

Proof Systems and Proof Complexity

Marijn J.H. Heule

**Carnegie
Mellon
University**

<http://www.cs.cmu.edu/~mheule/15816-f19/>

Automated Reasoning and Satisfiability, September 24, 2019

Certificates

What makes a problem **hard**?

Certificate angle: can one **efficiently check** an alleged solution?



Consider chess: does white begin and win?

A winning strategy will be very costly to check.

Certificates

What makes a problem **hard**?

Certificate angle: can one **efficiently check** an alleged solution?



Consider chess: does white begin and win?

A winning strategy will be very costly to check.

Consider the sudoku on the right:
Is **searching** for the solution harder
than **verifying** a given solution?

	4	3						
						7	9	
			6					
			1	4		5		
9							1	
2								6
				7	2			
	5					8		
			9					

Certificates

What makes a problem **hard**?

Certificate angle: can one **efficiently check** an alleged solution?



Consider chess: does white begin and win?

A winning strategy will be very costly to check.

Consider the sudoku on the right:
Is **searching** for the solution harder
than **verifying** a given solution?

Intuition: yes!

However, many problems for which
we can efficiently check a solution
turn out to be easy **in practice**.

1	4	7	3	8	9	2	6	5
5	8	6	2	1	4	7	9	3
3	9	2	6	5	7	1	8	4
8	7	3	1	4	6	5	2	9
9	6	4	7	2	5	3	1	8
2	1	5	9	3	8	4	7	6
6	3	8	5	7	2	9	4	1
7	5	9	4	6	1	8	3	2
4	2	1	8	9	3	6	5	7

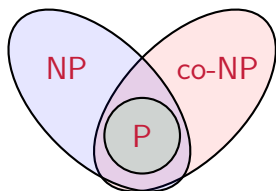
Certificates and Complexity

Complexity classes of decision problems:

P : efficiently computable answers.

NP : efficiently checkable yes-answers.

co-NP : efficiently checkable no-answers.



Cook-Levin Theorem [1971]: SAT is NP-complete.

Solving the $P \stackrel{?}{=} NP$ question is worth \$1,000,000 [Clay MI '00].

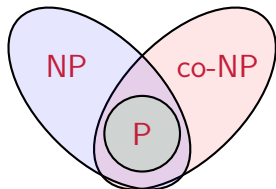
Certificates and Complexity

Complexity classes of decision problems:

P : efficiently computable answers.

NP : efficiently checkable yes-answers.

co-NP : efficiently checkable no-answers.



Cook-Levin Theorem [1971]: SAT is **NP-complete**.

Solving the **$P \stackrel{?}{=} NP$** question is worth **\$1,000,000 [Clay MI '00]**.

The beauty of NP: **guaranteed short** solutions.

The effectiveness of SAT solving: **fast solutions** in practice.

“NP is the new P!”

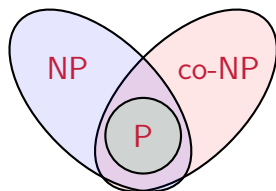
Certificates and Complexity

Complexity classes of decision problems:

P : efficiently computable answers.

NP : efficiently checkable yes-answers.

co-NP : efficiently checkable no-answers.



Cook-Levin Theorem [1971]: SAT is **NP-complete**.

Solving the **$P \stackrel{?}{=} NP$** question is worth **\$1,000,000 [Clay MI '00]**.

The beauty of NP: **guaranteed short** solutions.

The effectiveness of SAT solving: **fast solutions** in practice.

“NP is the new P!”

What about co-NP?

How to find **short** proofs for interesting problems **efficiently**?

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

Certifying Satisfiability and Unsatisfiability

- Certifying **satisfiability** of a formula is easy:

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

Certifying Satisfiability and Unsatisfiability

- Certifying **satisfiability** of a formula is easy:

- Just consider a **satisfying assignment**: $x\bar{y}z$

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

- We can easily check that the assignment is satisfying:
Just check for every clause if it has a satisfied literal!

Certifying Satisfiability and Unsatisfiability

■ Certifying **satisfiability** of a formula is easy:

- Just consider a **satisfying assignment**: $x\bar{y}z$

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

- We can easily check that the assignment is satisfying:
Just check for every clause if it has a satisfied literal!

■ Certifying **unsatisfiability** is not so easy:

- If a formula has n variables, there are 2^n possible assignments.
- ➔ Checking whether **every** assignment falsifies the formula is **costly**.
- More compact certificates of unsatisfiability are desirable.
➔ **Proofs**

What Is a Proof in SAT?

- In general, a **proof** is a **string** that **certifies the unsatisfiability** of a formula.
 - Proofs are **efficiently** (usually **polynomial-time**) **checkable**...

What Is a Proof in SAT?

- In general, a **proof** is a **string** that **certifies the unsatisfiability** of a formula.
 - Proofs are **efficiently** (usually **polynomial-time**) **checkable**...
... but can be of exponential size with respect to a formula.

What Is a Proof in SAT?

- In general, a **proof** is a **string** that **certifies the unsatisfiability** of a formula.
 - Proofs are **efficiently** (usually **polynomial-time**) **checkable**...
... but can be of exponential size with respect to a formula.
- **Example:** Resolution proofs
 - A **resolution proof** is a sequence C_1, \dots, C_m of clauses.
 - Every clause is either contained in the formula or derived from two earlier clauses via the **resolution rule**:

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D}$$

- C_m is the **empty clause** (containing no literals), denoted by \perp .
- There exists a resolution proof for every unsatisfiable formula.

Resolution Proofs

■ **Example:** $F = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$

■ **Resolution proof:**

$(\bar{x} \vee \bar{y} \vee z), (\bar{z}), (\bar{x} \vee \bar{y}), (x \vee \bar{y}), (\bar{y}), (\bar{u} \vee y), (\bar{u}), (u), \perp$

Resolution Proofs

■ **Example:** $F = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$

■ **Resolution proof:**

$(\bar{x} \vee \bar{y} \vee z), (\bar{z}), (\bar{x} \vee \bar{y}), (x \vee \bar{y}), (\bar{y}), (\bar{u} \vee y), (\bar{u}), (u), \perp$

$$\frac{\frac{\frac{\bar{x} \vee \bar{y} \vee z \quad \bar{z}}{\bar{x} \vee \bar{y}} \quad x \vee \bar{y}}{\bar{y}}}{\bar{u} \vee y} \quad \bar{u} \quad u}{\perp}$$

Resolution Proofs

■ **Example:** $F = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$

■ **Resolution proof:**

$(\bar{x} \vee \bar{y} \vee z), (\bar{z}), (\bar{x} \vee \bar{y}), (x \vee \bar{y}), (\bar{y}), (\bar{u} \vee y), (\bar{u}), (u), \perp$

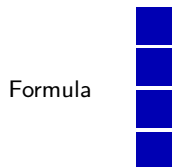
$$\frac{\frac{\frac{\bar{x} \vee \bar{y} \vee z \quad \bar{z}}{\bar{x} \vee \bar{y}} \quad x \vee \bar{y}}{\bar{y}} \quad \bar{u} \vee y}{\bar{u}} \quad u}{\perp}$$

■ **Drawbacks** of resolution:

- For **many** seemingly simple formulas, there are **only** resolution proofs of **exponential size**.
- **State-of-the-art solving techniques** are **not succinctly expressible**.

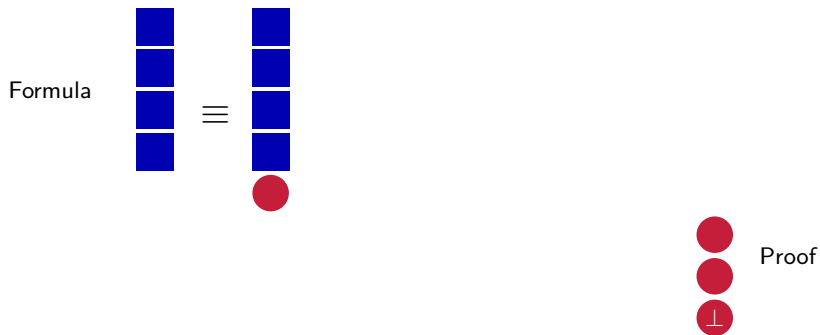
Clausal Proofs

Reduce the size of the proof by only storing added clauses



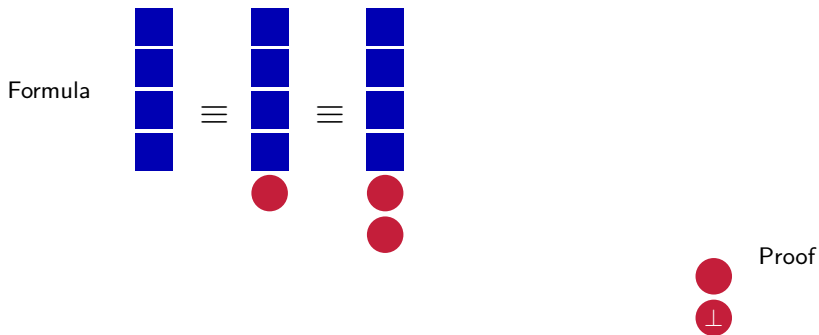
Clausal Proofs

Reduce the size of the proof by only storing added clauses



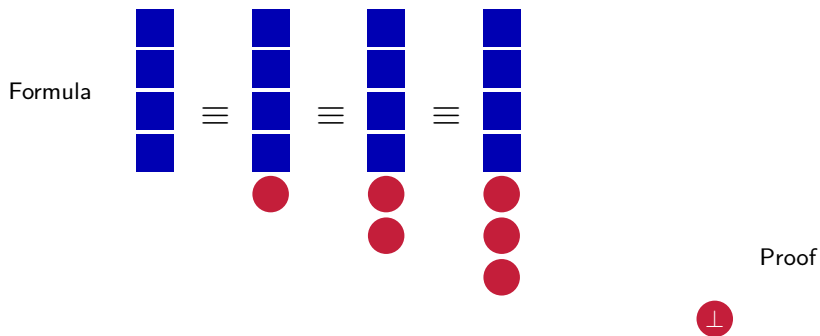
Clausal Proofs

Reduce the size of the proof by only storing added clauses



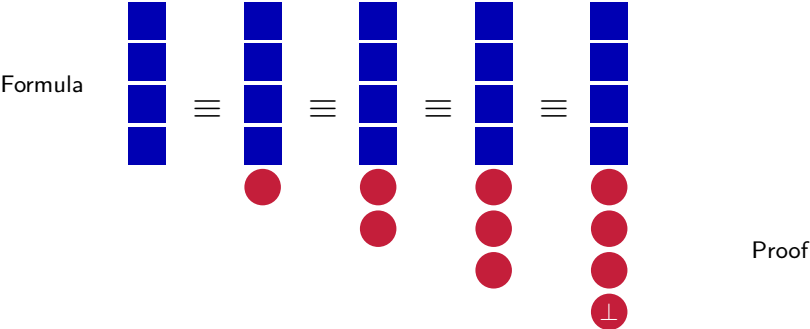
Clausal Proofs

Reduce the size of the proof by only storing added clauses



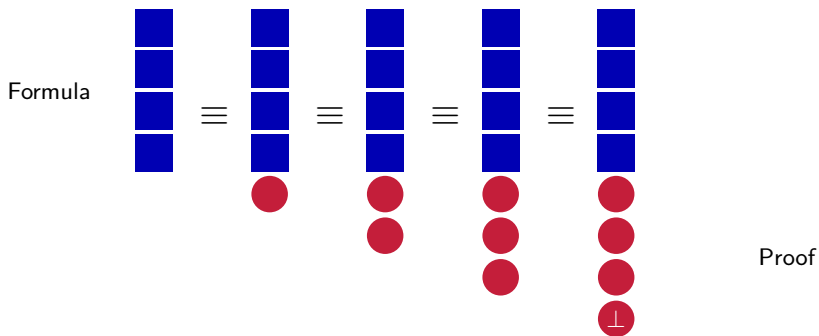
Clausal Proofs

Reduce the size of the proof by only storing added clauses



Clausal Proofs

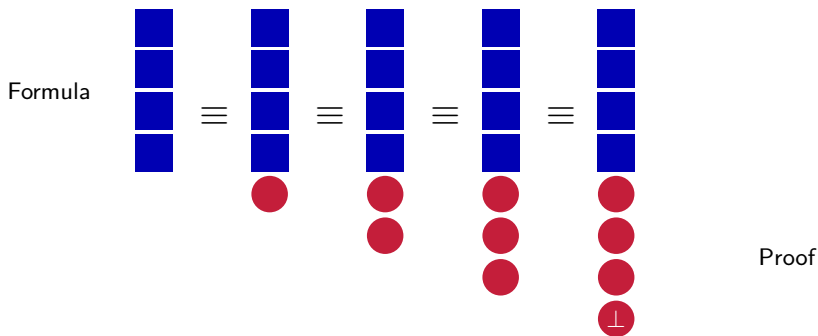
Reduce the size of the proof by only storing added clauses



- Checking whether additions preserve satisfiability should be **efficient**.
- Clauses whose addition preserves satisfiability are called **redundant**.

Clausal Proofs

Reduce the size of the proof by only storing added clauses



- Checking whether additions preserve satisfiability should be **efficient**.
- Clauses whose addition preserves satisfiability are called **redundant**.

➔ **Idea:** Allow only the addition of clauses that fulfill an **efficiently checkable redundancy criterion**.

Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.

Example

$$F = (a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (b \vee c \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee d) \wedge (a \vee c \vee d) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee b \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.

Example

$$F = (a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (b \vee c \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee d) \wedge \\ (a \vee c \vee d) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee b \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

$$\alpha = \{a = 0, b = 0\}$$

Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_{\perp} C$) if UP on $F|_{\alpha}$ results in a conflict.

Example

$$F = (a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (b \vee c \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee d) \wedge \\ (a \vee c \vee d) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee b \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

$$\alpha = \{a = 0, b = 0, c = 0\}$$

Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.

Example

$$F = (a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (b \vee c \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee d) \wedge \\ (a \vee c \vee d) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee b \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

$$\alpha = \{a = 0, b = 0, c = 0, d = 0\}$$

Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.

Example

$$F = (a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee c) \wedge (b \vee c \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee d) \wedge (a \vee c \vee d) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee b \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

$$\alpha = \{a = 0, b = 0, c = 0, d = 0\}$$

$$\frac{\frac{(a \vee c \vee d) \quad (b \vee c \vee \bar{d})}{(a \vee b \vee c)} \quad (a \vee b \vee \bar{c})}{(a \vee b)}$$

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

Early work on reasoning beyond resolution

The early SAT decision procedures used the **Pure Literal rule** [Davis and Putnam 1960; Davis, Logemann and Loveland 1962]:

$$\frac{\bar{x} \notin F}{(x)} \text{ (pure)}$$

Early work on reasoning beyond resolution

The early SAT decision procedures used the **Pure Literal rule** [Davis and Putnam 1960; Davis, Logemann and Loveland 1962]:

$$\frac{\bar{x} \notin F}{(x)} \text{ (pure)}$$

Extended Resolution (ER) [Tseitin 1966]

- Combines resolution with the **Extension rule**:

$$\frac{x \notin F \quad \bar{x} \notin F}{(x \vee \bar{a} \vee \bar{b}) \wedge (\bar{x} \vee a) \wedge (\bar{x} \vee b)} \text{ (ER)}$$

- Equivalently, adds the definition $x := \text{AND}(a, b)$
- Can be considered the **first interference-based proof system**
- Is very powerful: **No known lower bounds**

Short Proofs of Pigeon Hole Formulas [Cook 1967]

Can $n+1$ pigeons be placed in n holes (at-most-one pigeon per hole)?

$$PHP_n := \bigwedge_{1 \leq p \leq n+1} (x_{1,p} \vee \dots \vee x_{n,p}) \wedge \bigwedge_{1 \leq h \leq n} \bigwedge_{1 \leq p < q \leq n+1} (\bar{x}_{h,p} \vee \bar{x}_{h,q})$$

Resolution proofs of PHP_n formulas are **exponential** [Haken 1985]

Cook constructed **polynomial-sized** ER proofs of PHP_n formulas

Short Proofs of Pigeon Hole Formulas [Cook 1967]

Can $n+1$ pigeons be placed in n holes (at-most-one pigeon per hole)?

$$PHP_n := \bigwedge_{1 \leq p \leq n+1} (x_{1,p} \vee \dots \vee x_{n,p}) \wedge \bigwedge_{1 \leq h \leq n} \bigwedge_{1 \leq p < q \leq n+1} (\bar{x}_{h,p} \vee \bar{x}_{h,q})$$

Resolution proofs of PHP_n formulas are **exponential** [Haken 1985]

Cook constructed **polynomial-sized** ER proofs of PHP_n formulas

However, these proofs require introducing new variables:

- Hard to find such proofs automatically
- Existing ER approaches produce exponentially large proofs
- How to get rid of this hurdle? First approach: blocked clauses...

Traditional Proofs vs. Interference-Based Proofs

- In **traditional** proof systems, everything that is **inferred**, is **logically implied** by the premises.

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D} \text{ (RES)} \qquad \frac{A \quad A \rightarrow B}{B} \text{ (MP)}$$

Traditional Proofs vs. Interference-Based Proofs

- In **traditional** proof systems, everything that is **inferred**, is **logically implied** by the premises.

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D} \text{ (RES)} \qquad \frac{A \quad A \rightarrow B}{B} \text{ (MP)}$$

- ➡ Inference rules reason about the **presence** of facts.
- If certain premises are present, infer the conclusion.

Traditional Proofs vs. Interference-Based Proofs

- In **traditional** proof systems, everything that is **inferred**, is **logically implied** by the premises.

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D} \text{ (RES)} \qquad \frac{A \quad A \rightarrow B}{B} \text{ (MP)}$$

- ➔ Inference rules reason about the **presence** of facts.
 - If certain premises are present, infer the conclusion.
- **Different approach**: Allow **not only implied conclusions**.
 - **Require only** that the addition of facts preserves **satisfiability**.
 - Reason also about the **absence** of facts.
- ➔ This leads to **interference-based proof systems**.

Blocked Clauses [Kullmann 1999]

Definition (Blocking literal)

A literal x blocks clause $(C \vee x)$ w.r.t. a CNF formula F if for every clause $(D \vee \bar{x}) \in F$, the resolvent $C \vee D$ is a tautology.

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Blocked Clauses [Kullmann 1999]

Definition (Blocking literal)

A literal x blocks clause $(C \vee x)$ w.r.t. a CNF formula F if for every clause $(D \vee \bar{x}) \in F$, the resolvent $C \vee D$ is a tautology.

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.

First clause is not blocked.

Second clause is blocked by both a and \bar{c} .

Third clause is blocked by c

Theorem

Adding or removing a blocked clause preserves satisfiability.

Blocked Clause Addition and Blocked Clause Elimination

The Blocked Clause proof system (BC) combines the resolution rule with the addition of blocked clauses.

- BC generalizes ER [Kullmann 1999]

- Recall

$$\frac{x \notin F \quad \bar{x} \notin F}{(x \vee \bar{a} \vee \bar{b}) \wedge (\bar{x} \vee a) \wedge (\bar{x} \vee b)} \text{ (ER)}$$

- The ER clauses are blocked on the literals x and \bar{x} w.r.t. F

Blocked Clause Addition and Blocked Clause Elimination

The Blocked Clause proof system (BC) combines the resolution rule with the addition of blocked clauses.

- BC generalizes ER [Kullmann 1999]

- Recall

$$\frac{x \notin F \quad \bar{x} \notin F}{(x \vee \bar{a} \vee \bar{b}) \wedge (\bar{x} \vee a) \wedge (\bar{x} \vee b)} \text{ (ER)}$$

- The ER clauses are blocked on the literals x and \bar{x} w.r.t. F

Blocked clause elimination used in preprocessing and inprocessing

- Simulates many circuit optimization techniques
- Removes redundant Pythagorean Triples

DRAT: An Interference-Based Proof System

- Popular **example** of an interference-based proof system: **DRAT**
- DRAT allows the addition of **RATs** (defined below) to a formula.
 - It can be **efficiently checked** if a clause is a RAT.
 - RATs are **not necessarily implied** by the formula.
 - But RATs are redundant: their **addition preserves satisfiability**.
- DRAT also allows clause **deletion**
 - Initially introduced to check proofs **more efficiently**
 - Clause deletion may **introduce clause addition** options (interference)

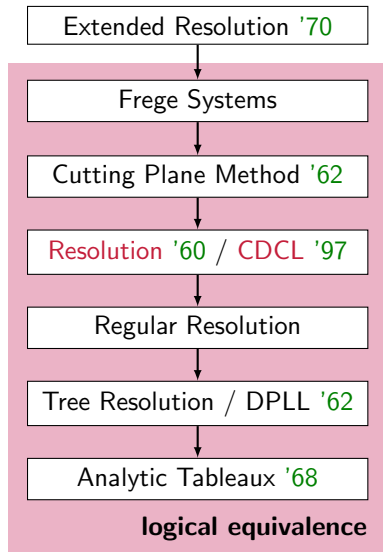
DRAT: An Interference-Based Proof System

- Popular **example** of an interference-based proof system: **DRAT**
- DRAT allows the addition of **RATs** (defined below) to a formula.
 - It can be **efficiently checked** if a clause is a RAT.
 - RATs are **not necessarily implied** by the formula.
 - But RATs are redundant: their **addition preserves satisfiability**.
- DRAT also allows clause **deletion**
 - Initially introduced to check proofs **more efficiently**
 - Clause deletion may **introduce clause addition** options (interference)

A clause $(C \vee x)$ is a **resolution asymmetric tautology** (RAT) on x w.r.t. a CNF formula F if for every clause $(D \vee \bar{x}) \in F$, the resolvent $C \vee D$ is **implied by F via unit-propagation**, i.e., $F \vdash_1 C \vee D$.

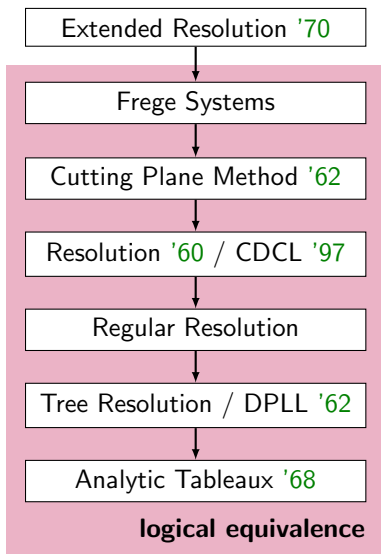
Proof Search in Strong Proof Systems

Existence of Short Proofs

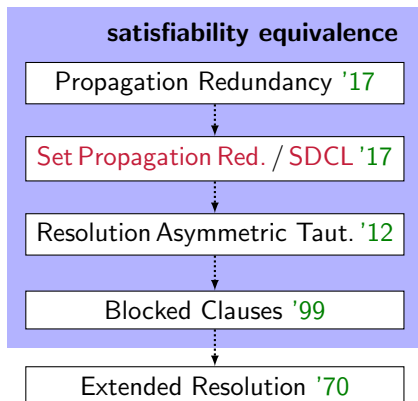


Proof Search in Strong Proof Systems

Existence of Short Proofs



Finding Short Proofs



Express solving techniques compactly
[Järvisalo, Heule, and Biere '12]
Short proofs without new variables
[Heule, Kiesel, and Biere '17A]

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

Redundant Clauses

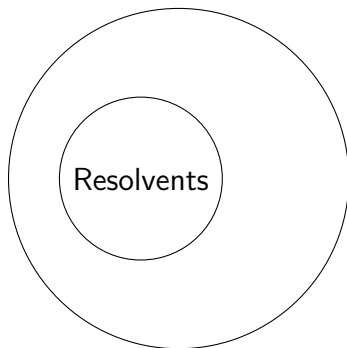
- Strong proof systems allow addition of **many redundant clauses**.



All Redundant Clauses

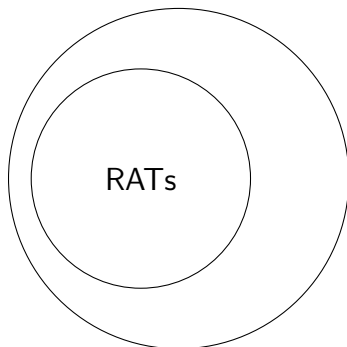
Redundant Clauses

- Strong proof systems allow addition of **many redundant clauses**.



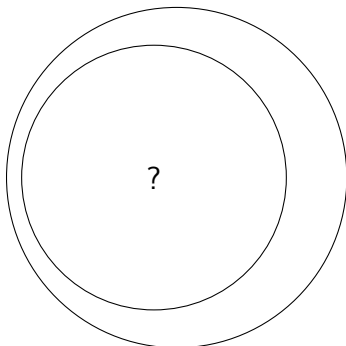
Redundant Clauses

- Strong proof systems allow addition of **many redundant clauses**.



Redundant Clauses

- Strong proof systems allow addition of **many redundant clauses**.

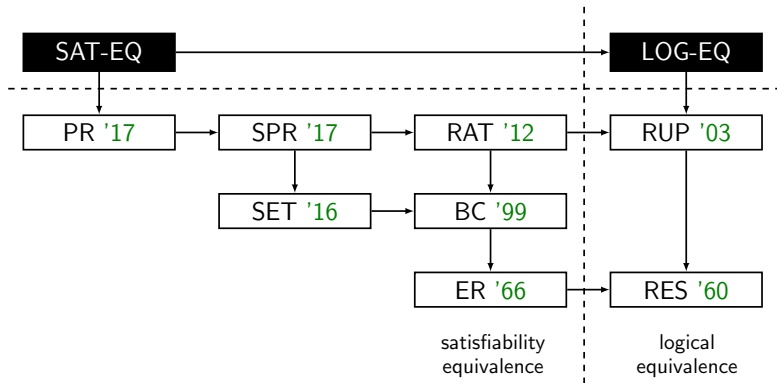


- Are **stronger** redundancy notions still **efficiently checkable**?

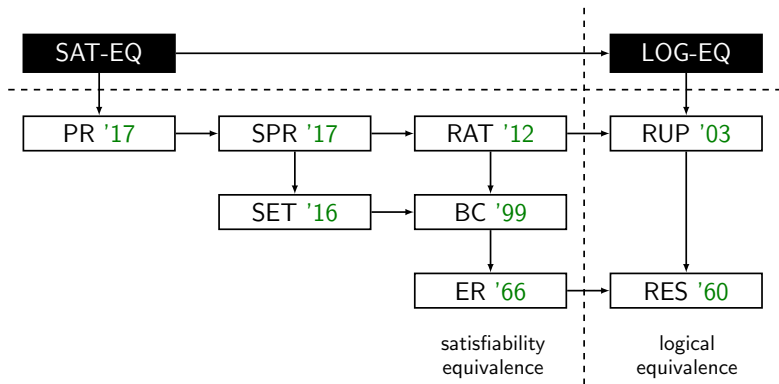
New Propositional Proof Systems

- We introduced **new clause-redundancy notions**:
 - Propagation-redundant (PR) clauses
 - Set-propagation-redundant (SPR) clauses
 - Literal-propagation-redundant (LPR) clauses
- LPR clauses coincide with RAT.
- SPR clauses strictly generalize RATs.
- PR clauses strictly generalize SPR clauses.
- The redundancy notions provide the basis for **new proof systems**.

New Proof Systems for Propositional Logic



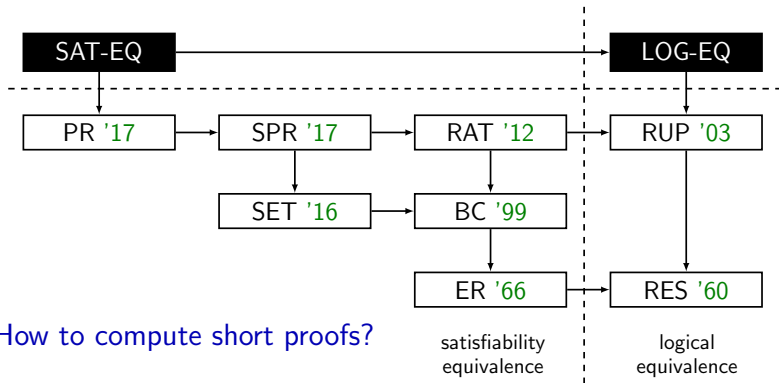
New Proof Systems for Propositional Logic



RAT simulates PR [Heule and Biere 2018]

ER simulates RAT [Kiesl, Rebola-Pardo, Heule 2018]

New Proof Systems for Propositional Logic



How to compute short proofs?

RAT simulates PR [Heule and Biere 2018]

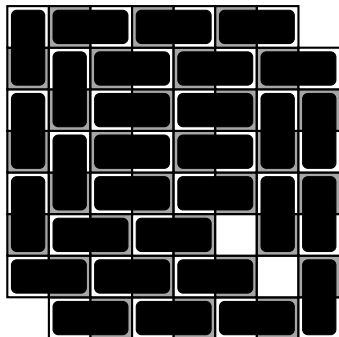
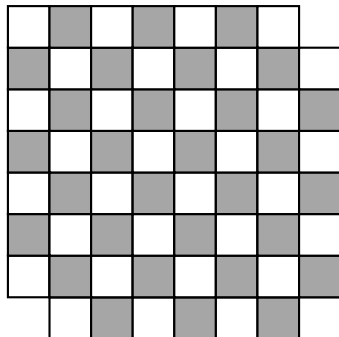
ER simulates RAT [Kiesl, Rebola-Pardo, Heule 2018]

Stronger Proof Systems: What Are They Good For?

- The new proof systems can give **short proofs** of formulas that are considered **hard**.
- We have **short SPR and PR proofs** for the well-known **pigeon hole formulas** (linear in the size of the input).
 - Pigeon hole formulas have **only exponential-size resolution proofs**.
 - If the **addition of new variables via definitions** is allowed, there are polynomial-size proofs.
- Strong proof systems do **not** require new variables.
 - ➔ Search space of possible clauses is **finite**.
 - ➔ Makes **search** for such clauses **easier**.

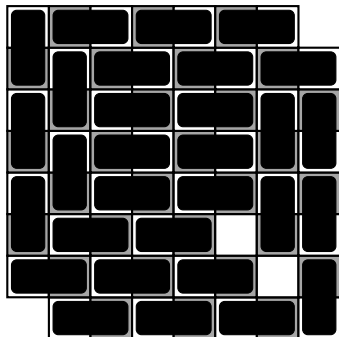
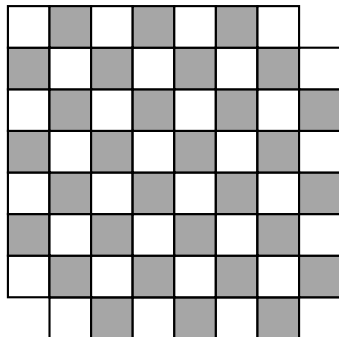
Mutilated Chessboards: “A Tough Nut to Crack” [McCarthy]

Can a chessboard be fully covered with dominos after removing two diagonally opposite corner squares?



Mutilated Chessboards: “A Tough Nut to Crack” [McCarthy]

Can a chessboard be fully covered with dominos after removing two diagonally opposite corner squares?

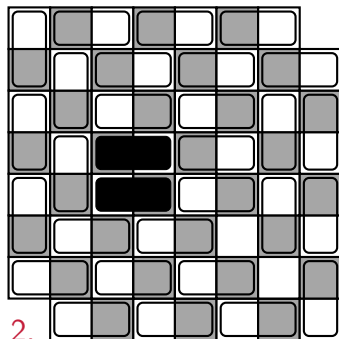
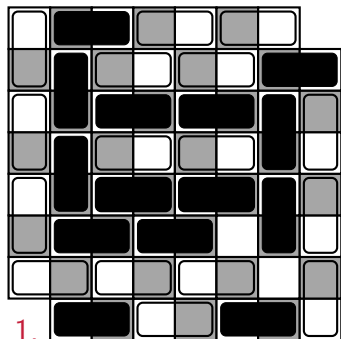


Easy to refute based on the following two observations:

- There are more white squares than black squares; and
- A domino covers exactly one white and one black square.

Without Loss of Satisfaction

One of the crucial techniques in SAT solvers is to **generalize a conflicting state** and use it to constrain the problem.



The used proof system can have a big impact on the size:

1. Resolution can only reduce the 30 dominos to 14 (left); and
2. “Without loss of satisfaction” can reduce them to 2 (right).

Redundancy as an Implication

A formula G is **at least as satisfiable** as a formula F if $F \models G$.

Given a formula F and assignment α , we denote with $F|_{\alpha}$ the **reduced formula** after removing from F all clauses satisfied by α and all literals falsified by α .

Theorem

*Let F be a formula, C a clause, and α the smallest assignment that falsifies C . Then, C is **redundant** w.r.t. F iff there exists an assignment ω such that 1) ω satisfies C ; and 2) $F|_{\alpha} \models F|_{\omega}$.*

This is the **strongest notion** of redundancy. However, it cannot be checked in polynomial time (assuming $P \neq NP$), unless **bounded**.

Checking Redundancy Using Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_1 C$) if UP on $F|\alpha$ results in a conflict.
- Implied by UP is used in SAT solvers to determine redundancy of learned clauses and therefore \vdash_1 is a **natural restriction** of \models .
- We bound $F|\alpha \models F|\omega$ by $F|\alpha \vdash_1 F|\omega$.
- **Example:** $F = (x \vee y \vee z) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee z)$ and $G = (z)$. Observe that $F \models G$, but that $F \not\vdash_1 G$.

Hand-crafted PR Proofs of Pigeon Hole Formulas

We manually constructed PR proofs of the famous pigeon hole formulas and the two-pigeons-per-hole family.

- The proofs consist only of **binary and unit** clauses.
- Only **original variables** appear in the proof.
- All proofs are **linear** in the size of the formula.
- ➔ The PR proofs are smaller than Cook's **ER proofs**.
- All resolution proofs of these formulas are **exponential** in size.

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

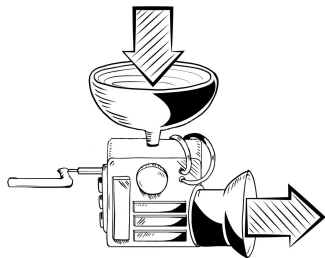
Challenges

Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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



Satisfiability Solving (Highly Simplified)

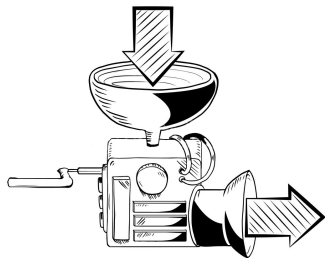
SAT problem:

Given a propositional formula, is it satisfiable?

Input Formula in CNF



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

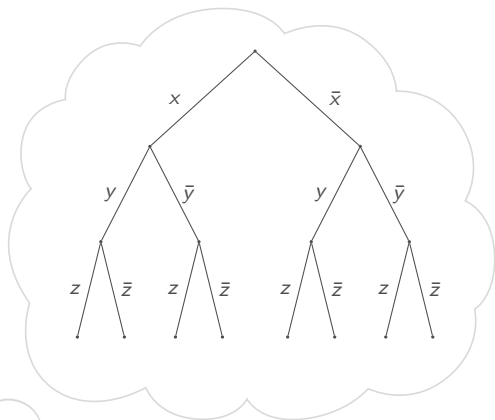
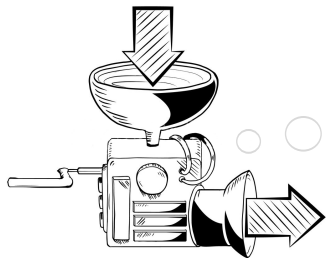


Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

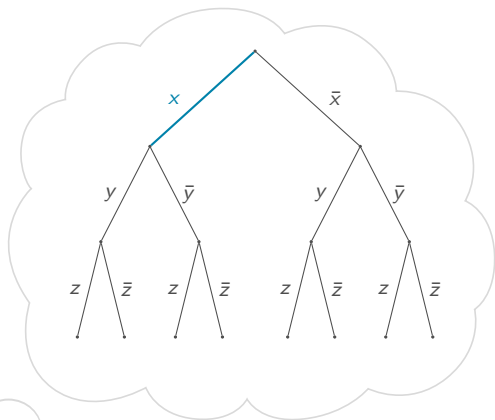
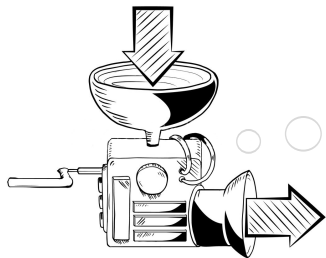


Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

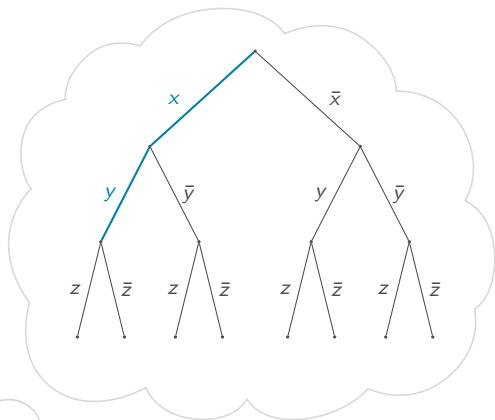
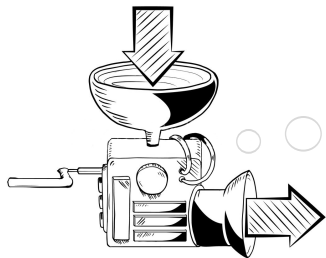


Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

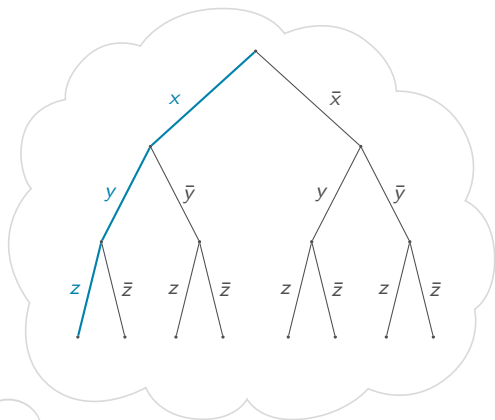
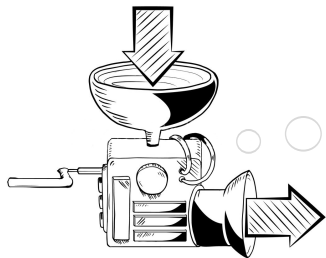


Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

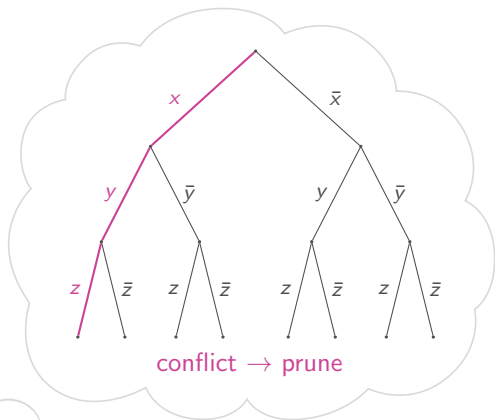
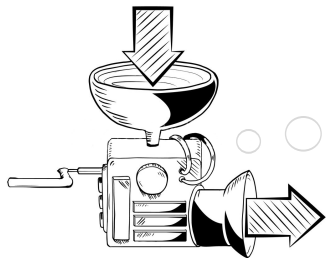


Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

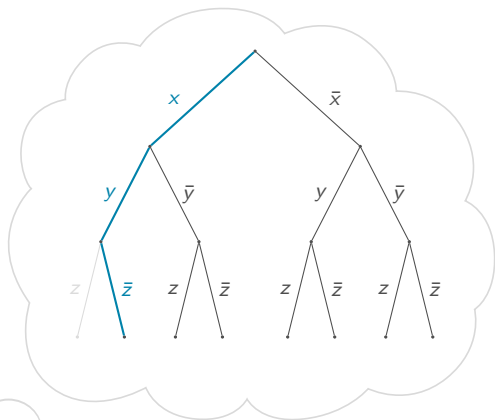
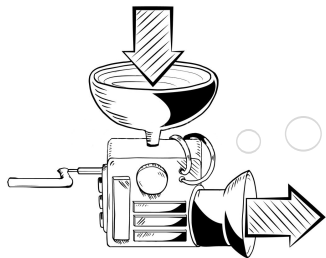


Satisfiability Solving (Highly Simplified)

SAT problem:

Given a propositional formula, is it satisfiable?

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

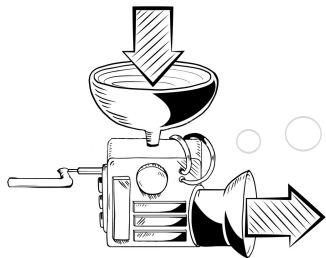


Satisfiability Solving (Highly Simplified)

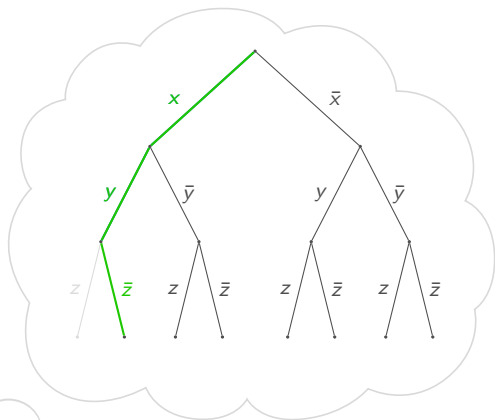
SAT problem:

Given a propositional formula, is it satisfiable?

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



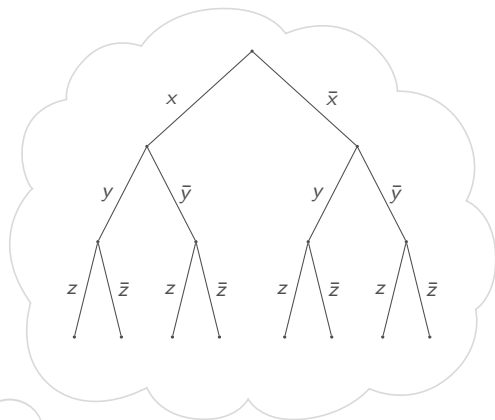
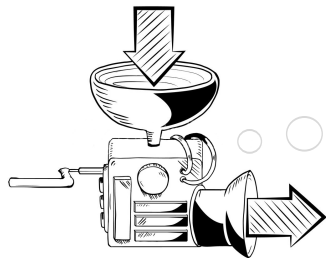
Satisfiable



Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?
Even satisfiable branches?

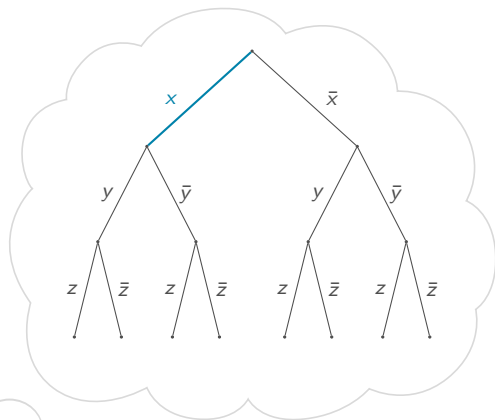
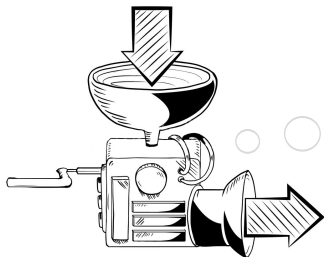
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?
Even satisfiable branches?

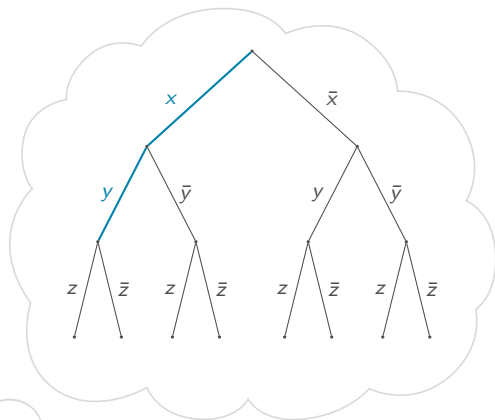
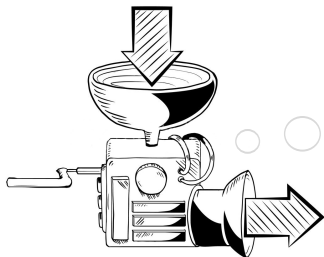
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?
Even satisfiable branches?

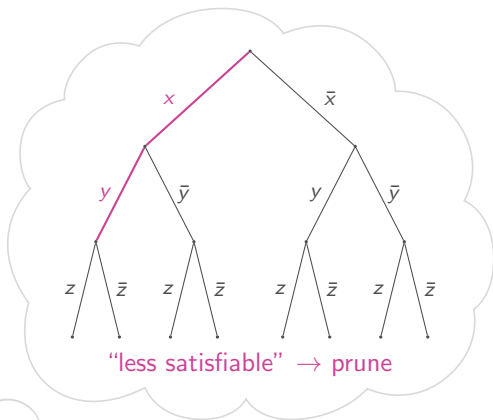
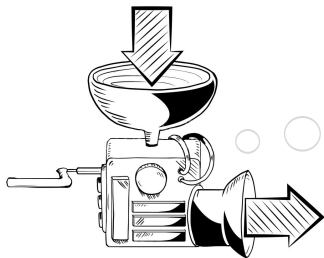
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?
Even satisfiable branches?

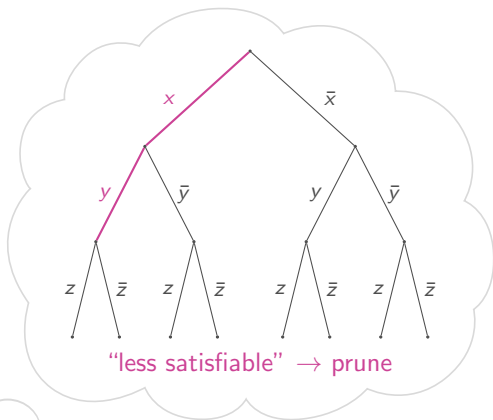
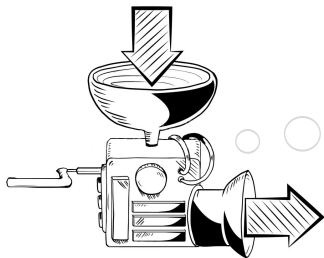
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



Key Idea: Prune Less Satisfiable Branches

Can we **prune** earlier?
Even satisfiable branches?

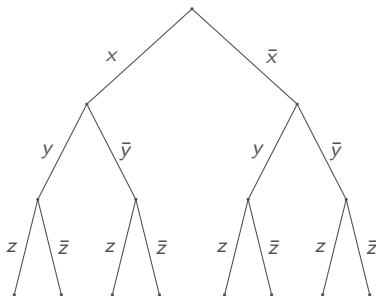
$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$



How to prune? Add **redundant** clauses!

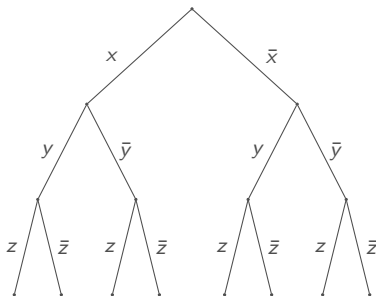
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.



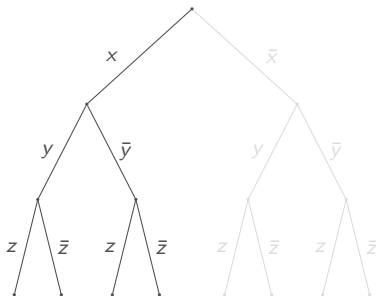
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.



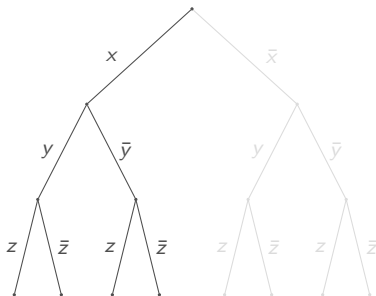
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.



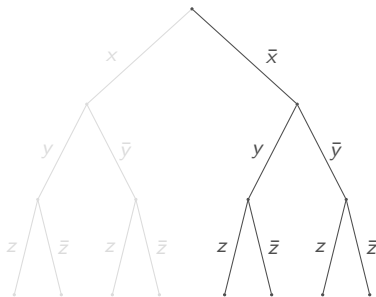
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.
- **Other Examples:**



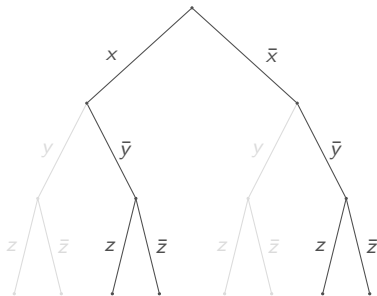
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.
- **Other Examples:** (\bar{x})



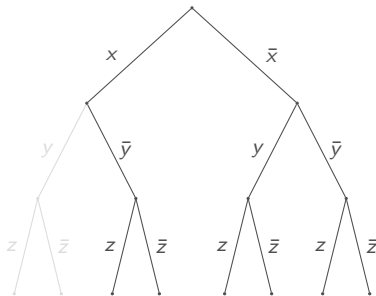
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.
- **Other Examples:** (\bar{x}) (\bar{y})



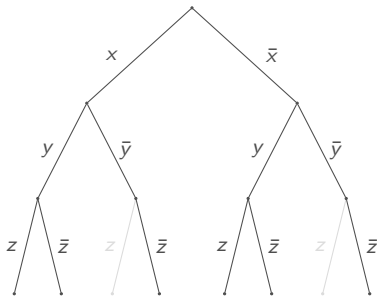
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.
- **Other Examples:** (\bar{x}) (\bar{y}) $(\bar{x} \vee \bar{y})$



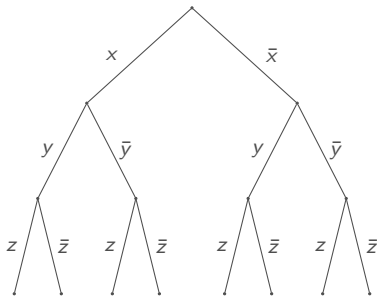
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.
- **Other Examples:** (\bar{x}) (\bar{y}) $(\bar{x} \vee \bar{y})$ $(y \vee \bar{z})$



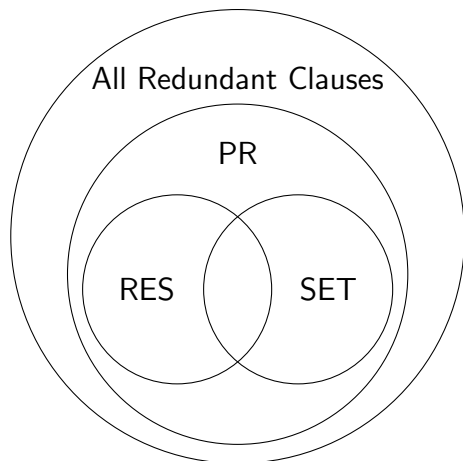
Pruning via Clause Addition

- A clause prunes all branches that falsify the clause.
- **Example:** The clause (x) prunes all branches where x is false.
- **Other Examples:** (\bar{x}) (\bar{y}) $(\bar{x} \vee \bar{y})$ $(y \vee \bar{z})$ $(x \vee \bar{x})$



Redundant Clauses

A clause C is **redundant** w.r.t. a formula F if and only if F and $F \wedge C$ are either both satisfiable or both unsatisfiable.



PR = Propagation Redundant Clauses [CADE'17]

RES = Resolvents

SET = Set-Blocked Clauses [IJCAR'16]

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in **SAT**!

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in **SAT**!

Given a formula F and a clause C . Let α denote the smallest assignment that falsifies C . The **positive reduct** of F and α is a formula which is satisfiable if and only if C is SET w.r.t. F .

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in **SAT**!

Given a formula F and a clause C . Let α denote the smallest assignment that falsifies C . The **positive reduct** of F and α is a formula which is satisfiable if and only if C is SET w.r.t. F .

Positive reducts are typically very easy to solve!

Finding Redundant Clauses: The Positive Reduct

Determining whether a clause C is SET or PR w.r.t. a formula F is an NP-complete problem.

How to find SET and PR clauses? Encode it in **SAT**!

Given a formula F and a clause C . Let α denote the smallest assignment that falsifies C . The **positive reduct** of F and α is a formula which is satisfiable if and only if C is SET w.r.t. F .

Positive reducts are typically very easy to solve!

Key Idea: While solving a formula F , check whether the positive reduct of F and the current assignment α is **satisfiable**. In that case, **prune** the branch α .

The Positive Reduct: An Example

Given a formula F and a clause C . Let α denote the smallest assignment that falsifies C . The **positive reduct** of F and α , denoted by $p(F, \alpha)$, is the formula that contains C and all *assigned*(D, α) with $D \in F$ and D is satisfied by α .

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Let $C_1 = (\bar{x})$, so $\alpha_1 = x$.

The positive reduct $p(F, \alpha_1) = (\bar{x}) \wedge (x) \wedge (x)$ is **unsatisfiable**.

Let $C_2 = (\bar{x} \vee \bar{y})$, so $\alpha_2 = x y$.

The positive reduct $p(F, \alpha_2) = (\bar{x} \vee \bar{y}) \wedge (x \vee y) \wedge (x \vee \bar{y})$ is **satisfiable**.

Autarkies

A non-empty assignment α is an **autarky** for formula F if every clause $C \in F$ that is **touched** by α is also **satisfied** by α .

A **pure literal** and a **satisfying assignment** are autarkies.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Assignment $\alpha_1 = \bar{z}$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Assignment $\alpha_2 = x \bar{y} z$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Autarkies

A non-empty assignment α is an **autarky** for formula F if every clause $C \in F$ that is **touched** by α is also **satisfied** by α .

A **pure literal** and a **satisfying assignment** are autarkies.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Assignment $\alpha_1 = \bar{z}$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Assignment $\alpha_2 = x \bar{y} z$ is an autarky: $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Given an assignment α , $F|_{\alpha}$ denotes a formula F without the clauses satisfied by α and without the literals falsified by α .

Theorem ([Monien and Speckenmeyer 1985])

Let α be an autarky for formula F .

Then, F and $F|_{\alpha}$ are satisfiability equivalent.

Conditional Autarkies

An assignment $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}}$ is a **conditional autarky** for formula F if α_{aut} is an autarky for $F|_{\alpha_{\text{con}}}$.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.

Let $\alpha_{\text{con}} = x$ and $\alpha_{\text{aut}} = \bar{y}$, then $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}} = x\bar{y}$ is a conditional autarky for F :

$$\alpha_{\text{aut}} = \bar{y} \text{ is an autarky for } F|_{\alpha_{\text{con}}} = (\bar{y} \vee \bar{z}).$$

Conditional Autarkies

An assignment $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}}$ is a **conditional autarky** for formula F if α_{aut} is an autarky for $F|_{\alpha_{\text{con}}}$.

Example

Consider the formula $F := (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$.
Let $\alpha_{\text{con}} = x$ and $\alpha_{\text{aut}} = \bar{y}$, then $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}} = x\bar{y}$ is a conditional autarky for F :

$$\alpha_{\text{aut}} = \bar{y} \text{ is an autarky for } F|_{\alpha_{\text{con}}} = (\bar{y} \vee \bar{z}).$$

Let $\alpha = \alpha_{\text{con}} \cup \alpha_{\text{aut}}$ be a **conditional autarky** for formula F .
Then F and $F \wedge (\alpha_{\text{con}} \rightarrow \alpha_{\text{aut}})$ are satisfiability-equivalent.

In the above example, we could therefore learn $(\bar{x} \vee \bar{y})$.

Learning PR clauses

Theorem

Given a formula F and an assignment α . Every satisfying assignment ω of $p(F, \alpha)$ is a conditional autarky of F .

Recall: Given a formula F and a clause C . Let α denote the smallest assignment that falsifies C . C is SET w.r.t. F if and only if $p(F, \alpha)$ is **satisfiable**.

Let assignment ω satisfy $p(F, \alpha)$. Removing all but one of the literals in C that are satisfied by ω results in a **PR clause** w.r.t. F .

Pseudo-Code of CDCL (formula F)

```
1    $\alpha := \emptyset$ 
2   forever do
3      $\alpha := \text{Simplify}(F, \alpha)$ 
4     if  $F|_{\alpha}$  contains a falsified clause then
5        $C := \text{AnalyzeConflict}()$ 
6       if  $C$  is the empty clause then return unsatisfiable
7        $F := F \cup \{C\}$ 
8        $\alpha := \text{BackJump}(C, \alpha)$ 
13  else
14     $I := \text{Decide}()$ 
15    if  $I$  is undefined then return satisfiable
16     $\alpha := \alpha \cup \{I\}$ 
```

Pseudo-Code of SDCL (formula F)

```
1   $\alpha := \emptyset$ 
2  forever do
3     $\alpha := \text{Simplify}(F, \alpha)$ 
4    if  $F|_{\alpha}$  contains a falsified clause then
5       $C := \text{AnalyzeConflict}()$ 
6      if  $C$  is the empty clause then return unsatisfiable
7       $F := F \cup \{C\}$ 
8       $\alpha := \text{BackJump}(C, \alpha)$ 
9    else if  $p(F, \alpha)$  is satisfiable then
10      $C := \text{AnalyzeWitness}()$ 
11      $F := F \cup \{C\}$ 
12      $\alpha := \text{BackJump}(C, \alpha)$ 
13   else
14      $I := \text{Decide}()$ 
15     if  $I$  is undefined then return satisfiable
16      $\alpha := \alpha \cup \{I\}$ 
```

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

Theoretical Challenges

Lower bounds for interference-based proof systems with new variables will be hard, but what about **without new variables**?

- Lower bound for BC w/o new variables? Pigeon-hole formulas?
- Lower bound for SET w/o new variables? Tseitin formulas?
- Lower bound for PR w/o new variables?!

Theoretical Challenges

Lower bounds for interference-based proof systems with new variables will be hard, but what about **without new variables**?

- Lower bound for BC w/o new variables? Pigeon-hole formulas?
- Lower bound for SET w/o new variables? Tseitin formulas?
- Lower bound for PR w/o new variables?!

What is the power of **conditional autarky reasoning**?

Theoretical Challenges

Lower bounds for interference-based proof systems with new variables will be hard, but what about **without new variables**?

- Lower bound for BC w/o new variables? Pigeon-hole formulas?
- Lower bound for SET w/o new variables? Tseitin formulas?
- Lower bound for PR w/o new variables?!

What is the power of **conditional autarky reasoning**?

Can the new proof systems without new variables **simulate** old ones, in particular **Frege systems** (or the other way around)?

What about **cutting planes**?

Theoretical Challenges

Lower bounds for interference-based proof systems with new variables will be hard, but what about **without new variables**?

- Lower bound for BC w/o new variables? Pigeon-hole formulas?
- Lower bound for SET w/o new variables? Tseitin formulas?
- Lower bound for PR w/o new variables?!

What is the power of **conditional autarky reasoning**?

Can the new proof systems without new variables **simulate** old ones, in particular **Frege systems** (or the other way around)?

What about **cutting planes**?

Can we design stronger proof systems that make it even **easier to compute** short proofs?

Practical Challenges

The current version of SDCL is just the beginning:

- Which heuristics allow learning short PR clauses?
- How to construct an AnalyzeWitness procedure?
- Can the positive reduct be improved?

Can **local search** be used to find short proofs of unsatisfiability?

Constructing positive reducts (or similar formulas) efficiently:

- Generating a positive reduct is **more costly** than solving them
- Can we design **data-structures** to cheaply compute them?

Proof Systems and Proof Complexity

Marijn J.H. Heule

**Carnegie
Mellon
University**

<http://www.cs.cmu.edu/~mheule/15816-f19/>

Automated Reasoning and Satisfiability, September 24, 2019