Proof Systems and Proof Complexity

Marijn J.H. Heule

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f22/

Automated Reasoning and Satisfiability

September 21, 2022

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:

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

Certifying Satisfiability and Unsatisfiability

- Certifying satisfiability of a formula is easy:
  - Just consider a satisfying assignment: $x\overline{y}z$

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

  - 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: $x\overline{y}z$

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

  - 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 $C_1, \ldots, C_m$ of clauses.
  - Every clause is either contained in the formula or derived from two earlier clauses via the resolution rule:

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

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

Resolution Proofs

- Example: $F = (\overline{x} \lor \overline{y} \lor z) \land (\overline{z}) \land (x \lor \overline{y}) \land (\overline{u} \lor y) \land (u)$
- Resolution proof:
  $(\overline{x} \lor \overline{y} \lor z), (\overline{z}), (\overline{x} \lor \overline{y}), (x \lor \overline{y}), (\overline{y}), (\overline{u} \lor y), (\overline{u}), (u), \bot$

Resolution Proofs

- Example: $F = (\bar{x} \lor \bar{y} \lor z) \land (\bar{z}) \land (x \lor \bar{y}) \land (\bar{u} \lor y) \land (u)$
- Resolution proof:
  $(\bar{x} \lor \bar{y} \lor z), (\bar{z}), (\bar{x} \lor \bar{y}), (x \lor \bar{y}), (\bar{y}), (\bar{u} \lor y), (\bar{u}), (u), \perp$

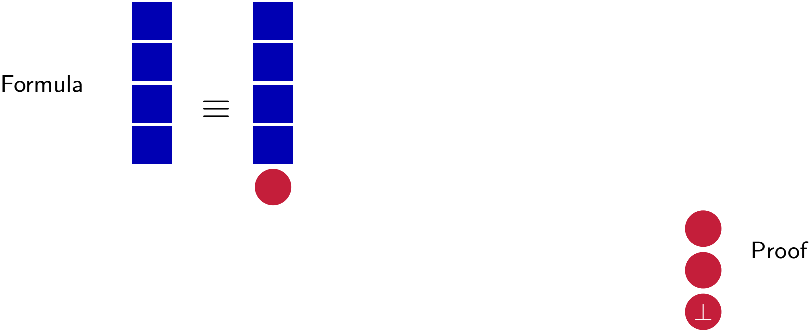
$$\dfrac{\dfrac{\bar{u} \lor y \qquad \dfrac{\dfrac{\bar{x} \lor \bar{y} \lor z \qquad \bar{z}}{\bar{x} \lor \bar{y}} \qquad x \lor \bar{y}}{\bar{y}}}{\bar{u}} \qquad u}{\perp}$$

Clausal Proofs

Reduce the size of the proof by only storing added clauses

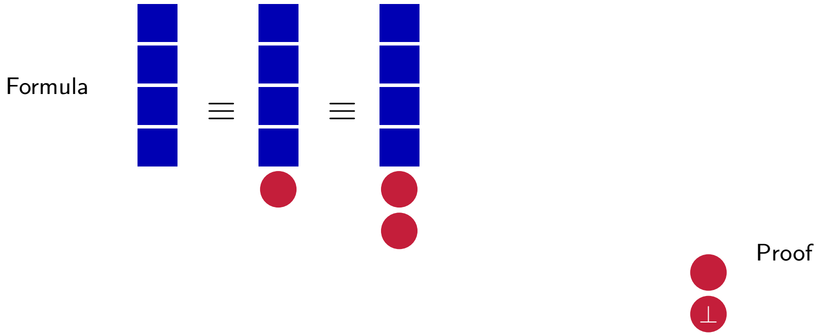
Clausal Proofs

Reduce the size of the proof by only storing added clauses



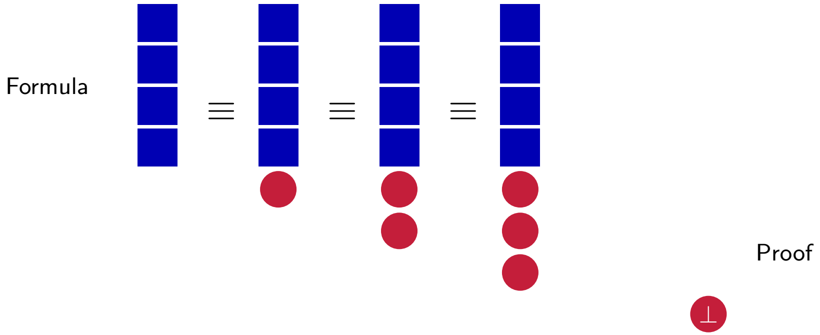
Clausal Proofs

Reduce the size of the proof by only storing added clauses



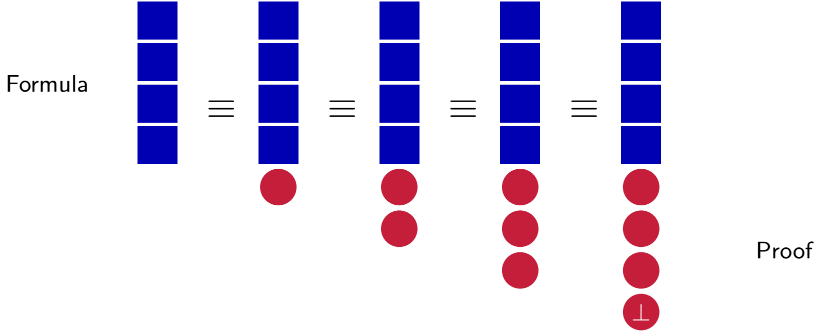
Clausal Proofs

Reduce the size of the proof by only storing added clauses



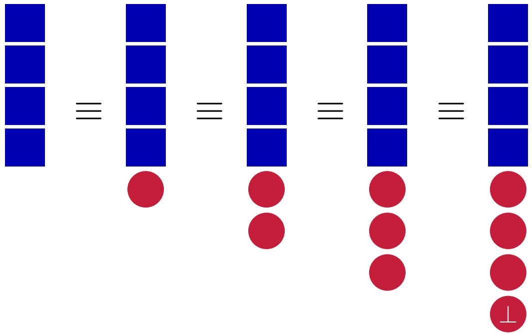
Clausal Proofs

Reduce the size of the proof by only storing added clauses



Clausal Proofs

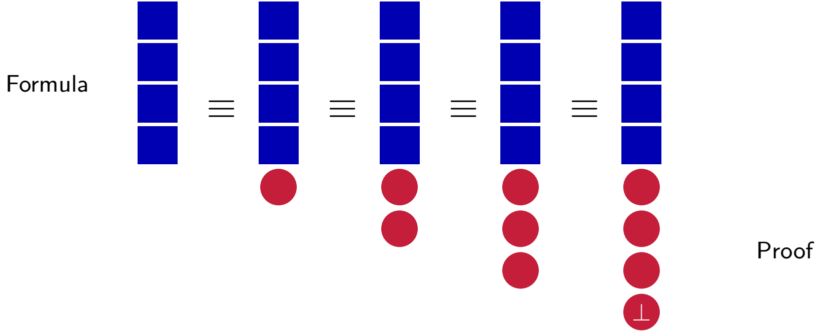
Reduce the size of the proof by only storing added clauses



- 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 F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

Example

$$\begin{aligned} F = \; & (a \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor c) \land (b \lor c \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor d) \; \land \\ & (a \lor c \lor d) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor b \lor d) \land (a \lor \bar{b} \lor \bar{d}) \end{aligned}$$

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. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \wedge \neg C$ results in a conflict.

Example

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

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

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. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

Example

$$F = (a \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor c) \land (b \lor c \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor d) \land (a \lor c \lor d) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor b \lor d) \land (a \lor \bar{b} \lor \bar{d})$$

clause	$(a \lor b)$	$(a \lor b \lor \bar{c})$
units	$\bar{a} \land \bar{b}$	$\bar{c}$

Reverse Unit Propagation

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

Example

$$F = (a \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor c) \land (b \lor c \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor d) \land (a \lor c \lor d) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor b \lor d) \land (a \lor \bar{b} \lor \bar{d})$$

clause	$(a \lor b)$	$(a \lor b \lor \bar{c})$	$(b \lor c \lor \bar{d})$
units	$\bar{a} \land \bar{b}$	$\bar{c}$	$\bar{d}$

Reverse Unit Propagation

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

Example

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

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 F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \wedge \neg C$ results in a conflict.

Example

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

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

$$\dfrac{\dfrac{(a \vee c \vee d) \qquad (b \vee c \vee \bar{d})}{(a \vee b \vee c)} \qquad (a \vee b \vee \bar{c})}{(a \vee b)}$$

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.

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

Traditional Proofs vs. Interference-Based Proofs

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

$$\frac{C \lor x \qquad \overline{x} \lor D}{C \lor D}\,\text{(RES)} \qquad\qquad \frac{A \qquad A \rightarrow B}{B}\,\text{(MP)}$$

➥ 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.

$$\frac{C \lor x \qquad \overline{x} \lor D}{C \lor D}\ \text{(RES)} \qquad\qquad \frac{A \qquad A \rightarrow B}{B}\ \text{(MP)}$$

➥ 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]:

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

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]:

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

Extended Resolution (ER) [Tseitin 1966]

- Combines resolution with the Extension rule:

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

- Equivalently, adds the definition $x := \mathrm{AND}(a, b)$
- 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)?

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

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)?

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

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 (Block Clause)

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

Blocked Clauses [Kullmann 1999]

Definition (Block Clause)

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

Example

Consider the formula $(a \vee b) \wedge (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\overline{c}$.
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

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

- The ER clauses are blocked on the literals x and $\overline{x}$ 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 $$\frac{x \notin F \qquad \overline{x} \notin F}{(x \vee \overline{a} \vee \overline{b}) \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee b)} \text{ (ER)}$$
- The ER clauses are blocked on the literals x and $\overline{x}$ 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

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

Proof Search in Strong Proof Systems

Existence of Short Proofs

- Extended Resolution '70
- Frege Systems
- Cutting Plane Method '62
- Resolution '60 / CDCL '97
- Regular Resolution
- Tree Resolution / DPLL '62
- Analytic Tableaux '68

logical equivalence

marijn@cmu.edu

Proof Search in Strong Proof Systems

Existence of Short Proofs

Extended Resolution '70
↓
Frege Systems
↓
Cutting Plane Method '62
↓
Resolution '60 / CDCL '97
↓
Regular Resolution
↓
Tree Resolution / DPLL '62
↓
Analytic Tableaux '68

logical equivalence

Finding Short Proofs

satisfiability equivalence

Propagation Redundancy '17
↓
Set Propagation Red. / SDCL '17
↓
Resolution Asymmetric Taut. '12
↓
Blocked Clauses '99
↓
Extended Resolution '70

Express solving techniques compactly
[Järvisalo, Heule, and Biere '12]

Short proofs without new variables
[Heule, Kiesl, and Biere '17A]

Proofs of Unsatisfiability

Beyond Resolution

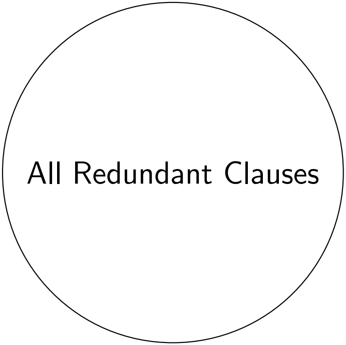
Propagation Redundancy

Satisfaction-Driven Clause Learning

Challenges

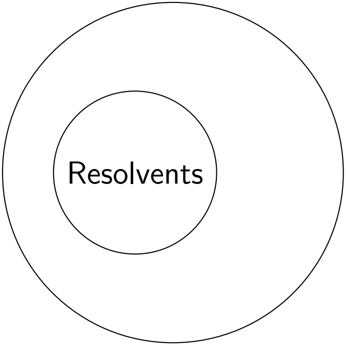
Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.



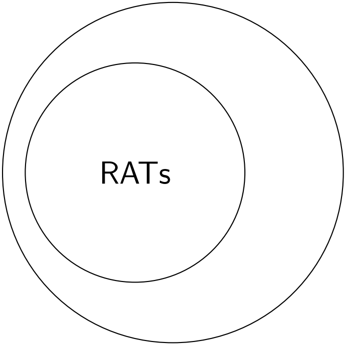
Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.



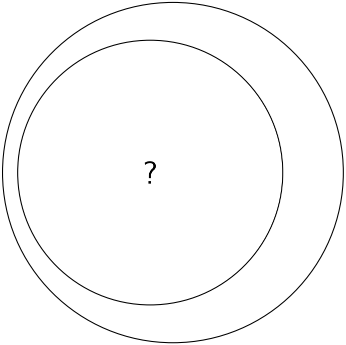
Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.



Redundant Clauses

- Strong proof systems allow addition of many redundant clauses.

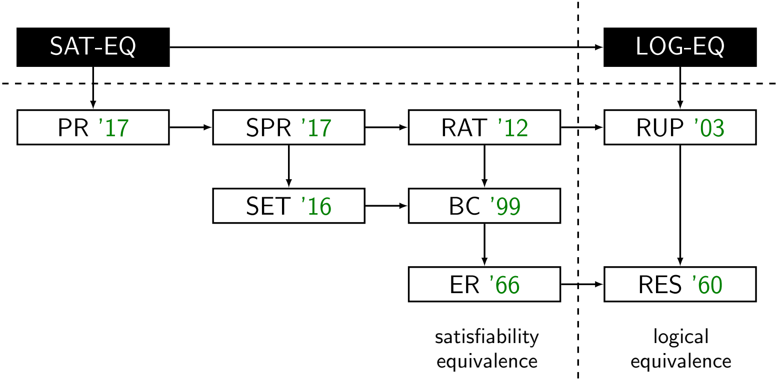


- Are stronger redundancy notions still efficiently checkable?

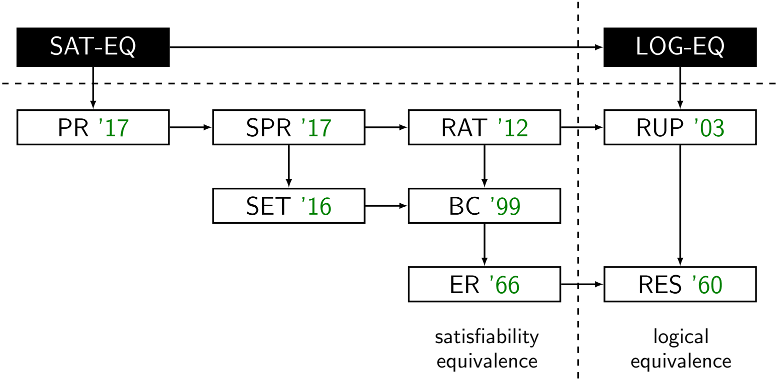
New Propositional Proof Systems

- 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



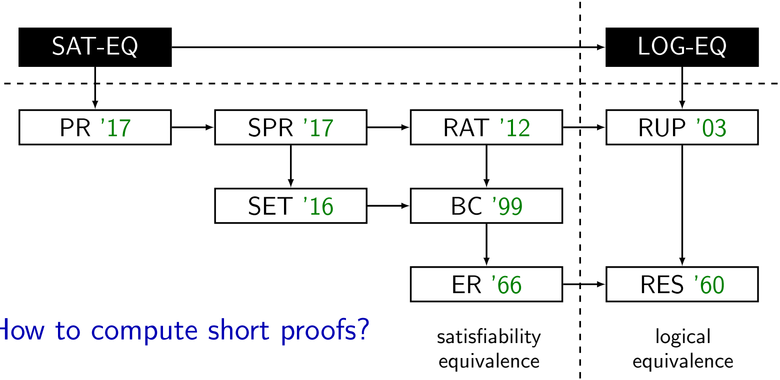
New Proof Systems for Propositional Logic



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?

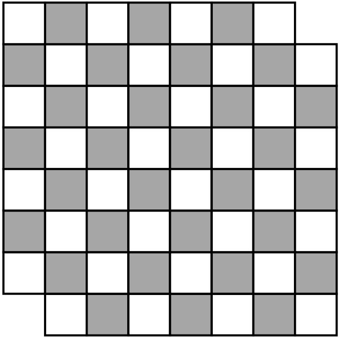
- The new proof systems can give short proofs of formulas that are considered hard.

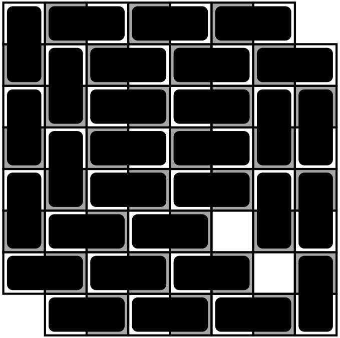
- We have short SPR and PR proofs for the well-known pigeon hole formulas (linear in the size of the input).
  - Pigeon hole formulas have only exponential-size resolution proofs.
  - If the addition of new variables via definitions is allowed, there are polynomial-size proofs.

- 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]

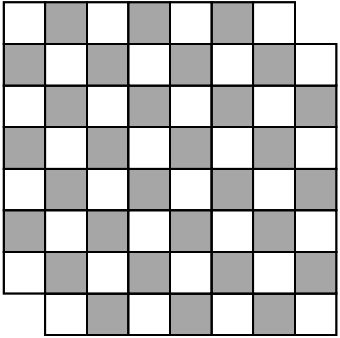
Can a chessboard be fully covered with dominos after removing two diagonally opposite corner squares?

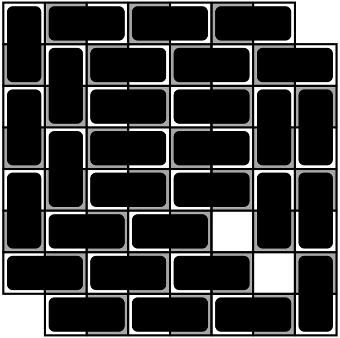




Mutilated Chessboards: "A Tough Nut to Crack" [McCarthy]

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



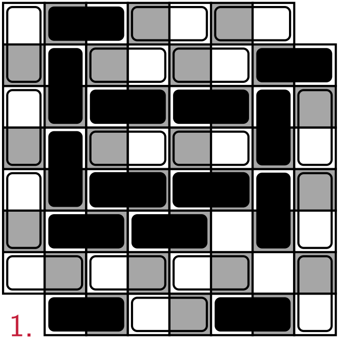


Easy to refute based on the following two observations:

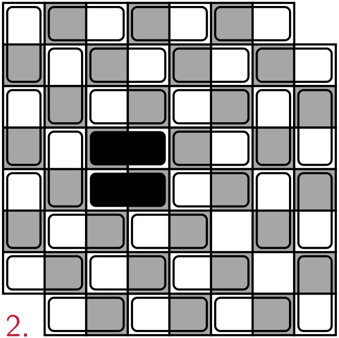
- There are more white squares than black squares; and
- A domino covers exactly one white and one black square.

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.



1.



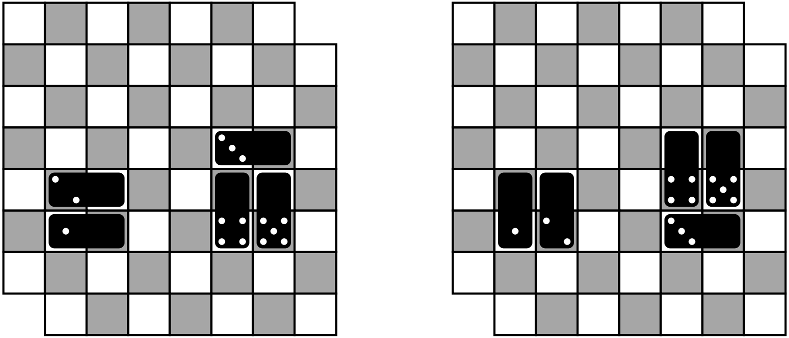
2.

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 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, 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 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:
  $F = (x \vee y \vee z) \wedge (\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y} \vee z)$
  and $G = (z)$. Observe that $F \models G$, but that $F \nvdash_1 G$.

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

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 $assigned(D, \alpha)$ with $D \in F$ and D is satisfied by α.

Example

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

Let $C_1 = (\overline{x})$, so $\alpha_1 = x$.
The positive reduct $p(F, \alpha_1) = (\overline{x}) \wedge (x) \wedge (x)$ is unsatisfiable.

Let $C_2 = (\overline{x} \vee \overline{y})$, so $\alpha_2 = x\,y$.
The positive reduct $p(F, \alpha_2) = (\overline{x} \vee \overline{y}) \wedge (x \vee y) \wedge (x \vee \overline{y})$ 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 $F := (x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$.
Assignment $\alpha_1 = \overline{z}$ is an autarky:
$(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$. Assignment $\alpha_2 = x\,\overline{y}\,z$ is an autarky: $(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$.

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 $F := (x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$.
Assignment $\alpha_1 = \overline{z}$ is an autarky:
$(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$. Assignment $\alpha_2 = x\,\overline{y}\,z$ is an autarky: $(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$.

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

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

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

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

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 $(\overline{x} \vee \overline{y})$.

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)

```
α := ∅
forever do
    α := Simplify (F, α)
    if F|α contains a falsified clause then
        C := AnalyzeConflict ()
        if C is the empty clause then return unsatisfiable
        F := F ∪ {C}
        α := BackJump (C, α)
    else
        l := Decide ()
        if l is undefined then return satisfiable
        α := α ∪ {l}
```

Pseudo-Code of SDCL (formula F)

$\alpha := \emptyset$
forever do
  $\alpha :=$ Simplify (F, α)
  if $F|_\alpha$ contains a falsified clause **then**
    $C :=$ AnalyzeConflict $()$
    if C is the empty clause **then return** unsatisfiable
    $F := F \cup \{C\}$
    $\alpha :=$ BackJump (C, α)
  else if $p(F, \alpha)$ is satisfiable **then**
    $C :=$ AnalyzeWitness $()$
    $F := F \cup \{C\}$
    $\alpha :=$ BackJump (C, α)
  else
    $l :=$ Decide $()$
    if l is undefined **then return** satisfiable
    $\alpha := \alpha \cup \{l\}$

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?