Computable Short Proofs

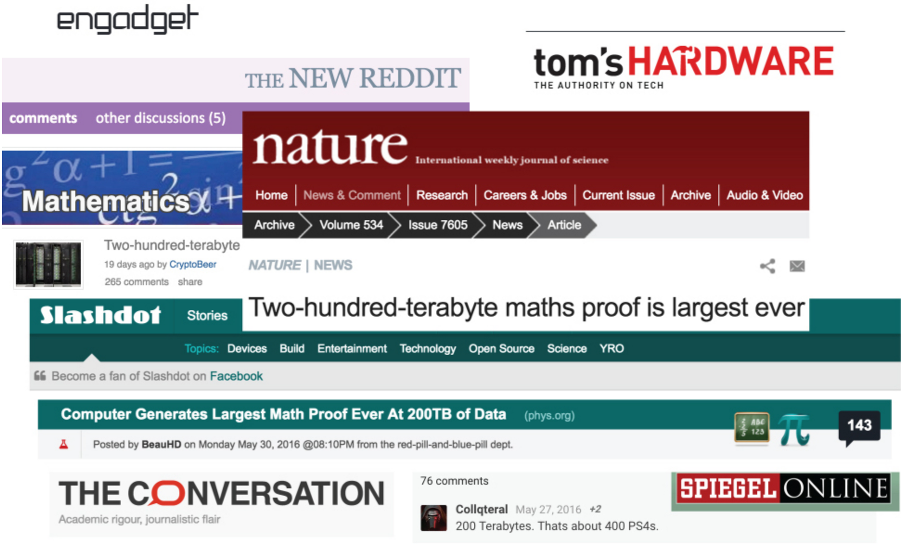
Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

joint work with Benjamin Kiesl and Armin Biere

Oaxaca SAT workshop August 27, 2018

"The Largest Math Proof Ever" [Nature 2016]



Introduction on Proofs

Certifying Satisfiability and Unsatisfiability

- Certifying satisfiability of a formula is easy:

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

Certifying Satisfiability and Unsatisfiability

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

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

  - 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: $x\bar{y}z$

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

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

Resolution Proofs

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

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

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

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

- 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.

To cope with these drawbacks, we need advanced techniques...

Interference-Based Proof Systems

Traditional Proofs vs. Interference-Based Proofs

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

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\,(\text{res}) \qquad \frac{A \qquad A \rightarrow B}{B}\,(\text{mp})$$

Traditional Proofs vs. Interference-Based Proofs

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

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\,(\text{res}) \qquad\qquad \frac{A \qquad A \rightarrow B}{B}\,(\text{mp})$$

➥ 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.

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\ (\text{res}) \qquad\qquad \frac{A \qquad A \rightarrow B}{B}\ (\text{mp})$$

➥ 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.

Reasoning about Absence is as old as SAT Solving

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

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

Reasoning about Absence is as old as SAT Solving

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

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

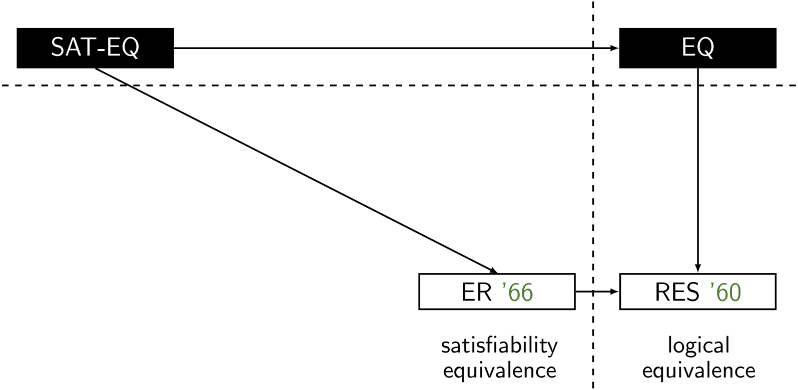
Extended Resolution (ER) [Tseitin 1966]

- Combines resolution with the Extension rule:

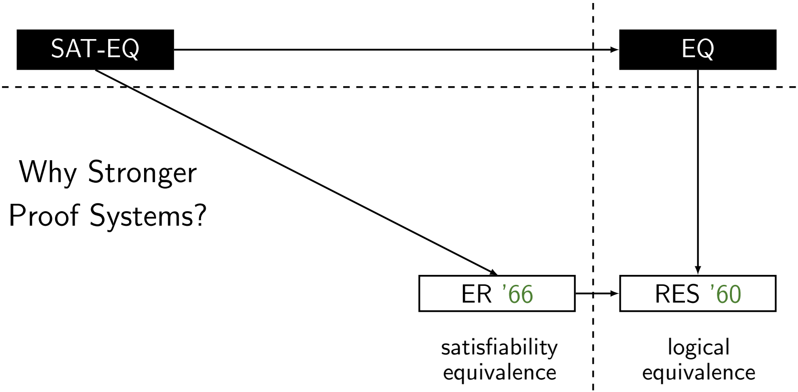
$$\frac{x \notin F \qquad \overline{x} \notin F}{(x \vee \overline{a} \vee \overline{b}) \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee b)} \text{ (er)}$$

- 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

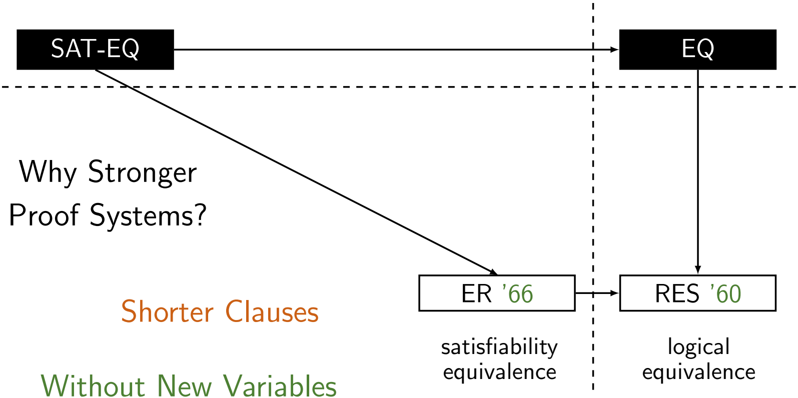
Classical Proof Systems for Propositional Logic



Classical Proof Systems for Propositional Logic



Classical Proof Systems for Propositional Logic



Why Stronger Proof Systems?

Shorter Clauses

Without New Variables

Easier to Compute

Without New Variables

Short Proofs of Pigeon Hole Formulas [Cook 1967]

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

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

Resolution proofs of PHP_n formulas are exponential [Haken 1985]

Cook constructed polynomial-sized ER proofs of PHP_n formulas

Short Proofs of Pigeon Hole Formulas [Cook 1967]

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

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

Resolution proofs of PHP_n formulas are exponential [Haken 1985]

Cook constructed polynomial-sized ER proofs of PHP_n formulas

However, these proofs require introducing new variables:

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

Blocked Clauses [Kullmann 1999]

Definition (Blocking literal)

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

Definition (Blocked clause)

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

Blocked Clauses [Kullmann 1999]

Definition (Blocking literal)

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

Definition (Blocked clause)

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

Example

Consider the formula $(a \lor b) \land (a \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c.

Blocked Clauses [Kullmann 1999]

Definition (Blocking literal)

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

Definition (Blocked clause)

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

Example

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

Theorem

Adding or removing a blocked clause preserves satisfiability.

Blocked Clause Addition and Blocked Clause Elimination

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

- BC generalizes ER [Kullmann 1999]
- Recall

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

- 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 [JAR 2012]
- Removes redundant Pythagorean Triples [SAT 2016]

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 [JAR 2012]
- Removes redundant Pythagorean Triples [SAT 2016]

However, blocked clauses do not offer enough expressivity

- Increase expressivity with autarky-based reasoning...

Autarkies [Monien and Speckenmeyer 1985]

An autarky is an assignment that satisfies every clause it touches.

A pure literal and a satisfying assignment are autarkies.

Example

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

Autarkies [Monien and Speckenmeyer 1985]

An autarky is an assignment that satisfies every clause it touches.

A pure literal and a satisfying assignment are autarkies.

Example

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

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

Theorem

Let α be an autarky for formula F.
Then, F and $F|_\alpha$ are satisfiability equivalent.

Conditional Autarkies [Heule, Kiesl, Seidl, Biere 2017]

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

Example

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

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

Conditional Autarkies [Heule, Kiesl, Seidl, Biere 2017]

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

Example

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

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

Theorem

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

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

Conditional Autarkies and Blocked Clauses

Blocked clauses and conditional autarkies are strongly related:

Theorem

A clause $(c_1 \vee \cdots \vee c_i \vee x)$ *is blocked on* x *w.r.t. formula* F *if and only if* $\overline{c}_1 \wedge \cdots \wedge \overline{c}_i \wedge x$ *is a conditional autarky for* F.

Conditional Autarkies and Blocked Clauses

Blocked clauses and conditional autarkies are strongly related:

Theorem

A clause $(c_1 \vee \cdots \vee c_i \vee x)$ *is blocked on* x *w.r.t. formula* F *if and only if* $\overline{c}_1 \wedge \cdots \wedge \overline{c}_i \wedge x$ *is a conditional autarky for* F.

The blocking literal set generalizes the blocking literal concept resulting in set-blockedness [Kiesl, Seidl, Tompits, Biere 2016]

Theorem ([Heule, Kiesl, Seidl, Biere 2017])

A clause $(c_1 \vee \cdots \vee c_i \vee x_1 \vee \cdots \vee x_k)$ *is set-blocked* (SBC) *on literal set* $\{x_1, \ldots, x_k\}$ *w.r.t. formula* F *if and only if* $\overline{c}_1 \wedge \cdots \wedge \overline{c}_i \wedge x_1 \wedge \cdots \wedge x_k$ *is a conditional autarky for* F.

Conditional Autarkies and Pigeon Hole Formulas

$$
\begin{array}{l}
(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \quad \wedge \quad (x_{1,3} \vee x_{2,3} \vee x_{3,3} \vee \cdots \vee x_{n,3}) \wedge \\
(x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge \ldots \wedge (x_{1,n+1} \vee x_{2,n+1} \vee x_{3,n+1} \vee \cdots \vee x_{n,n+1}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,2}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,2}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,2}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,2}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,3}) \wedge \\
(\overline{x}_{1,2} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,2} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,2} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,2} \vee \overline{x}_{n,3}) \wedge \\
\qquad \qquad \ldots \quad \wedge \quad \ldots \\
(\overline{x}_{1,n} \vee \overline{x}_{1,n+1}) \wedge (\overline{x}_{2,n} \vee \overline{x}_{2,n+1}) \wedge (\overline{x}_{3,n} \vee \overline{x}_{3,n+1}) \wedge \ldots \wedge (\overline{x}_{n,n} \vee \overline{x}_{n,n+1})
\end{array}
$$

Conditional Autarkies and Pigeon Hole Formulas

$$
\begin{array}{c}
(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \quad \wedge \quad (x_{1,3} \vee x_{2,3} \vee x_{3,3} \vee \cdots \vee x_{n,3}) \wedge \\
(x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge \ldots \wedge (x_{1,n+1} \vee x_{2,n+1} \vee x_{3,n+1} \vee \cdots \vee x_{n,n+1}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,2}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,2}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,2}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,2}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,3}) \wedge \\
(\overline{x}_{1,2} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,2} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,2} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,2} \vee \overline{x}_{n,3}) \wedge \\
\ldots \quad \wedge \quad \ldots \\
(\overline{x}_{1,n} \vee \overline{x}_{1,n+1}) \wedge (\overline{x}_{2,n} \vee \overline{x}_{2,n+1}) \wedge (\overline{x}_{3,n} \vee \overline{x}_{3,n+1}) \wedge \ldots \wedge (\overline{x}_{n,n} \vee \overline{x}_{n,n+1})
\end{array}
$$

- Consider $\alpha_{\text{con}} = \overline{x}_{1,3} \wedge \overline{x}_{1,4} \wedge \cdots \wedge \overline{x}_{1,n+1} \wedge \overline{x}_{2,3} \wedge \overline{x}_{2,4} \wedge \cdots \wedge \overline{x}_{2,n+1}$

Conditional Autarkies and Pigeon Hole Formulas

$$
\begin{array}{l}
(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \quad \wedge \quad (x_{1,3} \vee x_{2,3} \vee x_{3,3} \vee \cdots \vee x_{n,3}) \wedge \\
(x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge \ldots \wedge (x_{1,n+1} \vee x_{2,n+1} \vee x_{3,n+1} \vee \cdots \vee x_{n,n+1}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,2}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,2}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,2}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,2}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,3}) \wedge \\
(\overline{x}_{1,2} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,2} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,2} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,2} \vee \overline{x}_{n,3}) \wedge \\
\qquad \ldots \quad \wedge \quad \ldots \\
(\overline{x}_{1,n} \vee \overline{x}_{1,n+1}) \wedge (\overline{x}_{2,n} \vee \overline{x}_{2,n+1}) \wedge (\overline{x}_{3,n} \vee \overline{x}_{3,n+1}) \wedge \ldots \wedge (\overline{x}_{n,n} \vee \overline{x}_{n,n+1})
\end{array}
$$

- Consider $\alpha_{\text{con}} = \overline{x}_{1,3} \wedge \overline{x}_{1,4} \wedge \cdots \wedge \overline{x}_{1,n+1} \wedge \overline{x}_{2,3} \wedge \overline{x}_{2,4} \wedge \cdots \wedge \overline{x}_{2,n+1}$
- Notice that $\alpha_{\text{aut}} = x_{1,1} \wedge \overline{x}_{1,2} \wedge \overline{x}_{2,1} \wedge x_{2,2}$ is an autarky of $F|\alpha_{\text{con}}$

Conditional Autarkies and Pigeon Hole Formulas

$$
\begin{array}{l}
(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \quad \wedge \quad (x_{1,3} \vee x_{2,3} \vee x_{3,3} \vee \cdots \vee x_{n,3}) \wedge \\
(x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge \ldots \wedge (x_{1,n+1} \vee x_{2,n+1} \vee x_{3,n+1} \vee \cdots \vee x_{n,n+1}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,2}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,2}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,2}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,2}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,3}) \wedge \\
(\overline{x}_{1,2} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,2} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,2} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,2} \vee \overline{x}_{n,3}) \wedge \\
\qquad \ldots \quad \wedge \quad \ldots \\
(\overline{x}_{1,n} \vee \overline{x}_{1,n+1}) \wedge (\overline{x}_{2,n} \vee \overline{x}_{2,n+1}) \wedge (\overline{x}_{3,n} \vee \overline{x}_{3,n+1}) \wedge \ldots \wedge (\overline{x}_{n,n} \vee \overline{x}_{n,n+1})
\end{array}
$$

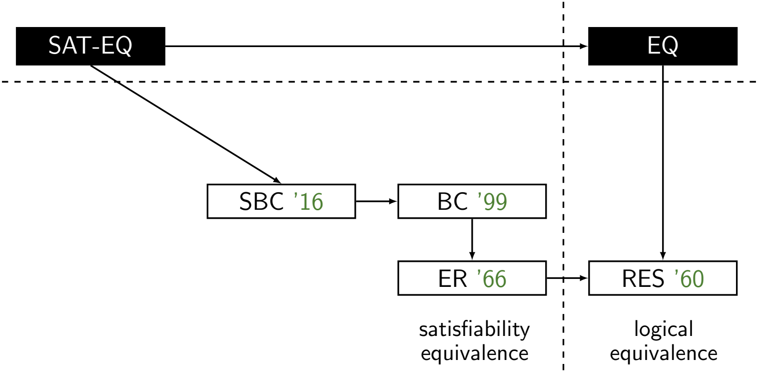
- Consider $\alpha_{\text{con}} = \overline{x}_{1,3} \wedge \overline{x}_{1,4} \wedge \cdots \wedge \overline{x}_{1,n+1} \wedge \overline{x}_{2,3} \wedge \overline{x}_{2,4} \wedge \cdots \wedge \overline{x}_{2,n+1}$
- Notice that $\alpha_{\text{aut}} = x_{1,1} \wedge \overline{x}_{1,2} \wedge \overline{x}_{2,1} \wedge x_{2,2}$ is an autarky of $F\,|\,\alpha_{\text{con}}$
- Resolution can reduce the constraint $\alpha_{\text{con}} \rightarrow \alpha_{\text{aut}}$ to $(\overline{x}_{1,2} \vee \overline{x}_{2,1})$

Conditional Autarkies and Pigeon Hole Formulas

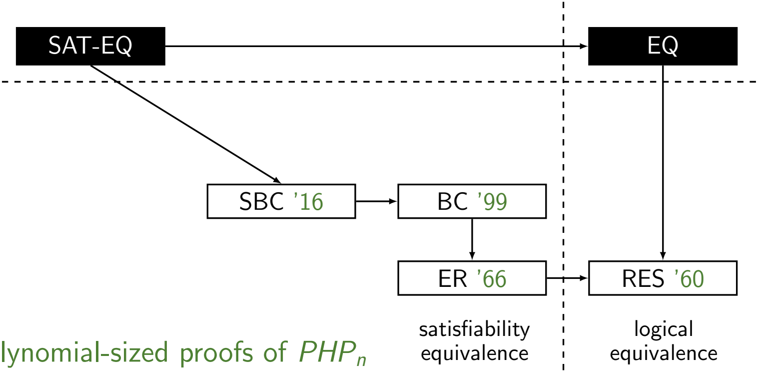
$$\begin{array}{c}
(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \quad \wedge \quad (x_{1,3} \vee x_{2,3} \vee x_{3,3} \vee \cdots \vee x_{n,3}) \wedge \\
(x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge \ldots \wedge (x_{1,n+1} \vee x_{2,n+1} \vee x_{3,n+1} \vee \cdots \vee x_{n,n+1}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,2}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,2}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,2}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,2}) \wedge \\
(\overline{x}_{1,1} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,1} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,1} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,1} \vee \overline{x}_{n,3}) \wedge \\
(\overline{x}_{1,2} \vee \overline{x}_{1,3}) \wedge (\overline{x}_{2,2} \vee \overline{x}_{2,3}) \wedge (\overline{x}_{3,2} \vee \overline{x}_{3,3}) \wedge \ldots \wedge (\overline{x}_{n,2} \vee \overline{x}_{n,3}) \wedge \\
\ldots \quad \wedge \quad \ldots \\
(\overline{x}_{1,n} \vee \overline{x}_{1,n+1}) \wedge (\overline{x}_{2,n} \vee \overline{x}_{2,n+1}) \wedge (\overline{x}_{3,n} \vee \overline{x}_{3,n+1}) \wedge \ldots \wedge (\overline{x}_{n,n} \vee \overline{x}_{n,n+1})
\end{array}$$

- Consider $\alpha_{\text{con}} = \overline{x}_{1,3} \wedge \overline{x}_{1,4} \wedge \cdots \wedge \overline{x}_{1,n+1} \wedge \overline{x}_{2,3} \wedge \overline{x}_{2,4} \wedge \cdots \wedge \overline{x}_{2,n+1}$
- Notice that $\alpha_{\text{aut}} = x_{1,1} \wedge \overline{x}_{1,2} \wedge \overline{x}_{2,1} \wedge x_{2,2}$ is an autarky of $F|\alpha_{\text{con}}$
- Resolution can reduce the constraint $\alpha_{\text{con}} \rightarrow \alpha_{\text{aut}}$ to $(\overline{x}_{1,2} \vee \overline{x}_{2,1})$
- Allows constructing poly-sized proofs in SBC w/o new variables

Proof Systems based on Conditional Autarkies



Proof Systems based on Conditional Autarkies



Polynomial-sized proofs of PHP_n in SBC without new variables

However, many clauses are long (making proof search hard)

Shorter Clauses

Reverse Unit Propagation [Goldberg and Novikov 2003]

- 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.
- $F \vdash_1 C$ is also known as Reverse Unit Propagation (RUP).
- Learned clauses in CDCL solvers are RUP clauses.
- RUP typically summarizes dozens of resolution steps.

DRAT: An Interference-Based Proof System [SAT 2014]

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

DRAT: An Interference-Based Proof System [SAT 2014]

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

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

Redundancy as an Implication [CADE 2017]

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

Theorem ([Heule, Kiesl, Biere 2017])

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) $F|_\alpha \vDash F|_\omega$.

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

Propagation Redundancy [CADE 2017]

- Implied by F via UP is used in SAT solvers to determine redundancy of learned clauses and therefore $\vdash_1$ is a natural restriction of $\vDash$.

- We bound $F|_\alpha \vDash F|_\omega$ by $F|_\alpha \vdash_1 F|_\omega$.

Propagation Redundancy [CADE 2017]

- Implied by F via 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 $F|_\alpha \vdash_1 F|_\omega$.

Definition (Propagation Redundant Clause)

Let F be a formula, C a clause, and α the smallest assignment that falsifies C. Then, C is propagation redundant (PR) w.r.t. F if there exists an assignment ω satisfying C with $F|_\alpha \vdash_1 F|_\omega$.

PR and Pigeon Hole Formulas [CADE 2017]

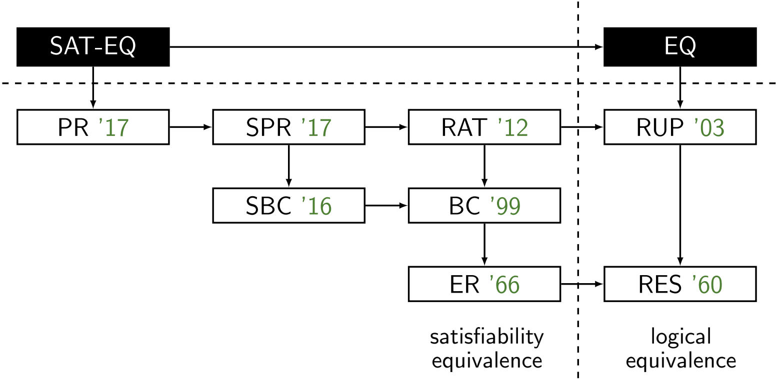
Can $n{+}1$ pigeons be placed in n holes (at-most-one pigeon per hole)?

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

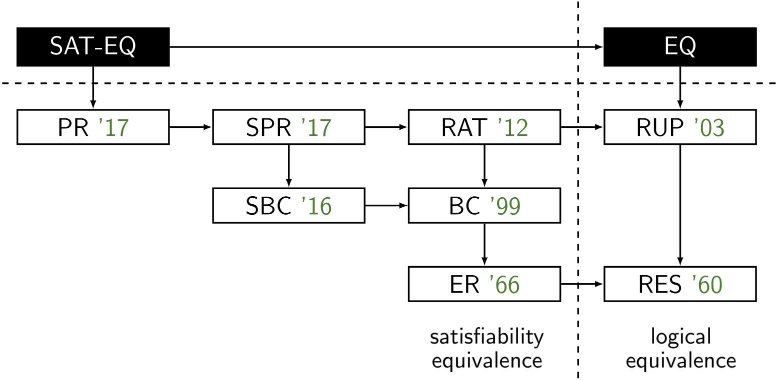
Any $(\overline{x}_{h,p} \vee \overline{x}_{k,q})$ with $h \neq k$ and $p \neq q$ is a PR clause w.r.t. PHP_n

- with witness $\omega = \overline{x}_{h,p} \wedge x_{k,p} \wedge x_{h,q} \wedge \overline{x}_{k,q}$
- learning n binary clauses $(\overline{x}_{1,1} \vee \overline{x}_{n,q})$ with $q \in \{2, .., n+1\}$ allows learning the unit clause $(\overline{x}_{1,1})$

New Proof Systems for Propositional Logic



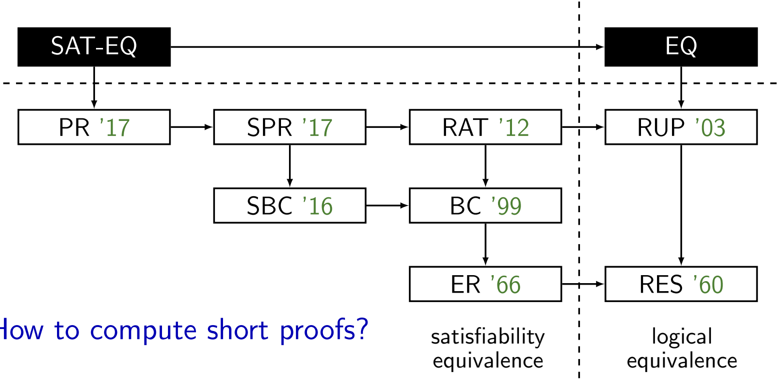
New Proof Systems for Propositional Logic



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]

Satisfaction-Driven Clause Learning

Finding PR Clauses: The Positive Reduct [HVC 2017]

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

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

Finding PR Clauses: The Positive Reduct [HVC 2017]

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

How to find SBC 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 SBC w.r.t. F.

Finding PR Clauses: The Positive Reduct [HVC 2017]

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

How to find SBC 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 SBC w.r.t. F.

Positive reducts are typically very easy to solve!

Finding PR Clauses: The Positive Reduct [HVC 2017]

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

How to find SBC 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 SBC 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 [HVC 2017]

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

Example

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

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

Let $C_2 = (\bar{x} \vee \bar{y})$, so $\alpha_2 = x\, y$. The positive reduct $p(F, \alpha_2) = (\bar{x} \vee \bar{y}) \wedge (x \vee y) \wedge (x \vee \bar{y})$ is satisfiable.

The Positive Reduct: An Example [HVC 2017]

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

Example

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

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

Let $C_2 = (\bar{x} \lor \bar{y})$, so $\alpha_2 = x\, y$. The positive reduct $p(F, \alpha_2) = (\bar{x} \lor \bar{y}) \land (x \lor y) \land (x \lor \bar{y})$ is satisfiable.

Theorem

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

Pseudo-Code of CDCL (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
    $l :=$ Decide $()$
    if l is undefined **then return** satisfiable
    $\alpha := \alpha \cup \{l\}$

Pseudo-Code of SDCL (formula F) [HVC 2017]

$\alpha := \emptyset$
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
    $\alpha := \alpha \cup \{l\}$

Benchmark Suite: Pigeon Hole Formulas [HVC 2017]

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

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

The binary clauses encode the constraint $\leq_1 (x_{h,1}; \ldots; x_{h,n+1})$.

There exists more compact encodings, such as the sequential counter and minimal encoding, for at-most-one constraints.

We include these encodings to evaluate the robustness of the solver.

Tool Comparison

We used three tools in our evaluation:

- EBDDRES: A tool based on binary decision diagrams that can convert a refutation into an extended resolution proof.
- GLUCOSER: A SAT solver with extended learning, i.e., a technique that introduces new variables and could potentially solve pigeon hole formulas in polynomial time.
- LINGELING (PR): Our SDCL solver.

Results on Small Pigeon Hole Formulas [HVC 2017]

	input		EBDDRES		GLUCOSER		LINGELING (PR)	
formula	#var	#cls	time	#node	time	#lemma	time	#lemma
PHP_{10}-std	110	561	1.00	3M	22.71	329,470	0.07	329
PHP_{11}-std	132	738	3.47	9M	146.61	1,514,845	0.11	439
PHP_{12}-std	156	949	10.64	27M	307.29	2,660,358	0.16	571
PHP_{13}-std	182	1,197	30.81	76M	982.84	6,969,736	0.22	727
PHP_{10}-seq	220	311	OF	——	1.62	25,712	0.07	327
PHP_{11}-seq	264	375	OF	——	6.94	77,747	0.10	437
PHP_{12}-seq	312	445	OF	——	19.40	174,084	0.14	569
PHP_{13}-seq	364	521	OF	——	172.76	1,061,318	0.18	725
PHP_{10}-min	180	281	28.60	81M	0.64	15,777	0.06	329
PHP_{11}-min	220	342	143.92	399M	1.82	34,561	0.10	439
PHP_{12}-min	264	409	OF	——	9.87	121,321	0.13	571
PHP_{13}-min	312	482	OF	——	57.66	483,789	0.18	727

OF = 32-bit overflow

Results on Large Pigeon Hole Formulas [HVC 2017]

	input		EBDDRES		GLUCOSER		LINGELING (PR)	
formula	#var	#cls	time	#node	time	#lemma	time	#lemma
PHP_{20}-std	420	4,221	OF	——	TO	——	1.61	2,659
PHP_{30}-std	930	13,981	OF	——	TO	——	13.45	8,989
PHP_{40}-std	1,640	32,841	OF	——	TO	——	67.41	21,319
PHP_{50}-std	2,550	63,801	OF	——	TO	——	241.14	41,649
PHP_{20}-seq	840	1,221	OF	——	TO	——	1.05	2,657
PHP_{30}-seq	1,860	2,731	OF	——	TO	——	6.55	8,987
PHP_{40}-seq	3,280	4,841	OF	——	TO	——	27.10	21,317
PHP_{50}-seq	5,100	7,551	OF	——	TO	——	86.30	41,647
PHP_{20}-min	760	1,161	OF	——	TO	——	1.03	2,659
PHP_{30}-min	1,740	2,641	OF	——	TO	——	6.30	8,989
PHP_{40}-min	3,120	4,721	OF	——	TO	——	26.65	21,319
PHP_{50}-min	4,900	7,401	OF	——	TO	——	85.00	41,649

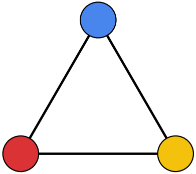
OF = 32-bit overflow TO = timeout of 9000 seconds

One More Thing...

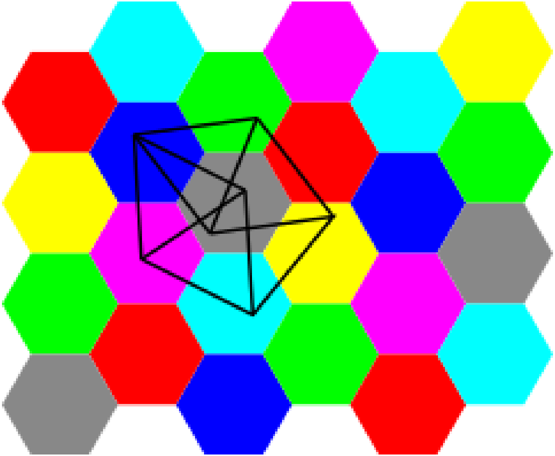
Chromatic Number of the Plane

The Hadwiger-Nelson problem:
How many colors are required to color the plane such that each pair of points that are exactly 1 apart are colored differently?

The answer must be three or more because three points can be mutually 1 apart—and thus must be colored differently.



Bounds since the 1950s



- The Moser Spindle graph shows the lower bound of 4
- A coloring of the plane showing the upper bound of 7

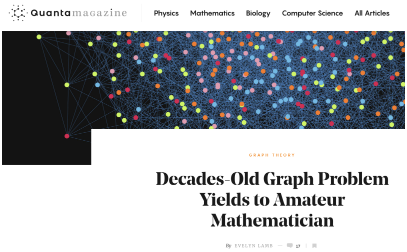
First Progress in Decades [De Grey 2018]

The first meaningful progress on this problem was by Aubrey de Grey, who found a unit-distance graph with chromatic number 5.

He published a graph with 1581 vertices on April 8, 2018.

Aubrey de Grey is known for his research to extend life.

The New Result Started a Media Hype



Quantamagazine | Physics | Mathematics | Biology | Computer Science | All Articles

GRAPH THEORY

Decades-Old Graph Problem Yields to Amateur Mathematician

By EVELYN LAMB — 17

INSIDE 新創 趨勢 娛樂 評論 焦點 訂閱：LINE

趨勢

業餘數學家為一道填色難題帶來突破！

2018/4/26 · TNL · 四色定理、填色難題、數學

本篇來自 關鍵評論網 ，INSIDE 經授權轉載。

1950 年秋天，當時仍在芝加哥大學就讀的數學家尼爾遜（Edward Nelson）對於「四色問題」感到興趣。這個問題是：

N+1 | Астрономия | Физика | Биология | Роботы и дроны

«Движение вверх» | Научные закрытия | Блокчейн | «Мастерская будущего»

Раскраска для математиков

Как покрасить плоскость?

Одна из самых красивых и до сих пор не решенных задач математики формулируется следующим образом. Попытаемся раскрасить плоскость так, чтобы никакие две точки, находящиеся на расстоянии одного сантиметра друг от друга, не оказались покрашены в один цвет. Какое минимальное число цветов для этого потребуется? Несмотря на кажущуюся простоту, за почти 70 лет существования этой задачи точного ответа до сих пор нет, притом что над ним

nrc.nl› abonnementen

Man van 'we leven ooit 1.000 jaar' brengt oplossing oud pixelprobleem dichterbij

Wiskunde

De bekende Britse verouderingswetenschapper Aubrey de Grey heeft zich met succes op een meetkundevraagstuk uit de jaren vijftig van de vorige eeuw gestort. Zijn vondst werd zeer veel besproken op blogs van wiskundigen.

Alex van den Brandhof 24 april 2018

Een gedachtenexperiment: stel je een foto voor waarvan de pixels oneindig klein zijn. Dus hoever je ook inzoomt, de individuele pixels zijn nooit te herkennen, omdat ze oppervlakte nul hebben. Elke pixel - dat zijn er uiteraard oneindig veel - heeft een kleur. De vraag is: hoeveel verschillende kleuren zijn er minimaal nodig om ervoor te zorgen dat twee pixels die op een vaste afstand, bijvoorbeeld 1 centimeter, van elkaar vandaan liggen, nooit dezelfde kleur hebben?

Propositional Proofs for Graph Validation and Shrinking

Checking that a unit-distance graph has chromatic number 5:

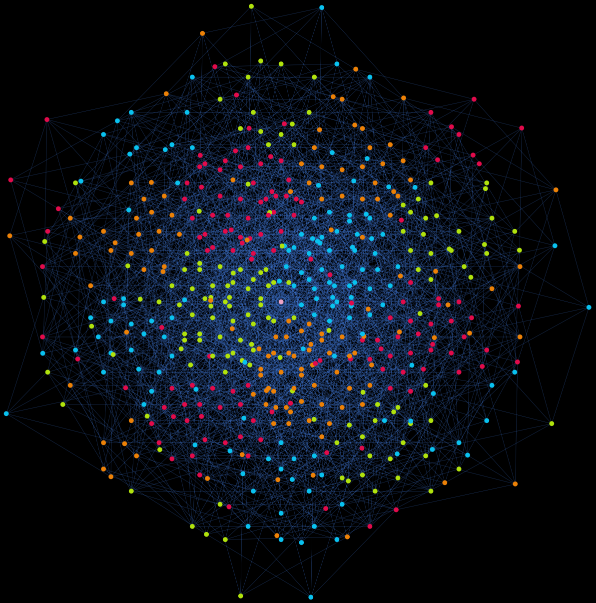
- Show that there exists a 5-coloring
- While there is no 4-coloring (formula is UNSAT)
- Unsatisfiable core represents a subgraph

SAT solvers find short proofs of unsatisfiability for these formulas

Proof minimization techniques allow further reduction

Combining the techniques allows finding much smaller graphs

Record by Proof Minimization: 553 Vertices [Heule 2018]



Challenges and Conclusions

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 SBC 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 SBC 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 SBC 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 SBC 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?

Conclusions

We introduced new redundancy notions for SAT.

Proof systems based on these redundancy notions are strong.

- They allow for short proofs without new variables; and
- They are more suitable for mechanized proof search.

Conclusions

We introduced new redundancy notions for SAT.

Proof systems based on these redundancy notions are strong.

- They allow for short proofs without new variables; and
- They are more suitable for mechanized proof search.

SDCL generalizes the well-known CDCL paradigm by allowing to prune branches that are potentially satisfiable:

- Such branches can be found using the positive reduct;
- Pruning can be expressed in the PR proof system;
- Runtime and proofs can be exponentially smaller.