Proof Complexity and SAT Solving Survey

Jakob Nordström

University of Copenhagen and Lund University

Dagstuhl Workshop 22411

"Theory and Practice of SAT and Combinatorial Solving"

October 10, 2022

The Boolean Satisfiability (SAT) Problem

SAT

Given a propositional logic formula F, is there a satisfying assignment for F?

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

- 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?

The Boolean Satisfiability (SAT) Problem

SAT

Given a propositional logic formula F, is there a satisfying assignment for F?

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

- 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?

Can we use computers to solve this problem efficiently?

The Same Problem in Three Different Shapes

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

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?

Solving SAT in Theory and Practice

- Problem mentioned in Gödel's letter in 1956 to von Neumann
- Topic of intense research in computer science ever since 1960s
- NP-complete, so probably very hard worst case [Coo71, Lev73]
- But enormous progress last 20–25 years on conflict-driven clause learning (CDCL) SAT solvers [BS97, MS99, MMZ^{+}01]
- Today large-scale real-world problems with hundreds of thousands or millions of variables solved routinely
- But. . . There are also small formulas (just ∼100 variables) that are completely beyond reach for even the very best SAT solvers

Solving SAT in Theory and Practice

- Problem mentioned in Gödel's letter in 1956 to von Neumann
- Topic of intense research in computer science ever since 1960s
- NP-complete, so probably very hard worst case [Coo71, Lev73]
- But enormous progress last 20–25 years on conflict-driven clause learning (CDCL) SAT solvers [BS97, MS99, MMZ$^+$01]
- Today large-scale real-world problems with hundreds of thousands or millions of variables solved routinely
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach for even the very best SAT solvers

How can we rigorously analyse SAT solving algorithms?

Solving Sat in Theory and Practice

- Problem mentioned in Gödel's letter in 1956 to von Neumann
- Topic of intense research in computer science ever since 1960s
- NP-complete, so probably very hard worst case [Coo71, Lev73]
- But enormous progress last 20–25 years on conflict-driven clause learning (CDCL) SAT solvers [BS97, MS99, MMZ$^+$01]
- Today large-scale real-world problems with hundreds of thousands or millions of variables solved routinely
- But... There are also small formulas (just $\sim$100 variables) that are completely beyond reach for even the very best SAT solvers

How can we rigorously analyse SAT solving algorithms?

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

Algorithmic View of Proof Complexity

For any algorithm deciding satisfiability formula F, describe which rules of reasoning it uses

Algorithmic View of Proof Complexity

For any algorithm deciding satisfiability formula F, 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 deciding satisfiability formula F, 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 deciding F 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 deciding satisfiability formula F, 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 deciding F using rules in this proof system?
2. Can short proofs in the proof system be found efficiently?

Focus of this talk: Question 1 for different proof systems/algorithms
Study unsatisfiable formulas — proof of satisfiability easy

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
   - Polynomial Calculus and Gröbner Bases
   - Cutting Planes and Pseudo-Boolean Solving
3. Some Proof Systems We Won't 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 ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \lor \cdots \lor \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \land \cdots \land C_m$: 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 ℓ: 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?

For instance, what about our example formula?

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

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

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

x

A DPLL Toy Example

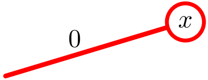
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

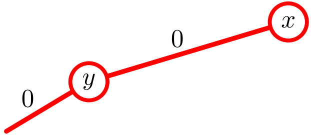
$$F = (\quad z) \land (\quad \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

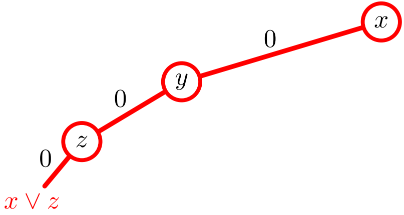
$$F = (x \vee z) \wedge (\quad \overline{z}) \wedge (\quad \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u}) \wedge (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

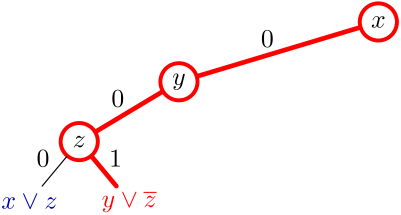
$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

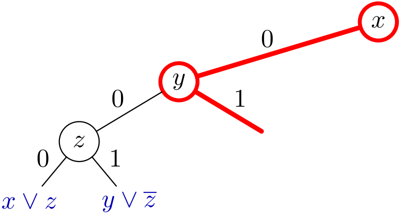
$$F = (\quad z) \land (y \lor \overline{z}) \land (\qquad u) \land (\quad \overline{u}) \\ \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

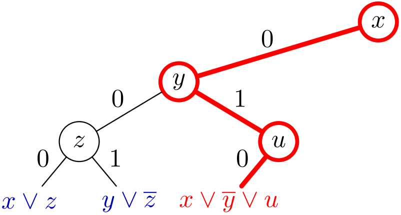
$$
\begin{aligned}
F = \quad & (\quad\; z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\quad\; \overline{u}) \\
& \land (\quad\; v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

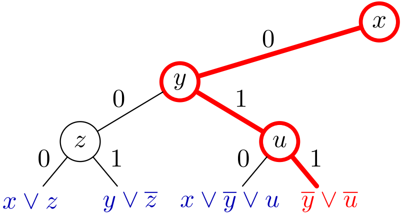
$$
\begin{aligned}
F = \quad & (\qquad z) \land (y \lor \overline{z}) \land (\qquad\qquad u) \land (\overline{y} \lor \overline{u}) \\
& \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\qquad w) \land (\overline{x} \lor \qquad \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

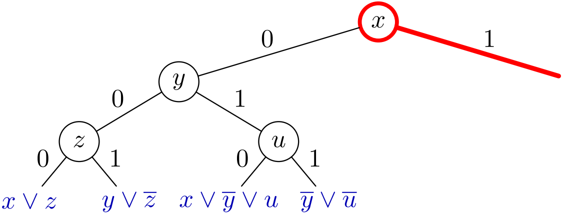
$$\begin{aligned} F = \quad & (x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u}) \\ & \wedge (u \vee v) \wedge (\quad \overline{v}) \wedge (\overline{u} \vee w) \wedge (\quad \overline{u} \vee \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

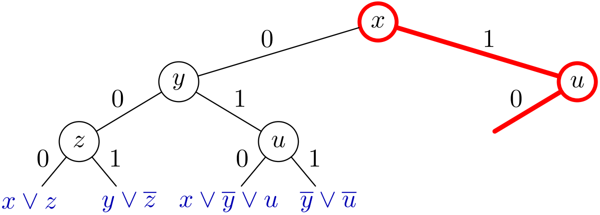
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \quad) \land (\overline{y} \lor \overline{u}) \land (\quad v) \land (\quad \overline{v}) \land (\overline{u} \lor w) \land (\quad \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

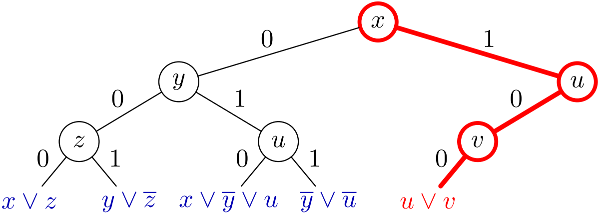
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \quad) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\quad \overline{v}) \land (\overline{u} \lor w) \land (\quad \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

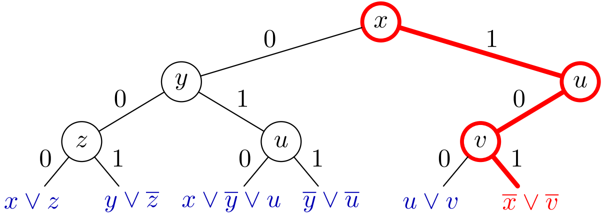
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \quad) \land (\overline{y} \lor \overline{u})$$
$$\land\ (\quad v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\quad \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

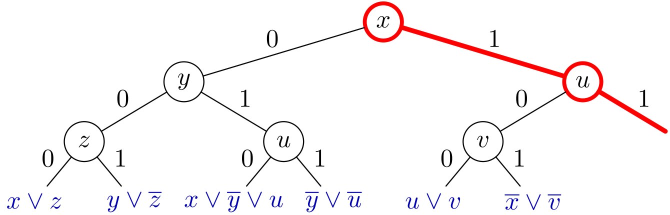
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\quad \overline{v}) \land (\quad w) \land (\quad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

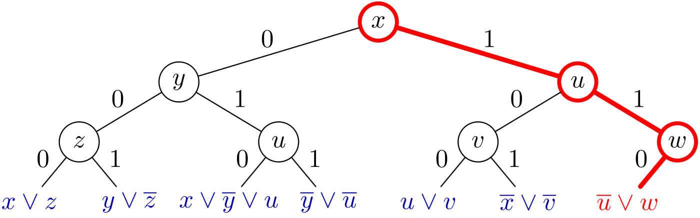
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{v}) \land (\overline{u} \lor w) \land (\overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

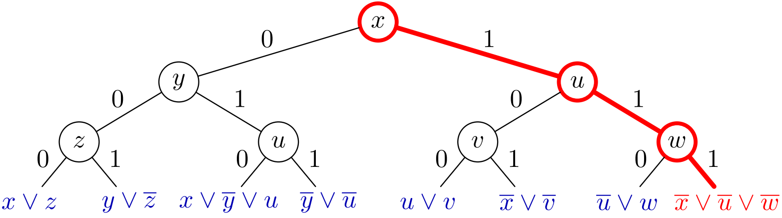
$$F = (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\quad \overline{v}) \land (\quad w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

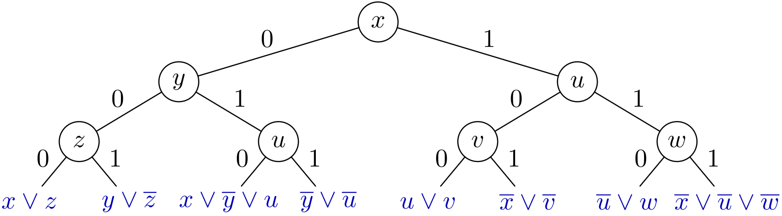
$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



State-of-the-art SAT solvers: Ingredients

Many more ingredients in modern conflict-driven clause learning (CDCL) SAT solvers (as pioneered in [BS97, MS99, MMZ⁺01]), e.g.:

- Branching or decision heuristic (choice of pivot variables crucial)
- When reaching leaf, 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)

Let us briefly discuss some of these ingredients

Variable Assignment Heuristics

Unit propagation

- Suppose current assignment ρ falsifies all literals in $C = \ell_1 \lor \ell_2 \lor \cdots \lor \ell_k$ except one (say ℓ_k) — C is unit under ρ
- Then ℓ_k has to be true, so set it to true
- Known as unit progagation or Boolean constraint progagation
- Always propagate if possible — in modern solvers aim for 99% of assignments being unit propagations

Variable Assignment Heuristics

Unit propagation

- Suppose current assignment ρ falsifies all literals in $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ except one (say ℓ_k) — C is unit under ρ
- Then ℓ_k has to be true, so set it to true
- Known as unit progagation or Boolean constraint progagation
- Always propagate if possible — in modern solvers aim for 99% of assignments being unit propagations

VSIDS (Variable state independent decaying sum)

- When backtracking, score $+1$ for variables "causing conflict"
- Also multiply all scores with factor $\kappa < 1$ — exponential filter rewarding variables involved in recent conflicts
- When no propagations, decide on variable with highest score

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \lor y \lor \overline{z}$ to avoid these decisions being made again — decision learning scheme

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \lor y \lor \overline{z}$ to avoid these decisions being made again — decision learning scheme
- In practice, more advanced learning schemes

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \lor y \lor \overline{z}$ to avoid these decisions being made again — decision learning scheme
- In practice, more advanced learning schemes
- Derive new clause from clauses unit propagating on the way to conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

Decisions, Unit Propagations, and Conflict

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

Decisions, Unit Propagations, and Conflict

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

Decisions, Unit Propagations, and Conflict

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

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)

Decisions, Unit Propagations, and Conflict

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

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)

Decisions, Unit Propagations, and Conflict

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

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

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

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

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

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

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

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 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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$\boxed{p \stackrel{\mathsf{d}}{=} 0}$
$u \stackrel{p \lor \overline{u}}{=} 0$
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

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

Could backtrack by removing last decision level & flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Could backtrack by removing last decision 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!

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

- $p \stackrel{\mathsf{d}}{=} 0$
- $u \stackrel{p \vee \overline{u}}{=} 0$
- $q \stackrel{\mathsf{d}}{=} 0$
- $r \stackrel{q \vee r}{=} 1$
- $w \stackrel{\overline{r} \vee w}{=} 1$
- $x \stackrel{\mathsf{d}}{=} 0$
- $y \stackrel{u \vee x \vee y}{=} 1$
- $z \stackrel{x \vee \overline{y} \vee z}{=} 1$
- $\stackrel{\overline{y} \vee \overline{z}}{\perp}$

$x \vee \overline{y}$

Could backtrack by removing last decision level & flipping last decision

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

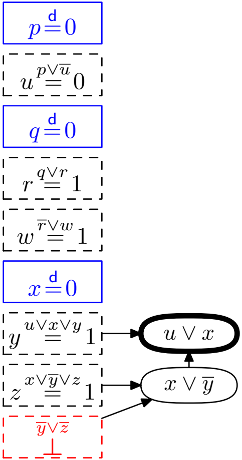
Case analysis over z for last two clauses:

- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \vee \overline{y}$

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 removing last decision 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:

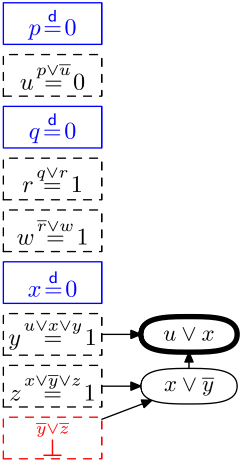
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \lor \overline{y}$

Repeat until UIP clause with only 1 variable 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

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

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

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

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

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

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

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

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

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

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

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

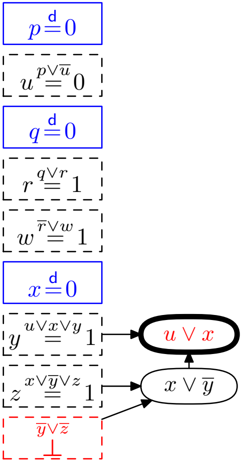
$u \stackrel{p \lor \overline{u}}{=} 0$

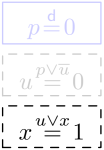
Assertion level 1 (max for non-UIP literal in learned clause) — keep trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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





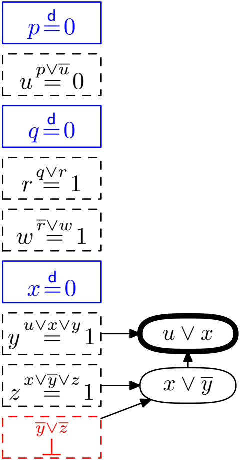
Assertion level 1 (max for non-UIP literal in learned clause) — keep trail to that level

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

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



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

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

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

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

Assertion level 1 (max for non-UIP literal in learned clause) — keep trail to that level

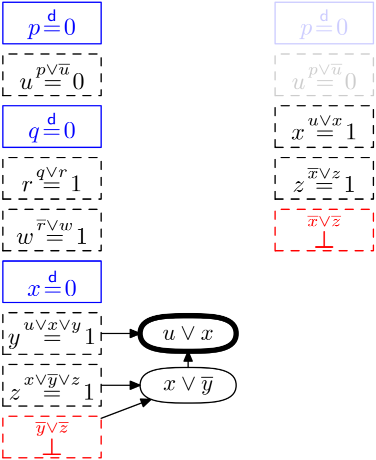
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before...

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

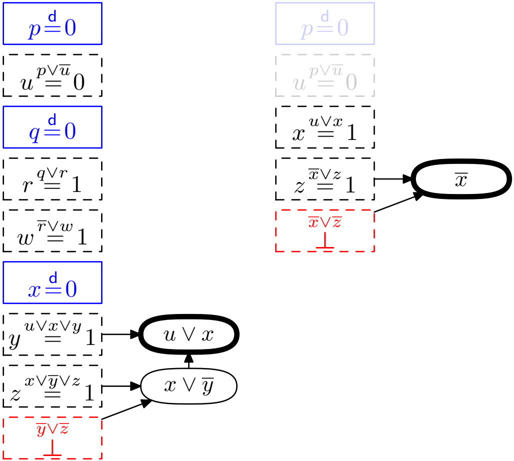
$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

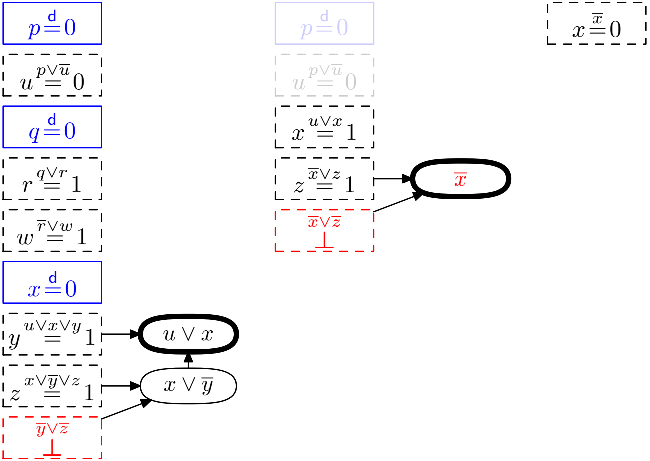
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

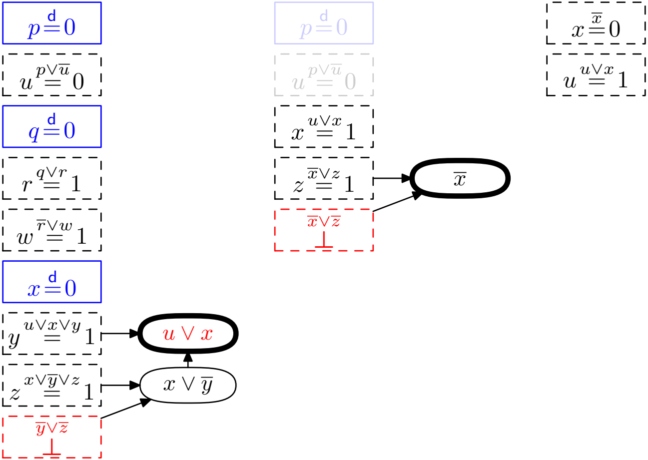
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

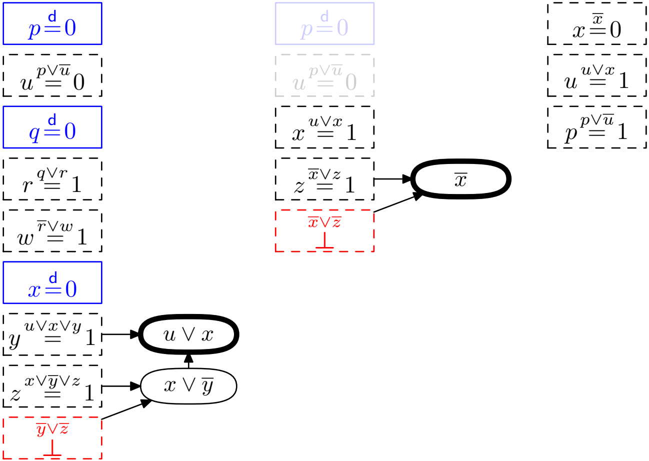
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

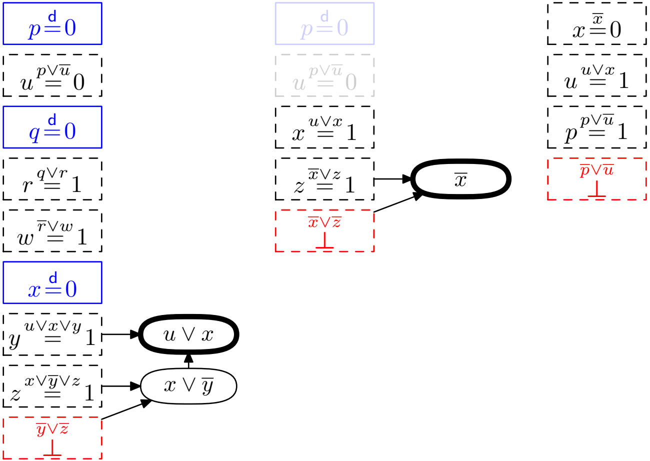
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

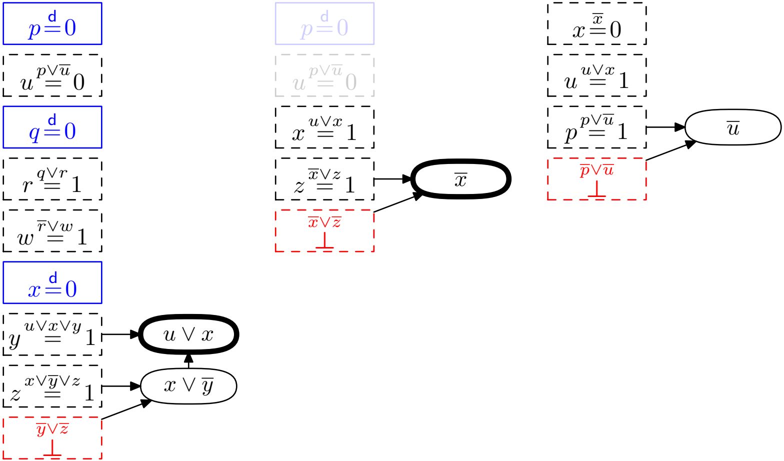
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

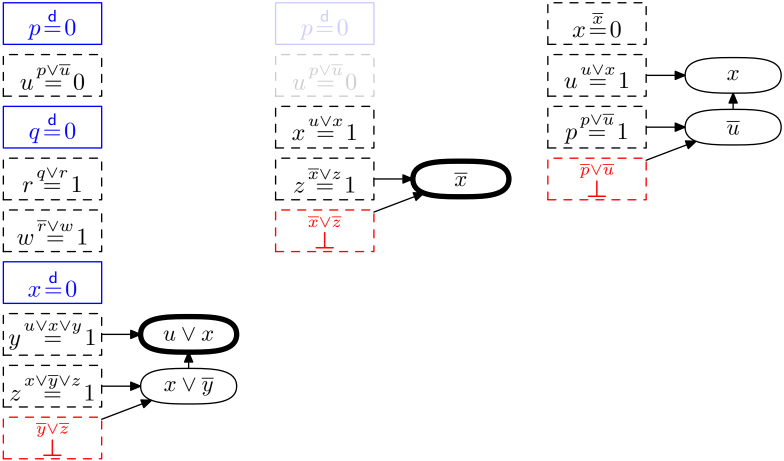
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

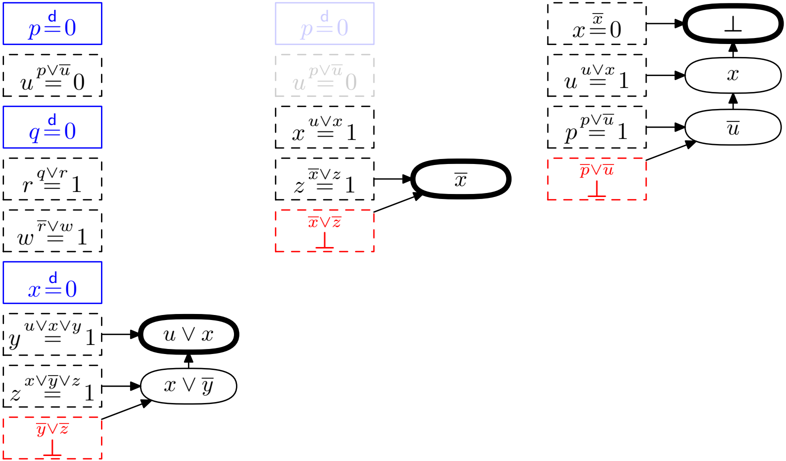
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



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

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

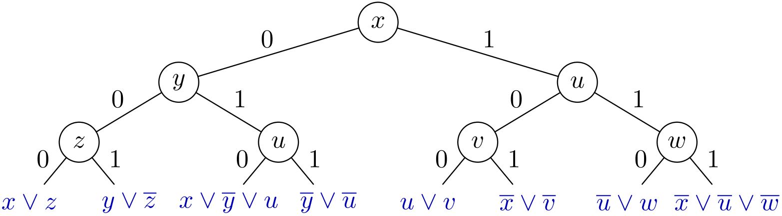
DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

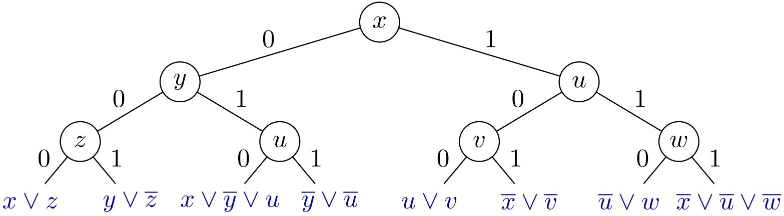
Look at our example again



DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

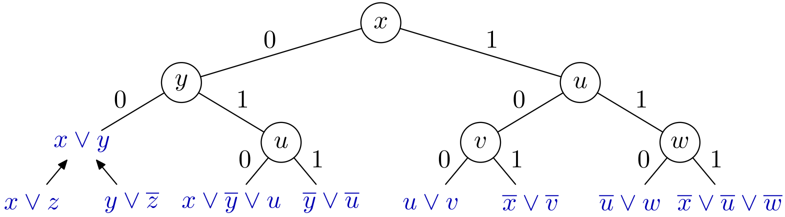


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

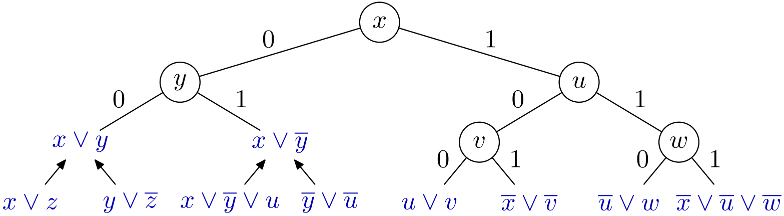


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

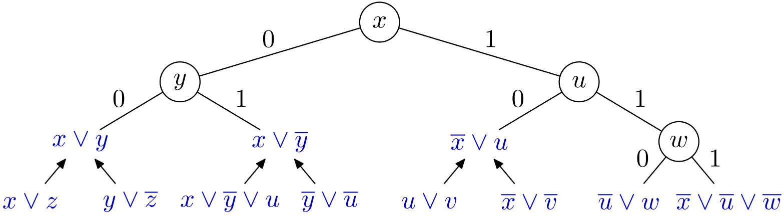


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

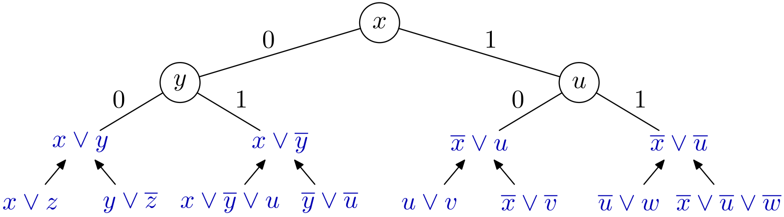


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

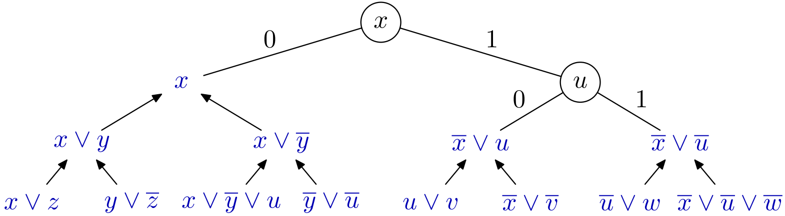


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

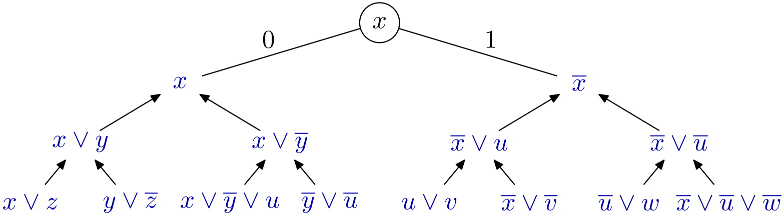


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

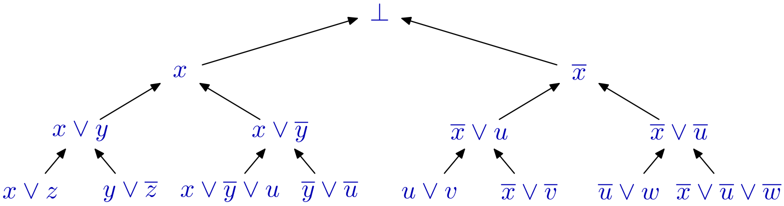


and apply resolution rule $\frac{C_1 \lor x \quad C_2 \lor \overline{x}}{C_1 \lor C_2}$ bottom-up

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 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

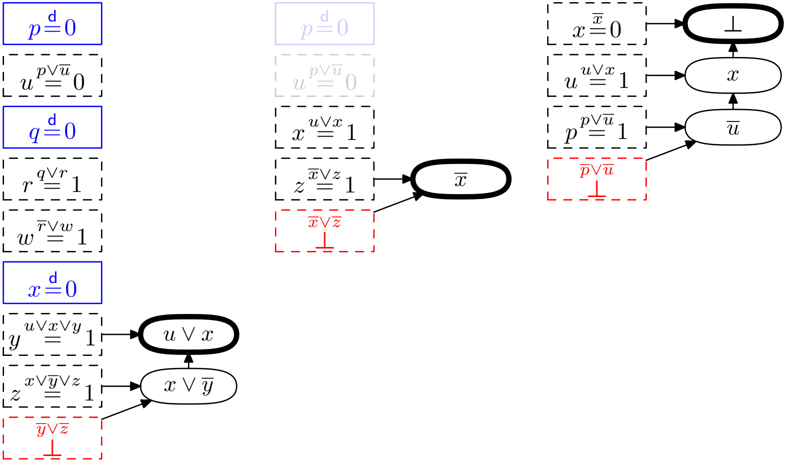
- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution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

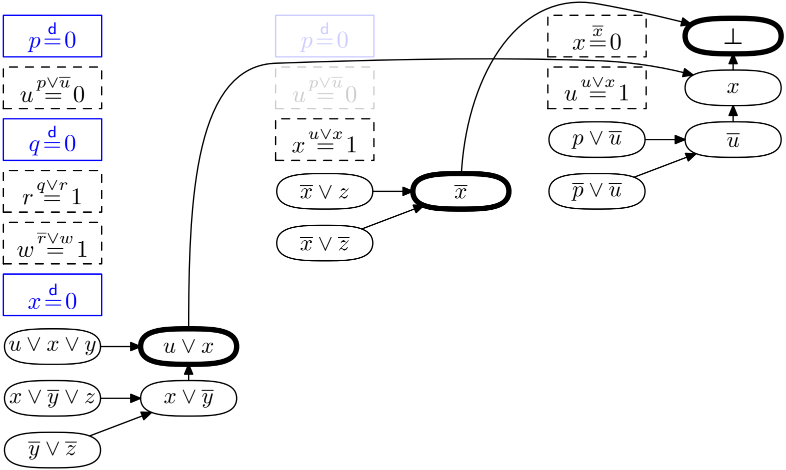
CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution. . .



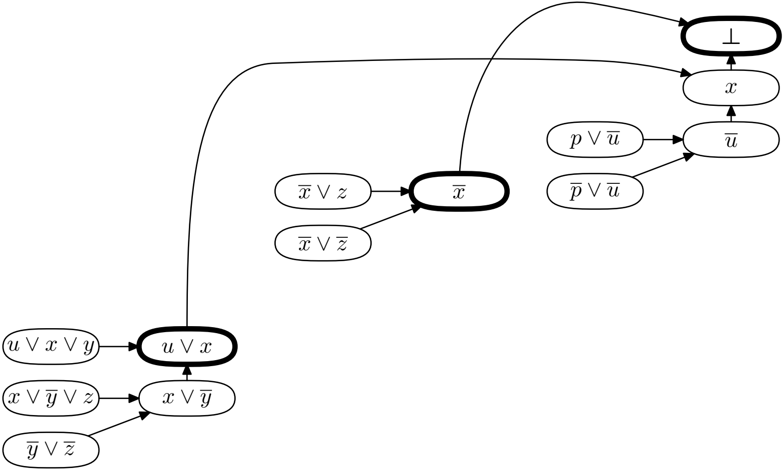
CDCL and Resolution Proofs

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 (DAG-like, not tree-like) from CDCL execution

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, 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 $i \to$ 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"; $1 \leq i \leq n+1$; $1 \leq j \leq n$

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

Can also add "functionality" and "onto" axioms

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

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

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

Examples of Hard Formulas For Resolution (2/3)

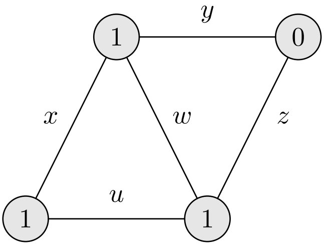
Tseitin formulas [Urq87]
"Sum of degrees of vertices in graph is even"

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



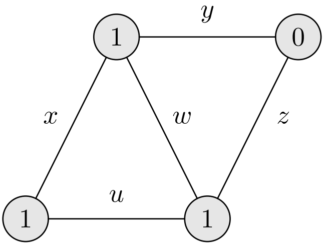
Examples of Hard Formulas For Resolution (2/3)

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



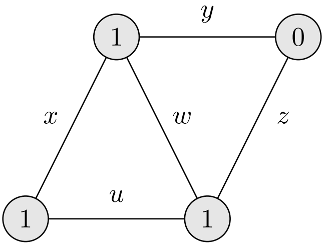
$$
\begin{array}{ll}
(u \lor x) & \land\ (y \lor \overline{z}) \\
\land\ (\overline{u} \lor \overline{x}) & \land\ (\overline{y} \lor z) \\
\land\ (w \lor x \lor y) & \land\ (u \lor w \lor z) \\
\land\ (w \lor \overline{x} \lor \overline{y}) & \land\ (u \lor \overline{w} \lor \overline{z}) \\
\land\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

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]

Δn randomly sampled k-clauses over n variables

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

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

Examples of Hard Formulas for Resolution (3/3)

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

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

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

And more...

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

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

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

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

Hilbert's Nullstellensatz

Consider any system of polynomial equations

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

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

Hilbert's Nullstellensatz

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)

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

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

Nullstellensatz Example (Not Expanded out)

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

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

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)

Polynomial Calculus [CEI96, ABRW02]

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 generated by these polynomials

Polynomial Calculus [CEI96, ABRW02]

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 generated by these polynomials

- Compute polynomials in this ideal $\mathcal{I}$ step by step

Polynomial Calculus [CEI96, ABRW02]

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 generated by these polynomials

- Compute polynomials in this ideal $\mathcal{I}$ step by step
- $p_i \in \mathcal{I}$, $x_j^2 - x_j \in \mathcal{I}$, and $x_j + x_j' - 1 \in \mathcal{I}$ for all $i \in [m]$, $j \in [n]$

Polynomial Calculus [CEI96, ABRW02]

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 generated by these polynomials

- Compute polynomials in this ideal $\mathcal{I}$ step by step
- $p_i \in \mathcal{I}$, $x_j^2 - x_j \in \mathcal{I}$, and $x_j + x_j' - 1 \in \mathcal{I}$ for all $i \in [m]$, $j \in [n]$
- If $p, q \in \mathcal{I}$, then $\alpha p + \beta q \in \mathcal{I}$ for any $\alpha, \beta \in \mathbb{F}$

Polynomial Calculus [CEI96, ABRW02]

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 generated by these polynomials

- Compute polynomials in this ideal $\mathcal{I}$ step by step
- $p_i \in \mathcal{I}$, $x_j^2 - x_j \in \mathcal{I}$, and $x_j + x_j' - 1 \in \mathcal{I}$ for all $i \in [m]$, $j \in [n]$
- If $p, q \in \mathcal{I}$, then $\alpha p + \beta q \in \mathcal{I}$ for any $\alpha, \beta \in \mathbb{F}$
- If $p \in \mathcal{I}$, then $m \cdot p \in \mathcal{I}$ for any monomial $m = \prod_j x_j$

Polynomial Calculus Derivations and Refutations

- A polynomial calculus derivation is a sequence of polynomials in the ideal generated by p_i, $x_j^2 - x_j$, and $x_j + x'_j - 1$
- Derivation rules (from previous slide):
  - Axioms p_i, $x_j^2 - x_j$, and $x_j + x'_j - 1$
  - Linear combination $p, q \Rightarrow \alpha p + \beta q$
  - Monomial multiplication $p \Rightarrow m \cdot p$
- A refutation ends with the polynomial 1
- Complexity measures:
  - Size: total number of monomials in all polynomials in sequence 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

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

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

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

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]

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. $\preceq$)

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

Gröbner Bases: Buchberger's Algorithm

Buchberger's algorithm (**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 (**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

Computes so-called Gröbner basis

Fact: At termination, $1 \in \mathcal{G} \Leftrightarrow$ polynomial equations infeasible

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?!

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?!

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?

What About Algebraic SAT Solvers?

- Excitement about Gröbner basis approach after [CEI96]
- 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]
- 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]
- 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)?
- But very successful work on circuit verification in [KFB20, KB20, KBK20a, KBK20b, KB21, KBBN22]

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

Cutting Planes Derivations and Refutations

- A cutting planes derivation is a sequence of 0-1 integer linear inequalities derived from
  - 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 $\sum c a_i x_i \geq A \Rightarrow \sum a_i x_i \geq \lceil A/c \rceil$
- 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; $1 \leq \ell \leq m-1$, $1 \leq i \leq n$

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

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]

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]

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?

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) $(s_k \in \mathbb{R}[x_1, \ldots, x_n])$

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

SA, SoS, and Other Proof Systems

Sherali-Adams models linear programming (LP) hierarchies

Sums of squares models semidefinite programming (SDP) hierarchies

SA, SoS, and 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 $\mathbb{R}$) while Sherali-Adams and polynomial calculus are incomparable [Ber18]

SA, SoS, and 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 $\mathbb{R}$) while Sherali-Adams and polynomial calculus are incomparable [Ber18]

Sums of squares very strong proof system, except it cannot do parity reasoning efficiently [GV01, Gri01]

Survey [FKP19] is 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 $\mathcal{S} := \mathcal{S} \cup \left\{\sum_i a_i \ell_i \geq A\right\}$
4. Recurse with $\mathcal{S} := \mathcal{S} \cup \left\{\sum_i a_i \ell_i \leq A - 1\right\}$

Complexity measures:

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

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 of representing all coefficients

Cutting planes is simulated efficiently by stabbing planes [BFI+18]

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 of representing all coefficients

Cutting planes is simulated efficiently by stabbing planes [BFI+18]

Stabbing planes with polynomial-size coefficient can be simulated by cutting planes with quasi-polynomial overhead [DT20, FGI+21]

Extended Resolution [Tse68]

Resolution rule

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

Extension rule introducing clauses

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

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

Extended Resolution and SAT Solving

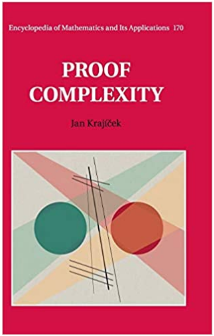
- Closely related (and equivalent) to *DRAT* proof system used to justify correctness of some preprocessing techniques [JHB12]
- *DRAT* also used for SAT solver proof logging
- Attempts to combine extended resolution with CDCL in, e.g., [AKS10, Hua10]
- Without restrictions, corresponds to extremely 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 $\longleftrightarrow$ DPLL and 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 $\longleftrightarrow$ DPLL and CDCL
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali-Adams
- Sums of squares
- Stabbing planes
- Extended resolution

Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ DPLL and CDCL
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali-Adams
- Sums of squares
- Stabbing planes
- Extended resolution

Proof complexity can

- Help analyse state-of-the-art algorithms
- Give ideas for new approaches
- Be a fun playground for theory-practice interaction!

Summing up This Presentation

Overview of some proof systems used in combinatorial solving:

- Resolution $\longleftrightarrow$ DPLL and CDCL
- Nullstellensatz and polynomial calculus $\longleftrightarrow$ Gröbner bases
- Cutting planes $\longleftrightarrow$ pseudo-Boolean solving

Very brief discussion of some other proof systems:

- Sherali-Adams
- Sums of squares
- Stabbing planes
- Extended resolution

Proof complexity can

- Help analyse state-of-the-art algorithms
- Give ideas for new approaches
- Be a fun playground for theory-practice interaction!

Thank you for your attention!

References I

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

[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*.

[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*.

References II

[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.

[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*.

References III

[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 IV

[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.

[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.

References V

[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.

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

References VI

[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.

[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(04):551–582, July 2009.

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

References VII

[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.

[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.

[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.

References VIII

[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.

[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*.

References IX

[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.

[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.

[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.

References X

[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.

[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.

References XI

[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.

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

[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`.

References XII

[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.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

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

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

References XIII

[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*.

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

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

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

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

[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.

References XIV

[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.

[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.