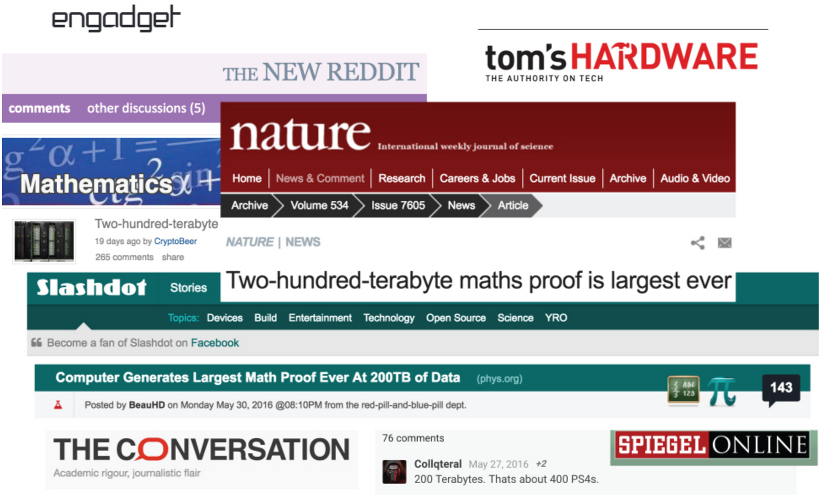
Short Proofs in Strong Proof Systems

Marijn J.H. Heule

Carnegie Mellon University

April 8, 2022

"The Largest Math Proof Ever"



Proofs of Unsatisfiability

Beyond Resolution

Finding Short Proofs

Short Proofs via Preprocessing

Future Work and Challenges

Proofs of Unsatisfiability

Beyond Resolution

Finding Short Proofs

Short Proofs via Preprocessing

Future Work and Challenges

Certifying Satisfiability and Unsatisfiability

- Certifying satisfiability of a formula is easy:

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

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: $F = (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{z}) \wedge (x \vee \overline{y}) \wedge (\overline{u} \vee y) \wedge (u)$
- Resolution proof:
  $(\overline{x} \vee \overline{y} \vee z), (\overline{z}), (\overline{x} \vee \overline{y}), (x \vee \overline{y}), (\overline{y}), (\overline{u} \vee y), (\overline{u}), (u), \perp$

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:
  $(\bar{x} \lor \bar{y} \lor z), (\bar{z}), (\bar{x} \lor \bar{y}), (x \lor \bar{y}), (\bar{y}), (\bar{u} \lor y), (\bar{u}), (u), \bot$

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

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:
  $(\bar{x} \lor \bar{y} \lor z), (\bar{z}), (\bar{x} \lor \bar{y}), (x \lor \bar{y}), (\bar{y}), (\bar{u} \lor y), (\bar{u}), (u), \bot$

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

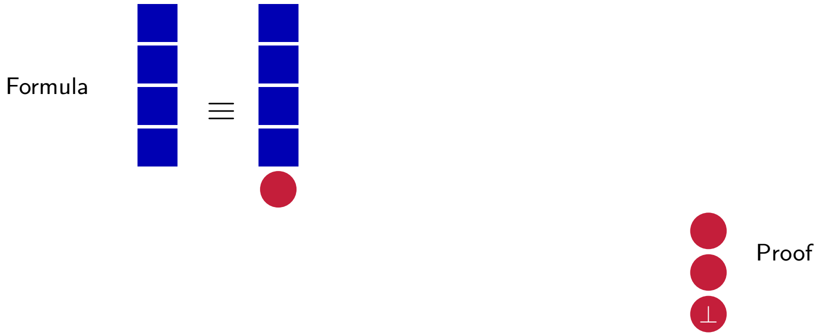
- Drawbacks of resolution:
  - For many seemingly simple formulas, there are only resolution proofs of exponential size.
  - State-of-the-art solving techniques are not succinctly expressible.

Clausal Proofs

Reduce the size of the proof by only storing added clauses

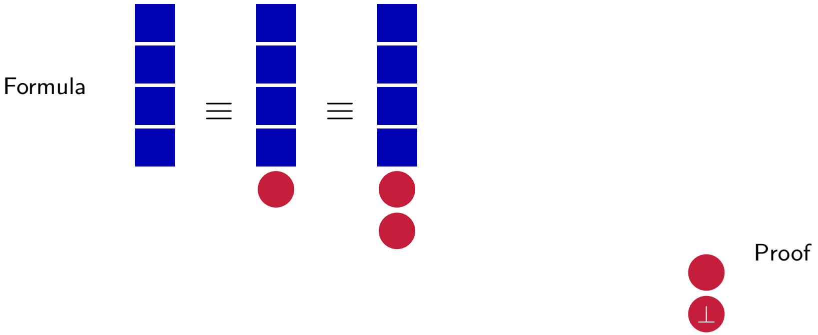
Clausal Proofs

Reduce the size of the proof by only storing added clauses



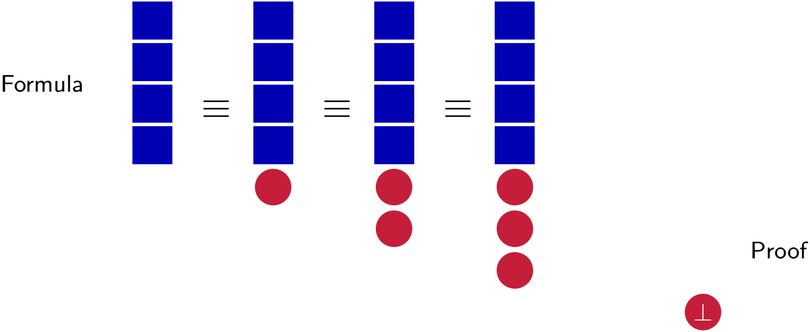
Clausal Proofs

Reduce the size of the proof by only storing added clauses



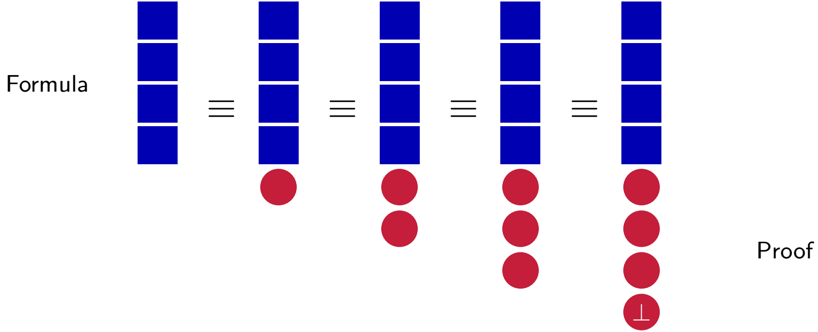
Clausal Proofs

Reduce the size of the proof by only storing added clauses



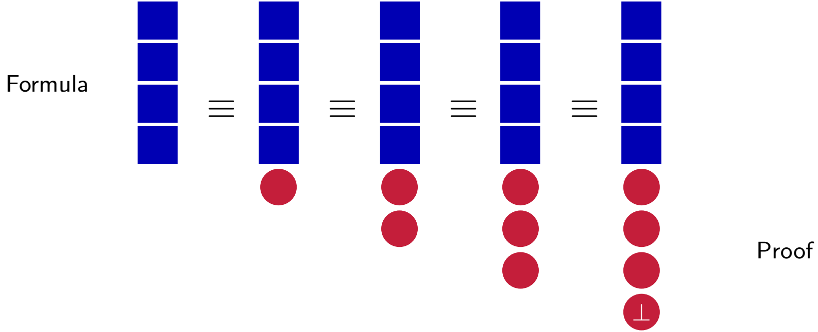
Clausal Proofs

Reduce the size of the proof by only storing added clauses



Clausal Proofs

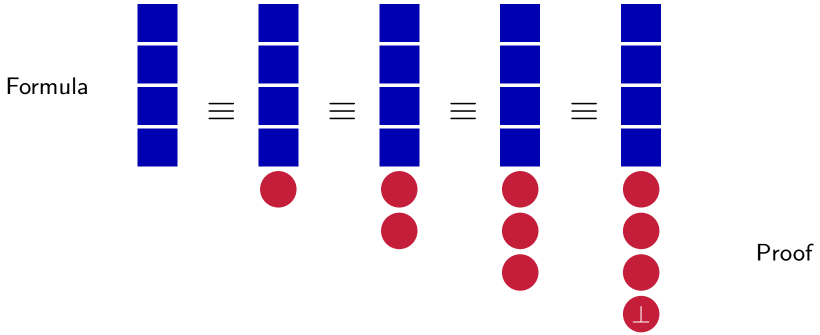
Reduce the size of the proof by only storing added clauses



- 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 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

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

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, 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

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

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

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 $F \vdash_1 C$) if UP on $F|_\alpha$ results in a conflict.

Example

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

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

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

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

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

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

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

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

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

Proofs of Unsatisfiability

Beyond Resolution

Finding Short Proofs

Short Proofs via Preprocessing

Future Work and Challenges

Traditional Proofs vs. Interference-Based Proofs

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

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

Traditional Proofs vs. Interference-Based Proofs

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

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

➥ 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)?

Resolution proofs are exponential in n [Haken 1985]

Cook constructed polynomial-sized ER proofs

Short Proofs of Pigeon Hole Formulas [Cook 1967]

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

Resolution proofs are exponential in n [Haken 1985]

Cook constructed polynomial-sized ER proofs

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

Blocked Clauses [Kullmann 1999]

Definition (Block Clause)

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

Or equivalently: A clause $(C \lor x)$ is a blocked on x w.r.t. a CNF formula F if x is pure in $F|\overline{C}$.

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.

Or equivalently: A clause $(C \vee x)$ is a blocked on x w.r.t. a CNF formula F if x is pure in $F|\overline{C}$.

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

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

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

- 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

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 Red. (PR) '17

Set Prop. Red. (SPR) / SDCL '17

Res. Asym. Taut. (RAT) '12

Blocked Clauses (BC) '99

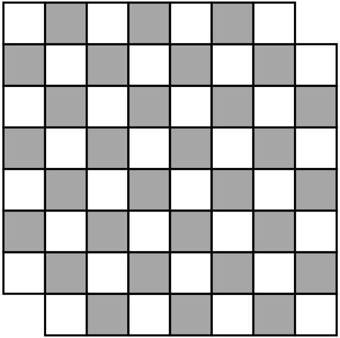
Extended Resolution (ER) '70

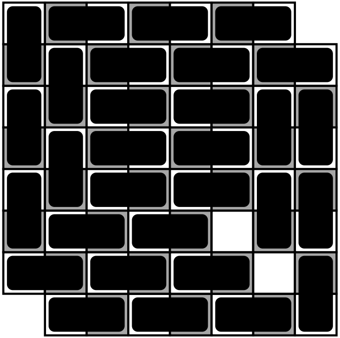
Express solving techniques compactly [Järvisalo, Heule, and Biere '12]

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

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

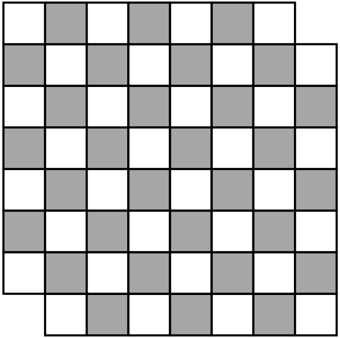
Can a chessboard be fully covered with dominos after removing two diagonally opposite corner squares?

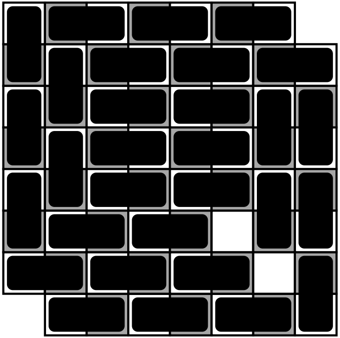




Mutilated Chessboards: "A Tough Nut to Crack" [McCarthy]

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



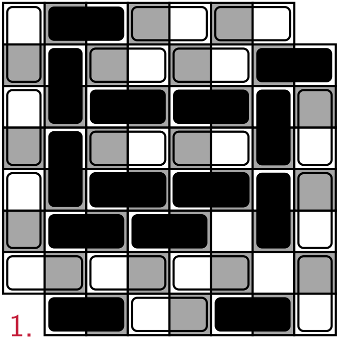


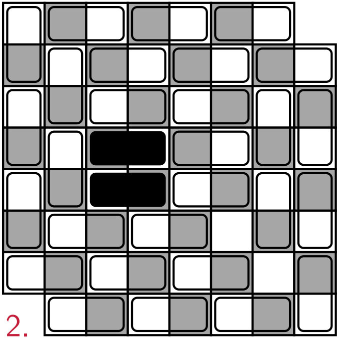
Easy to refute based on the following two observations:

- 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.

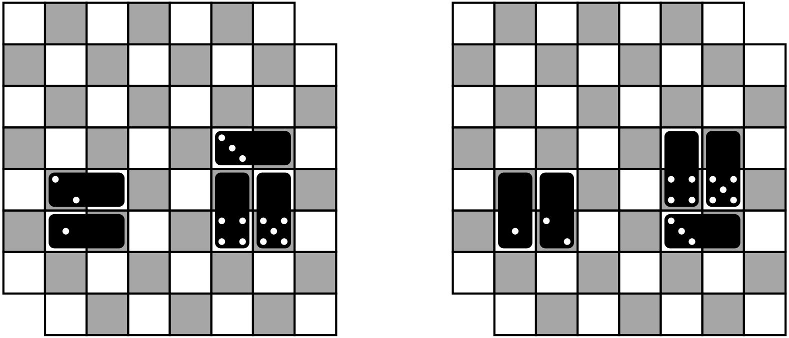

2.

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

Proofs of Unsatisfiability

Beyond Resolution

Finding Short Proofs

Short Proofs via Preprocessing

Future Work and Challenges

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

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

- $\alpha_1 = \overline{z}$ is an autarky: $(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z})$.
- $\alpha_2 = x\,\overline{y}\,z$ 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 a non-empty 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 a non-empty 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 $(\overline{x} \vee \overline{y})$.

Finding Conditional Autarkies

The positive reduct of a formula F and an assignment α, denoted by $p(F, \alpha)$, is the formula that contains clause $C = \overline{\alpha}$ 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 $\alpha_1 = x$, so $C_1 = (\overline{x})$ The positive reduct $p(F, \alpha_1) = (\overline{x}) \wedge (x) \wedge (x)$ is unsatisfiable.
- Let $\alpha_2 = x\,y$, so $C_2 = (\overline{x} \vee \overline{y})$. The positive reduct $p(F, \alpha_2) = (\overline{x} \vee \overline{y}) \wedge (x \vee y) \wedge (x \vee \overline{y})$ is satisfiable.

Finding Conditional Autarkies

The positive reduct of a formula F and an assignment α, denoted by $p(F, \alpha)$, is the formula that contains clause $C = \overline{\alpha}$ 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 $\alpha_1 = x$, so $C_1 = (\overline{x})$ The positive reduct $p(F, \alpha_1) = (\overline{x}) \wedge (x) \wedge (x)$ is unsatisfiable.
- Let $\alpha_2 = x\,y$, so $C_2 = (\overline{x} \vee \overline{y})$. The positive reduct $p(F, \alpha_2) = (\overline{x} \vee \overline{y}) \wedge (x \vee y) \wedge (x \vee \overline{y})$ is satisfiable.

Theorem ([Heule, Kiesl, Biere '17B])

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

Satisfaction-Driven Clause Learning [Heule, Kiesl, Biere '17B]

SDCL generalizes CDCL and finds proofs in the SPR proof system.

CDCL in a nutshell:

1. Main loop combines efficient problem simplification with cheap, but effective decision heuristics; ($> 90\%$ of time)
2. Reasoning kicks in if the current state is conflicting;
3. The current state is analyzed and turned into a constraint;
4. The constraint is added to the problem, the heuristics are updated, and the algorithm (partially) restarts.

Satisfaction-Driven Clause Learning [Heule, Kiesl, Biere '17B]

SDCL generalizes CDCL and finds proofs in the SPR proof system.

SDCL in a nutshell:

1. Main loop combines efficient problem simplification with cheap, but effective decision heuristics; ($>90\%$ of time)
2. ~~Reasoning kicks in if the current state is conflicting;~~
2. Reasoning kicks in if there exists a state that is at least as satisfiable as the current state; (*NP-complete check*)
3. The current state is analyzed and turned into a constraint;
4. The constraint is added to the problem, the heuristics are updated, and the algorithm (partially) restarts.

Satisfaction-Driven Clause Learning [Heule, Kiesl, Biere '17B]

SDCL generalizes CDCL and finds proofs in the SPR proof system.

SDCL in a nutshell:

1. Main loop combines efficient problem simplification with cheap, but effective decision heuristics; ($>90\%$ of time)
2. ~~Reasoning kicks in if the current state is conflicting;~~
2. Reasoning kicks in if there exists a state that is at least as satisfiable as the current state; (*NP-complete check*)
3. The current state is analyzed and turned into a constraint;
4. The constraint is added to the problem, the heuristics are updated, and the algorithm (partially) restarts.

Short proofs for problems that are hard for resolution

including pigeonhole, Tseitin, and mutilated chessboard problems

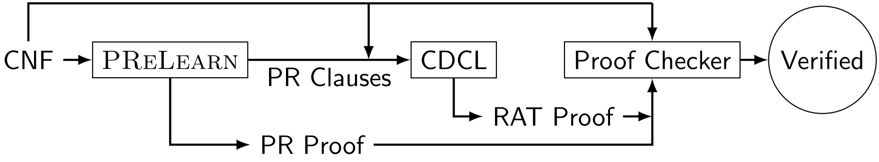
Preprocessing: PReLearn [Reeves, Heule, Bryant 2022]

Key observation: If PR clauses are useful, then we only need PR clauses of length at most 2.

Idea: only look for PR clauses of length at most 2

For each literal $l \in F$, check if $(l \vee k)$ is a PR clause.

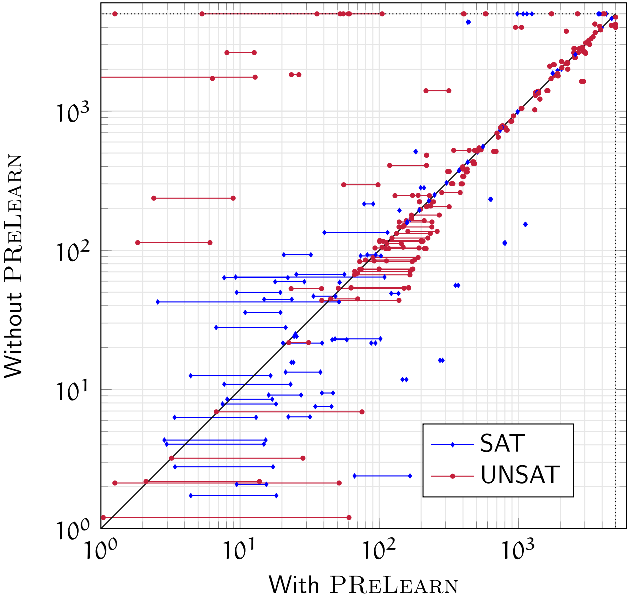
- Restrict the search by k occurring in $F \setminus F|l$.
- Use the positive reduct to check for PR.



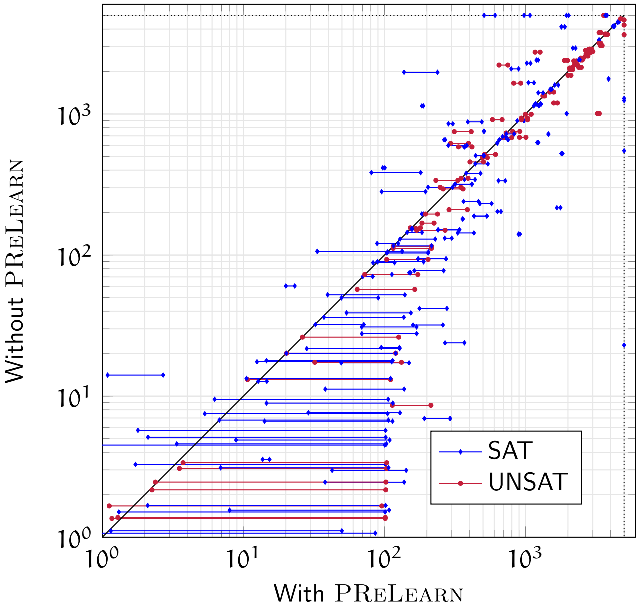
Preprocessing: Mutilated Chessboard

N	Time	Rounds	Avg.	Units	Bin.	Avg. Units	Avg. Bin.
8	0.14	1	0.14	30	164	30.00	164.00
12	4.94	1	4.94	103	1,045	103.00	1,045.00
16	62.47	2	31.23	195	3,988	97.50	1,994.00
20	513.12	6	85.52	339	1,4470	56.50	2,411.67
24	4,941.38	26	190.05	512	64,038	19.69	2,463.00

Preprocessing: Small Formulas (up to 10K clauses)



Preprocessing: Medium-Sized Formulas (10K-50K clauses)



Preprocessing: Biggest Impact on Performance

Size	Value	With	Without	Clauses	Formula	Year
0-10k	UNSAT	1.26	–	2,033	ph12*	2013
0-10k	UNSAT	35.69	–	20,179	Pb-chnl15-16_c18*	2019
0-10k	UNSAT	105.01	–	46,759	Pb-chnl20-21_c18	2019
0-10k	UNSAT	59.99	–	1,633	randomG-Mix-n17-d05	2021
0-10k	UNSAT	61.08	–	1,472	randomG-n17-d05	2021
0-10k	UNSAT	407.51	–	1,640	randomG-n18-d05	2021
0-10k	UNSAT	584.95	–	1,706	randomG-Mix-n18-d05	2021
0-10k	SAT	1,082.62	–	9,650	fsf-300-354-2-2-3-2.23.opt	2013
0-10k	SAT	1,250.82	–	10,058	fsf-300-354-2-2-3-2.46.opt	2013
10k-50k	SAT	1,076.34	–	804	sp5-26-19-bin-stri-flat	2021
10k-50k	SAT	608.48	–	901	sp5-26-19-una-nons-tree	2021
10k-50k	SAT	–	22.99	254	Ptn-7824-b13	2016
10k-50k	SAT	–	549.27	133	Ptn-7824-b09	2016
10k-50k	SAT	–	1,246.42	39	Ptn-7824-b02	2016
10k-50k	SAT	–	1,290.49	121	Ptn-7824-b08	2016
10k-50k	UNSAT	–	3,650.21	31,860	rphp4_110_shuffled	2016
10k-50k	UNSAT	–	4,273.88	31,531	rphp4_115_shuffled	2016

Future Work: Arbitrarily Complex Solvers

Verifying efficient automated reasoning tools is a daunting task:

- Tools are constantly modified and improved; and
- Even top-pier and "experimentally correct" solvers turned out to be buggy. [Järvisalo, Heule, Biere '12]

Verified checkers of certificates in strong proof systems:

- Don't worry about correctness or completeness of tools;
- Facilitates making tools more complex and efficient; while
- Full confidence in results. [Heule, Hunt, Kaufmann, Wetzler '17]

Future Work: Arbitrarily Complex Solvers

Verifying efficient automated reasoning tools is a daunting task:

- Tools are constantly modified and improved; and
- Even top-pier and "experimentally correct" solvers turned out to be buggy. [Järvisalo, Heule, Biere '12]

Verified checkers of certificates in strong proof systems:

- Don't worry about correctness or completeness of tools;
- Facilitates making tools more complex and efficient; while
- Full confidence in results. [Heule, Hunt, Kaufmann, Wetzler '17]

ACL2 Isabelle CAKEML

Formally verified checkers now also used in industry

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?