Time-Space Trade-offs in Proof Complexity (and SAT Solving?)

Jakob Nordström

School of Computer Science and Communication
KTH Royal Institute of Technology

Algorithms, Complexity and Machine Learning:
A Tribute to Kurt Mehlhorn
Chalmers University of Technology
October 23, 2014

A Fundamental Theoretical Problem. . .

Problem

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

A Fundamental Theoretical Problem...

Problem

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

TAUTOLOGY: Fundamental problem in theoretical computer science ever since Stephen Cook's NP-completeness paper in 1971

(And significance realized much earlier — cf. Gödel's letter 1956)

These days recognized as one of the main challenges for all of mathematics as identified by the Clay Mathematics Institute

Widely believed intractable in worst case — deciding whether this is so is one of the famous million dollar Millennium Problems

. . . with Huge Practical Implications

- All known algorithms run in exponential time in worst case
- But enormous progress on applied computer programs last 10–20 years (with important contributions from Chalmers)
- These so-called SAT solvers routinely deployed to solve large-scale real-world problems with millions of variables
- Used in e.g. hardware verification, software testing, software package management, artificial intelligence, cryptography, bioinformatics, and more
- But we also know small formulas with only about a hundred variables that trip up even state-of-the-art SAT solvers

Theoretical Understanding of SAT Solver Performance?

- Best known algorithms based on simple DPLL method (Davis-Putnam-Logemann-Loveland) from early 1960s extended with conflict-driven clause learning (CDCL)
- Can we gain better theoretical understanding of potential and limitations of CDCL SAT solvers?
- Key concerns in SAT solving: time and memory management
- What are the connections between these resources? Are they correlated? Are there trade-offs?
- This talk: What can the field of proof complexity say about these questions?

Tautologies and CNF Formulas

Conjunctive normal form (CNF)

ANDs of ORs of variables or negated variables
(or conjunctions of disjunctive clauses over literals)

Example:

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

Tautologies and CNF Formulas

Conjunctive normal form (CNF)

ANDs of ORs of variables or negated variables
(or conjunctions of disjunctive clauses over literals)

Example:

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

Proving that a formula in propositional logic is **always** satisfied

$\Updownarrow$

Proving that a CNF formula is **never** satisfied

I.e., evaluates to false however you set the variables

Tautologies and CNF Formulas

Conjunctive normal form (CNF)

ANDs of ORs of variables or negated variables
(or conjunctions of disjunctive clauses over literals)

Example:

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

Proving that a formula in propositional logic is **always** satisfied

$\Updownarrow$

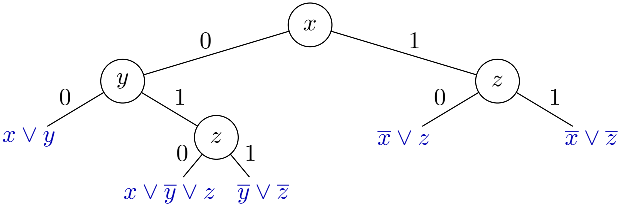
Proving that a CNF formula is **never** satisfied
I.e., evaluates to false however you set the variables

(Sidenote: Can assume k-CNF — all clauses of constant size $\leq k$)

A Very Simplified Description of DPLL

Visualize execution of DPLL algorithm as search tree

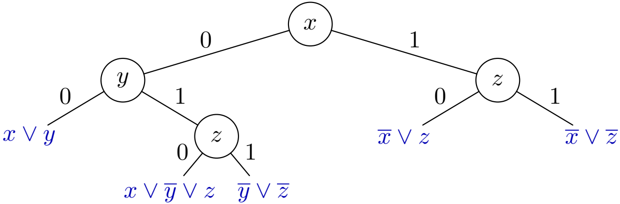
- Branch on variable assignments in internal nodes
- Stop in leaves when falsfied clause found



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)

The Resolution Proof System

Resolution rule:

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

The Resolution Proof System

Resolution rule:

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

Observation

If F is a satisfiable CNF formula and D is derived from clauses $C_1, C_2 \in F$ by the resolution rule, then $F \land D$ is satisfiable.

The Resolution Proof System

Resolution rule:

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

Observation

If F is a satisfiable CNF formula and D is derived from clauses $C_1, C_2 \in F$ by the resolution rule, then $F \land D$ is satisfiable.

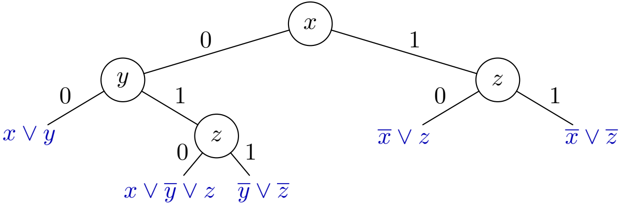
Prove F unsatisfiable by deriving the unsatisfiable empty clause $\bot$ from F by resolution

Proof of unsatisfiability = Refutation of formula
Will use terms "proof" and "refutation" as synonyms

DPLL and Resolution

A DPLL execution is essentially a resolution proof

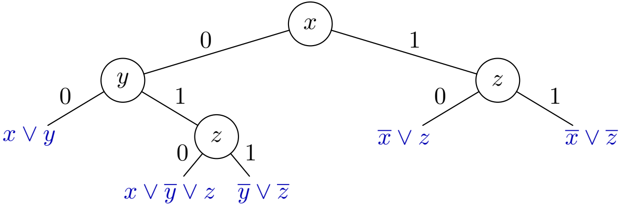
Look at our example again:



DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

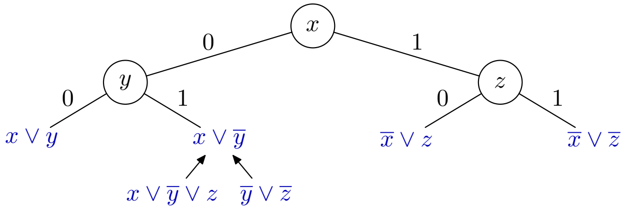


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

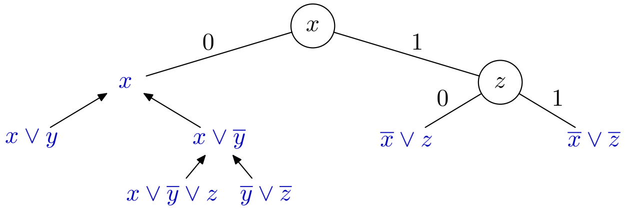


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

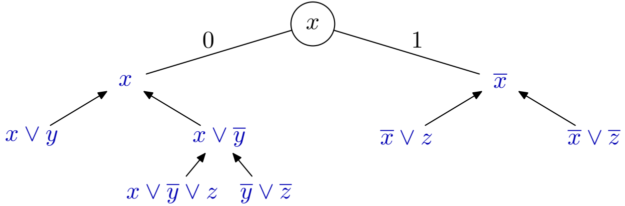


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

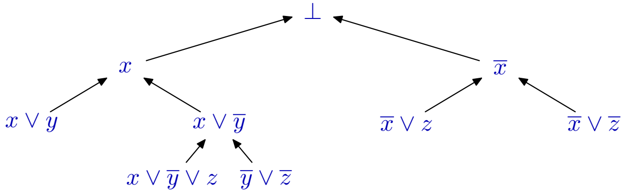


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again:

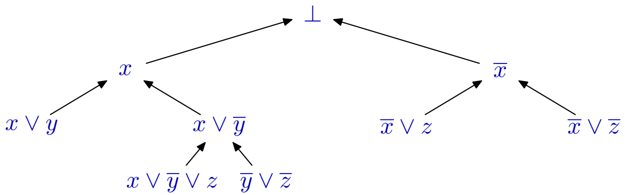


and apply resolution rule bottom-up

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

The Formal Model

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 ends when empty clause $\perp$ derived

The Formal Model

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 ends when empty clause $\perp$ derived

Can represent refutation as

- annotated list or
- DAG

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$(7, 8)$

The Formal Model

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

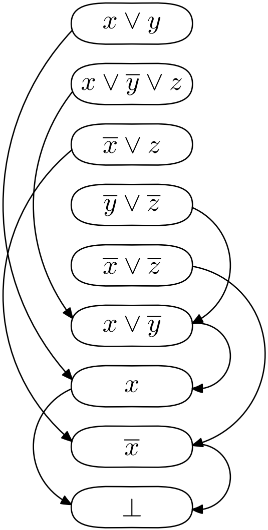
Derive new clauses by resolution rule

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

Refutation ends when empty clause $\bot$ derived

Can represent refutation as

- annotated list or
- DAG



Resolution Size/Length

Let N = size of formula (total # literals)

Size/length = # clauses in refutation

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 in e.g. [Urq87, CS88, BW01]

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

Resolution Space

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

Resolution Space

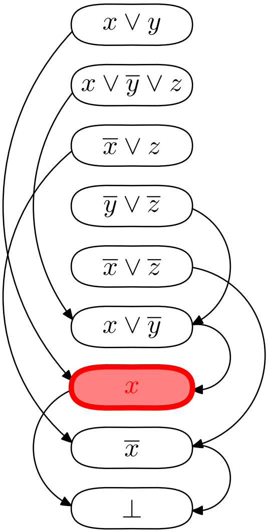
Space = max # clauses in memory when performing refutation

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 . . .



Resolution Space

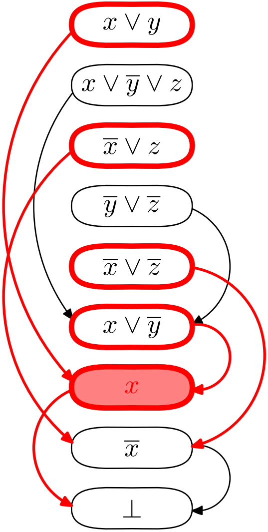
Space = max # clauses in memory when performing refutation

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]

- Build search tree of depth $\leq N$
- Derive root clause of one subtree
- Keep in memory while doing other subtree; then resolve
- $\#$ clauses needed in memory scales like tree height

Bounds on Resolution Space

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

- Build search tree of depth $\leq N$
- Derive root clause of one subtree
- Keep in memory while doing other subtree; then resolve
- $\#$ clauses needed in memory scales like tree height

Matching $\Omega(N)$ lower bounds in e.g. [ET01, ABRW02, BG03]

Bounds on Resolution Space

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

- Build search tree of depth $\leq N$
- Derive root clause of one subtree
- Keep in memory while doing other subtree; then resolve
- $\#$ clauses needed in memory scales like tree height

Matching $\Omega(N)$ lower bounds in e.g. [ET01, ABRW02, BG03]

Two comments/questions:

- Lower bounds hold even for "magic algorithms" making optimal choices — maybe much stronger in practice?
- Linear upper bounds hold for exponential-size proofs — what about space for reasonably-sized proofs?

Comparing Size and Space

Some “rescaling” needed to get meaningful comparisons of size and space

- Size exponential in formula size in worst case
- Space at most linear
- So natural to compare space to logarithm of size

Size-Space Correlations?

$\exists$ constant space refutation $\Rightarrow$ $\exists$ polynomial size refutation [AD03]

What about other direction — does small size imply small space?

Size-Space Correlations?

$\exists$ constant space refutation $\Rightarrow$ $\exists$ polynomial size refutation [AD03]

What about other direction — does small size imply small space?
No, false in strongest sense possible

Theorem ([BN08])

There are k-CNF formula families of size N

- *refutable in size $\mathcal{O}(N)$*
- *requiring space $\Omega(N/\log N)$*

Optimal separation — given proof size $\mathcal{O}(N)$, always possible to achieve proof space $\mathcal{O}(N/\log N)$

Size-Space Trade-offs

Can also show collection of size-space trade-off results

Formulas are simple and explicit

Theorem ((informal) [BN11])

There are k-CNF formulas for which

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

So no meaningful simultaneous optimization possible for size and space in the worst case

How to Get a Handle on Time-Space Relations?

Questions about time-space trade-offs fundamental in theoretical computer science

How to Get a Handle on Time-Space Relations?

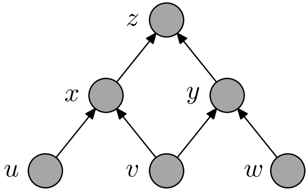
Questions about time-space trade-offs fundamental in theoretical computer science

In particular, well-studied (and well-understood) for pebble games modelling calculations described by directed acyclic graphs ([CS76] and many others)

- Time needed for calculation: # pebbling moves
- Space needed for calculation: max # pebbles required

The Black-White Pebble Game

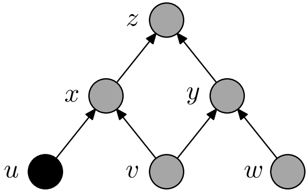
Goal: get single black pebble on sink vertex z of G



# moves	0
Current # pebbles	0
Max # pebbles so far	0

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

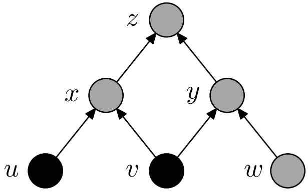


# moves	1
Current # pebbles	1
Max # pebbles so far	1

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

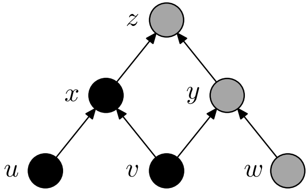


# moves	2
Current # pebbles	2
Max # pebbles so far	2

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

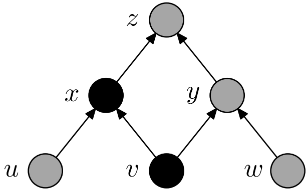


# moves	3
Current # pebbles	3
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

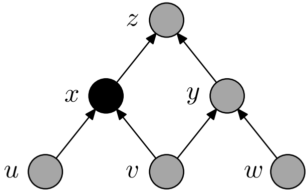


# moves	4
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

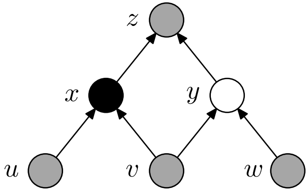


# moves	5
Current # pebbles	1
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

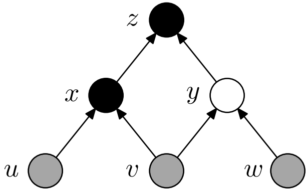


# moves	6
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

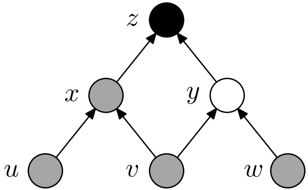


# moves	7
Current # pebbles	3
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

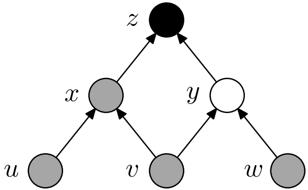


# moves	8
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

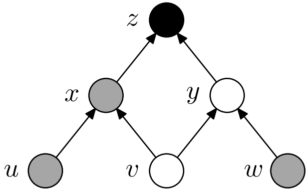


# moves	8
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble if all predecessors have pebbles

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

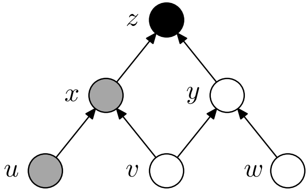


# moves	9
Current # pebbles	3
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble if all predecessors have pebbles

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

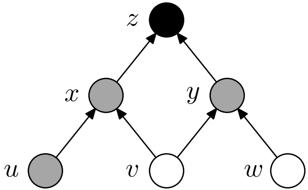


# moves	10
Current # pebbles	4
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble if all predecessors have pebbles

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

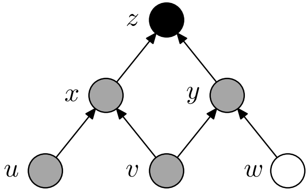


# moves	11
Current # pebbles	3
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble if all predecessors have pebbles

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G

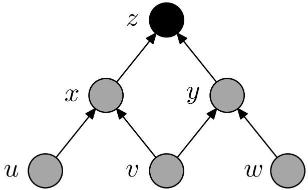


# moves	12
Current # pebbles	2
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble if all predecessors have pebbles

The Black-White Pebble Game

Goal: get single black pebble on sink vertex z of G



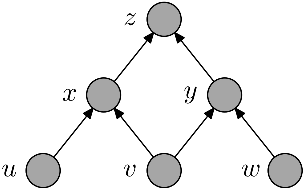
# moves	13
Current # pebbles	1
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all predecessors (vertices with edges to v) have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble if all predecessors have pebbles

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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. $\overline{z}$

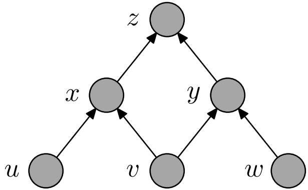


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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. $\overline{z}$

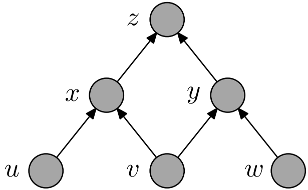


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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. $\overline{z}$

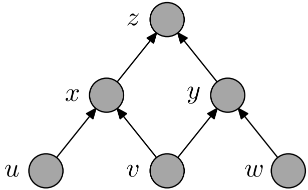


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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. $\overline{z}$

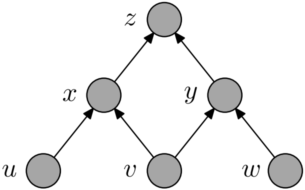


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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. $\overline{z}$



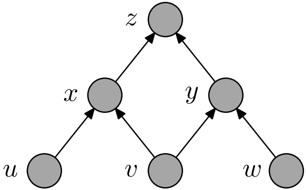
- sources are true
- truth propagates upwards
- but sink is false

Extensive literature on pebbling from 1970s and 80s

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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

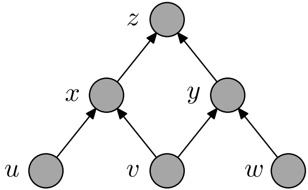
Extensive literature on pebbling from 1970s and 80s

In particular, the kind of time-space separations and trade-offs we want for resolution are known to hold for pebbling

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

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 from 1970s and 80s

In particular, the kind of time-space separations and trade-offs we want for resolution are known to hold for pebbling

Hope that pebbling properties of DAGs somehow carry over to resolution refutations of pebbling contradictions

A Problem and a Fix: Variable Substitution

Problem: Pebbling contradictions supereasy (solved by unit propagation) — no nontrivial lower bounds possible

A Problem and a Fix: Variable Substitution

Problem: Pebbling contradictions supereasy (solved by unit propagation) — no nontrivial lower bounds possible

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

A Problem and a Fix: Variable Substitution

Problem: Pebbling contradictions supereasy (solved by unit propagation) — no nontrivial lower bounds possible

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

$$\overline{x} \lor z$$

$$\Downarrow$$

$$\neg(x_1 \oplus x_2) \lor (z_1 \oplus z_2)$$

$$\Downarrow$$

$$\begin{aligned} & (x_1 \lor \overline{x}_2 \lor z_1 \lor z_2) \\ \land\ & (x_1 \lor \overline{x}_2 \lor \overline{z}_1 \lor \overline{z}_2) \\ \land\ & (\overline{x}_1 \lor x_2 \lor z_1 \lor z_2) \\ \land\ & (\overline{x}_1 \lor x_2 \lor \overline{z}_1 \lor \overline{z}_2) \end{aligned}$$

A Problem and a Fix: Variable Substitution

Problem: Pebbling contradictions supereasy (solved by unit propagation) — no nontrivial lower bounds possible

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

$$\overline{x} \vee z$$

$$\Downarrow$$

$$\neg(x_1 \oplus x_2) \vee (z_1 \oplus z_2)$$

$$\Downarrow$$

$$\begin{aligned} & (x_1 \vee \overline{x}_2 \vee z_1 \vee z_2) \\ \wedge\ & (x_1 \vee \overline{x}_2 \vee \overline{z}_1 \vee \overline{z}_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee z_1 \vee z_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee \overline{z}_1 \vee \overline{z}_2) \end{aligned}$$

Now CNF formula inherits pebbling graph properties!

Trade-offs in the Superlinear Space Regime?

But...

- Pebbling contradictions always refutable in linear size and linear space simultaneously
- If exists small proof, always possible to find in linear space?
- Or are there formulas for which small proofs require superlinear space?

Trade-offs in the Superlinear Space Regime?

But. . .

- Pebbling contradictions always refutable in linear size and linear space simultaneously
- If exists small proof, always possible to find in linear space?
- Or are there formulas for which small proofs require superlinear space?

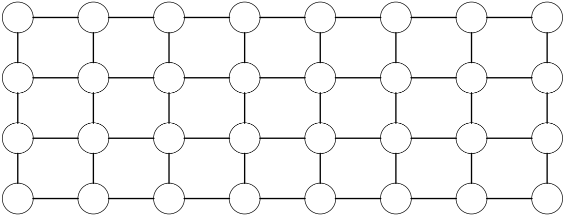
Theorem (informal [BBI12, BNT13])

For every $s \in \mathbb{N}^+$ there are k-CNF formulas for which

- *exist small proofs in size $N^{s+\mathcal{O}(1)}$ and space $N^{s+\mathcal{O}(1)}$*
- *exist space-efficient proofs in space $\mathcal{O}\left(s \log^2 N\right)$*
- *any proof in space $\mathcal{O}\left(N^{s/2}\right)$ requires superpolynomial size*

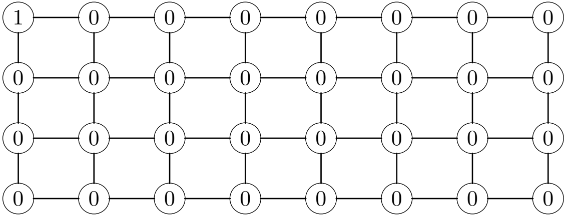
Tseitin Formulas over Long, Skinny Grids

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



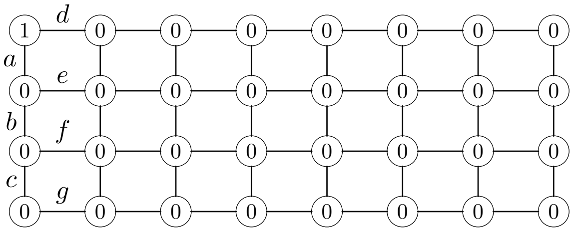
Tseitin Formulas over Long, Skinny Grids

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



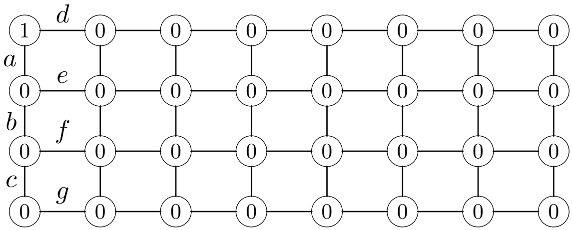
Tseitin Formulas over Long, Skinny Grids

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



Tseitin Formulas over Long, Skinny Grids

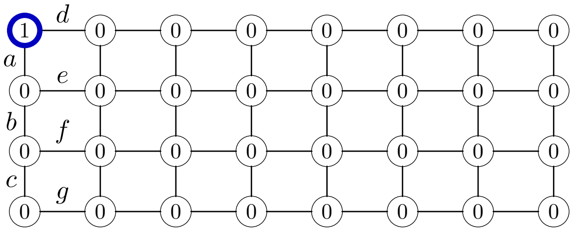
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Let variables = edges
- Write down clauses encoding constraints
  "vertex label = parity of incident edges"



Tseitin Formulas over Long, Skinny Grids

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

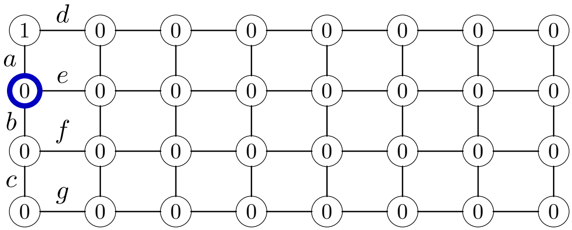
$(a \lor d)$
$\land\ (\overline{a} \lor \overline{d})$



Tseitin Formulas over Long, Skinny Grids

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

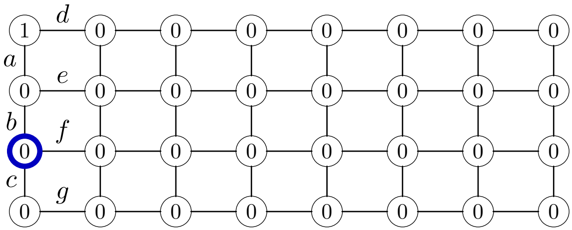
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e})
\end{aligned}
$$



Tseitin Formulas over Long, Skinny Grids

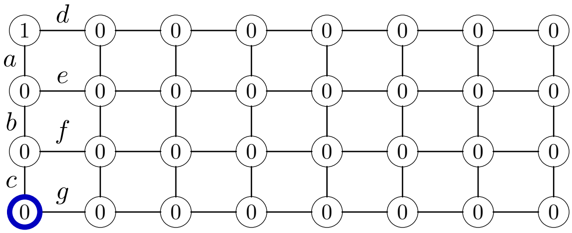
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Let variables = edges
- Write down clauses encoding constraints
  "vertex label = parity of incident edges"

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



Tseitin Formulas over Long, Skinny Grids

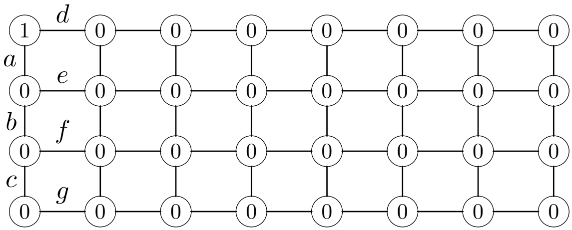
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Let variables = edges
- Write down clauses encoding constraints "vertex label = parity of incident edges"



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

Tseitin Formulas over Long, Skinny Grids

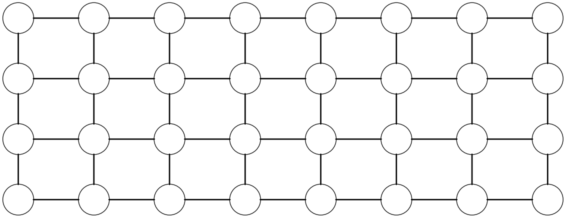
- Take $w \times m$ grid, $w = \mathcal{O}(\log m)$
- Label vertices $0/1$ with total charge odd
- Let variables = edges
- Write down clauses encoding constraints "vertex label = parity of incident edges"
- Unsatisfiable — every edge counted twice, so total sum can't be odd



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

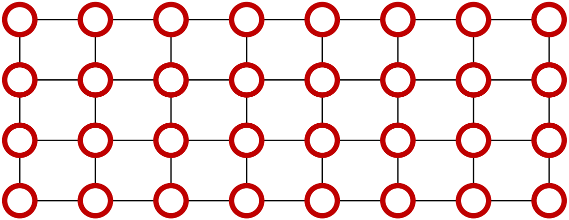
Small-Space "Divide-and-Conquer" Proof

- Build DPLL search tree querying edges



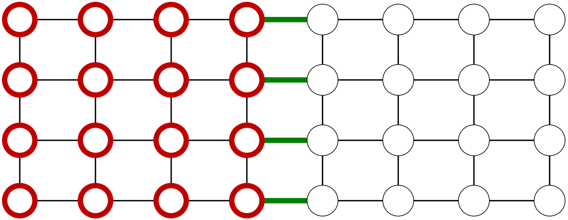
Small-Space "Divide-and-Conquer" Proof

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



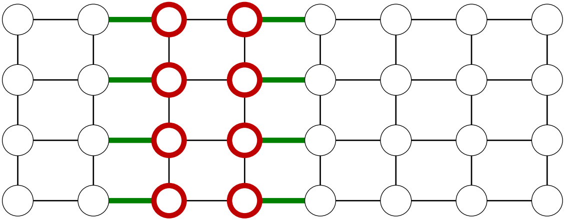
Small-Space "Divide-and-Conquer" Proof

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



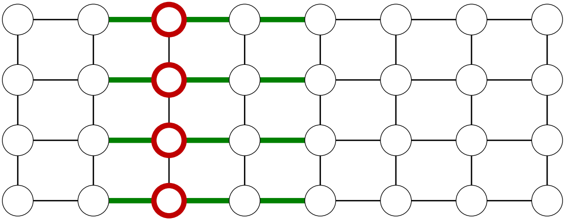
Small-Space "Divide-and-Conquer" Proof

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



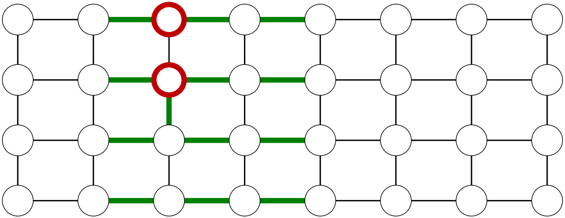
Small-Space "Divide-and-Conquer" Proof

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



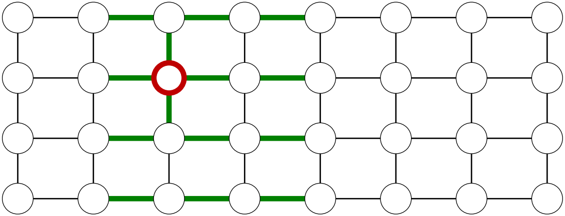
Small-Space "Divide-and-Conquer" Proof

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



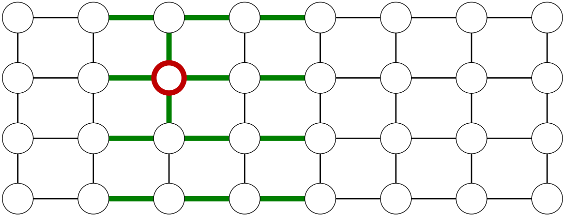
Small-Space "Divide-and-Conquer" Proof

- 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



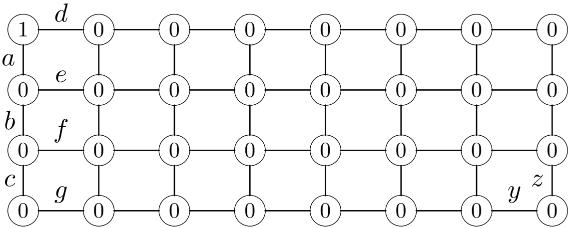
Small-Space "Divide-and-Conquer" Proof

- 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)



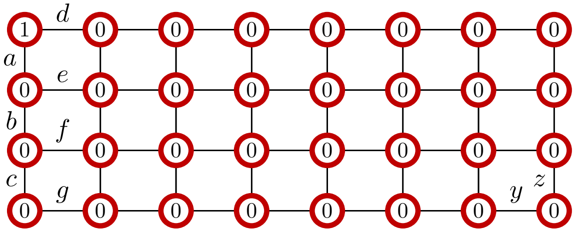
Small-Size "Dynamic programming" Proof

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



Small-Size "Dynamic programming" Proof

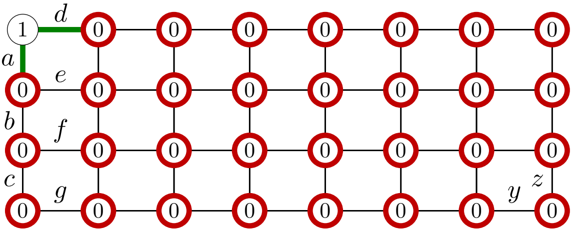
- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex



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

$a + d = 1$

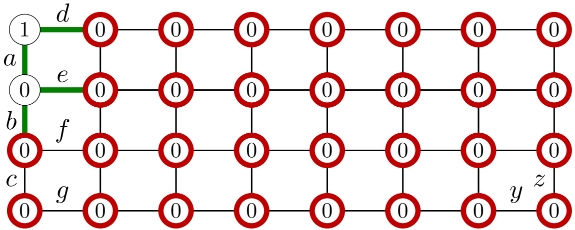


Small-Size "Dynamic programming" Proof

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

$a + d = 1$

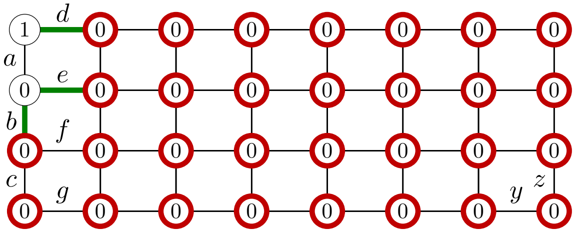
$a + b + e = 0$



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

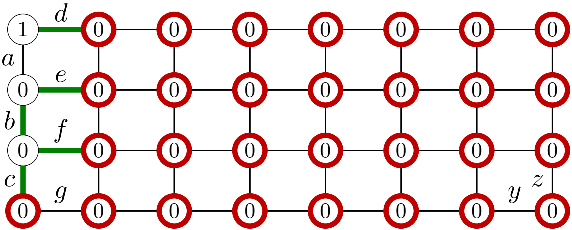
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$



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

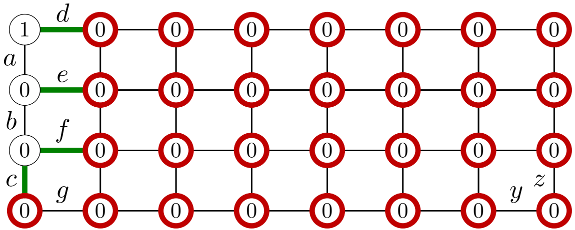
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$



Small-Size "Dynamic programming" Proof

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

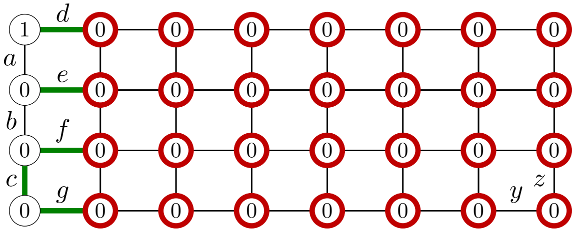
$$a+d=1$$
$$a+b+e=0$$
$$b+d+e=1$$
$$b+c+f=0$$
$$c+d+e+f=1$$



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

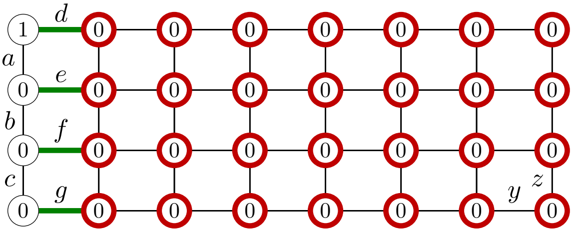
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$



Small-Size “Dynamic programming” Proof

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

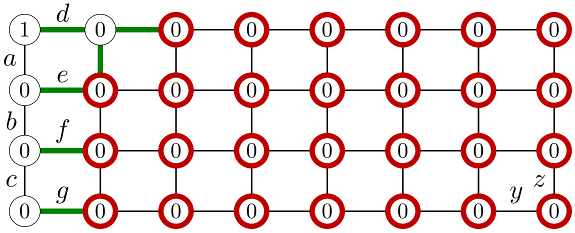
$$a+d=1$$
$$a+b+e=0$$
$$b+d+e=1$$
$$b+c+f=0$$
$$c+d+e+f=1$$
$$c+g=0$$
$$d+e+f+g=1$$



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

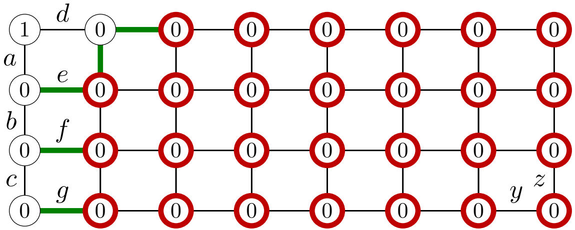
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

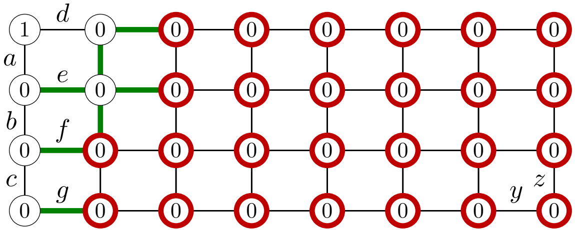
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic programming" Proof

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

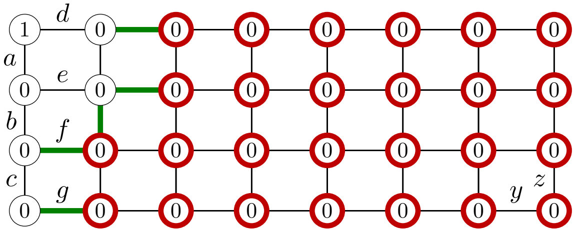
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic programming" Proof

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

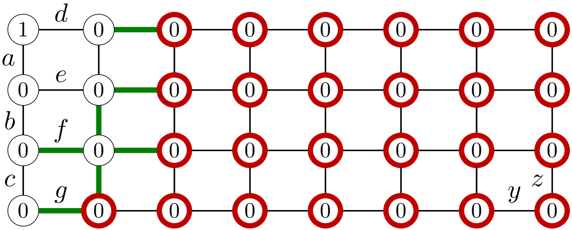
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

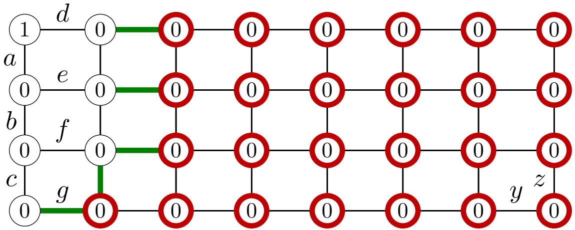
$$
\begin{aligned}
a + d &= 1 \\
a + b + e &= 0 \\
b + d + e &= 1 \\
b + c + f &= 0 \\
c + d + e + f &= 1 \\
c + g &= 0 \\
d + e + f + g &= 1 \\
&\vdots
\end{aligned}
$$



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

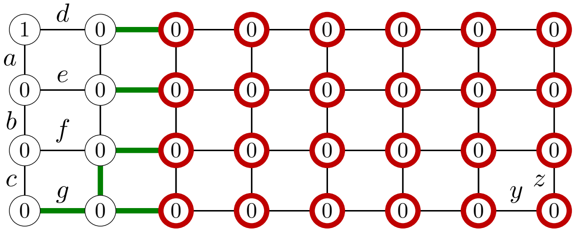
$$a+d=1$$
$$a+b+e=0$$
$$b+d+e=1$$
$$b+c+f=0$$
$$c+d+e+f=1$$
$$c+g=0$$
$$d+e+f+g=1$$
$$\vdots$$



Small-Size "Dynamic programming" Proof

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

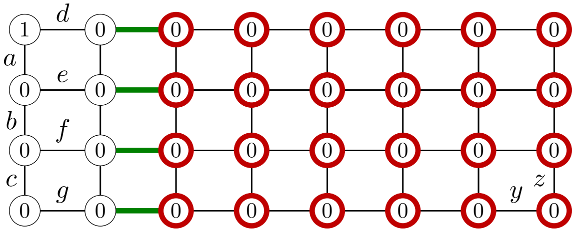
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



Small-Size "Dynamic programming" Proof

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

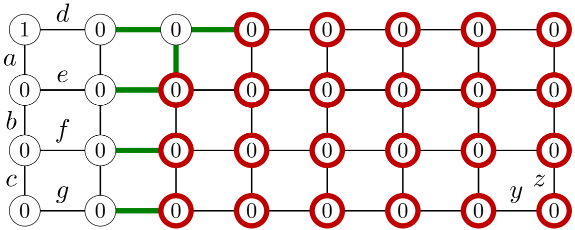
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

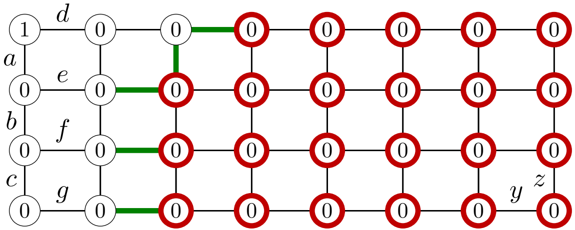
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

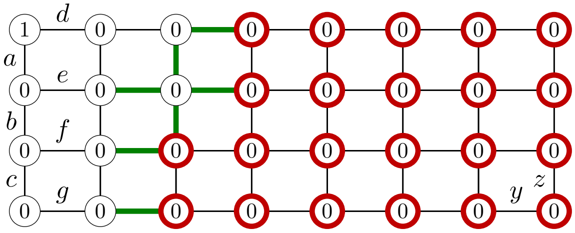
$$
\begin{aligned}
a+d&=1\\
a+b+e&=0\\
b+d+e&=1\\
b+c+f&=0\\
c+d+e+f&=1\\
c+g&=0\\
d+e+f+g&=1\\
&\vdots
\end{aligned}
$$



Small-Size “Dynamic programming” Proof

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

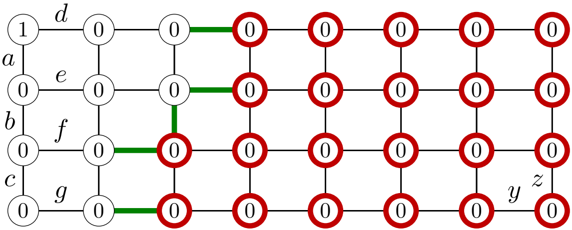
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

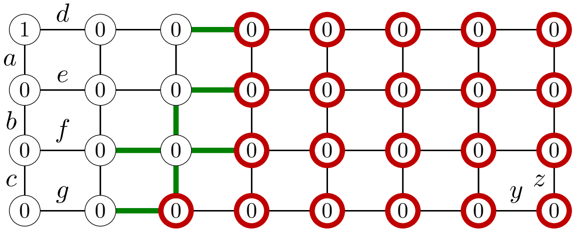
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

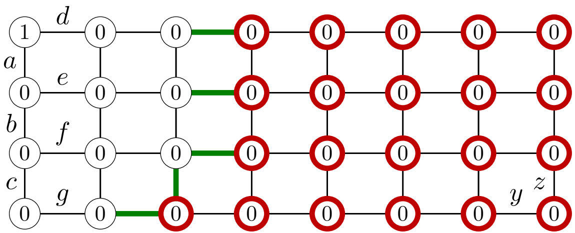
$$a+d=1$$
$$a+b+e=0$$
$$b+d+e=1$$
$$b+c+f=0$$
$$c+d+e+f=1$$
$$c+g=0$$
$$d+e+f+g=1$$
$$\vdots$$



Small-Size "Dynamic programming" Proof

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

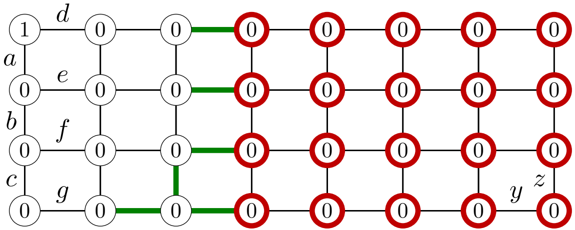
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

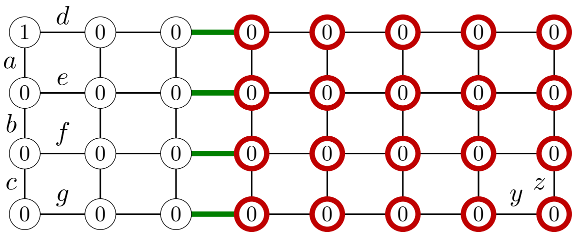
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

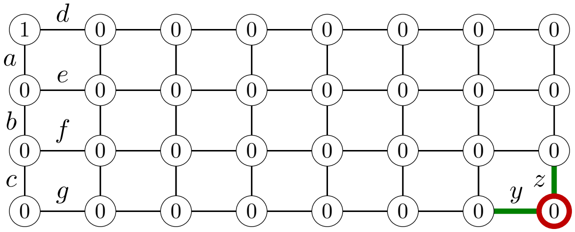
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$



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

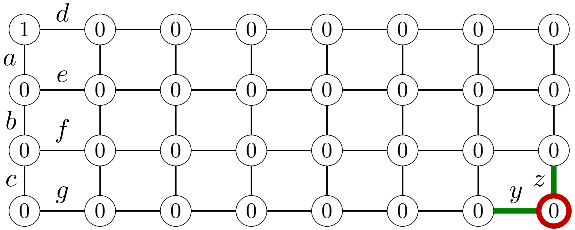
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$
$$y + z = 1$$



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

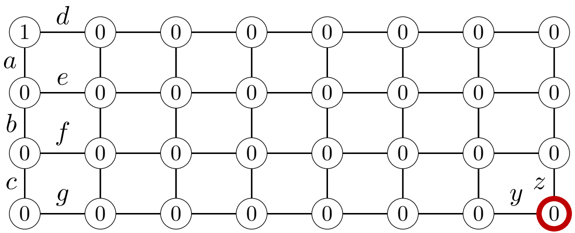
$$a + d = 1$$
$$a + b + e = 0$$
$$b + d + e = 1$$
$$b + c + f = 0$$
$$c + d + e + f = 1$$
$$c + g = 0$$
$$d + e + f + g = 1$$
$$\vdots$$
$$y + z = 1$$
$$y + z = 0$$



Small-Size "Dynamic programming" Proof

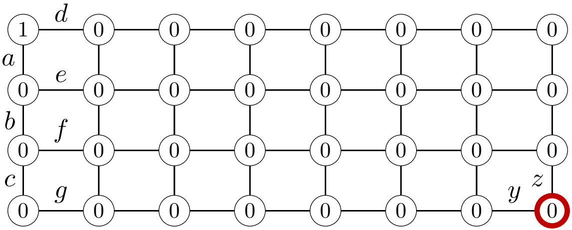
- View constraints as linear equations $\bmod 2$
- Sum constraints vertex by vertex
- Can be done in resolution by completeness
  But parity of $w+1$ variables $\Rightarrow 2^w$ clauses
- Total of mw summations

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



Small-Size "Dynamic programming" Proof

- View constraints as linear equations $\bmod 2$
- 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
- Small proof size $\mathcal{O}(mw2^w) = poly(m)$
  However, space $\approx$ size — superlinear!



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

Polynomial Calculus

- Translate CNF to polynomials and do algebraic manipulations (so-called Gröbner basis computations) [CEI96, ABRW02]
- Much stronger proof system, but most results we covered for resolution carry over (proofs are different and harder, though!)
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus
- 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

- Translate CNF to linear inequalities over the reals and prove no integral points in polytope [CCT87]
- Again much stronger proof system than resolution
- Very little known about size, space, or trade-offs
- Some work on pseudo-Boolean SAT solvers using (subset of) cutting planes
- Seems hard to make competitive with CDCL on CNFs — one key problem is to recover cardinality constraints
- But if cardinality constraints can be detected, solvers can do really well (at least on combinatorial benchmarks) [BBLM14]

Summing up

Overview of results on size-space trade-offs in proof complexity (more details in survey [Nor13])

- Resolution fairly well understood
- Most results in this talk carry over to polynomial calculus (algebraic reasoning)
- Cutting planes (geometric/pseudo-Boolean methods) very much less understood — several longstanding open questions

Summing up

Overview of results on size-space trade-offs in proof complexity (more details in survey [Nor13])

- Resolution fairly well understood
- Most results in this talk carry over to polynomial calculus (algebraic reasoning)
- Cutting planes (geometric/pseudo-Boolean methods) very much less understood — several longstanding open questions

Two open questions:

- Do these time-space trade-offs actually show up in SAT solving practice? (Under investigation. . .)
- How can we build efficient SAT solvers based on stronger proof systems than resolution?

Summing up

Overview of results on size-space trade-offs in proof complexity (more details in survey [Nor13])

- Resolution fairly well understood
- Most results in this talk carry over to polynomial calculus (algebraic reasoning)
- Cutting planes (geometric/pseudo-Boolean methods) very much less understood — several longstanding open questions

Two open questions:

- Do these time-space trade-offs actually show up in SAT solving practice? (Under investigation. . .)
- How can we 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 appeared in *STOC '00*.

[AD03] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. In *Proceedings of the 18th IEEE Annual Conference on Computational Complexity (CCC '03)*, pages 239–247, July 2003. Journal version in [AD08].

[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 appeared in *CCC '03*.

[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.

References II

[BBLM14] 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.

[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 appeared in *CCC '01*.

[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 III

[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.

[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 appeared in *STOC '99*.

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

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

[CS76] Stephen A. Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13(1):25–37, 1976. Preliminary version appeared in *STOC '74*.

References IV

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

[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*.

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

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