# Proof complexity and SAT solving

Jakob Nordström

University of Copenhagen and Lund University

SAT/SMT/AR Summer School
Nancy, France
June 29, 2024

# Three Simple Problems. . .

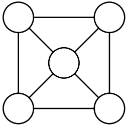
**Colouring**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

# Three Simple Problems. . .

**COLOURING**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

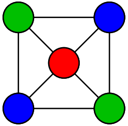


$3$-colouring?

# Three Simple Problems. . .

**COLOURING**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

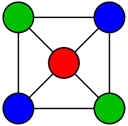


$3$-colouring? Yes

# Three Simple Problems. . .

**Colouring**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?



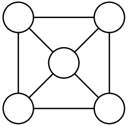
$3$-colouring? Yes, but no $2$-colouring

# Three Simple Problems. . .

**Clique**

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

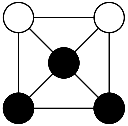
## Three Simple Problems...



3-clique?

### Clique

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

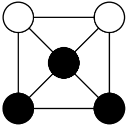


$3$-clique? Yes

**Clique**

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

# Three Simple Problems...



3-clique? Yes, but no 4-clique

### Clique

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

# Three Simple Problems. . .

### Colouring

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

### Clique

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

### Sat

Given propositional logic formula, is there a satisfying assignment?

# Three Simple Problems...

**COLOURING**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

**CLIQUE**

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

**SAT**

Given propositional logic formula, is there a satisfying assignment?

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

# Three Simple Problems...

**COLOURING**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

**CLIQUE**

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

**SAT**

Given propositional logic formula, is there a satisfying assignment?

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

- Variables should be set to **true** or **false**
- Constraint $(x \lor \neg y \lor z)$: means $x$ or $z$ should be true or $y$ false
- $\land$ means all constraints should hold simultaneously
- Is there a truth value assignment satisfying all constraints?

# Three Simple Problems...

**Colouring**

Does the graph $G = (V, E)$ have a colouring with $k$ colours such that all neighbours have distinct colours?

**Clique**

Is there a clique in the graph $G = (V, E)$ with $k$ vertices that are all pairwise connected by edges in $E$?

**Sat**

Given propositional logic formula, is there a satisfying assignment?

| | |
|---|---|
| Colouring: | frequency allocation for mobile base stations |
| Clique: | bioinformatics, computational chemistry |
| Sat: | easily models these and many other problems |

# . . . with Huge Practical Implications

- Some more examples of problems that can be encoded as propositional logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - operations research
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (100,000s or even 1,000,000s of variables)
- Can we use computers to solve these problems efficiently?

# Solving NP in Theory and Practice

- SAT mentioned already in Gödel's famous letter in 1956 to von Neumann
- Topic of intense research in computer science ever since 1960s

## Solving NP in Theory and Practice

- SAT mentioned already in Gödel's famous letter in 1956 to von Neumann
- Topic of intense research in computer science ever since 1960s
- NP-complete, so probably very hard [Coo71, Lev73]
- Assuming P $\neq$ NP, even impossible to meaningfully approximate
  - Colouring [Kho01, Zuc07]
  - Clique [Hås99]
  - SAT [Hås01]

## Solving NP in Theory and Practice

- SAT mentioned already in Gödel's famous letter in 1956 to von Neumann
- Topic of intense research in computer science ever since 1960s
- NP-complete, so probably very hard [Coo71, Lev73]
- Assuming P $\neq$ NP, even impossible to meaningfully approximate
  - COLOURING [Kho01, Zuc07]
  - CLIQUE [Hås99]
  - SAT [Hås01]
- Except that in practice, there are good algorithms for
  - COLOURING [DLMM08, DLMO09, DLMM11]
  - CLIQUE [Pro12, McC17]

  and amazing conflict-driven clause learning (CDCL) solvers [BS97, MS99, MMZ$^{+}$01] that solve huge SAT formulas

How can we understand real-world algorithms for NP-hard problems?

**This talk:** Use proof complexity (not only conceivable answer)

# Algorithmic View of Proof Complexity

For any algorithm solving NP problem, describe which rules of reasoning it uses

# Algorithmic View of Proof Complexity

For any algorithm solving NP problem, describe which rules of reasoning it uses

View this method of reasoning as formal proof system, with each single step efficiently verifiable

# Algorithmic View of Proof Complexity

For any algorithm solving NP problem, describe which rules of reasoning it uses

View this method of reasoning as formal proof system, with each single step efficiently verifiable

Efficiency of algorithm splits into two questions:

1. Is there a short proof using rules in this proof system?
2. Can short proofs in the proof system be found efficiently?

# Algorithmic View of Proof Complexity

For any algorithm solving NP problem, describe which rules of reasoning it uses

View this method of reasoning as formal proof system, with each single step efficiently verifiable

Efficiency of algorithm splits into two questions:

1. Is there a short proof using rules in this proof system?
2. Can short proofs in the proof system be found efficiently?

**Focus of this presentation:** Question 1 for different proof systems/algorithms
Study infeasible problems — proofs of feasibility are trivial

# Algorithmic View of Proof Complexity

For any algorithm solving NP problem, describe which rules of reasoning it uses

View this method of reasoning as formal proof system, with each single step efficiently verifiable

Efficiency of algorithm splits into two questions:

1. Is there a short proof using rules in this proof system?
2. Can short proofs in the proof system be found efficiently?

**Focus of this presentation:** Question 1 for different proof systems/algorithms
Study infeasible problems — proofs of feasibility are trivial

Question 2: Topic for separate lecture(s) — lots of recent exciting progress; mostly negative (worst-case) results that proof search is hard, e.g., [AM20, GKMP20, dRGN$^+$21]

# Applications of Proof Complexity

Three applied reasons for proof complexity:

1. Understand real-world applied algorithmic paradigms [**this lecture** ]

2. Get ideas for algorithmic improvements [EN18, EN20, DGD+21, DGN21, KBBN22]
   (See, e.g., tutorials https://www.youtube.com/watch?v=LZ8VztiplaQ and https://www.youtube.com/watch?v=wD_2tx1rTaw about RoundingSat)

3. Enhance algorithms to write machine-verifiable certificates of correctness [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BGMN23, BBN+23, MM23, GMM+24, HOGN24, BBN+24, DMM+24, IOT+24, MMN24]
   (See tutorial https://www.youtube.com/watch?v=s_5BIi4I22w about VeriPB)

# Outline

1. DPLL, CDCL, and Resolution
   - Davis-Putnam-Logemann-Loveland (DPLL) Method
   - Conflict-Driven Clause Learning (CDCL)
   - Resolution Proof System
2. Algebraic and Semi-algebraic Approaches
   - Nullstellensatz
   - Gröbner Bases and Polynomial Calculus
   - Pseudo-Boolean Solving and Cutting Planes
3. Some More Advanced Proof Systems We Might Not Have Time for
   - Sherali-Adams and Sums of Squares
   - Stabbing Planes
   - Extended Resolution

# Formal Description of SAT Problem

- Variable $x$: takes value **true** $(= 1)$ or **false** $(= 0)$
- Literal $\ell$: variable $x$ or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

**The SATISFIABILITY (or just SAT) Problem**

Given a CNF formula $F$, is it satisfiable?

# Formal Description of SAT Problem

- Variable $x$: takes value **true** $(= 1)$ or **false** $(= 0)$
- Literal $\ell$: variable $x$ or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

**The SATISFIABILITY (or just SAT) Problem**

Given a CNF formula $F$, is it satisfiable?

Here is our example formula again:

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

# The Same Problem in Three Different Shapes

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

# The Same Problem in Three Different Shapes

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$\begin{aligned}
(1-x)(1-z) &= 0 \\
(1-y)z &= 0 \\
(1-x)y(1-u) &= 0 \\
yu &= 0 \\
(1-u)(1-v) &= 0 \\
xv &= 0 \\
u(1-w) &= 0 \\
xuw &= 0
\end{aligned}$$

For **true** $= 1$ and **false** $= 0$, is there a $\{0,1\}$-valued solution?

# The Same Problem in Three Different Shapes

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$\begin{aligned}
1 - x - z + xz &= 0 \\
z - yz &= 0 \\
y - xy - yu + xyu &= 0 \\
yu &= 0 \\
1 - u - v + uv &= 0 \\
xv &= 0 \\
u - uw &= 0 \\
xuw &= 0
\end{aligned}$$

For **true** $= 1$ and **false** $= 0$, is there a $\{0, 1\}$-valued solution?

# The Same Problem in Three Different Shapes

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

| | |
|---|---|
| $1 - x - z + xz = 0$ | $x + z \geq 1$ |
| $z - yz = 0$ | $y + (1 - z) \geq 1$ |
| $y - xy - yu + xyu = 0$ | $x + (1 - y) + u \geq 1$ |
| $yu = 0$ | $(1 - y) + (1 - u) \geq 1$ |
| $1 - u - v + uv = 0$ | $u + v \geq 1$ |
| $xv = 0$ | $(1 - x) + (1 - v) \geq 1$ |
| $u - uw = 0$ | $(1 - u) + w \geq 1$ |
| $xuw = 0$ | $(1 - x) + (1 - u) + (1 - w) \geq 1$ |

For **true** $= 1$ and **false** $= 0$, is there a $\{0, 1\}$-valued solution?

# The Same Problem in Three Different Shapes

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$\begin{aligned}
1 - x - z + xz &= 0 \\
z - yz &= 0 \\
y - xy - yu + xyu &= 0 \\
yu &= 0 \\
1 - u - v + uv &= 0 \\
xv &= 0 \\
u - uw &= 0 \\
xuw &= 0
\end{aligned}$$

$$\begin{aligned}
x + z &\geq 1 \\
y - z &\geq 0 \\
x - y + u &\geq 0 \\
-y - u &\geq -1 \\
u + v &\geq 1 \\
-x - v &\geq -1 \\
-u + w &\geq 0 \\
-x - u - w &\geq -2
\end{aligned}$$

For **true** $= 1$ and **false** $= 0$, is there a $\{0,1\}$-valued solution?

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

## DPLL (somewhat simplified description)

1. If $F$ contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

## DPLL (somewhat simplified description)

1. If $F$ contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
2. If $F$ contains no clauses, report "satisfiable" and terminate

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

## DPLL (somewhat simplified description)

1. If $F$ contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
2. If $F$ contains no clauses, report "satisfiable" and terminate
3. Otherwise pick some variable $x$ in $F$

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

## DPLL (somewhat simplified description)

1. If $F$ contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
2. If $F$ contains no clauses, report "satisfiable" and terminate
3. Otherwise pick some variable $x$ in $F$
4. Set $x = 0$, simplify $F$ and make recursive call

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

## DPLL (somewhat simplified description)

1. If $F$ contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
2. If $F$ contains no clauses, report "satisfiable" and terminate
3. Otherwise pick some variable $x$ in $F$
4. Set $x = 0$, simplify $F$ and make recursive call
5. Set $x = 1$, simplify $F$ and make recursive call

# DPLL: Attempting Smart Case Analysis

The foundation of state-of-the-art SAT solvers is the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

## DPLL (somewhat simplified description)

1. If $F$ contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
2. If $F$ contains no clauses, report "satisfiable" and terminate
3. Otherwise pick some variable $x$ in $F$
4. Set $x = 0$, simplify $F$ and make recursive call
5. Set $x = 1$, simplify $F$ and make recursive call
6. If result in both cases "unsatisfiable", then report "unsatisfiable" and return

# A DPLL Toy Example

$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

# A DPLL Toy Example

$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

# A DPLL Toy Example

$$
\begin{aligned}
F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\
& \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

$x$

# A DPLL Toy Example

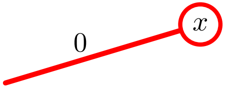
$$F = (\quad z) \wedge (y \vee \overline{z}) \wedge (\quad \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge\ (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

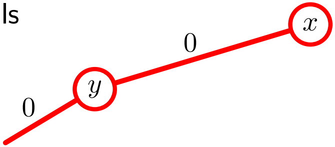
$$F = (\quad z) \wedge (\quad \overline{z}) \wedge (\quad \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge\, (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

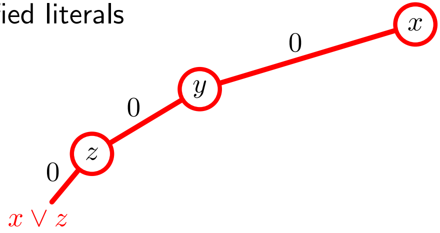
$$F = (x \lor z) \land (\quad \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

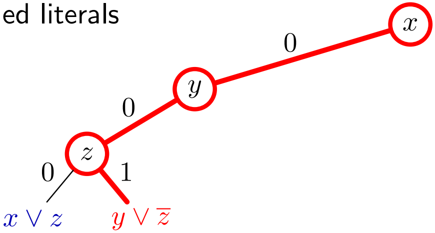
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

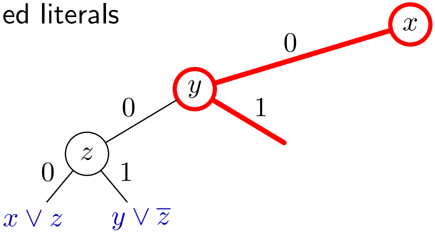
$$F = (\qquad z) \land (y \lor \overline{z}) \land (\qquad\qquad u) \land (\qquad \overline{u})$$
$$\land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

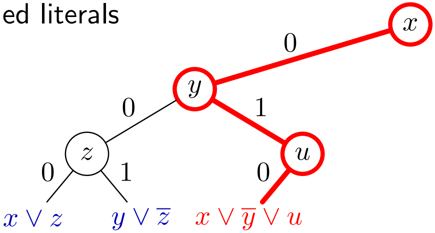
$$F = (\quad z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\quad \overline{u}) \wedge (\quad v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

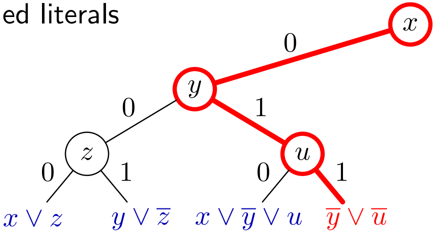
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad u) \land (\overline{y} \lor \overline{u}) \\ \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\quad w) \land (\overline{x} \lor \quad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

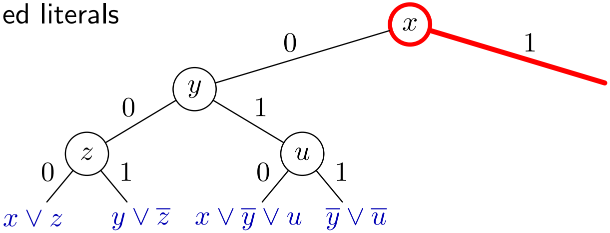
$$
\begin{aligned}
F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\
& \land (u \lor v) \land (\quad \overline{v}) \land (\overline{u} \lor w) \land (\quad \overline{u} \lor \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

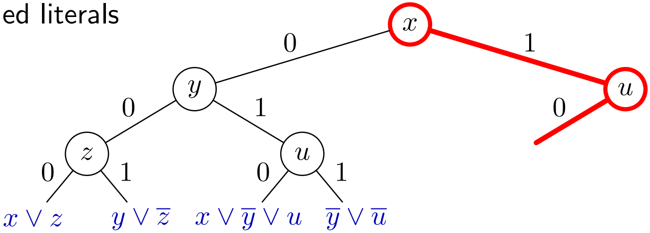
$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \qquad ) \land (\overline{y} \lor \overline{u}) \\ & \land ( \qquad v) \land ( \qquad \overline{v}) \land (\overline{u} \lor w) \land ( \qquad \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

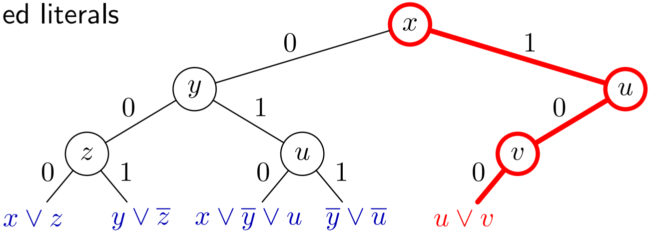
$$F = \quad (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \qquad) \land (\overline{y} \lor \overline{u})$$
$$\land\ (u \lor v) \land (\qquad \overline{v}) \land (\overline{u} \lor w) \land (\qquad \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

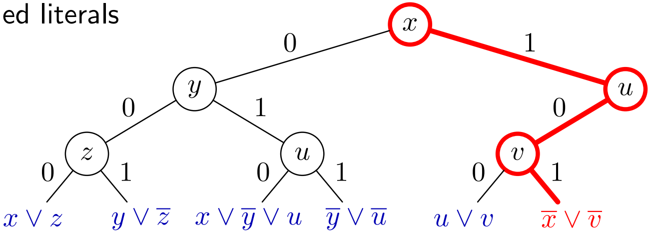
$$
\begin{aligned}
F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \quad ) \land (\overline{y} \lor \overline{u}) \\
\land\ & ( \quad v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land ( \quad \overline{u} \lor \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

“Simplify formula” by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

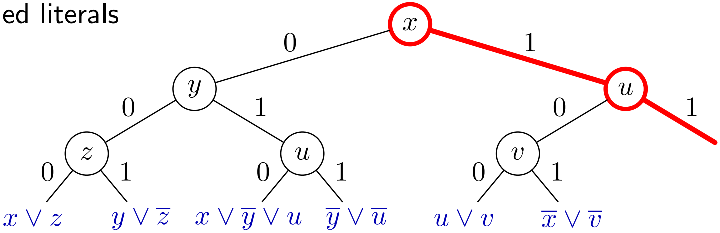
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{v}) \land (w) \land (\overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

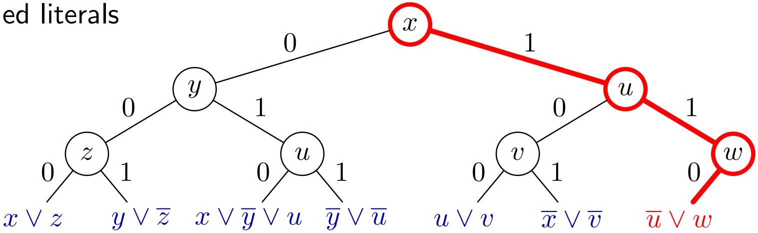
$$F = (x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u}) \wedge (u \vee v) \wedge (\overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

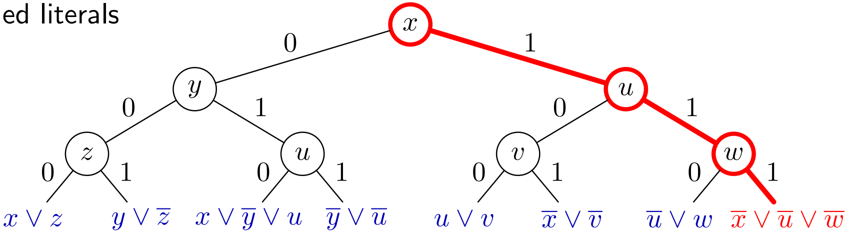
$$\begin{aligned} F = \; & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{v}) \land (w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# A DPLL Toy Example

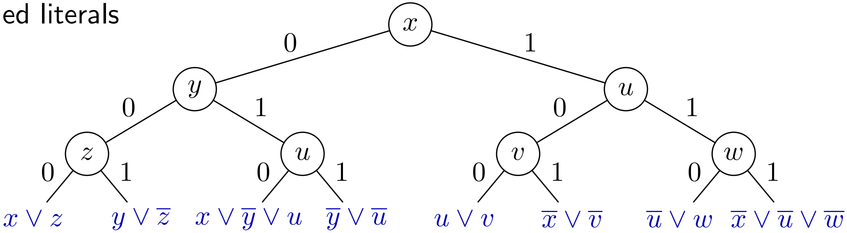
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



# State-of-the-Art SAT Solving in One Slide

High-level description of modern conflict-driven clause learning (CDCL) SAT solving (as pioneered in [BS97, MS99, MMZ$^+$01]):

- Try to build satisfying assignment for formula (branching or decision heuristic crucial)
- When partial assignment violates formula, compute explanation for conflict and add to formula as new clause (clause learning)
- Every once in a while, restart from beginning (but save computed info)

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \vee \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \vee \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \vee r}{=} 1$

$w \stackrel{\overline{r} \vee w}{=} 1$

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$p \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{p \vee \overline{u}}{=} 0$

$q \stackrel{\mathsf{d}}{=} 0$

$r \stackrel{q \vee r}{=} 1$

$w \stackrel{\overline{r} \vee w}{=} 1$

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \vee x \vee y}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \vee \overline{u}}{=} 0$

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \vee r}{=} 1$

$w \stackrel{\overline{r} \vee w}{=} 1$

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$

$y \stackrel{u \vee x \vee y}{=} 1$

$z \stackrel{x \vee \overline{y} \vee z}{=} 1$

**Decision**

Free choice to assign value to variable

Notation $p \stackrel{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$

$u \overset{p \vee \overline{u}}{=} 0$

$\boxed{q \overset{\mathsf{d}}{=} 0}$

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

$\boxed{x \overset{\mathsf{d}}{=} 0}$

$y \overset{u \vee x \vee y}{=} 1$

$z \overset{x \vee \overline{y} \vee z}{=} 1$

$\overset{\overline{y} \vee \overline{z}}{\perp}$

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict-Driven Clause Learning (CDCL) by Example

Two kinds of assignments — illustrate on example formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$
$u \overset{p \lor \overline{u}}{=} 0$
decision level 1

$\boxed{q \overset{\mathsf{d}}{=} 0}$
$r \overset{q \lor r}{=} 1$
$w \overset{\overline{r} \lor w}{=} 1$
decision level 2

$\boxed{x \overset{\mathsf{d}}{=} 0}$
$y \overset{u \lor x \lor y}{=} 1$
$z \overset{x \lor \overline{y} \lor z}{=} 1$
$\overset{\overline{y} \lor \overline{z}}{\bot}$
decision level 3

**Decision**

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

**Unit propagation**

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \stackrel{\mathsf{d}}{=} 0$

$u \stackrel{p \lor \overline{u}}{=} 0$

decision level 1

$q \stackrel{\mathsf{d}}{=} 0$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

decision level 2

$x \stackrel{\mathsf{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

$\stackrel{\overline{y} \lor \overline{z}}{\bot}$

decision level 3

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \overset{\mathsf{d}}{=} 0}$

$u \overset{p \lor \overline{u}}{=} 0$

decision level 1

$\boxed{q \overset{\mathsf{d}}{=} 0}$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

decision level 2

$\boxed{x \overset{\mathsf{d}}{=} 0}$

$y \overset{u \lor x \lor y}{=} 1$

$z \overset{x \lor \overline{y} \lor z}{=} 1$

$\overset{\overline{y} \lor \overline{z}}{\bot}$

decision level 3

Could backtrack by erasing conflict level & flipping last decision

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$

$u \stackrel{p \lor \overline{u}}{=} 0$

decision level 1

$\boxed{q \stackrel{\mathsf{d}}{=} 0}$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

decision level 2

$\boxed{x \stackrel{\mathsf{d}}{=} 0}$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

$\stackrel{\overline{y} \lor \overline{z}}{\bot}$

decision level 3

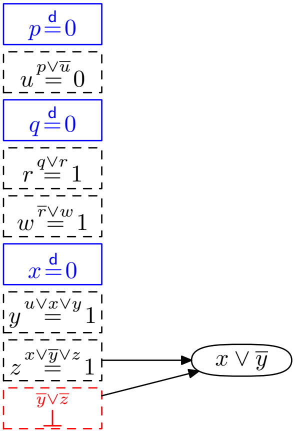
Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$




Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

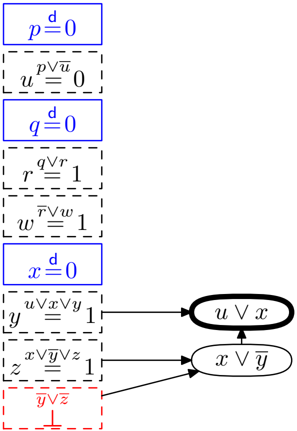
Case analysis over $z$ for last two clauses:

- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge clauses & remove $z$ — must satisfy $x \lor \overline{y}$

# Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$




Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Case analysis over $z$ for last two clauses:

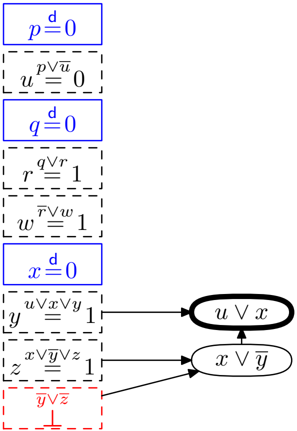
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge clauses & remove $z$ — must satisfy $x \lor \overline{y}$

Repeat until UIP clause with only $1$ variable at conflict level after last decision — learn and backjump

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

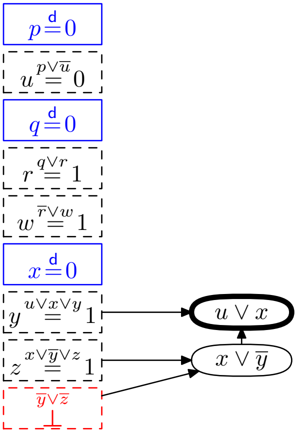
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$





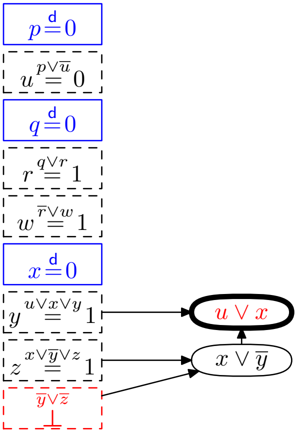


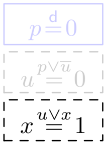
Assertion level $1$ (2nd largest level in learned clause) — trim trail to that level

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



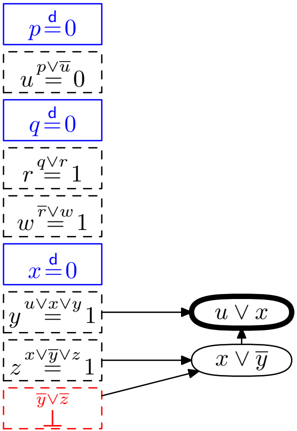

Assertion level $1$ (2nd largest level in learned clause) — trim trail to that level

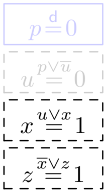
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$




Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

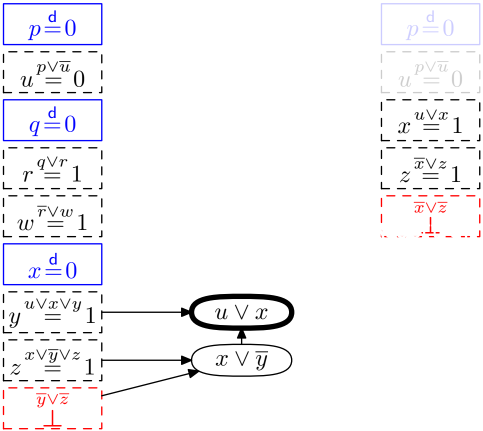
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before...

# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

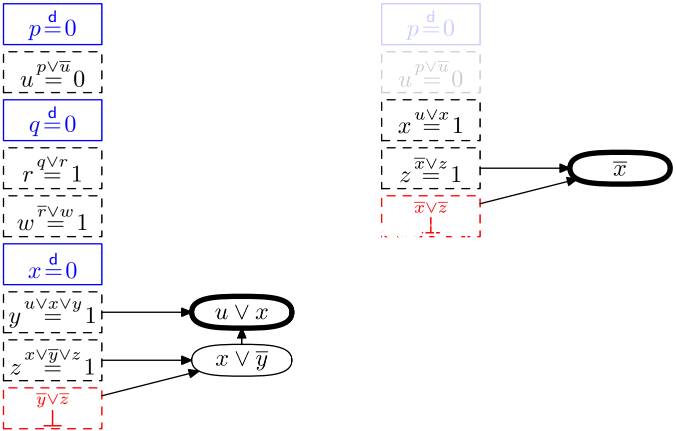
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

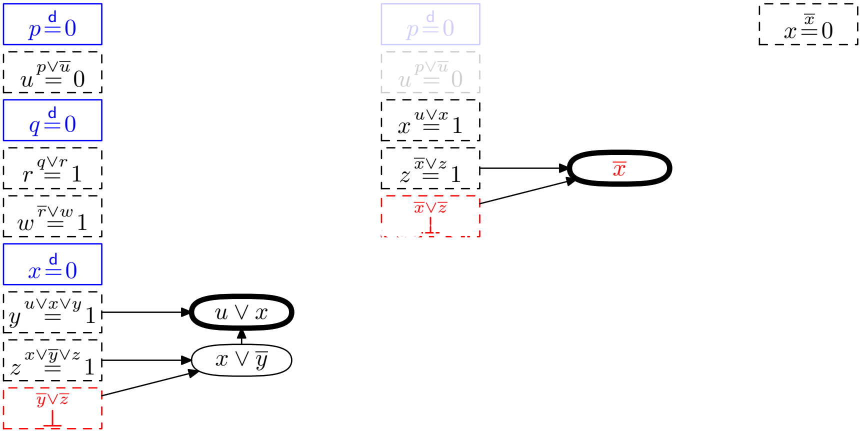
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

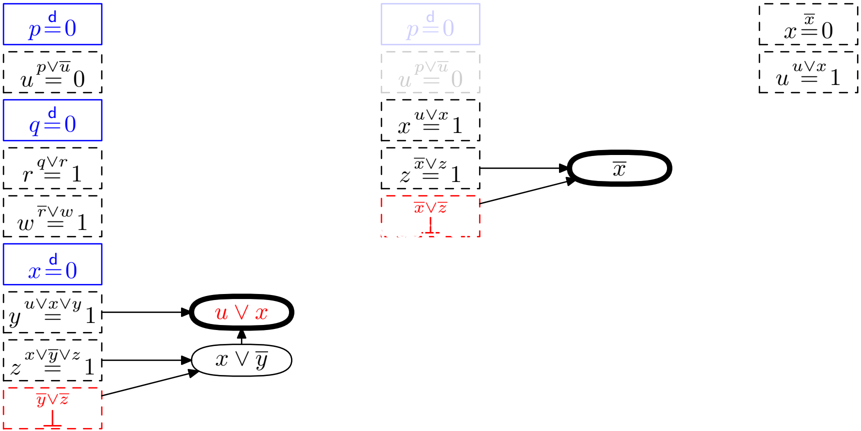
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

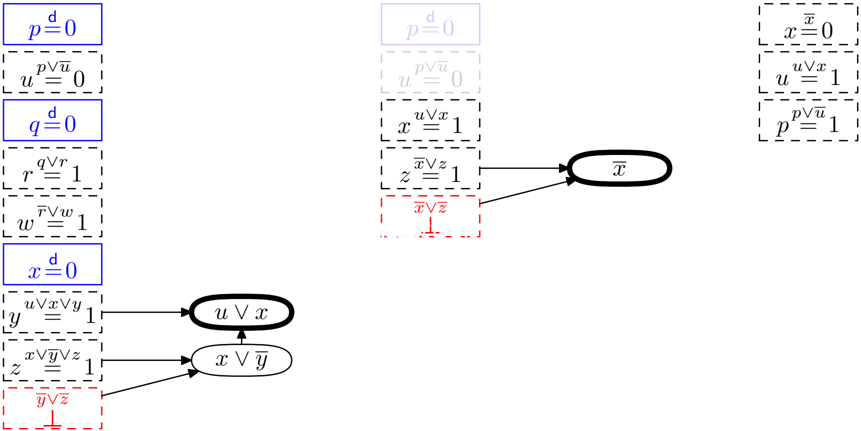
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

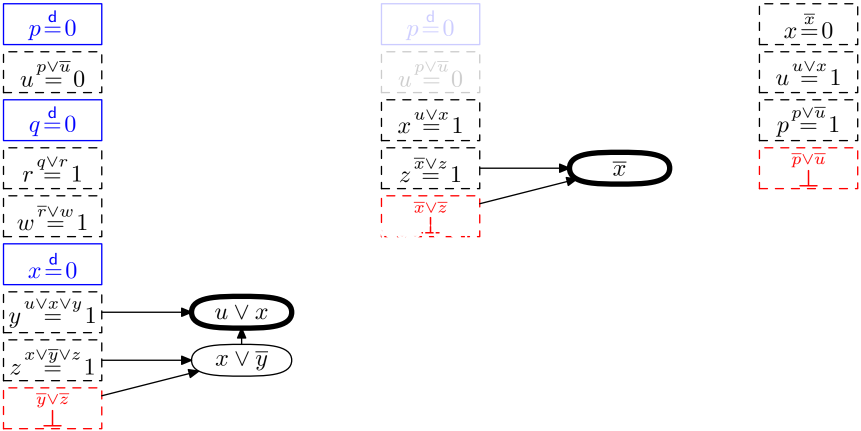
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

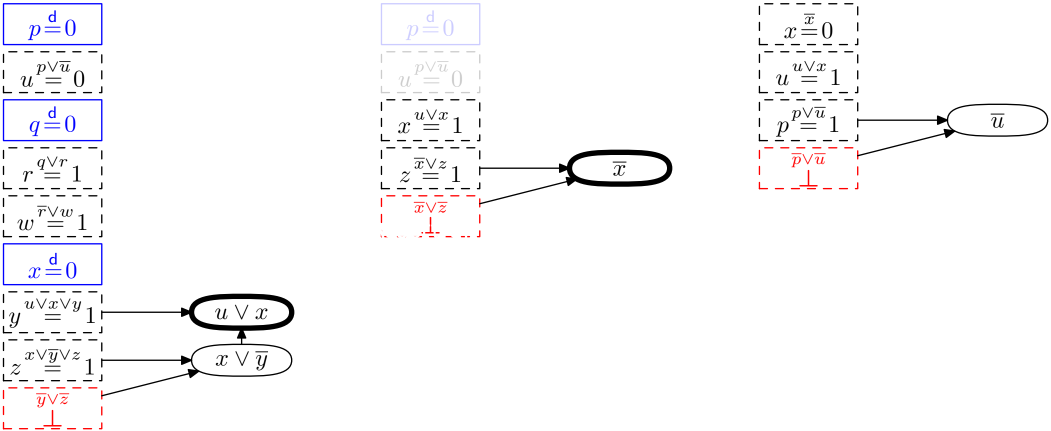
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

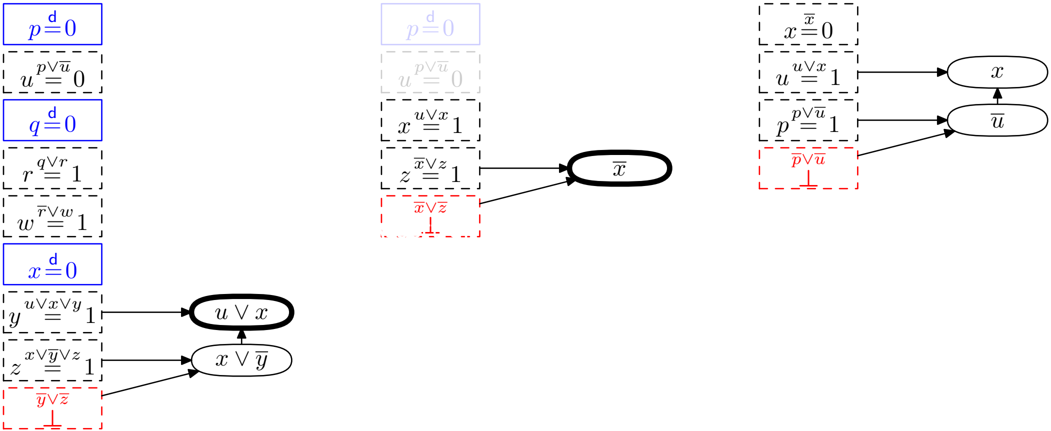
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

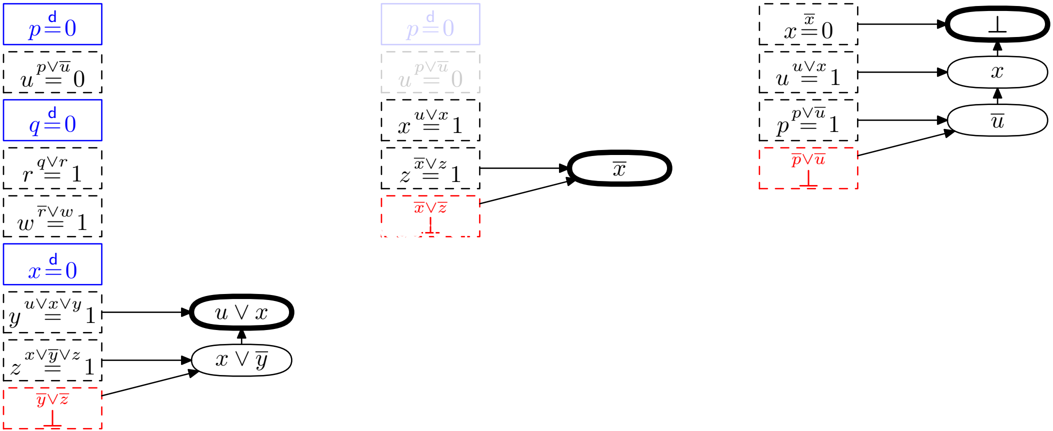
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



# Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



# SAT Solver Analysis and the Resolution Proof System

How to make rigorous analysis of SAT solver performance?

Many intricate, hard-to-understand heuristics

So focus instead on underlying method of reasoning

# SAT Solver Analysis and the Resolution Proof System

How to make rigorous analysis of SAT solver performance?
Many intricate, hard-to-understand heuristics
So focus instead on underlying method of reasoning

**Resolution proof system [Bla37, Rob65]**

- Start with clauses of CNF formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

# Resolution Proofs by Contradction

Resolution rule:

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

**Observation**

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

# Resolution Proofs by Contradction

Resolution rule:

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

**Observation**

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

So can prove $F$ unsatisfiable by deriving the unsatisfiable empty clause (denoted $\perp$) from $F$ by resolution

Such proof by contradiction also called resolution refutation

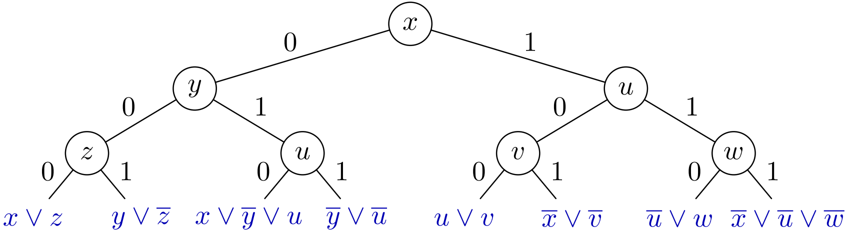
# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

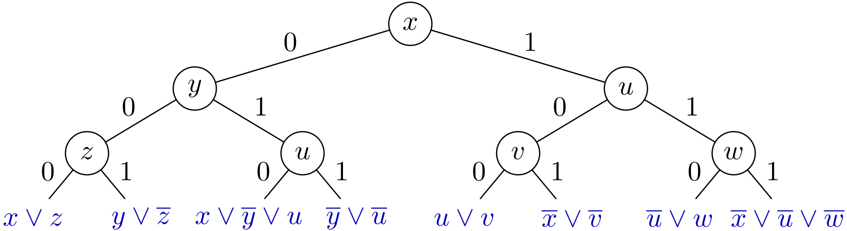
Look at our example again



# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

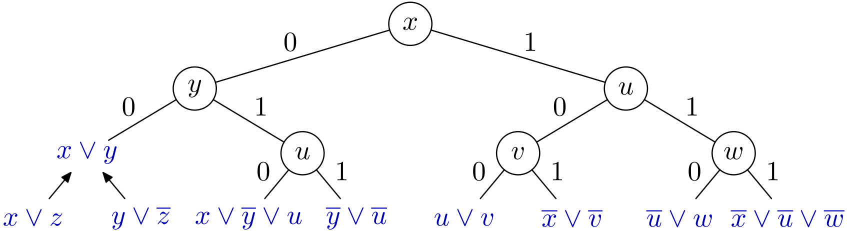



and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

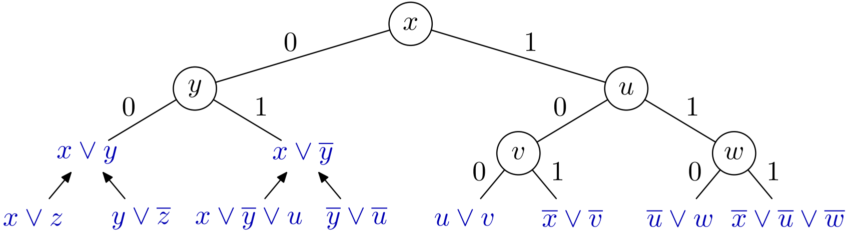



and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

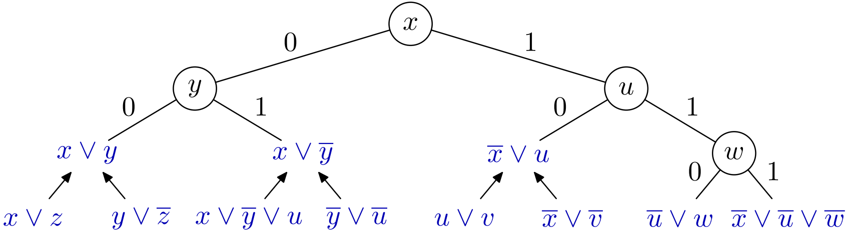



and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

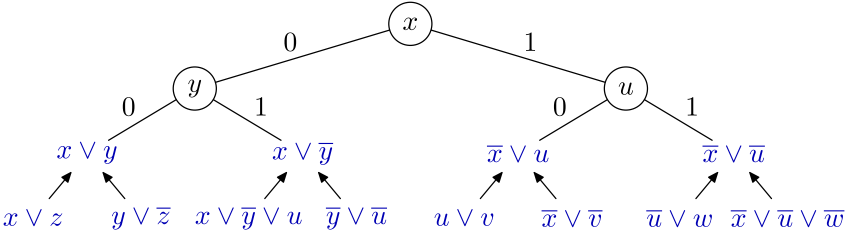



and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

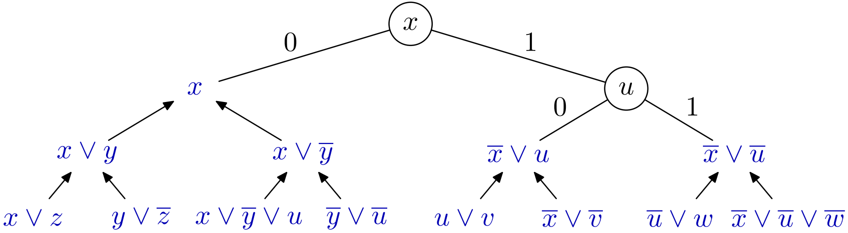



and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

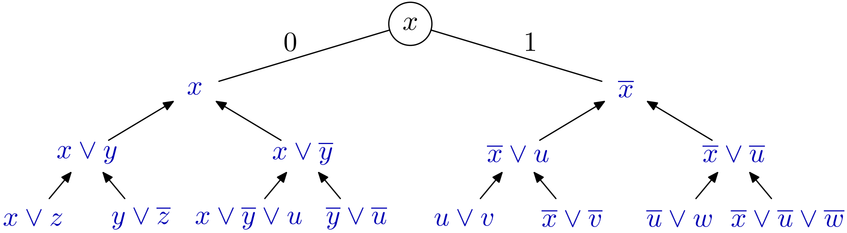



and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

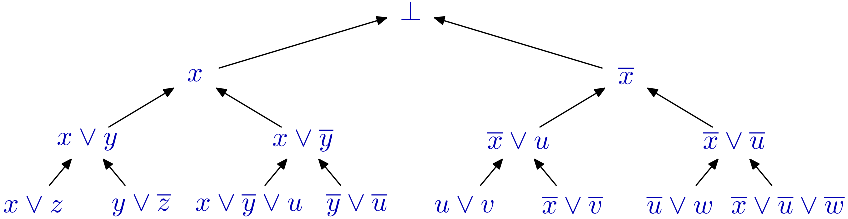



and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

# DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again




and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

# DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution

# DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show

# DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once
  (to use a clause twice, we have to derive it twice from scratch)

# DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution $\Rightarrow$ lower bounds on DPLL running time

# DPLL Running Time and Tree-Like Resolution Proof Size

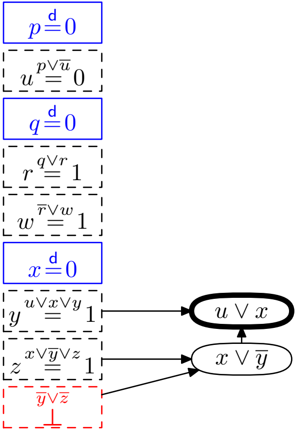
- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution $\Rightarrow$ lower bounds on DPLL running time
- Conflict-driven clause learning adds "shortcut edges" in tree, but still yields resolution proof

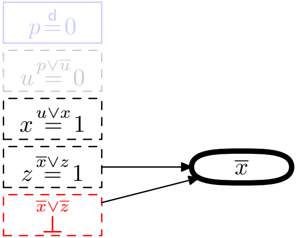
# CDCL and Resolution Proofs

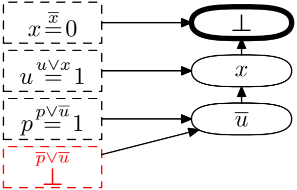
Obtain resolution proof...

# CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution. . .

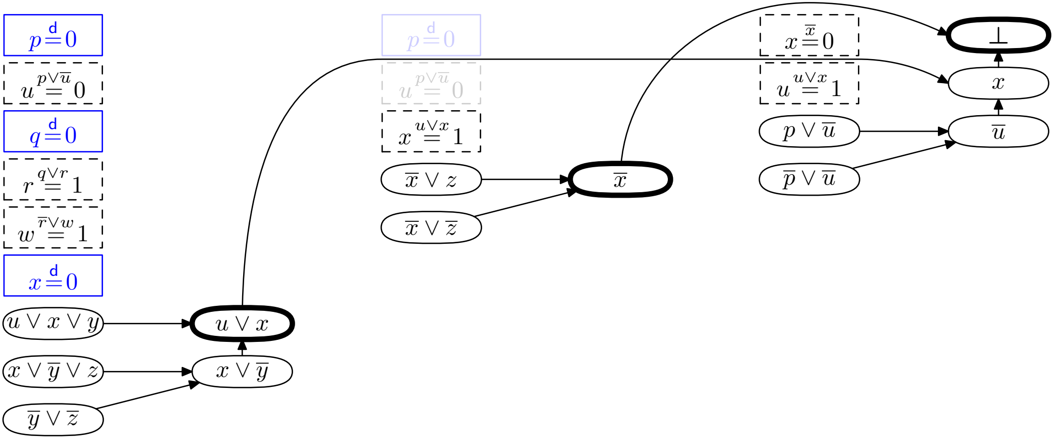


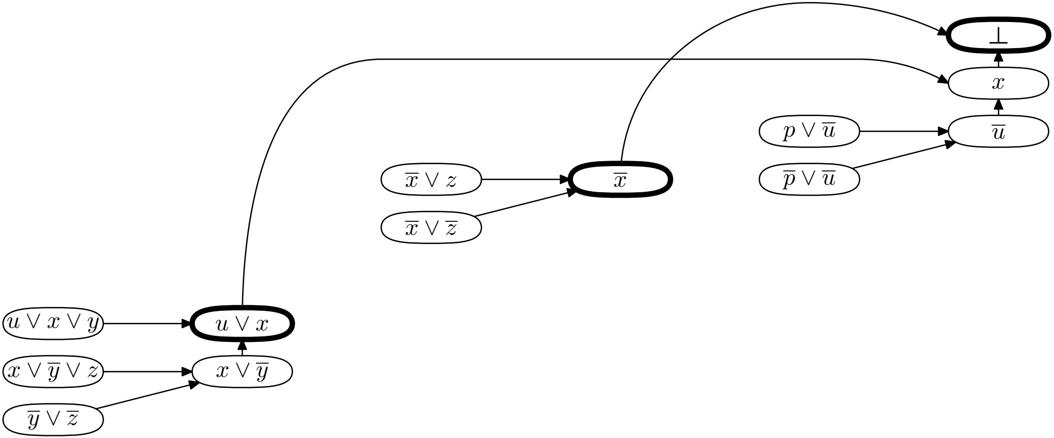
# CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



# CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



# CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from CDCL execution

# CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from CDCL execution
- Requires an argument, of course, but you have seen enough in this presentation to be able to fill in the required details. . .

# CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from CDCL execution
- Requires an argument, of course, but you have seen enough in this presentation to be able to fill in the required details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*

# CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from CDCL execution
- Requires an argument, of course, but you have seen enough in this presentation to be able to fill in the required details. . .
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on CDCL running time

# CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from CDCL execution
- Requires an argument, of course, but you have seen enough in this presentation to be able to fill in the required details. . .
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on CDCL running time
- Lower (and upper) bounds for different methods of reasoning about propositional logic formulas studied in proof complexity

# CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from CDCL execution
- Requires an argument, of course, but you have seen enough in this presentation to be able to fill in the required details. . .
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on CDCL running time
- Lower (and upper) bounds for different methods of reasoning about propositional logic formulas studied in proof complexity

(*) Except for some preprocessing techniques, which is an important omission, but this gets complicated and we don't have time to go into details. . .

# Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")

# Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?

# Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?
- Paradox: resolution quite weak proof system; many strong proof complexity lower bounds for (seemingly) "obvious" formulas

# Examples of Hard Formulas For Resolution (1/3)

**Pigeonhole principle (PHP) formulas** [Hak85]

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

# Examples of Hard Formulas For Resolution (1/3)

**Pigeonhole principle (PHP) formulas** [Hak85]

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

Variables $p_{i,j}$ = "pigeon $i \rightarrow$ hole $j$"; $1 \leq i \leq n+1$; $1 \leq j \leq n$

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

Can also add "functionality" and "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 |

# Examples of Hard Formulas For Resolution (1/3)

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

Variables $p_{i,j}$ = "pigeon $i \rightarrow$ hole $j$"; $1 \leq i \leq n+1$; $1 \leq j \leq n$

| | |
|---|---|
| $p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$ | 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 |

Even onto functional PHP hard — **"resolution cannot count"**

Resolution proof requires $\exp(\Omega(n)) = \exp\big(\Omega(\sqrt[3]{N})\big)$ clauses
(measured in terms of formula size $N$)

# Examples of Hard Formulas For Resolution (2/3)

**Tseitin formulas** [Urq87]

"Sum of degrees of vertices in graph is even"

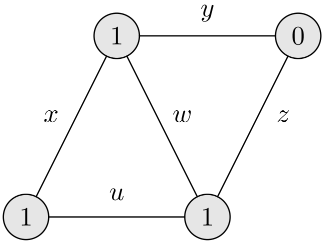
# Examples of Hard Formulas For Resolution (2/3)

**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



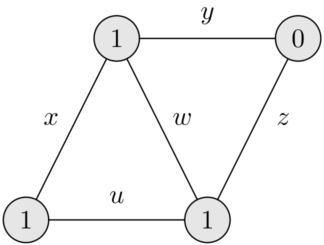
# Examples of Hard Formulas For Resolution (2/3)

**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




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

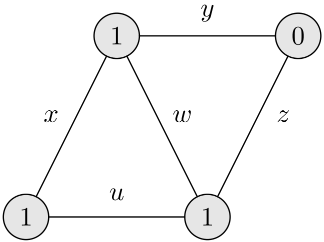
# Examples of Hard Formulas For Resolution (2/3)

**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




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

Requires proof size $\exp(\Omega(N))$ on well-connected so-called expander graphs —
**"resolution cannot count** $\mathrm{mod}\ 2$**"**

# Examples of Hard Formulas for Resolution (3/3)

**Random $k$-CNF formulas** [CS88]

$\Delta n$ 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 for Resolution (3/3)

**Random $k$-CNF formulas** [CS88]

$\Delta n$ 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...**

- Colouring [BCMM05]
- Zero-one designs [Spe10, VS10, MN14]
- Et cetera... (See, e.g., [BN21] for overview)

# Examples of Hard Formulas for Resolution (3/3)

**Random $k$-CNF formulas** [CS88]

$\Delta n$ 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...**

- COLOURING [BCMM05]
- Zero-one designs [Spe10, VS10, MN14]
- Et cetera... (See, e.g., [BN21] for overview)

But no such strong lower bounds known for CLIQUE!

- Refuting existence of $k$-clique should require proof size $n^{\Omega(k)}$
- Only known for restricted so-called regular resolution [ABdR$^+$21]

# SAT as System of Polynomial Equations

- Given CNF formula $F = \bigwedge_{i=1}^{m} C_i$

# SAT as System of Polynomial Equations

- Given CNF formula $F = \bigwedge_{i=1}^{m} C_i$
- Translate clauses

$$C = \bigvee_{i \in \mathcal{P}} x_i \vee \bigvee_{j \in \mathcal{N}} \overline{x}_j$$

to polynomial equations

$$\prod_{i \in \mathcal{P}} (1 - x_i) \cdot \prod_{j \in \mathcal{N}} x_j = 0$$

# SAT as System of Polynomial Equations

- Given CNF formula $F = \bigwedge_{i=1}^{m} C_i$
- Translate clauses

$$C = \bigvee_{i \in \mathcal{P}} x_i \vee \bigvee_{j \in \mathcal{N}} \overline{x}_j$$

  to polynomial equations

$$\prod_{i \in \mathcal{P}} (1 - x_i) \cdot \prod_{j \in \mathcal{N}} x_j = 0$$

- Add Boolean axioms

$$x_j^2 - x_j = 0$$

  for all variables

# Hilbert's Nullstellensatz

Consider any system of polynomial equations

$$\begin{aligned} p_1(x_1, \ldots, x_n) &= 0 \qquad\qquad & x_1^2 - x_1 &= 0 \\ p_2(x_1, \ldots, x_n) &= 0 & x_2^2 - x_2 &= 0 \\ &\vdots & &\vdots \\ p_m(x_1, \ldots, x_n) &= 0 & x_n^2 - x_n &= 0 \end{aligned}$$

in polynomial ring over field $\mathbb{F}$

# Hilbert's Nullstellensatz

Consider any system of polynomial equations

$$\begin{aligned} p_1(x_1, \ldots, x_n) &= 0 \\ p_2(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ p_m(x_1, \ldots, x_n) &= 0 \end{aligned} \qquad\qquad \begin{aligned} x_1^2 - x_1 &= 0 \\ x_2^2 - x_2 &= 0 \\ &\vdots \\ x_n^2 - x_n &= 0 \end{aligned}$$

in polynomial ring over field $\mathbb{F}$

## Hilbert's Nullstellensatz

System infeasible $\Leftrightarrow$ exist $q_i, r_j \in \mathbb{F}[x_1, \ldots, x_n]$ such that

$$\sum_{i=1}^{m} q_i(x_1, \ldots, x_n) \cdot p_i(x_1, \ldots, x_n) + \sum_{j=1}^{n} r_j(x_1, \ldots, x_n) \cdot (x_j^2 - x_j) = 1$$

# Nullstellensatz Proof System [BIK+94]

Nullstellensatz refutation of

$$p_i(x_1, \ldots, x_n) = 0 \qquad i \in [m]$$
$$x_j^2 - x_j = 0 \qquad j \in [n]$$

is (syntactic) equality

$$\sum_{i=1}^{m} q_i(x_1, \ldots, x_n) \cdot p_i(x_1, \ldots, x_n) + \sum_{j=1}^{n} r_j(x_1, \ldots, x_n) \cdot (x_j^2 - x_j) = 1$$

# Nullstellensatz Proof System [BIK+94]

Nullstellensatz refutation of

$$\begin{aligned} p_i(x_1, \ldots, x_n) &= 0 & \qquad i &\in [m] \\ x_j^2 - x_j &= 0 & \qquad j &\in [n] \end{aligned}$$

is (syntactic) equality

$$\sum_{i=1}^{m} q_i(x_1, \ldots, x_n) \cdot p_i(x_1, \ldots, x_n) + \sum_{j=1}^{n} r_j(x_1, \ldots, x_n) \cdot (x_j^2 - x_j) = 1$$

Complexity measures of refutations:

- Size: number of monomials (when all polynomials expanded out)
- Degree: highest total degree of any polynomial

# Nullstellensatz Example (Not Expanded out)

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

# Nullstellensatz Example (Not Expanded out)

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$(1-x)(1-z)$$
$$(1-y)z$$
$$(1-x)y(1-u)$$
$$yu$$
$$(1-u)(1-v)$$
$$xv$$
$$u(1-w)$$
$$xuw$$

# Nullstellensatz Example (Not Expanded out)

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$\begin{aligned}
& (1-y) \cdot (1-x)(1-z) \\
+\ & (1-x) \cdot (1-y)z \\
+\ & 1 \cdot (1-x)y(1-u) \\
+\ & (1-x) \cdot yu \\
+\ & x \cdot (1-u)(1-v) \\
+\ & (1-u) \cdot xv \\
+\ & x \cdot u(1-w) \\
+\ & 1 \cdot xuw
\end{aligned}$$

# Nullstellensatz Example (Not Expanded out)

$$(x \lor z) \land (y \lor \neg z) \land (x \lor \neg y \lor u) \land (\neg y \lor \neg u)$$
$$\land\ (u \lor v) \land (\neg x \lor \neg v) \land (\neg u \lor w) \land (\neg x \lor \neg u \lor \neg w)$$

$$
\begin{aligned}
& (1-y) \cdot (1-x)(1-z) \\
+\ & (1-x) \cdot (1-y)z \\
+\ & 1 \cdot (1-x)y(1-u) \\
+\ & (1-x) \cdot yu \\
+\ & x \cdot (1-u)(1-v) \\
+\ & (1-u) \cdot xv \\
+\ & x \cdot u(1-w) \\
+\ & 1 \cdot xuw \\
=\ & 1
\end{aligned}
$$

# Nullstellensatz Example (Not Expanded out)

$$(x \vee z) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee u) \wedge (\neg y \vee \neg u)$$
$$\wedge\ (u \vee v) \wedge (\neg x \vee \neg v) \wedge (\neg u \vee w) \wedge (\neg x \vee \neg u \vee \neg w)$$

$$\begin{aligned}
& (1-y) \cdot (1-x)(1-z) \\
+\ & (1-x) \cdot (1-y)z \\
+\ & 1 \cdot (1-x)y(1-u) \\
+\ & (1-x) \cdot yu \\
+\ & x \cdot (1-u)(1-v) \\
+\ & (1-u) \cdot xv \\
+\ & x \cdot u(1-w) \\
+\ & 1 \cdot xuw \\
=\ & 1
\end{aligned}$$

Size 27
Degree 3
(No use of Boolean axioms)

# Nullstellensatz Proof Search

- Solve linear system of equations with coefficients of polynomials $q_i$, $r_j$ as unknowns
- Used successfully to solve, e.g., graph colouring problems [DLMM08, DLMO09, DLMM11]
- Running time grows exponentially with degree, though high-degree refutations can be very small [BCIP02, dRMNR21]

# Dual Variables

- Annoying problem: $x_1 \lor x_2 \lor x_3$ translates to polynomial

$$(1 - x_1)(1 - x_2)(1 - x_3) = 1 - x_1 - x_2 - x_3 + x_1x_2 + x_1x_3 + x_2x_3 - x_1x_2x_3$$

# Dual Variables

- Annoying problem: $x_1 \lor x_2 \lor x_3$ translates to polynomial

$$(1 - x_1)(1 - x_2)(1 - x_3) = 1 - x_1 - x_2 - x_3 + x_1x_2 + x_1x_3 + x_2x_3 - x_1x_2x_3$$

- More generally, exponential blow-up in $\#$ positive literals

# Dual Variables

- Annoying problem: $x_1 \lor x_2 \lor x_3$ translates to polynomial

$$(1 - x_1)(1 - x_2)(1 - x_3) = 1 - x_1 - x_2 - x_3 + x_1x_2 + x_1x_3 + x_2x_3 - x_1x_2x_3$$

- More generally, exponential blow-up in $\#$ positive literals
- Fix: introduce dual variables $x_i'$ and axioms $x_i + x_i' - 1 = 0$

# Dual Variables

- Annoying problem: $x_1 \lor x_2 \lor x_3$ translates to polynomial

$$(1-x_1)(1-x_2)(1-x_3) = 1 - x_1 - x_2 - x_3 + x_1x_2 + x_1x_3 + x_2x_3 - x_1x_2x_3$$

- More generally, exponential blow-up in $\#$ positive literals
- Fix: introduce dual variables $x_i'$ and axioms $x_i + x_i' - 1 = 0$
- Translate $C = \bigvee_{i \in \mathcal{P}} x_i \lor \bigvee_{j \in \mathcal{N}} \overline{x}_j$ to polynomial equations

$$\prod_{i \in \mathcal{P}} x_i' \cdot \prod_{j \in \mathcal{N}} x_j = 0$$

# Dual Variables

- Annoying problem: $x_1 \lor x_2 \lor x_3$ translates to polynomial

$$(1-x_1)(1-x_2)(1-x_3) = 1 - x_1 - x_2 - x_3 + x_1x_2 + x_1x_3 + x_2x_3 - x_1x_2x_3$$

- More generally, exponential blow-up in # positive literals
- Fix: introduce dual variables $x_i'$ and axioms $x_i + x_i' - 1 = 0$
- Translate $C = \bigvee_{i \in \mathcal{P}} x_i \lor \bigvee_{j \in \mathcal{N}} \overline{x}_j$ to polynomial equations

$$\prod_{i \in \mathcal{P}} x_i' \cdot \prod_{j \in \mathcal{N}} x_j = 0$$

- Doesn't affect degree (obviously), but can decrease size exponentially [dRLNS21] (also for other algebraic proof systems)

# Dynamic Construction of Nullstellensatz Certificates

## Nullstellensatz again

Infeasibility of

$$\begin{aligned} p_i(x_1, \ldots, x_n) &= 0 & \qquad i &\in [m] \\ x_j^2 - x_j &= 0 & j &\in [n] \\ x_j + x_j' - 1 &= 0 & j &\in [n] \end{aligned}$$

$$\Updownarrow$$

1 lies in polynomial ideal $\mathcal{I}$ generated by these polynomials

# Dynamic Construction of Nullstellensatz Certificates

## Nullstellensatz again

Infeasibility of

$$\begin{aligned} p_i(x_1, \ldots, x_n) &= 0 & i &\in [m] \\ x_j^2 - x_j &= 0 & j &\in [n] \\ x_j + x_j' - 1 &= 0 & j &\in [n] \end{aligned}$$

$$\Updownarrow$$

1 lies in polynomial ideal $\mathcal{I}$ generated by these polynomials

- Ideal $\mathcal{I}$:
  1. $p, q \in \mathcal{I} \Rightarrow p + q \in \mathcal{I}$
  2. $p \in \mathcal{I} \Rightarrow r \cdot q \in \mathcal{I}$ for any $r$
- Compute polynomials in this ideal $\mathcal{I}$ step by step
- Use "multivariate division" to check whether 1 lies in ideal or not

# Gröbner Bases: Admissible Orderings and Leading Terms

Admissible ordering $\preceq$ on monomials $m, m', t$:

1. $m \preceq m' \Rightarrow t \cdot m \preceq t \cdot m'$
2. $m \preceq t \cdot m$

Examples:

- Lexicographic
- Degree-lexicographic

# Gröbner Bases: Admissible Orderings and Leading Terms

Admissible ordering $\preceq$ on monomials $m, m', t$:

1. $m \preceq m' \Rightarrow t \cdot m \preceq t \cdot m'$
2. $m \preceq t \cdot m$

Examples:

- Lexicographic
- Degree-lexicographic

Can write $p = \mathrm{lt}(p) + p'$ for $\mathrm{lt}(p)$ leading term (largest w.r.t. $\preceq$)

If $\mathrm{lt}(p) = t \cdot \mathrm{lt}(q)$, can reduce $p \bmod q$ by computing $p - t \cdot q$

# Gröbner Bases: Admissible Orderings and Leading Terms

Admissible ordering $\preceq$ on monomials $m, m', t$:

1. $m \preceq m' \Rightarrow t \cdot m \preceq t \cdot m'$
2. $m \preceq t \cdot m$

Examples:

- Lexicographic
- Degree-lexicographic

Can write $p = \mathrm{lt}(p) + p'$ for $\mathrm{lt}(p)$ leading term (largest w.r.t. $\preceq$)

If $\mathrm{lt}(p) = t \cdot \mathrm{lt}(q)$, can reduce $p \bmod q$ by computing $p - t \cdot q$

"Multivariate division": Reduce $p$ modulo all $q$ in set of polynomials $\mathcal{G}$ until no further reductions possible

$\mathcal{G}$ is a Gröbner basis if final result uniquely determined

# Gröbner Bases: Buchberger's Algorithm

## Buchberger's algorithm for computing Gröbner bases (**very** rough)

1. Let $\mathcal{G} :=$ all axioms
2. Pick unprocessed pair $p, q \in \mathcal{G}$ or terminate if none exists
3. Compute $p' = t_p \cdot p$ and $q' = t_q \cdot q$ to make leading terms cancel
4. Set $S := p' - q'$; reduce $S \bmod \mathcal{G}$ with multivariate division; add result to $\mathcal{G}$ if non-zero
5. Go to 2

# Gröbner Bases: Buchberger's Algorithm

## Buchberger's algorithm for computing Gröbner bases (**very** rough)

1. Let $\mathcal{G}$ := all axioms
2. Pick unprocessed pair $p, q \in \mathcal{G}$ or terminate if none exists
3. Compute $p' = t_p \cdot p$ and $q' = t_q \cdot q$ to make leading terms cancel
4. Set $S := p' - q'$; reduce $S \bmod \mathcal{G}$ with multivariate division; add result to $\mathcal{G}$ if non-zero
5. Go to 2

**Facts:**

- Buchberger's algorithm computes Gröbner basis
- At termination, $1 \in \mathcal{G} \Leftrightarrow$ polynomial equations infeasible

# Polynomial Calculus [CEI96, ABRW02]

- Compute polynomials in ideal $\mathcal{I}$ generated by $p_i$, $x_j^2 - x_j$, and $x_j + x'_j - 1$ step by step:
  - $p_i \in \mathcal{I}$, $x_j^2 - x_j \in \mathcal{I}$, and $x_j + x'_j - 1 \in \mathcal{I}$ (axioms)
  - If $p, q \in \mathcal{I}$, then $\alpha p + \beta q \in \mathcal{I}$ for any $\alpha, \beta \in \mathbb{F}$ (linear combination)
  - If $p \in \mathcal{I}$, then $m \cdot p \in \mathcal{I}$ for any monomial $m = \prod_j x_j$ (multiplication)

# Polynomial Calculus [CEI96, ABRW02]

- Compute polynomials in ideal $\mathcal{I}$ generated by $p_i$, $x_j^2 - x_j$, and $x_j + x'_j - 1$ step by step:
    - $p_i \in \mathcal{I}$, $x_j^2 - x_j \in \mathcal{I}$, and $x_j + x'_j - 1 \in \mathcal{I}$ (axioms)
    - If $p, q \in \mathcal{I}$, then $\alpha p + \beta q \in \mathcal{I}$ for any $\alpha, \beta \in \mathbb{F}$ (linear combination)
    - If $p \in \mathcal{I}$, then $m \cdot p \in \mathcal{I}$ for any monomial $m = \prod_j x_j$ (multiplication)
- A refutation is a derivation ending with the polynomial $1$
- Complexity measures:
    - Size: total number of monomials in all polynomials in derivation expanded out
    - Degree: highest total degree of any polynomial
- Polynomial calculus (much) stronger than Nullstellensatz w.r.t. both size and degree

# Polynomial Calculus Can Simulate Resolution

Polynomial calculus can always simulate resolution proofs efficiently step by step

# Polynomial Calculus Can Simulate Resolution

Polynomial calculus can always simulate resolution proofs efficiently step by step

**Example:** Resolution step

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

# Polynomial Calculus Can Simulate Resolution

Polynomial calculus can always simulate resolution proofs efficiently step by step

**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

$$\frac{x'yz' \qquad \dfrac{\dfrac{yz}{x'yz} \qquad \dfrac{z + z' - 1}{x'yz + x'yz' - x'y}}{-x'yz' + x'y}}{x'y}$$

# Polynomial Calculus is Strictly Stronger than Resolution

Polynomial calculus can be exponentially stronger than resolution

For instance:

- Tseitin formulas on expander graphs if $\mathbb{F} = \mathrm{GF}(2)$
- Onto functional pigeonhole principle over any field [Rii93]

# Polynomial Calculus is Strictly Stronger than Resolution

Polynomial calculus can be exponentially stronger than resolution

For instance:

- Tseitin formulas on expander graphs if $\mathbb{F} = \mathrm{GF}(2)$
- Onto functional pigeonhole principle over any field [Rii93]

But other versions of pigeonhole principle formulas remain hard:

- "vanilla" PHP [Raz98, AR03]
- onto PHP [AR03]
- functional PHP [MN15]

# Polynomial Calculus is Strictly Stronger than Resolution

Polynomial calculus can be exponentially stronger than resolution

For instance:

- Tseitin formulas on expander graphs if $\mathbb{F} = \mathrm{GF}(2)$
- Onto functional pigeonhole principle over any field [Rii93]

But other versions of pigeonhole principle formulas remain hard:

- “vanilla” PHP [Raz98, AR03]
- onto PHP [AR03]
- functional PHP [MN15]

Other hard formulas:

- Tseitin-like formulas for counting $\bmod p$ if $p \neq$ field characteristic [BGIP01]
- Random $k$-CNF formulas
  - all characteristics except $2$ [BI99]
  - all characteristics [AR03]

# COLOURING and CLIQUE for Polynomial Calculus

## COLOURING

- Exponential worst-case lower bounds in [LN17]
- Exponential average-case lower bounds in [CdRN$^+$23]

## CLIQUE

Essentially nothing known!

# What About Algebraic SAT Solvers?

- Excitement about Gröbner basis approach after [CEI96], but promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s...

# What About Algebraic SAT Solvers?

- Excitement about Gröbner basis approach after [CEI96], but promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s...
- 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)?

# What About Algebraic SAT Solvers?

- Excitement about Gröbner basis approach after [CEI96], but promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s. . .
- 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)?
- Work in [KFB20, KB20, KBK20a, KBK20b, KB21] on circuit verification quite successful, but struggles with monomial blow-up

# What About Algebraic SAT Solvers?

- Excitement about Gröbner basis approach after [CEI96], but promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s. . .
- 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)?
- Work in [KFB20, KB20, KBK20a, KBK20b, KB21] on circuit verification quite successful, but struggles with monomial blow-up
- Use dual variables! [KBBN22]

# Gröbner bases: Some Problems and Questions

1. Buchberger not a great SAT solving algorithm
   Slow and memory-intensive, and computes too much info
   Possible to use conflict-driven paradigm?!

2. Dual variables increase reasoning power exponentially [dRLNS21]
   But are immediately eliminated by multivariate division
   Possible to design dual-variable-aware Buchberger?!

3. Analysis of polynomial calculus uses degree-lexicographic ordering
   In computational algebra, many other orderings used
   Prove proof complexity separation results for different orderings?

# SAT as System of 0–1 Integer Linear Inequalities

- Given CNF formula $F = \bigwedge_{i=1}^{m} C_i$

# SAT as System of 0–1 Integer Linear Inequalities

- Given CNF formula $F = \bigwedge_{i=1}^{m} C_i$
- Translate clauses

$$C = \bigvee_{i \in \mathcal{P}} x_i \vee \bigvee_{j \in \mathcal{N}} \overline{x}_j$$

to 0-1 integer linear inequalities

$$\sum_{i \in \mathcal{P}} x_i + \sum_{j \in \mathcal{N}} (1 - x_j) \geq 1$$

# SAT as System of 0–1 Integer Linear Inequalities

- Given CNF formula $F = \bigwedge_{i=1}^{m} C_i$
- Translate clauses

$$C = \bigvee_{i \in \mathcal{P}} x_i \vee \bigvee_{j \in \mathcal{N}} \overline{x}_j$$

to 0-1 integer linear inequalities

$$\sum_{i \in \mathcal{P}} x_i + \sum_{j \in \mathcal{N}} (1 - x_j) \geq 1$$

- Add variable axioms

$$x_j \geq 0$$
$$-x_j \geq -1$$

for all variables

# Cutting Planes Proof System [CCT87]

Cutting planes introduced in [CCT87] to model integer linear programming algorithm in [Gom63, Chv73]

Can be applied to any system of $0$-$1$ integer linear inequalities

# Cutting Planes Proof System [CCT87]

Cutting planes introduced in [CCT87] to model integer linear programming algorithm in [Gom63, Chv73]

Can be applied to any system of $0$-$1$ integer linear inequalities

## Cutting planes derivation rules

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

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

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

# Cutting Planes Derivations and Refutations

- A cutting planes derivation is a sequence of $0$-$1$ integer linear inequalities derived using
  - Axioms (clauses and variable bounds)
  - Multiplication $\sum a_i x_i \geq A \Rightarrow \sum c a_i x_i \geq cA$
  - Addition $\sum a_i x_i \geq A$, $\sum b_i x_i \geq B \Rightarrow \sum (a_i + b_i) x_i \geq A + B$
  - Division $\sum a_i x_i \geq A \Rightarrow \sum \lceil a_i / c \rceil x_i \geq \lceil A/c \rceil$
- A refutation ends with the inequality $0 \geq 1$
- Complexity measures:
  - Length: $\#$ inequalities
  - Size: Count also bit size of representing all coefficients

# Cutting Planes vs. Resolution

- Cutting planes can simulate resolution reasoning efficiently and can be exponentially stronger (e.g., for PHP, just count and argue that #pigeons $>$ #holes)

# Cutting Planes vs. Resolution

- Cutting planes can simulate resolution reasoning efficiently and can be exponentially stronger (e.g., for PHP, just count and argue that #pigeons > #holes)
- And 0-1 linear inequalities are similar to but much more concise than CNF

Compare

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3$$

and

$$\begin{aligned}
&(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \\
\land &(x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \\
\land &(x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_5 \lor x_6) \\
\land &(x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \\
\land &(x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6)
\end{aligned}$$

# Hard Formulas for Cutting Planes

**Clique-colouring formulas** [Pud97]

"A graph with an $m$-clique is not $(m-1)$-colourable"

# Hard Formulas for Cutting Planes

**Clique-colouring formulas** [Pud97]

"A graph with an $m$-clique is not $(m-1)$-colourable"

Variables

- $p_{i,j}$ indicators of the edges in graph; $1 \leq i < j \leq n$
- $q_{k,i}$ identify members of $m$-clique; $1 \leq k \leq m$, $1 \leq i \leq n$
- $r_{i,\ell}$ specify colouring of vertices; $1 \leq \ell \leq m-1$, $1 \leq i \leq n$

# Hard Formulas for Cutting Planes

**Clique-colouring formulas** [Pud97]

"A graph with an $m$-clique is not $(m-1)$-colourable"

Variables

- $p_{i,j}$ indicators of the edges in graph; $1 \leq i < j \leq n$
- $q_{k,i}$ identify members of $m$-clique; $1 \leq k \leq m$, $1 \leq i \leq n$
- $r_{i,\ell}$ specify colouring of vertices; $1 \leq \ell \leq m-1$, $1 \leq i \leq n$

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

# More Hard Formulas for Cutting Planes?

Lower bound for clique-colouring formulas uses interpolation and circuit complexity

- From small cutting planes proof, build small circuit of special type that can decide whether graph has clique
- Prove separately that no such small circuits can exist
- Hence, no small cutting planes proofs can exist either

# More Hard Formulas for Cutting Planes?

Lower bound for clique-colouring formulas uses interpolation and circuit complexity

- From small cutting planes proof, build small circuit of special type that can decide whether graph has clique
- Prove separately that no such small circuits can exist
- Hence, no small cutting planes proofs can exist either

Cutting planes not well understood at all
Clear need for development of new analysis methods
Some exciting contributions in [HP17, FPPR22, GGKS20, Sok23]

# More Hard Formulas for Cutting Planes?

Lower bound for clique-colouring formulas uses interpolation and circuit complexity

- From small cutting planes proof, build small circuit of special type that can decide whether graph has clique
- Prove separately that no such small circuits can exist
- Hence, no small cutting planes proofs can exist either

Cutting planes not well understood at all
Clear need for development of new analysis methods
Some exciting contributions in [HP17, FPPR22, GGKS20, Sok23]

Nothing known for COLOURING or CLIQUE
Surprisingly, Tseitin formulas are at most quasi-polynomially hard for cutting planes [DT20]!

# SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean (PB) solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, SS06, LP10, EN18]

Perhaps counter-intuitively, hard to make competitive with CDCL

# SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean (PB) solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, SS06, LP10, EN18]

Perhaps counter-intuitively, hard to make competitive with CDCL

**Challenge 1: Conjunctive normal form**

- Pseudo-Boolean solvers terrible for CNF input
- Solvers can rewrite CNF to more helpful $0$-$1$ linear inequalities [BLLM14, EN20], but this doesn't work so well in practice
- Better to encode problem with $0$-$1$ inequalities from the start

# SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean (PB) solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, SS06, LP10, EN18]

Perhaps counter-intuitively, hard to make competitive with CDCL

**Challenge 1: Conjunctive normal form**

- Pseudo-Boolean solvers terrible for CNF input
- Solvers can rewrite CNF to more helpful $0$-$1$ linear inequalities [BLLM14, EN20], but this doesn't work so well in practice
- Better to encode problem with $0$-$1$ inequalities from the start

**Challenge 2: Increased degrees of freedom(!?)**

- Cutting planes much smarter method of reasoning
- But this makes it trickier to design smart search algorithms

# SAT Solvers Based on Cutting Planes?

So-called pseudo-Boolean (PB) solvers using (subset of) cutting planes reasoning developed in, e.g., [CK05, SS06, LP10, EN18]

Perhaps counter-intuitively, hard to make competitive with CDCL

**Challenge 1: Conjunctive normal form**

- Pseudo-Boolean solvers terrible for CNF input
- Solvers can rewrite CNF to more helpful $0$-$1$ linear inequalities [BLLM14, EN20], but this doesn't work so well in practice
- Better to encode problem with $0$-$1$ inequalities from the start

**Challenge 2: Increased degrees of freedom(!?)**

- Cutting planes much smarter method of reasoning
- But this makes it trickier to design smart search algorithms

Is it truly harder to build good pseudo-Boolean solvers?
Or has just so much more work has been put into CDCL solvers?

# Division Versus Saturation

Use negated literals as needed to get all $a_i$, $A$ positive

## Boolean derivation rules for $0$–$1$ integer linear inequalities

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

Saturation
$$\frac{\sum a_i \ell_i \geq A}{\sum \min\{a_i, A\} \cdot \ell_i \geq A}$$

# Division Versus Saturation

Use negated literals as needed to get all $a_i$, $A$ positive

## Boolean derivation rules for $0$–$1$ integer linear inequalities

$$\text{Division} \quad \frac{\sum a_i \ell_i \geq A}{\sum \lceil a_i / c \rceil \ell_i \geq \lceil A / c \rceil} \quad c \in \mathbb{N}^+$$

$$\text{Saturation} \quad \frac{\sum a_i \ell_i \geq A}{\sum \min\{a_i, A\} \cdot \ell_i \geq A}$$

- Complexity literature of cutting planes uses division [CCT87]
- Pseudo-Boolean solvers instead adopted saturation [CK05, LP10]
- Open how the two variants compare, but clear that division can sometimes be better in theory [GNY19]

# Division Versus Saturation

Use negated literals as needed to get all $a_i$, $A$ positive

**Boolean derivation rules for $0$–$1$ integer linear inequalities**

$$\text{Division} \quad \frac{\sum a_i \ell_i \geq A}{\sum \lceil a_i/c \rceil \ell_i \geq \lceil A/c \rceil} \quad c \in \mathbb{N}^+$$

$$\text{Saturation} \quad \frac{\sum a_i \ell_i \geq A}{\sum \min\{a_i, A\} \cdot \ell_i \geq A}$$

- Complexity literature of cutting planes uses division [CCT87]
- Pseudo-Boolean solvers instead adopted saturation [CK05, LP10]
- Open how the two variants compare, but clear that division can sometimes be better in theory [GNY19]
- . . . And most often also in practice [EN18], though not always [LBD⁺20]

# Sherali–Adams (SA) and Sums of Squares (SoS)

Refutation of $p_i \in \mathbb{R}[x_1, \ldots, x_n]$, $i \in [m]$, and $x_j^2 - x_j$, $j \in [n]$

**Nullstellensatz**

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) = 1$$

# Sherali–Adams (SA) and Sums of Squares (SoS)

Refutation of $p_i \in \mathbb{R}[x_1, \ldots, x_n]$, $i \in [m]$, and $x_j^2 - x_j$, $j \in [n]$

**Nullstellensatz**

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) = -1$$

# Sherali–Adams (SA) and Sums of Squares (SoS)

Refutation of $p_i \in \mathbb{R}[x_1, \ldots, x_n]$, $i \in [m]$, and $x_j^2 - x_j$, $j \in [n]$

**Nullstellensatz**

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) = -1$$

**Sherali–Adams (SA)** $(\alpha_k \in \mathbb{R}^+)$

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) + \sum_{k=1}^{t} \alpha_k \prod_{i \in \mathcal{P}_t} (1 - x_i) \cdot \prod_{j \in \mathcal{N}_t} x_j = -1$$

# Sherali–Adams (SA) and Sums of Squares (SoS)

Refutation of $p_i \in \mathbb{R}[x_1, \ldots, x_n]$, $i \in [m]$, and $x_j^2 - x_j$, $j \in [n]$

**Nullstellensatz**

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) = -1$$

**Sherali–Adams (SA)** $(\alpha_k \in \mathbb{R}^+)$

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) + \sum_{k=1}^{t} \alpha_k \prod_{i \in \mathcal{P}_t} (1 - x_i) \cdot \prod_{j \in \mathcal{N}_t} x_j = -1$$

**Sums of squares (SoS)** $(s_k \in \mathbb{R}[x_1, \ldots, x_n])$

$$\sum_{i=1}^{m} q_i \cdot p_i + \sum_{j=1}^{n} r_j \cdot (x_j^2 - x_j) + \sum_{k=1}^{s} s_k^2 = -1$$

# Sherali–Adams, Sums of Squares, and Relations to Other Proof Systems

**Sherali–Adams** models linear programming (LP) hierarchies

**Sums of squares** models semidefinite programming (SDP) hierarchies

# Sherali–Adams, Sums of Squares, and Relations to Other Proof Systems

**Sherali–Adams** models linear programming (LP) hierarchies

**Sums of squares** models semidefinite programming (SDP) hierarchies

Strict hierarchy (over $\mathbb{R}$):

- Nullstellensatz
- Sherali–Adams
- Sums of squares

Sums of squares is strictly stronger than polynomial calculus (over $\mathbb{R}$)
Sherali-Adams and polynomial calculus are incomparable [Ber18]

# Sherali–Adams, Sums of Squares, and Relations to Other Proof Systems

**Sherali–Adams** models linear programming (LP) hierarchies

**Sums of squares** models semidefinite programming (SDP) hierarchies

Strict hierarchy (over $\mathbb{R}$):

- Nullstellensatz
- Sherali–Adams
- Sums of squares

Sums of squares is strictly stronger than polynomial calculus (over $\mathbb{R}$)
Sherali-Adams and polynomial calculus are incomparable [Ber18]

Sums of squares very strong proof system (e.g., can reason about PHP)
But can't do parity reasoning efficiently [GV01, Gri01]

Survey [FKP19] recommended for more reading

# Stabbing Planes [BFI+18]

Intended to model modern $0$-$1$ integer linear programming

# Stabbing Planes [BFI+18]

Intended to model modern $0$-$1$ integer linear programming

**Stabbing planes refutation of set of $0$-$1$ integer linear inequalities $\mathcal{S}$**

1. If polytope $\mathcal{S}$ is empty over $\mathbb{R}$, terminate this branch

# Stabbing Planes [BFI+18]

Intended to model modern $0$-$1$ integer linear programming

**Stabbing planes refutation of set of $0$-$1$ integer linear inequalities $\mathcal{S}$**

1. If polytope $\mathcal{S}$ is empty over $\mathbb{R}$, terminate this branch
2. Otherwise, pick new inequality $\sum_i a_i \ell_i \geq A$ to branch on

# Stabbing Planes [BFI$^+$18]

Intended to model modern $0$-$1$ integer linear programming

**Stabbing planes refutation of set of $0$-$1$ integer linear inequalities $\mathcal{S}$**

1. If polytope $\mathcal{S}$ is empty over $\mathbb{R}$, terminate this branch
2. Otherwise, pick new inequality $\sum_i a_i\ell_i \geq A$ to branch on
3. Recurse with $\mathcal{S} := \mathcal{S} \cup \{\sum_i a_i\ell_i \geq A\}$

# Stabbing Planes [BFI+18]

Intended to model modern $0$-$1$ integer linear programming

**Stabbing planes refutation of set of $0$-$1$ integer linear inequalities $\mathcal{S}$**

1. If polytope $\mathcal{S}$ is empty over $\mathbb{R}$, terminate this branch
2. Otherwise, pick new inequality $\sum_i a_i \ell_i \geq A$ to branch on
3. Recurse with $\mathcal{S} := \mathcal{S} \cup \{\sum_i a_i \ell_i \geq A\}$
4. Recurse with $\mathcal{S} := \mathcal{S} \cup \{\sum_i a_i \ell_i \leq A - 1\}$

# Stabbing Planes [BFI+18]

Intended to model modern $0$-$1$ integer linear programming

**Stabbing planes refutation of set of $0$-$1$ integer linear inequalities $\mathcal{S}$**

1. If polytope $\mathcal{S}$ is empty over $\mathbb{R}$, terminate this branch
2. Otherwise, pick new inequality $\sum_i a_i \ell_i \geq A$ to branch on
3. Recurse with $\mathcal{S} := \mathcal{S} \cup \{\sum_i a_i \ell_i \geq A\}$
4. Recurse with $\mathcal{S} := \mathcal{S} \cup \{\sum_i a_i \ell_i \leq A - 1\}$

Complexity measures:

- Length: $\#$ branching nodes / sets $\mathcal{S}$
- Size: Count also bit size for representing all coefficients

# Stabbing Planes and Cutting Planes

Stabbing planes efficiently simulates cutting planes [BFI+18]

Stabbing planes probably much stronger!?

# Stabbing Planes and Cutting Planes

Stabbing planes efficiently simulates cutting planes [BFI+18]

Stabbing planes probably much stronger!?

Or maybe not... Stabbing planes with polynomial-size coefficient can be simulated by cutting planes with quasi-polynomial overhead [DT20, FGI+21]

# Stabbing Planes and Cutting Planes

Stabbing planes efficiently simulates cutting planes [BFI$^+$18]

Stabbing planes probably much stronger!?

Or maybe not... Stabbing planes with polynomial-size coefficient can be simulated by cutting planes with quasi-polynomial overhead [DT20, FGI$^+$21]

Very recent news: Interpolation and circuit complexity can be used to get similar lower bounds for stabbing planes as for cutting planes! [GP24]

# Stabbing Planes and Cutting Planes

Stabbing planes efficiently simulates cutting planes [BFI+18]

Stabbing planes probably much stronger!?

Or maybe not... Stabbing planes with polynomial-size coefficient can be simulated by cutting planes with quasi-polynomial overhead [DT20, FGI+21]

Very recent news: Interpolation and circuit complexity can be used to get similar lower bounds for stabbing planes as for cutting planes! [GP24]

Still possible that stabbing planes is exponentially more powerful than cutting planes, but hard to know what to believe

# Extended Resolution [Tse68]

**Resolution rule**

$$\frac{C_1 \lor x \qquad C_2 \lor \overline{x}}{C_1 \lor C_2}$$

**Extension rule** introducing clauses

$$a \lor \overline{x} \lor \overline{y} \qquad \overline{a} \lor x \qquad \overline{a} \lor y$$

for fresh variable $a$ (encoding that $a \leftrightarrow (x \land y)$ must hold)

# Extended Resolution and SAT Solving

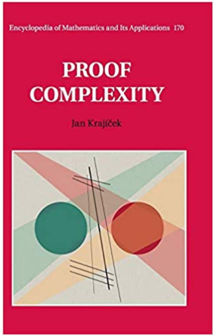
- Closely related (and equivalent) to *DRAT* system used to justify correctness of some SAT preprocessing techniques [JHB12]
- *DRAT* also used for SAT solver proof logging
- Attempts to combine extended resolution with CDCL in, e.g., [AKS10, Hua10]
- Without restrictions, corresponds to extremely strong extended Frege system [CR79] — pretty much no lower bounds known
- To analyse solvers using extended resolution, would need to:
  - Describe heuristics/rules actually used
  - See if possible to reason about such restricted proof system

# Some More References for Further Reading

**Handbook of Satisfiability**
(Especially chapter 7 ☺)


[BHvMW21]

**Proof Complexity**
by Jan Krajíček




[Kra19]

# Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ conflict-driven clause learning (CDCL)
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

# Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ conflict-driven clause learning (CDCL)
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali–Adams and sums of squares $\longleftrightarrow$ LP and SDP hierarchies
- Stabbing planes $\longleftrightarrow$ integer linear programming
- Extended resolution $\longleftrightarrow$ SAT pre- and inprocessing

# Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ conflict-driven clause learning (CDCL)
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali–Adams and sums of squares $\longleftrightarrow$ LP and SDP hierarchies
- Stabbing planes $\longleftrightarrow$ integer linear programming
- Extended resolution $\longleftrightarrow$ SAT pre- and inprocessing

Proof complexity useful to

- Analyse state-of-the-art algorithms (and provide methods for certifying correctness!)
- Give ideas for new approaches
- Provide a fun playground for theory-practice interaction!

# Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ conflict-driven clause learning (CDCL)
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali–Adams and sums of squares $\longleftrightarrow$ LP and SDP hierarchies
- Stabbing planes $\longleftrightarrow$ integer linear programming
- Extended resolution $\longleftrightarrow$ SAT pre- and inprocessing

Proof complexity useful to

- Analyse state-of-the-art algorithms (and provide methods for certifying correctness!)
- Give ideas for new approaches
- Provide a fun playground for theory-practice interaction! (And we're hiring!)

# Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ conflict-driven clause learning (CDCL)
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali–Adams and sums of squares $\longleftrightarrow$ LP and SDP hierarchies
- Stabbing planes $\longleftrightarrow$ integer linear programming
- Extended resolution $\longleftrightarrow$ SAT pre- and inprocessing

Proof complexity useful to

- Analyse state-of-the-art algorithms (and provide methods for certifying correctness!)
- Give ideas for new approaches
- Provide a fun playground for theory-practice interaction! (And we're hiring!)

Thank you for your attention!

# References I


[ABdR+21] Albert Atserias, Ilario Bonacina, Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Alexander Razborov. Clique is hard on average for regular resolution. *Journal of the ACM*, 68(4):23:1–23:26, August 2021. Preliminary version in *STOC '18*.

[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, April 2002. Preliminary version in *STOC '00*.

[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.

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

[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*.

# References II


[BBN+23] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

[BBN+24] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, September 2024. To appear.

[BCIP02] Joshua Buresh-Oppenheim, Matthew Clegg, Russell Impagliazzo, and Toniann Pitassi. Homogenization and the polynomial calculus. *Computational Complexity*, 11(3-4):91–108, 2002. Preliminary version in *ICALP '00*.

[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.

# References III


[Ber18] Christoph Berkholz. The relation between polynomial calculus, Sherali-Adams, and sum-of-squares proofs. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS '18)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14, February 2018.

[BFI+18] Paul Beame, Noah Fleming, Russell Impagliazzo, Antonina Kolokolova, Denis Pankratov, Toniann Pitassi, and Robert Robere. Stabbing planes. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS '18)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, January 2018.

[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*.

[BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

# References IV


[BHvMW21] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.

[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(4):501–519, 2010. Preliminary version in *FOCS '99*.

[BIK+94] Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS '94)*, pages 794–806, November 1994.

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

# References V


[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.

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [BHvMW21], chapter 7, pages 233–350.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

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

[CdRN+23] Jonas Conneryd, Susanna F. de Rezende, Jakob Nordström, Shuo Pang, and Kilian Risse. Graph colouring is hard on average for polynomial calculus and Nullstellensatz. In *Proceedings of the 64th Annual IEEE Symposium on Foundations of Computer Science (FOCS '23)*, pages 1–11, November 2023.

# References VI


[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. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(1):305–337, 1973.

[CK05] Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, March 2005. Preliminary version in *DAC '03*.

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

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979. Preliminary version in *STOC '74*.

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

# References VII


[DGD$^{+}$21] Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3750–3758, February 2021.

[DGN21] Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating $0$-$1$ integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1–4):26–55, October 2021. Preliminary version in *CPAIOR '20*.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DLMM08] Jesús A. De Loera, Jon Lee, Peter N. Malkin, and Susan Margulies. Hilbert's Nullstellensatz and an algorithm for proving combinatorial infeasibility. In *Proceedings of the 21st International Symposium on Symbolic and Algebraic Computation (ISSAC '08)*, pages 197–206, July 2008.

[DLMM11] Jesús A. De Loera, Jon Lee, Peter N. Malkin, and Susan Margulies. Computing infeasibility certificates for combinatorial problems through Hilbert's Nullstellensatz. *Journal of Symbolic Computation*, 46(11):1260–1283, November 2011.

# References VIII


[DLMO09] Jesús A. De Loera, Jon Lee, Susan Margulies, and Shmuel Onn. Expressing combinatorial problems by systems of polynomial equations and Hilbert's Nullstellensatz. *Combinatorics, Probability and Computing*, 18(4):551–582, July 2009.

[DMM+24] Emir Demirović, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, and Konstantin Sidorov. Pseudo-Boolean reasoning about states and transitions to certify dynamic programming and decision diagram algorithms. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, September 2024. To appear.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[dRGN+21] Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC '21)*, pages 209–222, June 2021.

[dRLNS21] Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Dmitry Sokolov. The power of negative reasoning. In *Proceedings of the 36th Annual Computational Complexity Conference (CCC '21)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:24, July 2021.

# References IX


[dRMNR21] Susanna F. de Rezende, Or Meir, Jakob Nordström, and Robert Robere. Nullstellensatz size-degree trade-offs from reversible pebbling. *Computational Complexity*, 30:4:1–4:45, February 2021.

[DT20] Daniel Dadush and Samarth Tiwari. On the complexity of branching proofs. In *Proceedings of the 35th Annual Computational Complexity Conference (CCC '20)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:35, July 2020.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

[EN18] Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.

[EN20] Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1495–1503, February 2020.

# References X


[FGI+21] Noah Fleming, Mika Göös, Russell Impagliazzo, Toniann Pitassi, Robert Robere, Li-Yang Tan, and Avi Wigderson. On the power and limitations of branch and cut. In *Proceedings of the 36th Annual Computational Complexity Conference (CCC '21)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:30, July 2021.

[FKP19] Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic proofs and efficient algorithm design. *Foundations and Trends in Theoretical Computer Science*, 14(1–2):1–221, December 2019.

[FPPR22] Noah Fleming, Denis Pankratov, Toniann Pitassi, and Robert Robere. Random $\theta(\log n)$-CNFs are hard for cutting planes. *Journal of the ACM*, 69(3):19:1–19:32, June 2022. Preliminary version in *FOCS '17*.

[GGKS20] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. *Theory of Computing*, 16(13):1–30, 2020. Preliminary version in *STOC '18*.

[GKMP20] Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC '20)*, pages 68–77, June 2020.

# References XI


[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

[GMM+24] Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 368h AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

# References XII


[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

[GNY19] Stephan Gocht, Jakob Nordström, and Amir Yehudayoff. On division versus saturation in pseudo-Boolean solving. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*, pages 1711–1718, August 2019.

[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.

[GP24] Max Gläser and Marc E. Pfetsch. Sub-exponential lower bounds for branch-and-bound with general disjunctions via interpolation. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '24)*, pages 3747–3764, January 2024.

# References XIII


[Gri01] Dima Grigoriev. Linear lower bound on degrees of Positivstellensatz calculus proofs for the parity. *Theoretical Computer Science*, 259(1–2):613–622, May 2001.

[GV01] Dima Grigoriev and Nicolai Vorobjov. Complexity of Null- and Positivstellensatz proofs. *Annals of Pure and Applied Logic*, 113(1–3):153–160, December 2001.

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

[Hås99] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999. Preliminary version in *FOCS '96*.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, July 2001. Preliminary version in *STOC '97*.

[HOGN24] Alexander Hoen, Andy Oertel, Ambros Gleixner, and Jakob Nordström. Certifying MIP-based presolve reductions for $0$–$1$ integer linear programs. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14742 of *Lecture Notes in Computer Science*, pages 310–328. Springer, May 2024.

# References XIV


[HP17] Pavel Hrubeš and Pavel Pudlák. Random formulas, monotone circuits, and interpolation. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS '17)*, pages 121–131, October 2017.

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

[IOT+24] Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR '24)*, July 2024. To appear.

[JHB12] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, June 2012.

[KB20] Daniela Kaufmann and Armin Biere. Nullstellensatz-proofs for multiplier verification. In *Proceedings of the 22nd International Workshop on Computer Algebra in Scientific Computing (CASC' 20)*, volume 12291 of *Lecture Notes in Computer Science*, pages 368–389. Springer, September 2020.

# References XV


[KB21] Daniela Kaufmann and Armin Biere. AMulet 2.0 for verifying multiplier circuits. In *Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '21)*, volume 12652 of *Lecture Notes in Computer Science*, pages 357–364. Springer, March-April 2021.

[KBBN22] Daniela Kaufmann, Paul Beame, Armin Biere, and Jakob Nordström. Adding dual variables to algebraic reasoning for circuit verification. In *Proceedings of the 25th Design, Automation and Test in Europe Conference (DATE '22)*, pages 1435–1440, March 2022.

[KBK20a] Daniela Kaufmann, Armin Biere, and Manuel Kauers. From DRUP to PAC and back. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE '20)*, pages 654–657, March 2020.

[KBK20b] Daniela Kaufmann, Armin Biere, and Manuel Kauers. Incremental column-wise verifiation of arithmetic circuits using computer algebra. *Formal Methods in Systems Design*, 56(1–3):22–54, 2020. Preliminary version in *FMCAD '17*.

[KFB20] Daniela Kaufmann, Mathias Fleury, and Armin Biere. The proof checkers Pacheck and Pastèque for the practical algebraic calculus. In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD '20)*, pages 264–269, September 2020.

# References XVI


[Kho01] Subhash Khot. Improved inapproximability results for MaxClique, chromatic number and approximate graph coloring. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '01)*, pages 600–609, October 2001.

[Kra19] Jan Krajíček. *Proof Complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, March 2019.

[LBD+20] Vincent Liew, Paul Beame, Jo Devriendt, Jan Elffers, and Jakob Nordström. Verifying properties of bit-vector multiplication using cutting planes reasoning. In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD '20)*, pages 194–204, September 2020.

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

[LN17] Massimo Lauria and Jakob Nordström. Graph colouring is hard for algorithms based on Hilbert's Nullstellensatz and Gröbner bases. In *Proceedings of the 32nd Annual Computational Complexity Conference (CCC '17)*, volume 79 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:20, July 2017.

# References XVII


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

[McC17] Ciaran McCreesh. *Solving Hard Subgraph Problems in Parallel*. PhD thesis, University of Glasgow, 2017.

[MM23] Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

[MMN24] Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14743 of *Lecture Notes in Computer Science*, pages 38–55. Springer, May 2024.

[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.

# References XVIII


[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*.

[Pro12] Patrick Prosser. Exact algorithms for maximum clique: A computational study. *Algorithms*, 5(4):545–587, November 2012.

[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.

# References XIX


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

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

[Sok23] Dmitry Sokolov. Random $(\log n)$-CNF are hard for cutting planes (again). Technical Report TR23-086, Electronic Colloquium on Computational Complexity (ECCC), June 2023.

[Spe10] Ivor Spence. sgen1: A generator of small but difficult satisfiability benchmarks. *Journal of Experimental Algorithmics*, 15:1.2:1–1.2:15, March 2010.

[SS06] Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, March 2006. Preliminary version in *DATE '05*.

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

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

# References XX


[VDB22] Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In *Proceedings of the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR '22)*, volume 13416 of *Lecture Notes in Computer Science*, pages 429–442. Springer, September 2022.

[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.

[Zuc07] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, August 2007. Preliminary version in *STOC '06*.