complexity Measures and CDCL Proof Search

Recall $\log(\text{length}) \lesssim \text{width} \lesssim \text{space}$

Length

- Lower bound on running time for CDCL
- CDCL polynomially simulates resolution [PD11]
- But short proofs may be worst-case intractable to find [AR08]

Proof Complexity Measures and CDCL Proof Search

Recall $\log(\text{length}) \lesssim$ width $\lesssim$ space

Length

- Lower bound on running time for CDCL
- CDCL polynomially simulates resolution [PD11]
- But short proofs may be worst-case intractable to find [AR08]

Width

- Searching in small width known heuristic in AI community
- Small width $\Rightarrow$ CDCL solver will run fast [AFT11]
- LBD measure in [AS09] kind of "generalised width" measure

Proof Complexity Measures and CDCL Proof Search

Recall $\log(\text{length}) \lesssim \text{width} \lesssim \text{space}$

Length

- Lower bound on running time for CDCL
- CDCL polynomially simulates resolution [PD11]
- But short proofs may be worst-case intractable to find [AR08]

Width

- Searching in small width known heuristic in AI community
- Small width $\Rightarrow$ CDCL solver will run fast [AFT11]
- LBD measure in [AS09] kind of "generalised width" measure

Space

- In practice, memory consumption important bottleneck
- Space complexity gives lower bound on clause database size
- Plus assumes solver knows **exactly** which clauses to keep $\Rightarrow$ in reality (much) more memory might be needed

Bridging the Gap Between Theory and Practice?

- CDCL hardness related to width and/or space?
  Preliminary work in [JMNŽ12] — no clear-cut answers

Bridging the Gap Between Theory and Practice?

- CDCL hardness related to width and/or space?
  Preliminary work in [JMNŽ12] — no clear-cut answers
- Or is CDCL as good as general resolution?
  Are [PD11] and [AFT11] results "true in practice"? Doubt it

Bridging the Gap Between Theory and Practice?

- CDCL hardness related to width and/or space?
  Preliminary work in [JMNŽ12] — no clear-cut answers
- Or is CDCL as good as general resolution?
  Are [PD11] and [AFT11] results "true in practice"? Doubt it
- CDCL explores only small part of resolution search space —
  Can time-space trade-offs occur in principle? Yes [EJL^{+}16]

Bridging the Gap Between Theory and Practice?

- CDCL hardness related to width and/or space?
  Preliminary work in [JMNŽ12] — no clear-cut answers
- Or is CDCL as good as general resolution?
  Are [PD11] and [AFT11] results "true in practice"? Doubt it
- CDCL explores only small part of resolution search space —
  Can time-space trade-offs occur in principle? Yes [EJL+16]
- Do such time-space trade-offs occur in practice?
  Great question — working on it; looks like yes [ENSS16]

Bridging the Gap Between Theory and Practice?

- CDCL hardness related to width and/or space?
  Preliminary work in [JMNŽ12] — no clear-cut answers
- Or is CDCL as good as general resolution?
  Are [PD11] and [AFT11] results "true in practice"? Doubt it
- CDCL explores only small part of resolution search space —
  Can time-space trade-offs occur in principle? Yes [EJL+16]
- Do such time-space trade-offs occur in practice?
  Great question — working on it; looks like yes [ENSS16]

Can we explain when CDCL does well and which formulas are hard?
Not mathematically well-defined question...
Still possible to run experiments and draw interesting conclusions?

Bridging the Gap Between Theory and Practice?

- CDCL hardness related to width and/or space?
  Preliminary work in [JMNŽ12] — no clear-cut answers
- Or is CDCL as good as general resolution?
  Are [PD11] and [AFT11] results "true in practice"? Doubt it
- CDCL explores only small part of resolution search space —
  Can time-space trade-offs occur in principle? Yes [EJL$^+$16]
- Do such time-space trade-offs occur in practice?
  Great question — working on it; looks like yes [ENSS16]

Can we explain when CDCL does well and which formulas are hard?
Not mathematically well-defined question. . .
Still possible to run experiments and draw interesting conclusions?

Some work in [KSM11], but diversity and sparsity of industrial benchmarks makes it hard to draw clear conclusions

Using Theory Benchmarks to Shed Light on CDCL? (1/2)

Generate scalable easy versions of benchmarks discussed in this talk
Have short resolution proofs, so no excuse for not doing well...

Run CDCL solver and vary settings to see how performance affected
Some preliminary findings (from upcoming paper [ENSS16]):

Using Theory Benchmarks to Shed Light on CDCL? (1/2)

Generate scalable easy versions of benchmarks discussed in this talk
Have short resolution proofs, so no excuse for not doing well. . .

Run CDCL solver and vary settings to see how performance affected
Some preliminary findings (from upcoming paper [ENSS16]):

Importance of restarts

- Sometimes very frequent restarts very important
- Crucial in [AFT11, PD11] to exploit full power of resolution
- Also seems to matter in practice for some formulas which are provably hard for subsystems of resolution

Using Theory Benchmarks to Shed Light on CDCL? (1/2)

Generate scalable easy versions of benchmarks discussed in this talk
Have short resolution proofs, so no excuse for not doing well. . .

Run CDCL solver and vary settings to see how performance affected
Some preliminary findings (from upcoming paper [ENSS16]):

Importance of restarts

- Sometimes very frequent restarts very important
- Crucial in [AFT11, PD11] to exploit full power of resolution
- Also seems to matter in practice for some formulas which are provably hard for subsystems of resolution

Clause erasure

- Theory says very aggressive clause removal could hurt badly
- Seem to see this on scaled-down versions of trade-off formulas

Using Theory Benchmarks to Shed Light on CDCL? (2/2)

Clause assessment

- Could heuristics identifying important clauses to keep compensate for aggressive erasures?
- Is LBD (literal block distance) such a heuristic? Maybe...

Using Theory Benchmarks to Shed Light on CDCL? (2/2)

Clause assessment

- Could heuristics identifying important clauses to keep compensate for aggressive erasures?
- Is LBD (literal block distance) such a heuristic? Maybe...

Variable branching

- Sometimes small variations in VSIDS decay factor (rate of forgetting) absolutely crucial (ordering principle)
- Does slow decay bring solver closer to tree-like resolution???

Using Theory Benchmarks to Shed Light on CDCL? (2/2)

Clause assessment

- Could heuristics identifying important clauses to keep compensate for aggressive erasures?
- Is LBD (literal block distance) such a heuristic? Maybe. . .

Variable branching

- Sometimes small variations in VSIDS decay factor (rate of forgetting) absolutely crucial (ordering principle)
- Does slow decay bring solver closer to tree-like resolution???

CDCL vs. resolution

- Sometimes CDCL fails miserably on easy formulas (Tseitin, even colouring [Mar06]) — VSIDS just goes dead wrong
- Sometimes strange easy-hard-easy patterns (zero-one designs)

Towards an Improved Understanding of CDCL

Open Problems

- *Could explanations of above phenomena help us understand CDCL better?*
- *Could experiments on easily scalable theoretical benchmarks yield other interesting insights?*

Towards an Improved Understanding of CDCL

Open Problems

- *Could explanations of above phenomena help us understand CDCL better?*
- *Could experiments on easily scalable theoretical benchmarks yield other interesting insights?*

And what about **community structure**? Not discussing this for several reasons:

- Intuitively very appealing, but not too much practical or theoretical work backing up this intuition
- Hard to analyze rigorously (and doesn't seem related to proof complexity, which is the focus of this talk)
- And you can only cover so much. . .

Polynomial Calculus

Introduced in [CEI96]; below modified version from [ABRW02]

Clauses interpreted as polynomial equations over finite field
Any field in theory; $\mathrm{GF}(2)$ in practice
Example: $x \lor y \lor \overline{z}$ gets translated to $xy\overline{z} = 0$
(Think of $0 \equiv \mathit{true}$ and $1 \equiv \mathit{false}$)

Polynomial Calculus

Introduced in [CEI96]; below modified version from [ABRW02]

Clauses interpreted as polynomial equations over finite field
Any field in theory; $\mathrm{GF}(2)$ in practice
Example: $x \lor y \lor \overline{z}$ gets translated to $xy\overline{z} = 0$
(Think of $0 \equiv \mathit{true}$ and $1 \equiv \mathit{false}$)

Derivation rules

Boolean axioms $\dfrac{}{x^2 - x = 0}$ Negation $\dfrac{}{x + \overline{x} = 1}$

Linear combination $\dfrac{p = 0 \qquad q = 0}{\alpha p + \beta q = 0}$ Multiplication $\dfrac{p = 0}{xp = 0}$

Goal: Derive $1 = 0 \Leftrightarrow$ no common root $\Leftrightarrow$ formula unsatisfiable

Size, Degree and Space

Clauses turn into monomials

Write out all polynomials as sums of monomials

W.l.o.g. all polynomials multilinear (because of Boolean axioms)

Size, Degree and Space

Clauses turn into monomials
Write out all polynomials as sums of monomials
W.l.o.g. all polynomials multilinear (because of Boolean axioms)

Size — analogue of resolution length
total # monomials in refutation counted with repetitions

Degree — analogue of resolution width
largest degree of monomial in refutation

(Monomial) space — analogue of resolution (clause) space
max # monomials in memory during refutation (with repetitions)

Polynomial Calculus Simulates Resolution

Polynomial calculus can simulate resolution proofs efficiently with respect to length/size, width/degree, and space simultaneously

- Can mimic resolution refutation step by step
- Hence worst-case upper bounds for resolution carry over

Polynomial Calculus Simulates Resolution

Polynomial calculus can simulate resolution proofs efficiently with respect to length/size, width/degree, and space simultaneously

- Can mimic resolution refutation step by step
- Hence worst-case upper bounds for resolution carry over

Example: Resolution step:

$$\frac{x \lor \overline{y} \lor z \qquad \overline{y} \lor \overline{z}}{x \lor \overline{y}}$$

Polynomial Calculus Simulates Resolution

Polynomial calculus can simulate resolution proofs efficiently with respect to length/size, width/degree, and space simultaneously

- Can mimic resolution refutation step by step
- Hence worst-case upper bounds for resolution carry over

Example: Resolution step:

$$\frac{x \lor \overline{y} \lor z \qquad \overline{y} \lor \overline{z}}{x \lor \overline{y}}$$

simulated by polynomial calculus derivation:

$$\dfrac{x\overline{y}z = 0 \qquad \dfrac{\dfrac{\overline{y}\,\overline{z} = 0}{x\overline{y}\,\overline{z} = 0} \qquad \dfrac{\dfrac{z + \overline{z} - 1 = 0}{\overline{y}z + \overline{y}\,\overline{z} - \overline{y} = 0}}{x\overline{y}z + x\overline{y}\,\overline{z} - x\overline{y} = 0}}{-x\overline{y}z + x\overline{y} = 0}}{x\overline{y} = 0}$$

Polynomial Calculus Strictly Stronger than Resolution

Polynomial calculus strictly stronger w.r.t. size and degree

- Tseitin formulas on expanders (just do Gaussian elimination)
- Onto functional pigeonhole principle [Rii93]

Polynomial Calculus Strictly Stronger than Resolution

Polynomial calculus strictly stronger w.r.t. size and degree

- Tseitin formulas on expanders (just do Gaussian elimination)
- Onto functional pigeonhole principle [Rii93]

Open Problem

Show that polynomial calculus is strictly stronger than resolution w.r.t. space

Size vs. Degree

- Degree upper bound ⇒ size upper bound [CEI96]
  Qualitatively similar to resolution bound
  A bit more involved argument
  Again essentially tight by [ALN16]

- Degree lower bound ⇒ size lower bound [IPS99]
  Precursor of [BW01] — can do same proof to get same bound

- Size-degree lower bound essentially optimal [GL10]
  Example: same ordering principle formulas

- Most size lower bounds for polynomial calculus derived via degree lower bounds (but machinery much less developed)

Examples of Hard Formulas w.r.t. Size (and Degree)

Pigeonhole principle formulas

Follows from [AR03]

Earlier work on other encodings in [Raz98, IPS99]

Hard even with functionality axioms added [MN15]

Tseitin formulas with "wrong modulus"

Can define Tseitin-like formulas counting $\mathrm{mod}\ p$ for $p \neq 2$

Hard if $p \neq$ characteristic of field [BGIP01]

Zero-one design formulas

Lower bound for both resolution and polynomial calculus in [MN14]

Random k-CNF formulas

Hard in all characteristics except 2 [BI99]

Lower bound for all characteristics in [AR03]

Polynomial Calculus Space

Monomial space lower bounds for

- pigeonhole principle [ABRW02]
- Random k-CNFs [BG15, BBG$^+$15]
- Tseitin formulas on (some) 4-regular expanders [FLM$^+$13]

Polynomial Calculus Space

Monomial space lower bounds for

- pigeonhole principle [ABRW02]
- Random k-CNFs [BG15, BBG$^+$15]
- Tseitin formulas on (some) 4-regular expanders [FLM$^+$13]

Open Problems

Prove polynomial calculus space lower bounds on

- *Tseitin formulas on any 3-regular expander*
- *3-CNF version of PHP formulas*

Open Problem (analogue of [AD08])

Is it true that space $\geq$ degree $+ \mathcal{O}(1)$?

Trade-offs for Polynomial Calculus

- Strong, essentially optimal space-degree trade-offs [BNT13]
  Same formulas as for resolution — same parameters
- Strong size-space trade-offs [BNT13]
  Same formulas as for resolution — some loss in parameters

Open Problem

Are there size-degree trade-offs in polynomial calculus?

[Tha14] works only for resolution (so far)

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]
- Promise of performance improvement failed to deliver

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution...

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus
- Fail to solve even onto FPHP formulas and some other formulas easy for polynomial calculus (should be fixable)

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus
- Fail to solve even onto FPHP formulas and some other formulas easy for polynomial calculus (should be fixable)
- Is it harder to build good algebraic SAT solvers, or is it just that too little work has been done (or both)?

Algebraic SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [CEI96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus
- Fail to solve even onto FPHP formulas and some other formulas easy for polynomial calculus (should be fixable)
- Is it harder to build good algebraic SAT solvers, or is it just that too little work has been done (or both)?
- Some shortcut seems to be needed — full Gröbner basis computation does too much work

Cutting Planes

Introduced in [CCT87] based on integer LP in [Gom63, Chv73]

Clauses interpreted as linear inequalities over the reals with integer coefficients

Example: $x \vee y \vee \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$
(Now $1 \equiv \mathit{true}$ and $0 \equiv \mathit{false}$ again)

Cutting Planes

Introduced in [CCT87] based on integer LP in [Gom63, Chv73]

Clauses interpreted as linear inequalities over the reals with integer coefficients

Example: $x \lor y \lor \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$

(Now $1 \equiv \mathit{true}$ and $0 \equiv \mathit{false}$ again)

Derivation rules

Variable axioms $\dfrac{}{0 \leq x \leq 1}$

Multiplication $\dfrac{\sum a_i x_i \geq A}{\sum c a_i x_i \geq cA}$

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

Division $\dfrac{\sum c a_i x_i \geq A}{\sum a_i x_i \geq \lceil A/c \rceil}$

Goal: Derive $0 \geq 1 \Leftrightarrow$ formula unsatisfiable

Size, Length and Space

Length = total # lines/inequalities in refutation

Size = sum also size of coefficients

Space = max # lines in memory during refutation

No (useful) analogue of width/degree

Size, Length and Space

Length = total # lines/inequalities in refutation

Size = sum also size of coefficients

Space = max # lines in memory during refutation

No (useful) analogue of width/degree

Cutting planes

- simulates resolution efficiently w.r.t. length/size and space simultaneously

Size, Length and Space

Length = total # lines/inequalities in refutation

Size = sum also size of coefficients

Space = max # lines in memory during refutation

No (useful) analogue of width/degree

Cutting planes

- simulates resolution efficiently w.r.t. length/size and space simultaneously
- is strictly stronger w.r.t. length/size — can refute PHP [CCT87] and zero-one design formulas efficiently

Size, Length and Space

Length = total # lines/inequalities in refutation

Size = sum also size of coefficients

Space = max # lines in memory during refutation

No (useful) analogue of width/degree

Cutting planes

- simulates resolution efficiently w.r.t. length/size and space simultaneously
- is strictly stronger w.r.t. length/size — can refute PHP [CCT87] and zero-one design formulas efficiently (and no cardinality encoding + CDCL can compensate for this)

Size, Length and Space

Length = total # lines/inequalities in refutation

Size = sum also size of coefficients

Space = max # lines in memory during refutation

No (useful) analogue of width/degree

Cutting planes

- simulates resolution efficiently w.r.t. length/size and space simultaneously
- is strictly stronger w.r.t. length/size — can refute PHP [CCT87] and zero-one design formulas efficiently (and no cardinality encoding + CDCL can compensate for this)
- is strictly stronger w.r.t. space — can refute any CNF in constant space 5 (!) [GPT15]

Size, Length and Space

Length = total # lines/inequalities in refutation

Size = sum also size of coefficients

Space = max # lines in memory during refutation

No (useful) analogue of width/degree

Cutting planes

- simulates resolution efficiently w.r.t. length/size and space simultaneously
- is strictly stronger w.r.t. length/size — can refute PHP [CCT87] and zero-one design formulas efficiently (and no cardinality encoding + CDCL can compensate for this)
- is strictly stronger w.r.t. space — can refute any CNF in constant space 5 (!) [GPT15] (But coefficients will be exponentially large — what if also coefficient size counted?)

Hard Formulas w.r.t. Cutting Planes Length

Clique-coclique formulas [Pud97]
"A graph with an m-clique is not $(m-1)$-colourable"

$p_{i,j}$ = indicator variables for edges in an n-vertex graph
$q_{k,i}$ = identifiers for members of m-clique in graph
$r_{i,\ell}$ = encoding of legal $(m-1)$-colouring of vertices

$q_{k,1} \lor q_{k,2} \lor \cdots \lor q_{k,n}$	some vertex is kth member of clique
$\overline{q}_{k,i} \lor \overline{q}_{k,j}$	kth clique member is uniquely defined
$p_{i,j} \lor \overline{q}_{k,i} \lor \overline{q}_{k',j}$	clique members are connected by edges
$r_{i,1} \lor r_{i,2} \lor \cdots \lor r_{i,m-1}$	every vertex i has a colour
$\overline{p}_{i,j} \lor \overline{r}_{i,\ell} \lor \overline{r}_{j,\ell}$	neighbours have distinct colours

Exponential lower bound via interpolation and circuit complexity
Technique very specifically tied to structure of formula

Open Problems for Cutting Planes Length and Space

Open Problems

Prove length lower bounds for cutting planes

- *for Tseitin formulas*
- *for random k-CNFs*
- *for any formula using other technique than interpolation*

Open Problems for Cutting Planes Length and Space

Open Problems

Prove length lower bounds for cutting planes

- *for Tseitin formulas*
- *for random k-CNFs*
- *for any formula using other technique than interpolation*

Open Problems

Prove space lower bounds for cutting planes

- *with constant-size coefficients (very weak bounds in [GPT15])*
- *with polynomial-size coefficients (nothing known)*

Size-Space Trade-offs for Cutting Planes?

- Short cutting planes refutations of (lifted) Tseitin formulas on expanders need large space [GP14] (but probably don't exist)
- Short cutting planes refutations of (some) pebbling formulas need large space [HN12, GP14] (and such refutations exist)

Results obtained via communication complexity

Size-Space Trade-offs for Cutting Planes?

- Short cutting planes refutations of (lifted) Tseitin formulas on expanders need large space [GP14] (but probably don't exist)
- Short cutting planes refutations of (some) pebbling formulas need large space [HN12, GP14] (and such refutations exist)

Results obtained via communication complexity

By [GPT15] get trade-offs with "constant space" upper bounds (but with coefficients of exponential size)

Doesn't seem like a too relevant a trade-off — exponential size coefficients doesn't feel like "small space"

Size-Space Trade-offs for Cutting Planes?

- Short cutting planes refutations of (lifted) Tseitin formulas on expanders need large space [GP14] (but probably don't exist)
- Short cutting planes refutations of (some) pebbling formulas need large space [HN12, GP14] (and such refutations exist)

Results obtained via communication complexity

By [GPT15] get trade-offs with "constant space" upper bounds (but with coefficients of exponential size)

Doesn't seem like a too relevant a trade-off — exponential size coefficients doesn't feel like "small space"

Open Problem

Are there trade-offs where the space-efficient CP refutations have small coefficients? (Say, of polynomial or even constant size)

Size-Space Trade-offs for Cutting Planes!

Breaking news: Yes, there are such trade-offs!

Theorem ([dRNV16])

There exist flavours of pebbling formulas such that

- $\exists$ *small-size refutations with constant-size coefficients*
- $\exists$ *small-space refutations with constant-size coefficients*
- *Decreasing the space even for refutations with exponentially large coefficients causes exponential blow-up of length*

Size-Space Trade-offs for Cutting Planes!

Breaking news: Yes, there are such trade-offs!

Theorem ([dRNV16])

There exist flavours of pebbling formulas such that

- $\exists$ *small-size refutations with constant-size coefficients*
- $\exists$ *small-space refutations with constant-size coefficients*
- *Decreasing the space even for refutations with exponentially large coefficients causes exponential blow-up of length*

- Results hold uniformly for resolution, polynomial calculus and cutting planes
- Again uses communication complexity (+ several other twists)
- Downside: Parameters worse than in previous results

What About Conflict-Driven Cutting Planes Solvers?

There are so-called pseudo-Boolean solvers using (subset of) cutting planes — seems hard to make competitive with CDCL

What About Conflict-Driven Cutting Planes Solvers?

There are so-called pseudo-Boolean solvers using (subset of) cutting planes — seems hard to make competitive with CDCL

Possible to combine reasoning power of cutting planes with efficiency of CDCL? Work in this direction in, e.g., Sat4j [LP10]

What About Conflict-Driven Cutting Planes Solvers?

There are so-called pseudo-Boolean solvers using (subset of) cutting planes — seems hard to make competitive with CDCL

Possible to combine reasoning power of cutting planes with efficiency of CDCL? Work in this direction in, e.g., Sat4j [LP10]

Several challenges:

- How detect unit propagation? Not enough to watch just 2 literals (or any finite number)

What About Conflict-Driven Cutting Planes Solvers?

There are so-called pseudo-Boolean solvers using (subset of) cutting planes — seems hard to make competitive with CDCL

Possible to combine reasoning power of cutting planes with efficiency of CDCL? Work in this direction in, e.g., Sat4j [LP10]

Several challenges:

- How detect unit propagation? Not enough to watch just 2 literals (or any finite number)
- Linear constraints more complicated than clauses — and integer arithmetic can become expensive

What About Conflict-Driven Cutting Planes Solvers?

There are so-called pseudo-Boolean solvers using (subset of) cutting planes — seems hard to make competitive with CDCL

Possible to combine reasoning power of cutting planes with efficiency of CDCL? Work in this direction in, e.g., Sat4j [LP10]

Several challenges:

- How detect unit propagation? Not enough to watch just 2 literals (or any finite number)
- Linear constraints more complicated than clauses — and integer arithmetic can become expensive
- Not obvious how to do conflict analysis
  - Can sometimes skip "resolution steps" in conflict analysis with propagating constraints on reason side — good or bad?
  - Can happen that "resolvent" is not conflicting — can be fixed in several ways, but what way is best?

Conflict-Driven CP Solvers: Two Concrete Obstacles

- **Roadblock 1:** Given CNF input, solvers cannot discover and use cardinality constraints (too limited form of addition)

Conflict-Driven CP Solvers: Two Concrete Obstacles

- **Roadblock 1:** Given CNF input, solvers cannot discover and use cardinality constraints (too limited form of addition)
- But given more helpful encoding, solvers can do really well (e.g., PHP and zero-one design formulas) [BLLM14]

Conflict-Driven CP Solvers: Two Concrete Obstacles

- **Roadblock 1:** Given CNF input, solvers cannot discover and use cardinality constraints (too limited form of addition)
- But given more helpful encoding, solvers can do really well (e.g., PHP and zero-one design formulas) [BLLM14]
- **Roadblock 2(?):** Solvers seem inefficient for systems of inequalities that have rational but not integral solutions (too limited form of division?)

Conflict-Driven CP Solvers: Two Concrete Obstacles

- **Roadblock 1:** Given CNF input, solvers cannot discover and use cardinality constraints (too limited form of addition)
- But given more helpful encoding, solvers can do really well (e.g., PHP and zero-one design formulas) [BLLM14]
- **Roadblock 2(?):** Solvers seem inefficient for systems of inequalities that have rational but not integral solutions (too limited form of division?)
- Fail on, e.g., even colouring formulas [Mar06] for no good reason
- Not well understood at all — work in progress

Building SAT Solvers on Extended Resolution?

- Resolution + introduce new variables to name subformulas
- Combined with CDCL in, e.g., [AKS10, Hua10]
- Without restrictions, corresponds to extended Frege system
- Extremely strong — pretty much no lower bounds known
- In order to analyze solvers using extended resolution, would need to:
  - Describe heuristics/rules actually used
  - See if possible to reason about such restricted proof system

Summing up This Presentation

Overview of resolution, polynomial calculus and cutting planes
(More details in survey papers [Nor13, Nor15])

- Resolution fairly well understood
- Polynomial calculus less so
- Cutting planes almost not at all

Summing up This Presentation

Overview of resolution, polynomial calculus and cutting planes
(More details in survey papers [Nor13, Nor15])

- Resolution fairly well understood
- Polynomial calculus less so
- Cutting planes almost not at all

Open problems motivated by applied SAT solving

- Can proof complexity measures shed more light on the hardness (or easiness) of SAT?
- Is it possible to build efficient SAT solvers based on stronger proof systems than resolution?

Summing up This Presentation

Overview of resolution, polynomial calculus and cutting planes
(More details in survey papers [Nor13, Nor15])

- Resolution fairly well understood
- Polynomial calculus less so
- Cutting planes almost not at all

Open problems motivated by applied SAT solving

- Can proof complexity measures shed more light on the hardness (or easiness) of SAT?
- Is it possible to build efficient SAT solvers based on stronger proof systems than resolution?

Thank you for your attention!

References I

[ABRW02] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. Preliminary version in *STOC '00*.

[AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version in *CCC '03*.

[AFT11] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, January 2011. Preliminary version in *SAT '09*.

[AKS10] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning SAT solvers. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 15–20, July 2010.

[ALN16] Albert Atserias, Massimo Lauria, and Jakob Nordström. Narrow proofs may be maximally long. *ACM Transactions on Computational Logic*, 17:19:1–19:30, May 2016. Preliminary version in *CCC '14*.

References II

[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*.

[AR08] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM Journal on Computing*, 38(4):1347–1363, October 2008. 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.

[BBG+15] Patrick Bennett, Ilario Bonacina, Nicola Galesi, Tony Huynh, Mike Molloy, and Paul Wollan. Space proof complexity for random 3-CNFs. Technical Report 1503.01613, arXiv.org, April 2015.

References III

[BBI12] Paul Beame, Chris Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12)*, pages 213–232, May 2012.

[BCMM05] Paul Beame, Joseph C. Culberson, David G. Mitchell, and Cristopher Moore. The resolution complexity of random graph k-colorability. *Discrete Applied Mathematics*, 153(1-3):25–47, December 2005.

[Ben09] Eli Ben-Sasson. Size-space tradeoffs for resolution. *SIAM Journal on Computing*, 38(6):2511–2525, May 2009. Preliminary version in *STOC '02*.

[BG01] María Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10(4):261–276, December 2001. Preliminary version in *FOCS '99*.

[BG03] Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, August 2003. Preliminary version in *CCC '01*.

References IV

[BG15] Ilario Bonacina and Nicola Galesi. A framework for space complexity in algebraic proof systems. *Journal of the ACM*, 62(3):23:1–23:20, June 2015. Preliminary version in *ITCS '13*.

[BGIP01] Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *Journal of Computer and System Sciences*, 62(2):267–289, March 2001. Preliminary version in *CCC '99*.

[BI99] Eli Ben-Sasson and Russell Impagliazzo. Random CNF's are hard for the polynomial calculus. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS '99)*, pages 415–421, October 1999. Journal version in [BI10].

[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.

References V

[BIS07] Paul Beame, Russell Impagliazzo, and Ashish Sabharwal. The resolution complexity of independent sets and vertex covers in random graphs. *Computational Complexity*, 16(3):245–297, October 2007.

[BLLM14] Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. 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 285–301. Springer, July 2014.

[BN08] Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*, pages 709–718, October 2008.

[BN11] Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS '11)*, pages 401–416, January 2011.

References VI

[BNT13] Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 813–822, May 2013.

[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.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*.

[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.

References VII

[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.

[Chv73] Vašek Chvátal. Edmond polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(1):305–337, 1973.

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

[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.

References VIII

[dRNV16] Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16)*, pages 466–485, October 2016. To appear.

[EJL+16] Jan Elffers, Jan Johannsen, Massimo Lauria, Thomas Magnard, Jakob Nordström, and Marc Vinyals. Trade-offs between time and memory in a tighter model of CDCL SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 160–176. Springer, July 2016.

[ENSS16] Jan Elffers, Jakob Nordström, Laurent Simon, and Karem A. Sakallah. Seeking practical CDCL insights from theoretical SAT benchmarks. Manuscript in preparation, 2016.

[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.

References IX

[ET01] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions of these results appeared in *STACS '99* and *CSL '99*.

[FLM+13] Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals. Towards an understanding of polynomial calculus: New separations and lower bounds (Extended abstract). In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP '13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 437–448. Springer, July 2013.

[GL10] Nicola Galesi and Massimo Lauria. Optimality of size-degree trade-offs for polynomial calculus. *ACM Transactions on Computational Logic*, 12:4:1–4:22, November 2010.

[Gom63] Ralph E. Gomory. An algorithm for integer solutions of linear programs. In R.L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

References X

[GP14] Mika Göös and Toniann Pitassi. Communication lower bounds via critical block sensitivity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC '14)*, pages 847–856, May 2014.

[GPT15] Nicola Galesi, Pavel Pudlák, and Neil Thapen. The space complexity of cutting planes refutations. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 433–447, June 2015.

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

[HN12] Trinh Huynh and Jakob Nordström. On the virtue of succinct proofs: Amplifying communication complexity hardness to time-space trade-offs in proof complexity (Extended abstract). In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12)*, pages 233–248, May 2012.

[Hua10] Jinbo Huang. Extended clause learning. *Artificial Intelligence*, 174(15):1277–1284, October 2010.

References XI

[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.

[JMNŽ12] Matti Järvisalo, Arie Matsliah, Jakob Nordström, and Stanislav Živný. Relating proof complexity measures and practical hardness of SAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 316–331. Springer, October 2012.

[KSM11] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT '11)*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer, June 2011.

[LP10] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[Mar06] Klas Markström. Locality and hard SAT-instances. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):221–227, 2006.

References XII

[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 XIII

[NH13] Jakob Nordström and Johan Håstad. Towards an optimal separation of space and length in resolution. *Theory of Computing*, 9:471–557, May 2013. Preliminary version in *STOC '08*.

[Nor09] Jakob Nordström. Narrow proofs may be spacious: Separating space and width in resolution. *SIAM Journal on Computing*, 39(1):59–121, May 2009. Preliminary version in *STOC '06*.

[Nor13] Jakob Nordström. Pebble games, proof complexity and time-space trade-offs. *Logical Methods in Computer Science*, 9:15:1–15:63, September 2013.

[Nor15] Jakob Nordström. On the interplay between proof complexity and SAT solving. *ACM SIGLOG News*, 2(3):19–44, July 2015.

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175:512–525, February 2011. Preliminary version in *CP '09*.

References XIV

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

[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.

[Spe10] Ivor Spence. sgen1: A generator of small but difficult satisfiability benchmarks. *Journal of Experimental Algorithmics*, 15:1.2:1.1–1.2:1.15, March 2010.

[Stå96] Gunnar Stålmarck. Short resolution proofs for a sequence of tricky formulas. *Acta Informatica*, 33(3):277–280, May 1996.

[Tha14] Neil Thapen. A trade-off between length and width in resolution. Technical Report TR14-137, Electronic Colloquium on Computational Complexity (ECCC), October 2014.

References XV

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

[VS10] Allen Van Gelder and Ivor Spence. Zero-one designs produce small hard SAT instances. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 388–397. Springer, July 2010.