Clausal Proofs for Pseudo-Boolean Reasoning

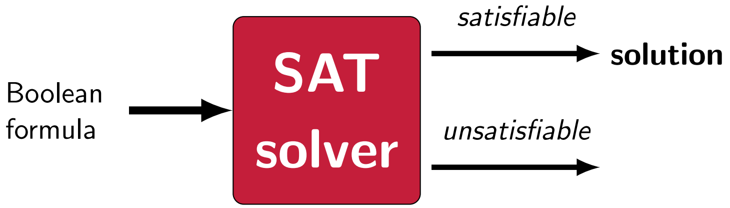
Randal E. Bryant
Carnegie Mellon University

Armin Biere
Albert-Ludwigs University

Marijn J. H. Heule
Carnegie Mellon University

TACAS, 2022

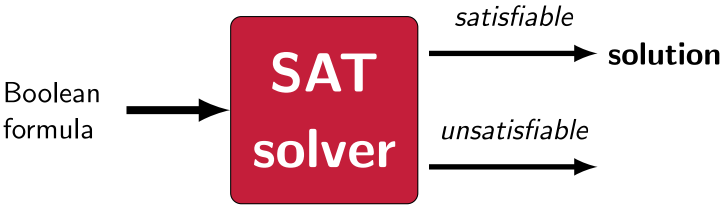
Context: Boolean Satisfiability Solvers



SAT Solvers Useful & Powerful

- Mathematical proofs
- Formal verification
- Optimization

Context: Boolean Satisfiability Solvers



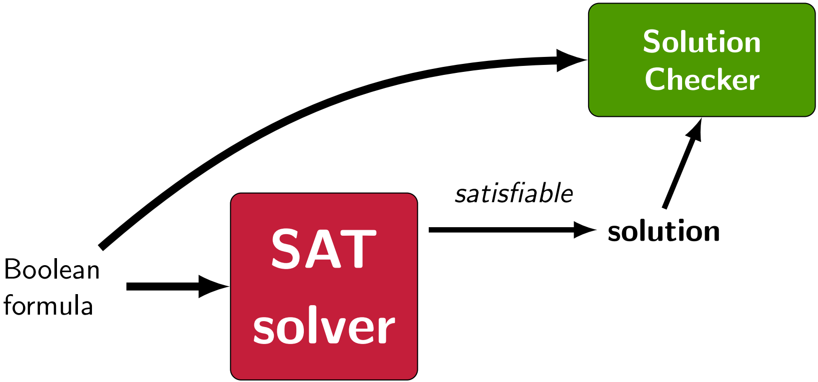
SAT Solvers Useful & Powerful

- Mathematical proofs
- Formal verification
- Optimization

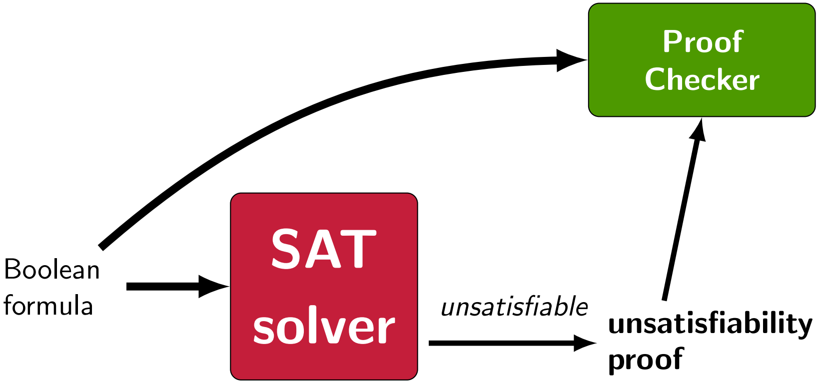
Can We Trust Them?

- No!
- Complex software with lots of optimizations
- KISSAT: 35K LOC

Trustworthy SAT Solvers: Satisfiable Formulas



Trustworthy SAT Solvers: Unsatisfiable Formulas



Checkable Proofs

- Step-by-step proof in standard logical framework
- Independently validated by proof checker

Impact of Proof Checking

Adoption

- Required for SAT competition entrants since 2016

Benefits

- Can clearly judge competition submissions
- Developers have improved quality of their solvers
- Firm foundation for use in mathematical proofs

Impact of Proof Checking

Adoption

- Required for SAT competition entrants since 2016

Benefits

- Can clearly judge competition submissions
- Developers have improved quality of their solvers
- Firm foundation for use in mathematical proofs

Unintended Consequences

- Narrowed focus to single SAT algorithm
  - Conflict-Driven Clause Learning (CDCL)
  - Search for solution, but learn conflicts
- Other powerful solution methods have languished.

Impact of Proof Checking

Adoption

- Required for SAT competition entrants since 2016

Benefits

- Can clearly judge competition submissions
- Developers have improved quality of their solvers
- Firm foundation for use in mathematical proofs

Unintended Consequences

- Narrowed focus to single SAT algorithm
  - Conflict-Driven Clause Learning (CDCL)
  - Search for solution, but learn conflicts
- Other powerful solution methods have languished.

Our Contribution

- Enable proof generation for algorithms based on *pseudo-Boolean reasoning*

Clausal Proofs

Conjunctive Normal Form (CNF) Input Formula

$$C_1, C_2, \ldots, C_m$$

Unsatisfiability Proof

$$C_1, C_2, \ldots, C_m, C_{m+1}, \ldots, C_t$$

- For all $i > m$:
  If $C_1, \ldots, C_{i-1}$ has a satisfying assignment,
  then so does $C_1, \ldots, C_{i-1}, C_i$.
- $C_t = \emptyset$
  - Unsatisfiable

Clausal Proof Frameworks

Resolution (Robinson, 1965)

- Proof rule guarantees *implication redundancy*:

$$\bigwedge_{1 \leq j < i} C_j \;\rightarrow\; C_i$$

Clausal Proof Frameworks

Resolution (Robinson, 1965)

- Proof rule guarantees *implication redundancy*:

$$\bigwedge_{1 \leq j < i} C_j \;\rightarrow\; C_i$$

Extended Resolution (Tseitin, 1967)

- Allow *extension variables*
  - Variable e shorthand for some formula F over input and previous extension variables
  - Add clauses encoding $e \leftrightarrow F$ to proof
- Can make proofs exponentially more compact

Clausal Proof Frameworks

Resolution (Robinson, 1965)

- Proof rule guarantees *implication redundancy*:

$$\bigwedge_{1 \leq j < i} C_j \;\rightarrow\; C_i$$

Extended Resolution (Tseitin, 1967)

- Allow *extension variables*
  - Variable e shorthand for some formula F over input and previous extension variables
  - Add clauses encoding $e \leftrightarrow F$ to proof
- Can make proofs exponentially more compact

Deletion Resolution Asymmetric Tautology (DRAT)

- Superset of extended resolution
- Variety of efficient checkers, including formally verified ones

Proof-Generating Solvers Based on BDDs

Implementations

- EBDDRES: Sinz, Biere, Jussila, 2006
- PGBDD: Bryant, Heule, 2021

Proof-Generating Solvers Based on BDDs

Implementations

- EBDDRES: Sinz, Biere, Jussila, 2006
- PGBDD: Bryant, Heule, 2021

Extended-Resolution Proof Generation

- Introduce extension variable for each BDD node
- Generate proof steps based on recursive structure of BDD algorithms
- Proof is (very) detailed justification of each BDD operation

Proof-Generating Solvers Based on BDDs

Implementations

- EBDDRES: Sinz, Biere, Jussila, 2006
- PGBDD: Bryant, Heule, 2021

Extended-Resolution Proof Generation

- Introduce extension variable for each BDD node
- Generate proof steps based on recursive structure of BDD algorithms
- Proof is (very) detailed justification of each BDD operation

Capabilities

- Can handle some problems that are intractable for CDCL
- Often requires careful guidance from user
- Often very sensitive to variable ordering

Proof-Generating Solvers Based on BDDs

Generate Sequence of Terms

$$T_1, T_2, \ldots, T_m, T_{m+1}, \ldots, T_p$$

- Each term T_i is Boolean function represented by BDD
- For $1 \leq i \leq m$, T_i is BDD representation of clause C_i
- For $i > m$, term T_i generated as conjunction or existential quantification of earlier terms:

$$\bigwedge_{1 \leq j < i} T_j \;\rightarrow\; T_i$$

- Final term $T_p = \bot$.

Proof Structure

- Prove that initial terms represent clauses
- Prove that implication holds for each successive term.

Pseudo-Boolean (PB) Formulas

- Integer Equations

$$\sum_{1 \leq i \leq n} a_i \, x_i \quad = \quad b$$

  - a_i, b integer constants
  - x_i 0-1 valued variables

- Ordering Constraints

$$\sum_{1 \leq i \leq n} a_i \, x_i \quad \geq \quad b$$

- Modular Equations

$$\sum_{1 \leq i \leq n} a_i \, x_i \quad \equiv \quad b \pmod{r}$$

  - r constant modulus
  - Parity constraints: $r = 2$

Incorporating Pseudo-Boolean Reasoning into SAT Solver

- Motivation: CDCL tends to do poorly on PB constraints

Parity Reasoning

- Detect CNF encodings of XOR/XNOR
- Apply Gaussian elimination over GF2
- E.g., Lingeling, CryptoMiniSAT
- Useful for both SAT and UNSAT problems

Incorporating Pseudo-Boolean Reasoning into SAT Solver

- Motivation: CDCL tends to do poorly on PB constraints

Parity Reasoning

- Detect CNF encodings of XOR/XNOR
- Apply Gaussian elimination over GF2
- E.g., Lingeling, CryptoMiniSAT
- Useful for both SAT and UNSAT problems

Constraint Reasoning

- Detect standard encodings of ordering constraints
- Apply Fourier-Motzin elimination over integers
- E.g., Lingeling
- Only useful for UNSAT problems

Incorporating Pseudo-Boolean Reasoning into SAT Solver

- Motivation: CDCL tends to do poorly on PB constraints

Parity Reasoning

- Detect CNF encodings of XOR/XNOR
- Apply Gaussian elimination over GF2
- E.g., Lingeling, CryptoMiniSAT
- Useful for both SAT and UNSAT problems

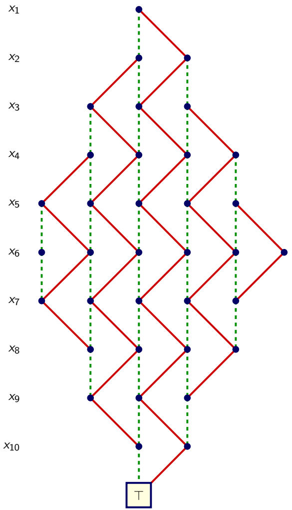
Constraint Reasoning

- Detect standard encodings of ordering constraints
- Apply Fourier-Motzin elimination over integers
- E.g., Lingeling
- Only useful for UNSAT problems

Proof Generation

- No previous solver could generate clausal proof
- Revert to CDCL when proof generation required

Representing Pseudo-Boolean Equations with BDDs



- Example equation:

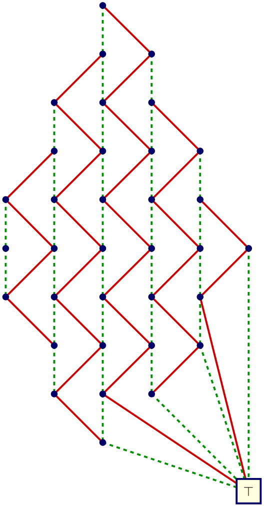
$$\begin{array}{ll} +x_1 + x_3 + x_5 + x_7 + x_9 & \\ -x_2 - x_4 - x_6 - x_8 - x_{10} & = \quad 0 \end{array}$$

- BDD size $\leq a_{\max} \cdot n^2$

$$a_{\max} \quad = \quad \max_{1 \leq i \leq n} |a_i|$$

  - Independent of variable ordering

Representing Ordering Constraints with BDDs



- Example constraint:

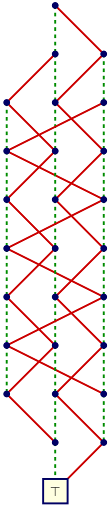
$$\begin{array}{l} +x_1 + x_3 + x_5 + x_7 + x_9 \\ -x_2 - x_4 - x_6 - x_8 - x_{10} \end{array} \geq 0$$

- BDD size $\leq a_{\max} \cdot n^2$

$$a_{\max} = \max_{1 \leq i \leq n} |a_i|$$

  - Independent of variable ordering

Representing Modular Equations with BDDs

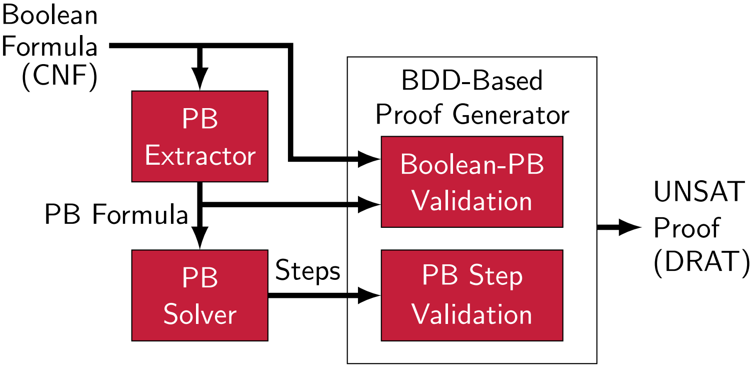


- Example equation:

$$\begin{array}{l} +x_1 + x_3 + x_5 + x_7 + x_9 \\ -x_2 - x_4 - x_6 - x_8 - x_{10} \end{array} \equiv 0 \pmod 3$$

- BDD size $\leq n \cdot r$
  - Independent of variable ordering

Integrating Pseudo-Boolean Reasoning into Proof-Generating SAT Solver



- Overall flow same as SAT solver
- PB solver does all of the reasoning
- BDDs serve only as mechanism for generating clausal proof

PGPBS (Proof-Generating Pseudo-Boolean Solver)

Implementation

- Augmented version of earlier solver PGBDD
- https://github.com/rebryant/pgpbs-artifact

Constraint Extraction

- CNF file input
- Detects PB constraints:
  - Equations: XOR/XNOR, Exactly-one
  - Ordering constraints: At-most-one, At-least-one
- Including ones using auxilliary variables
- Heuristic methods
- Generates schedule
  - How clauses grouped into constraints
  - Existentially quantify auxilliary variables

Integer Gaussian Elimination

System of Equations $E = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m\}$

$$\mathbf{e}_i : \quad \sum_{j=1,n} a_{i,j}\, x_j = b_i$$

Elimination Step

1. Choose pivot equation $\mathbf{e}_s$ and variable x_t such that $a_{s,t} \neq 0$
2. For each $i \neq s$:

$$\mathbf{e}_i \quad \leftarrow \quad \begin{cases} \mathbf{e}_i & a_{i,t} = 0 \\ -a_{i,t} \cdot \mathbf{e}_s + a_{s,t} \cdot \mathbf{e}_i, & a_{i,t} \neq 0 \end{cases}$$

   - Guarantees $a_{i,t} = 0$ for all $i \neq s$
   - Only requires addition and multiplication

3. Remove $\mathbf{e}_s$ from E and repeat until single equation left

Gaussian Elimination Results

Possible Outcomes

1. If encounter degenerate equation
   - Of form $0 = b$ for $b \neq 0$.
   - Has no solution
   - Occurs for problems we consider
2. Otherwise, if modular equation with $r = 2$
   - Can perform back substitution to find solution
3. Otherwise
   - Generated solution may not be 0-1 valued

Gaussian Elimination Results

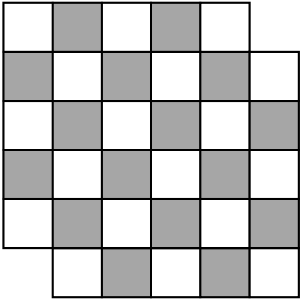
Possible Outcomes

1. If encounter degenerate equation
   - Of form $0 = b$ for $b \neq 0$.
   - Has no solution
   - Occurs for problems we consider
2. Otherwise, if modular equation with $r = 2$
   - Can perform back substitution to find solution
3. Otherwise
   - Generated solution may not be 0-1 valued

Validating Each Step:

- Given BDDs representing term functions T_{i_1} and T_{i_2}
- Validate $T_{i_1} \wedge T_{i_2} \rightarrow T_{i_1} + T_{i_2}$
- Use proof-generating BDD operations

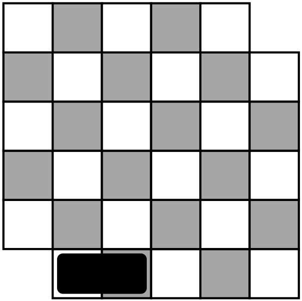
Mutilated Chessboard Problem



Definition

- $N \times N$ chessboard with 2 corners removed
- Cover with tiles, each covering two squares

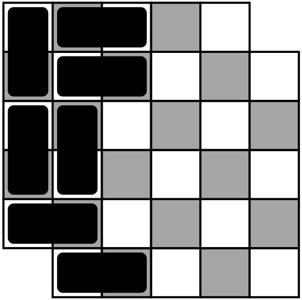
Mutilated Chessboard Problem



Definition

- $N \times N$ chessboard with 2 corners removed
- Cover with tiles, each covering two squares

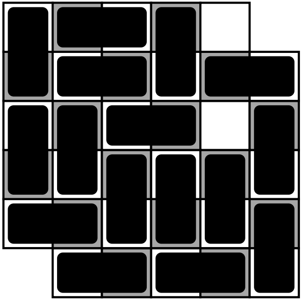
Mutilated Chessboard Problem



Definition

- $N \times N$ chessboard with 2 corners removed
- Cover with tiles, each covering two squares

Mutilated Chessboard Problem



Definition

- $N \times N$ chessboard with 2 corners removed
- Cover with tiles, each covering two squares

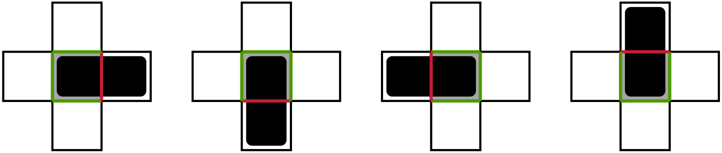
Solutions

- None
- More white squares than black
- Each tile covers one white and one black square

Proof

- All resolution proofs of exponential size

Encoding as SAT Problem

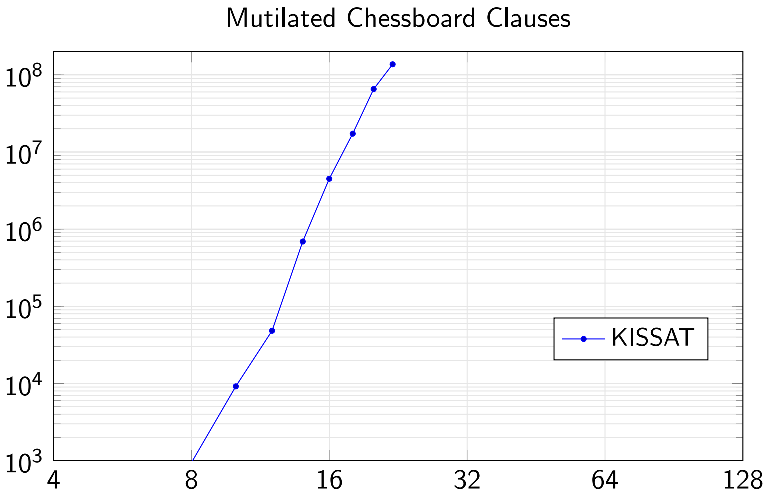


Boolean variable for each possible domino placement

Constraints

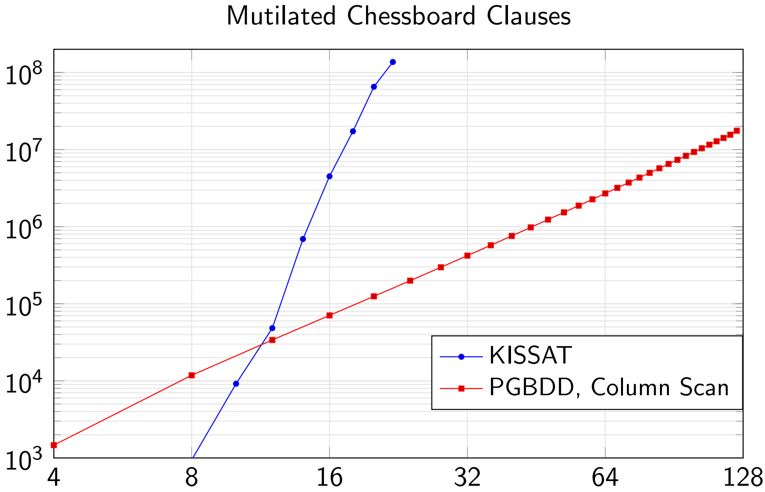
- For each square, exactly one of its covering placements $= 1$

Chess Proof Complexity: KISSAT



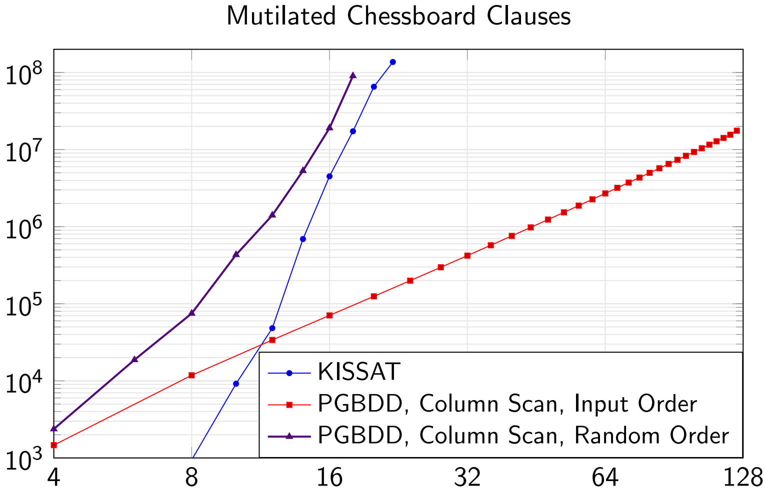
- Requires 12.6 hours for $N = 22$.
- Express complexity as number of clauses in generated proof

Chess Proof Complexity: Column Scanning (TACAS '21)



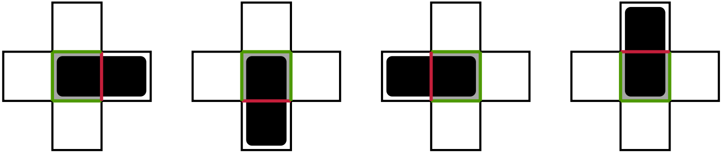
- Careful ordering of conjunction and quantification operations
- Scan columns, representing partial solutions with $O(N^2)$ nodes

Chess Proof: BDD Variable Ordering Sensitivity



- Column scanning highly dependent on variable ordering
- Also requires careful user guidance

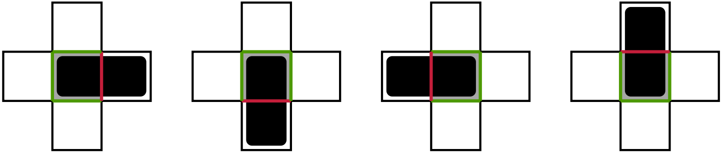
Pseudo-Boolean Solving of Mutilated Chessboard



- For every square i, j:

$$x_{E(i,j)} + x_{S(i,j)} + x_{W(i,j)} + x_{N(i,j)} \quad = \quad 1$$

Pseudo-Boolean Solving of Mutilated Chessboard



- For every square i, j:

$$x_{E(i,j)} + x_{S(i,j)} + x_{W(i,j)} + x_{N(i,j)} \quad = \quad 1$$

- Sum equations for white squares:

$$\sum_{x \in X} x \quad = \quad N^2/2$$

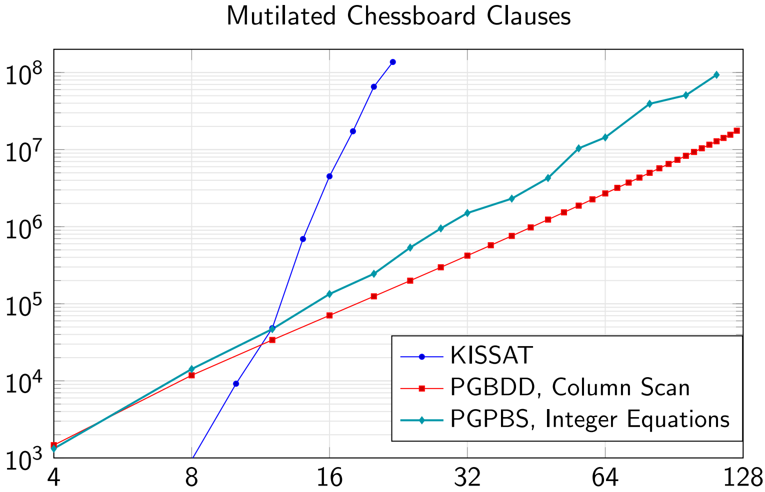
- Sum equations for black squares:

$$\sum_{x \in X} x \quad = \quad N^2/2 - 2$$

- Difference:

$$0 \quad = \quad 2$$

Chess Proof Complexity: Integer Equations



- Integer equations less efficient than column scanning
- But, insensitive to variable ordering; no user guidance required

Modulus Autodetection

- Apply Gaussian elimination to system of integer equations
  - Only requires multiplication and addition
- Encounter equation $0 = b$
- Observation:
  - If performed arithmetic modulo r, would get equation

$$0 \equiv b \pmod{r}$$

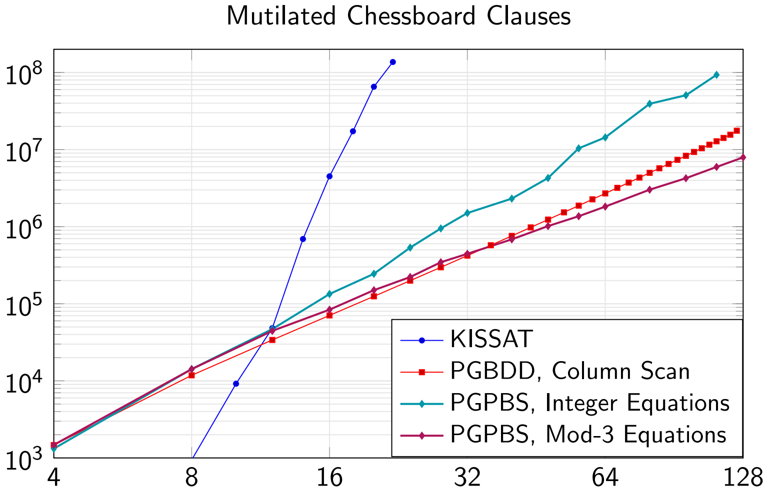
Modulus Autodetection

- Apply Gaussian elimination to system of integer equations
  - Only requires multiplication and addition
- Encounter equation $0 = b$
- Observation:
  - If performed arithmetic modulo r, would get equation

$$0 \equiv b \pmod{r}$$

- Generate proof when solving as system of modular equations
  - Choose least r such that $b \not\equiv 0 \pmod{r}$.
  - More efficient, since BDDs smaller
  - Totally automated

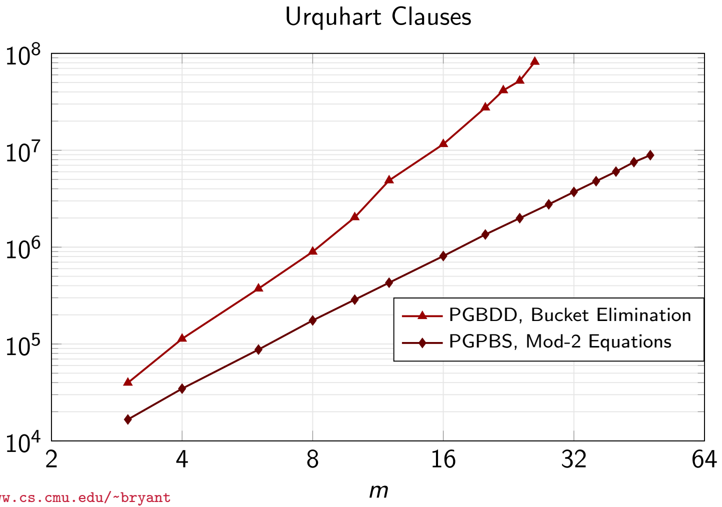
Chess Proof Complexity: Modular Equations



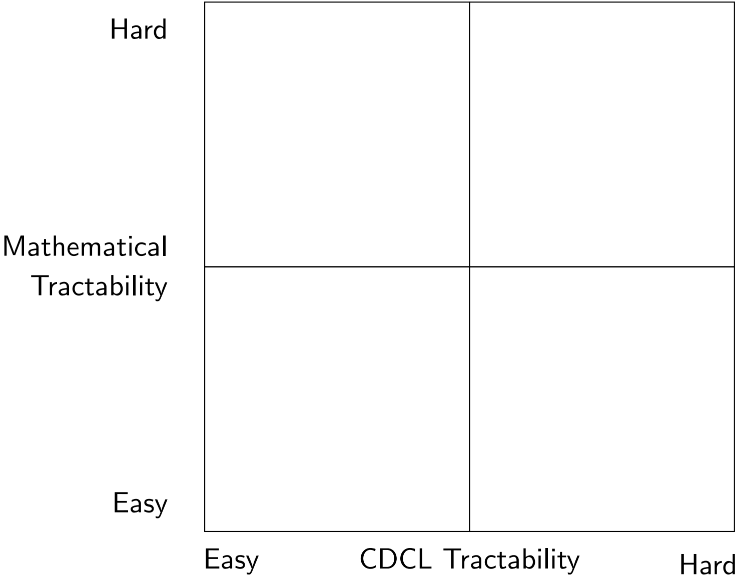
- Modular equations outperform column scanning
- Insensitive to variable ordering; no user guidance required

Urquhart Parity Benchmark (Li's Version)

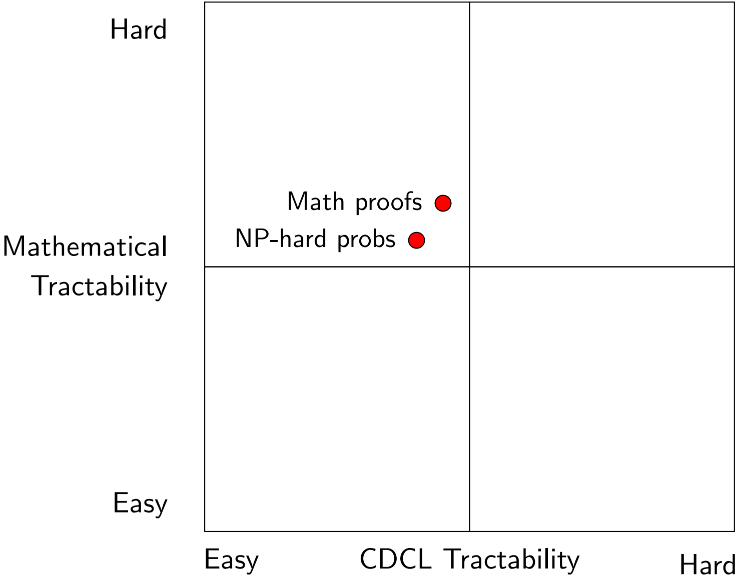
- Set of XOR constraints defined over graph with $2m^2$ nodes.
- KISSAT cannot solve even minimal instance ($m = 3$)
- Trivial with Gaussian elimination



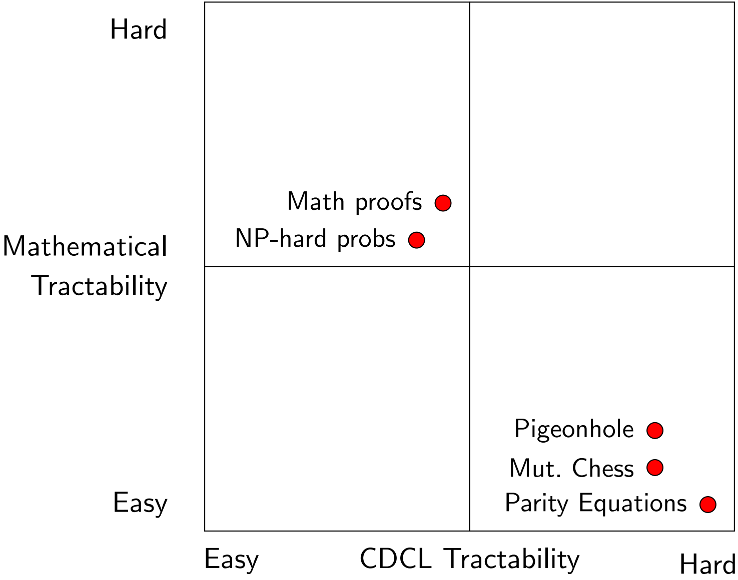
A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



Summary

Role of BDDs in SAT

- As primary reasoning method
  - Handle problems intractable for CDCL
  - Difficult to achieve full automation
- To enable proof generation for other reasoning methods
  - BDD algorithms expressed as extended-resolution proofs
  - Fully automated
  - Insensitive to variable ordering

Summary

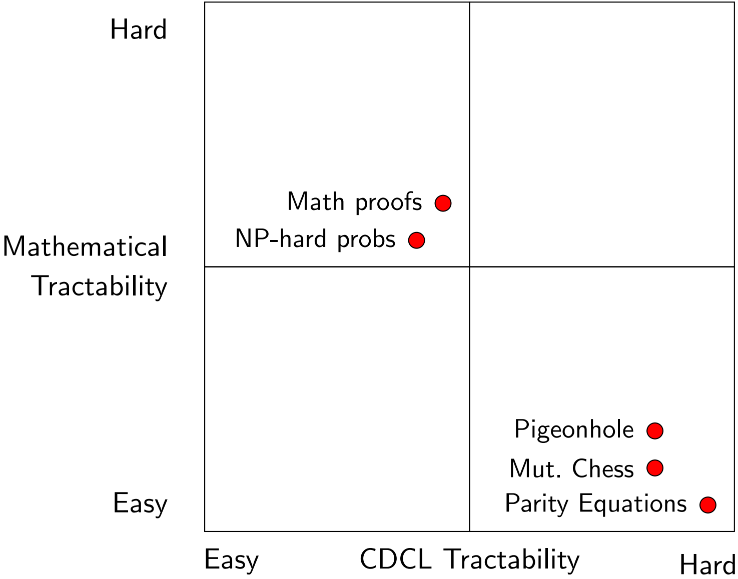
Role of BDDs in SAT

- As primary reasoning method
  - Handle problems intractable for CDCL
  - Difficult to achieve full automation
- To enable proof generation for other reasoning methods
  - BDD algorithms expressed as extended-resolution proofs
  - Fully automated
  - Insensitive to variable ordering

Future Work: Combine Multiple Approaches

- CDCL, BDDs, pseudo-Boolean reasoning, . . .
- Build on unique strengths of each
- Must be able to generate clausal proof

A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving