

# Proof Systems and Proof Complexity

**Marijn J.H. Heule**

**Carnegie  
Mellon  
University**

<http://www.cs.cmu.edu/~mheule/15816-f24/>

Automated Reasoning and Satisfiability

September 18, 2024

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:

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

# Certifying Satisfiability and Unsatisfiability

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

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

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

- 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**:  $x\bar{y}z$

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

- 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:**  $\Gamma = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$
- **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:**  $\Gamma = (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z}) \wedge (x \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (u)$

■ **Resolution proof:**

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

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

# Resolution Proofs

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

■ **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**.

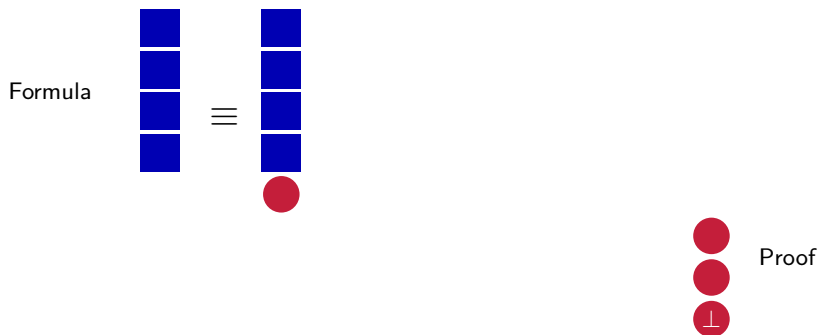
# Clausal Proofs

Reduce the size of the proof by only storing added clauses



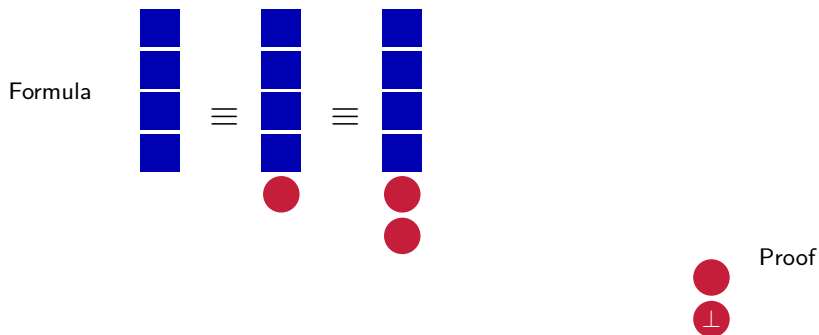
# Clausal Proofs

Reduce the size of the proof by only storing added clauses



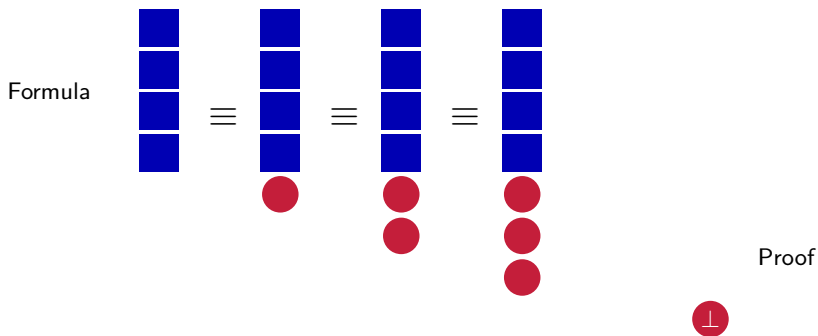
# Clausal Proofs

Reduce the size of the proof by only storing added clauses



# Clausal Proofs

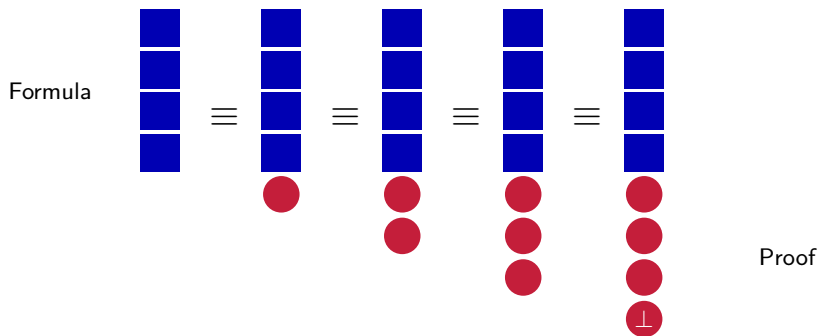
Reduce the size of the proof by only storing added clauses





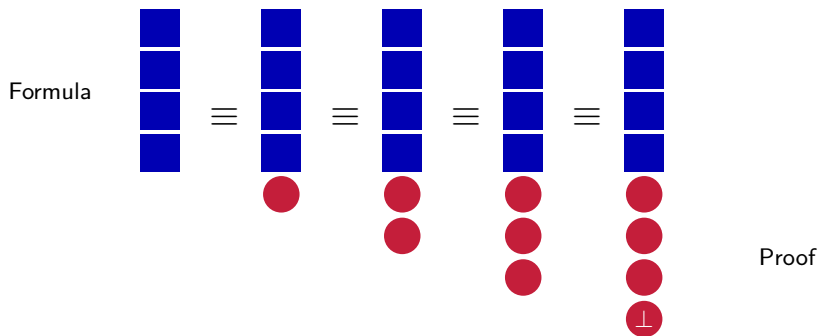
# Clausal Proofs

Reduce the size of the proof by only storing added clauses



## Clausal Proofs

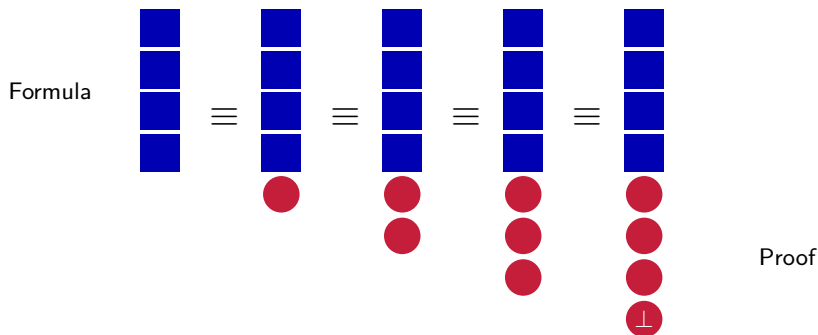
Reduce the size of the proof by only storing added clauses



- Clauses whose addition preserves satisfiability are **redundant**.
- Checking redundancy should be **efficient**.

# Clausal Proofs

Reduce the size of the proof by only storing added clauses



- Clauses whose addition preserves satisfiability are **redundant**.
  - Checking redundancy should be **efficient**.
- ➔ **Idea:** Only add 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  $\Gamma$  be a formula. A clause  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_1 C$ ) if UP on  $\Gamma \wedge \neg C$  results in a conflict.

### Example

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

## Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let  $\Gamma$  be a formula. A clause  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_1 C$ ) if UP on  $\Gamma \wedge \neg C$  results in a conflict.

### Example

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

clause	$(a \vee b)$
<hr/>	
units	$\bar{a} \wedge \bar{b}$

## Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let  $\Gamma$  be a formula. A clause  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_1 C$ ) if UP on  $\Gamma \wedge \neg C$  results in a conflict.

### Example

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

clause	$(a \vee b)$	$(a \vee b \vee \bar{c})$
units	$\bar{a} \wedge \bar{b}$	$\bar{c}$

## Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let  $\Gamma$  be a formula. A clause  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_1 C$ ) if UP on  $\Gamma \wedge \neg C$  results in a conflict.

### Example

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

clause	$(a \vee b)$	$(a \vee b \vee \bar{c})$	$(b \vee c \vee \bar{d})$
units	$\bar{a} \wedge \bar{b}$	$\bar{c}$	$\bar{d}$

## Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let  $\Gamma$  be a formula. A clause  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_1 C$ ) if UP on  $\Gamma \wedge \neg C$  results in a conflict.

### Example

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

clause	$(a \vee b)$	$(a \vee b \vee \bar{c})$	$(b \vee c \vee \bar{d})$	$(a \vee c \vee d)$
units	$\bar{a} \wedge \bar{b}$	$\bar{c}$	$\bar{d}$	$\perp$



## Reverse Unit Propagation

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let  $\Gamma$  be a formula. A clause  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_C C$ ) if UP on  $\Gamma \wedge \neg C$  results in a conflict.

### Example

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

clause	$(a \vee b)$	$(a \vee b \vee \bar{c})$	$(b \vee c \vee \bar{d})$	$(a \vee c \vee d)$
units	$\bar{a} \wedge \bar{b}$	$\bar{c}$	$\bar{d}$	$\perp$

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

Proofs of Unsatisfiability

**Beyond Resolution**

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

## 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**.

## 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]:

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

## 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]:

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

### Extended Resolution (ER) [Tseitin 1966]

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

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

- 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 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  $PHP_n$  are exponential [Haken 1985]

Cook constructed polynomial-sized ER proofs of  $PHP_n$



## Short Proofs of Pigeon Hole Formulas [Cook 1967]

Can  $n+1$  pigeons be 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  $PHP_n$  are **exponential** [Haken 1985]

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

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...

## Blocked Clauses [Kullmann 1999]

### Definition (Blocked Clause)

A clause  $(C \vee x)$  is a **blocked** on  $x$  w.r.t. a CNF formula  $\Gamma$  if for every clause  $(D \vee \bar{x}) \in \Gamma$ , resolvent  $C \vee D$  is a **tautology**.

## Blocked Clauses [Kullmann 1999]

### Definition (Blocked Clause)

A clause  $(C \vee x)$  is a **blocked** on  $x$  w.r.t. a CNF formula  $\Gamma$  if for every clause  $(D \vee \bar{x}) \in \Gamma$ , resolvent  $C \vee D$  is a **tautology**.

### 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 (un)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

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

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

# 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

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

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

**Blocked clause elimination** used in preprocessing and inprocessing

- Simulates many circuit optimization techniques
- Removes redundant Pythagorean Triples

# DRAT: An Interference-Based Proof System

- DRAT is a popular interference-based proof system
- DRAT allows adding 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

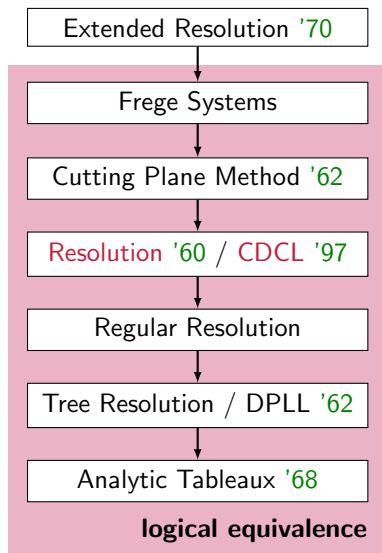
- DRAT is a popular interference-based proof system
- DRAT allows adding RATs (defined below) to a formula.
  - It can be efficiently checked if a clause is a RAT.
  - RATs are not necessarily implied by the formula.
  - But RATs are redundant: their addition preserves satisfiability.
- DRAT also allows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

## Definition (Resolution Asymmetric Tautology)

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

# Proof Search in Strong Proof Systems

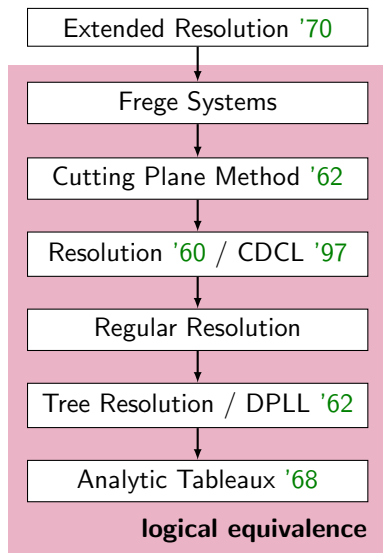
## Existence of Short Proofs



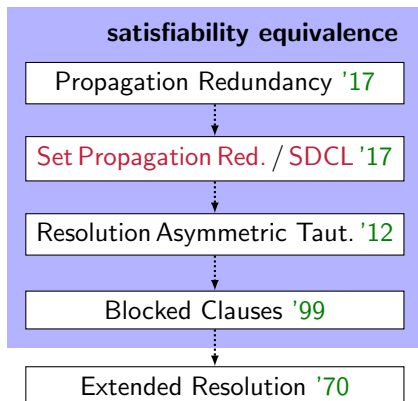


# Proof Search in Strong Proof Systems

## 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

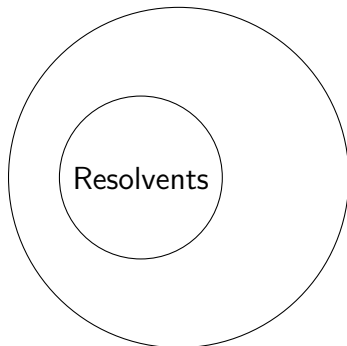
- Strong proof systems allow adding **many redundant clauses**.



All Redundant Clauses

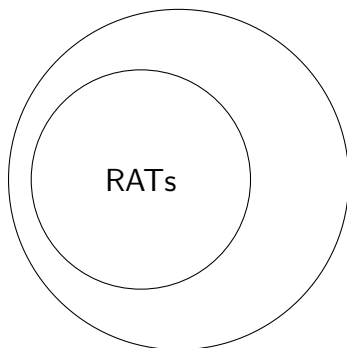
## Redundant Clauses

- Strong proof systems allow adding **many redundant clauses**.



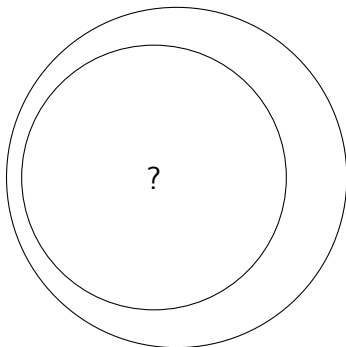
## Redundant Clauses

- Strong proof systems allow adding **many redundant clauses**.



## Redundant Clauses

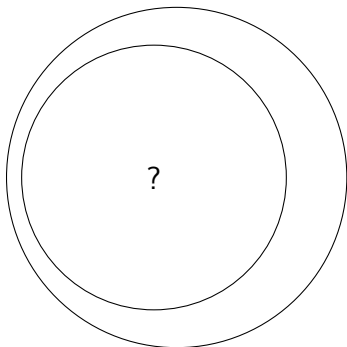
- Strong proof systems allow adding **many redundant clauses**.



- The new proof systems can give **short proofs** of formulas that are considered **hard**.

## Redundant Clauses

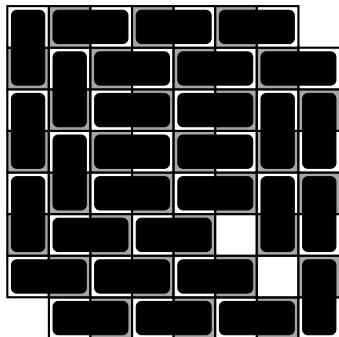
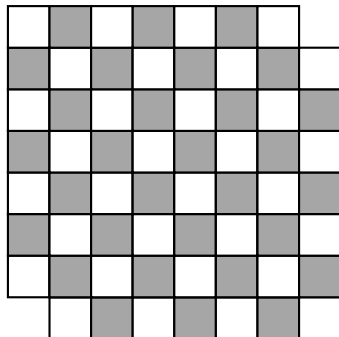
- Strong proof systems allow adding **many redundant clauses**.



- The new proof systems can give **short proofs** of formulas that are considered **hard**.
- Are **stronger** redundancy notions still **efficiently checkable**?

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

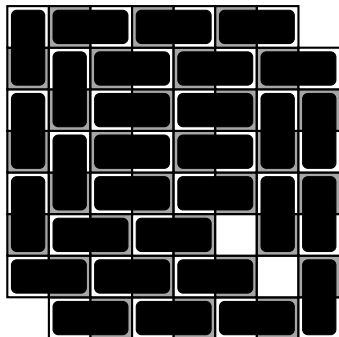
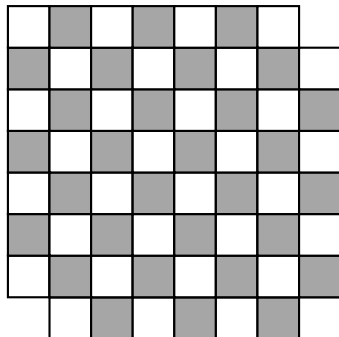
Can a chessboard be fully covered with dominos after removing two diagonally opposite corner squares?





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

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

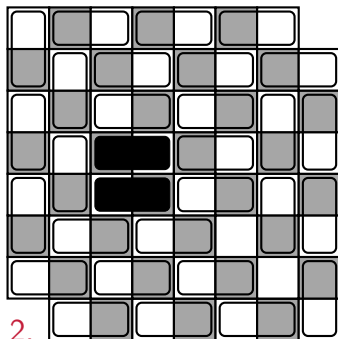
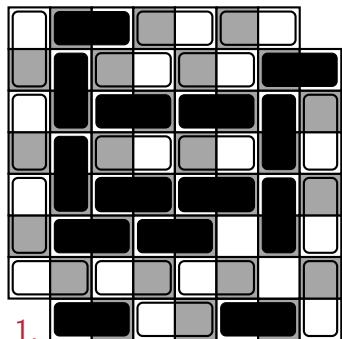


Easy to refute based on the following two observations:

- 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.

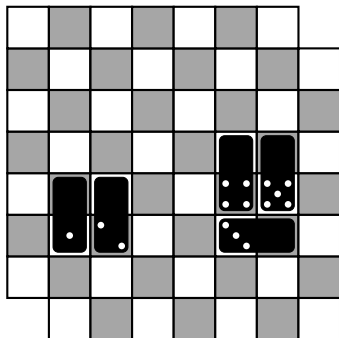
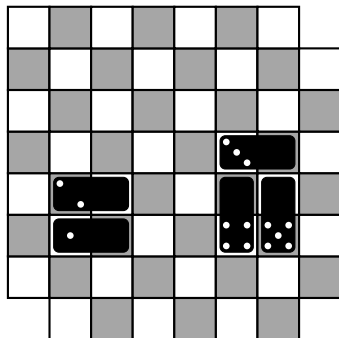


The used proof system can have a big impact on the size:

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

## Mutilated Chessboards: An alternative proof

Satisfaction-Driven Clause Learning (SDCL) is a new solving paradigm that finds proofs in the PR proof system [HKB '17]



SDCL can detect that the above two patterns can be **blocked**

- This reduces the number of explored states **exponentially**
- We produced **SPR** proofs that are linear in the formula size

## Redundancy as an Implication

A formula  $\Delta$  is **at least as satisfiable** as a formula  $\Gamma$  if  $\Gamma \models \Delta$ .

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

### Theorem

A clause  $C$  is **redundant** w.r.t. a formula  $\Gamma$  iff there exists an assignment  $\alpha$  such that

$$\Gamma \wedge \neg C \models (\Gamma \wedge C)|_{\alpha}$$

This is the **strongest notion** of redundancy. However, entailment ( $\models$ ) 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  $\Gamma$  be a formula,  $C$  a clause, and  $\alpha$  the smallest assignment that falsifies  $C$ .  $C$  is **implied by  $\Gamma$  via UP** (denoted by  $\Gamma \vdash_1 C$ ) if UP on  $\Gamma|_{\alpha}$  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  $\Gamma \wedge \neg C \models (\Gamma \wedge C)|_{\alpha}$  by  $\Gamma \wedge \neg C \vdash_1 (\Gamma \wedge C)|_{\alpha}$
- **Example:**  
 $\Gamma = (x \vee y \vee z) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee z)$   
and  $\Delta = (z)$ . Observe that  $\Gamma \models \Delta$ , but that  $\Gamma \not\vdash_1 \Delta$ .

# 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

## Autarkies

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

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

### Example

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

Assignment  $\alpha_1 = \bar{z}$  is an autarky:

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



## Autarkies

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

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

### Example

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

Assignment  $\alpha_1 = \bar{z}$  is an autarky:

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

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

**Theorem ([Monien and Speckenmeyer 1985])**

*Let  $\alpha$  be an autarky for formula  $\Gamma$ .*

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

## Conditional Autarkies

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

### Example

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

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

$$\alpha_{\text{aut}} = \bar{x} \text{ is an autarky for } \Gamma | \alpha_{\text{con}} = (\bar{x}) \wedge (\bar{z}).$$

## Conditional Autarkies

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

### Example

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

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

$$\alpha_{\text{aut}} = \bar{x} \text{ is an autarky for } \Gamma | \alpha_{\text{con}} = (\bar{x}) \wedge (\bar{z}).$$

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

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

## Finding Redundant Clauses: The Positive Reduct

Determining whether a clause  $C$  is PR w.r.t. a formula  $\Gamma$  is an NP-complete problem.

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

## Finding Redundant Clauses: The Positive Reduct

Determining whether a clause  $C$  is PR w.r.t. a formula  $\Gamma$  is an NP-complete problem.

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

### Theorem

Given formula  $\Gamma$  and assignment  $\alpha$ . A *satisfying assignment*  $\omega$  of *positive reduct*  $p(\Gamma, \alpha)$  is a *conditional autarky* of  $\Gamma$ .

## Finding Redundant Clauses: The Positive Reduct

Determining whether a clause  $C$  is PR w.r.t. a formula  $\Gamma$  is an NP-complete problem.

How to find PR clauses? Encode it in SAT!

### Theorem

Given formula  $\Gamma$  and assignment  $\alpha$ . A *satisfying assignment*  $\omega$  of *positive reduct*  $p(\Gamma, \alpha)$  is a *conditional autarky* of  $\Gamma$ .

Positive reducts are typically very easy to solve!

## Finding Redundant Clauses: The Positive Reduct

Determining whether a clause  $C$  is PR w.r.t. a formula  $\Gamma$  is an NP-complete problem.

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

### Theorem

Given formula  $\Gamma$  and assignment  $\alpha$ . A *satisfying assignment*  $\omega$  of *positive reduct*  $p(\Gamma, \alpha)$  is a *conditional autarky* of  $\Gamma$ .

Positive reducts are typically very easy to solve!

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

## The Positive Reduct: An Example

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

### Example

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

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

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

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

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



## Pseudo-Code of CDCL (formula $\Gamma$ )

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

## Pseudo-Code of SDCL (formula $\Gamma$ )

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

Proofs of Unsatisfiability

Beyond Resolution

Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

# Theoretical Challenges

What is the power of conditional autarky reasoning?

# Theoretical Challenges

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

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?