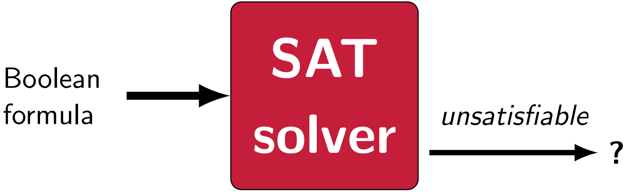
Generating Extended Resolution Proofs with a BDD-Based SAT Solver

Randal E. Bryant and Marijn J. H. Heule

Carnegie Mellon University

TACAS, 2021

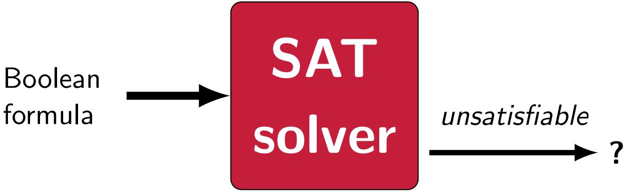
Boolean Satisfiability Solvers



SAT Solvers Useful & Powerful

- Formal verification
- Security verification
- Optimization

Boolean Satisfiability Solvers



SAT Solvers Useful & Powerful

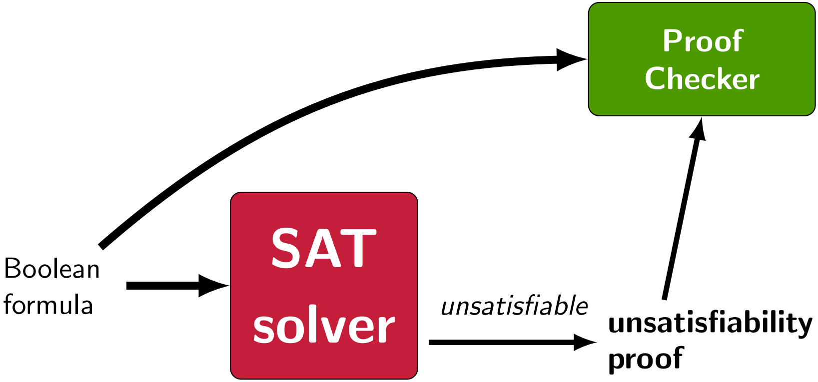
- Formal verification
- Security verification
- Optimization

Can We Trust Them?

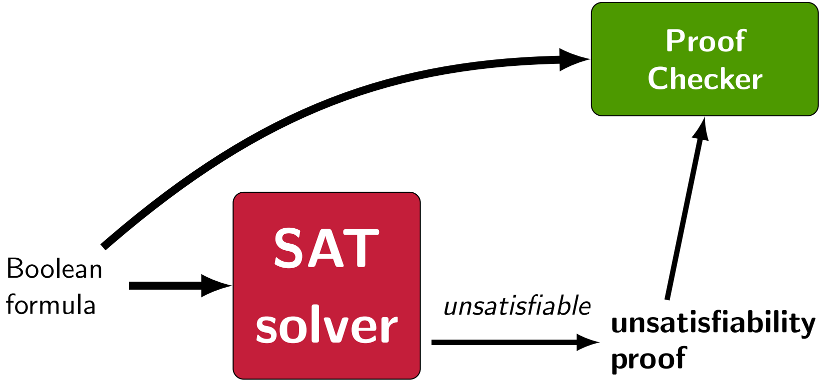
- No!
- Complex software with lots of optimizations

Proof Generating Solvers

Proof Generating Solvers



Proof Generating Solvers



Unsatisfiability Proof

- Step-by-step proof in some logical framework

Proof Checker

- Simple program
- May be formally verified

Background

- 2006 Sinz, Biere generate proofs with BDD-based SAT solver (conjunctions only)
- 2006 Jussila, Sinz, Biere allow limited use of existential quantification
- 2021 Bryant and Heule allow arbitrary existential quantification

Basics

Clauses

- $\neg u \vee v \vee w$ Disjunction of literals
- $\bot$ Empty clause (False)

Resolution Principle

$$\frac{\neg u \vee v \vee w \qquad \neg w \vee x \vee \neg z}{(\neg u \vee v) \vee (x \vee \neg z)}$$

- Generalization of implication
- See https://en.wikipedia.org/wiki/Resolution_(logic)

Clausal Proof

Step	Clause	Antecedents	Formula	
1	$\neg v \vee w$		$v \rightarrow w$	Input clauses
2	$\neg v \vee \neg w$		$v \rightarrow \neg w$	
3	v		v	
4	$\neg v$	$1, 2$	$\neg v$	Derived clauses
5	$\perp$	$3, 4$	$v \wedge \neg v$	

- Prove conjunction of input clauses unsatisfiable
- Add derived clauses
  - Provides list of antecedent clauses that resolve to new clause
- Finish with empty clause
  - Proof is series of inferences leading to contradiction

Extended Resolution (ER)

Can introduce extension variables

- Variable e that has not yet occurred in proof
- Must add *defining* clauses
  - Encode constraint of form $e \leftrightarrow F$
  - Boolean formula F over input and earlier extension variables

Extension variable e becomes shorthand for formula F

- Repeated use can yield exponentially smaller proof

Extended Resolution Example

Example: Prove following set of constraints unsatisfiable

Constraint	Clauses
$u \wedge v \rightarrow w$	$\neg u \vee \neg v \vee w$
$u \wedge v \rightarrow \neg w$	$\neg u \vee \neg v \vee \neg w$
$u \wedge v$	u
	v

- Strategy: Introduce extension variable e such that $e \leftrightarrow u \wedge v$

Constraint	Clauses
$u \wedge v \rightarrow e$	$e \vee \neg u \vee \neg v$
$e \rightarrow u$	$\neg e \vee u$
$e \rightarrow v$	$\neg e \vee v$

Extended Resolution Proof

Step	Clause	Antecedents	Formula	
1	$\neg u \vee \neg v \vee w$		$u \wedge v \rightarrow w$	
2	$\neg u \vee \neg v \vee \neg w$		$u \wedge v \rightarrow \neg w$	Input clauses
3	u		u	
4	v		v	
5	$e \vee \neg u \vee \neg v$		$u \wedge v \rightarrow e$	
6	$\neg e \vee u$		$e \rightarrow u$	Defining clauses
7	$\neg e \vee v$		$e \rightarrow v$	
8	$\neg e \vee \neg v \vee w$	1, 6	$e \wedge v \rightarrow w$	
9	$\neg e \vee w$	7, 8	$e \rightarrow w$	
10	$\neg e \vee \neg v \vee \neg w$	2, 6	$e \wedge v \rightarrow \neg w$	
11	$\neg e \vee \neg w$	7, 10	$e \rightarrow \neg w$	
12	$e \vee \neg v$	3, 5	$v \rightarrow e$	Derived clauses
13	e	4, 12	e	
14	$\neg e$	9, 11	$\neg e$	
15	$\bot$	13, 14	$e \wedge \neg e$	

Reduced, Ordered Binary Decision Diagrams (BDDs)

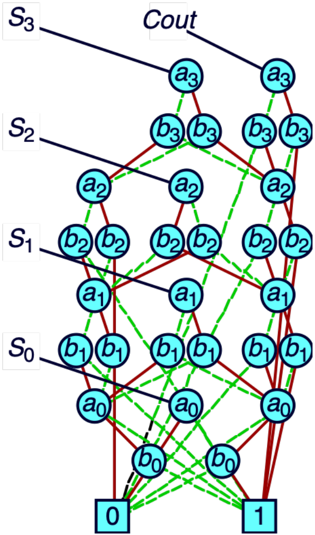
- Bryant, 1986

Representation

- Canonical representation of Boolean function
- Compact for many useful cases

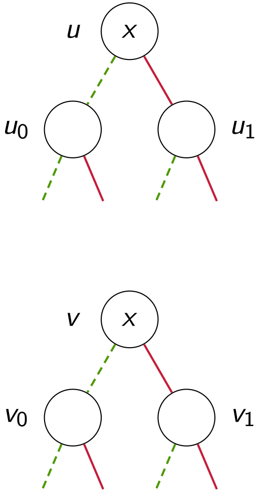
Algorithms

- Apply(f, g, op)
  - *op* is Boolean operation (e.g., $\wedge$, $\vee$)
  - BDD representation of $f \; op \; g$
- EQuant(f, X)
  - X set of variables
  - BDD representation of $\exists X f$



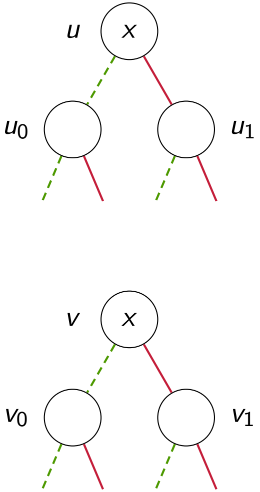
Apply Algorithm Recursion

$\mathsf{Apply}(u, v, \wedge)$



Apply Algorithm Recursion

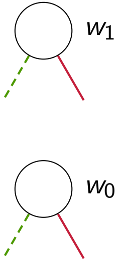
$\text{Apply}(u, v, \wedge)$



Recursion

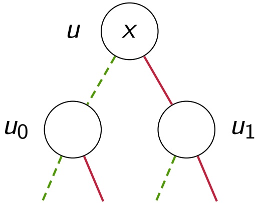
$\text{Apply}(u_1, v_1, \wedge) \;\rightarrow$

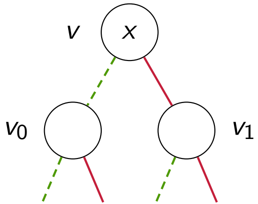
$\text{Apply}(u_0, v_0, \wedge) \;\rightarrow$



Apply Algorithm Recursion

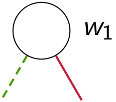
$\mathrm{Apply}(u, v, \wedge)$



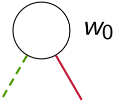


Recursion

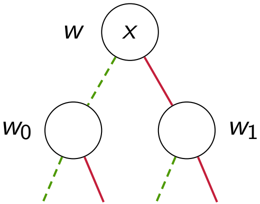
$\mathrm{Apply}(u_1, v_1, \wedge) \rightarrow$



$\mathrm{Apply}(u_0, v_0, \wedge) \rightarrow$

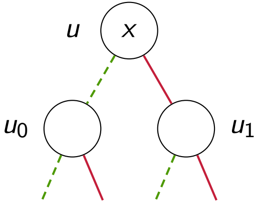


Result



Generating ER Proofs

- Create extension variable for each node in BDD
  - Notation: Same symbol for node and its extension variable



- Defining clauses encode constraint $u \leftrightarrow \mathrm{ITE}(x, u_1, u_0)$

Clause name	Formula	Clausal form
HD(u)	$x \rightarrow (u \rightarrow u_1)$	$\neg x \vee \neg u \vee u_1$
LD(u)	$\neg x \rightarrow (u \rightarrow u_0)$	$x \vee \neg u \vee u_0$
HU(u)	$x \rightarrow (u_1 \rightarrow u)$	$\neg x \vee \neg u_1 \vee u$
LU(u)	$\neg x \rightarrow (u_0 \rightarrow u)$	$x \vee \neg u_0 \vee u$

Proof-Generating Apply Operation

Integrate Proof Generation into Apply Operation

- When $\text{Apply}(u, v, \wedge)$ returns w, also generate proof $u \wedge v \rightarrow w$
- **Key Idea:** Proof based on the underlying logic of the Apply algorithm

Proof Structure

- Assume recursive calls generate proofs
  - $u_1 \wedge v_1 \rightarrow w_1$
  - $u_0 \wedge v_0 \rightarrow w_0$
- Combine with defining clauses for nodes u, v, and w

Apply Proof Structure

Defining Clauses

Clause	Formula	Clause	Formula
HD(u)	$x \rightarrow (u \rightarrow u_1)$	LD(u)	$\neg x \rightarrow (u \rightarrow u_0)$
HD(v)	$x \rightarrow (v \rightarrow v_1)$	LD(v)	$\neg x \rightarrow (v \rightarrow v_0)$
HU(w)	$x \rightarrow (w_1 \rightarrow w)$	LU(w)	$\neg x \rightarrow (w_0 \rightarrow w)$

Resolution Steps

$$
\dfrac{\dfrac{\begin{array}{l} x \rightarrow (u \rightarrow u_1) \\ x \rightarrow (v \rightarrow v_1) \\ x \rightarrow (w_1 \rightarrow w) \quad \color{blue}{u_1 \wedge v_1 \rightarrow w_1} \end{array}}{x \rightarrow (u \wedge v \rightarrow w)} \qquad \dfrac{\begin{array}{l} \neg x \rightarrow (u \rightarrow u_0) \\ \neg x \rightarrow (v \rightarrow v_0) \\ \neg x \rightarrow (w_0 \rightarrow w) \quad \color{blue}{u_0 \wedge v_0 \rightarrow w_0} \end{array}}{\neg x \rightarrow (u \wedge v \rightarrow w)}}{u \wedge v \rightarrow w}
$$

Quantification Operations

Operation EQuant(f, X)

- Abstract away details of satisfying (partial) solutions
- Not logically required for SAT solver
  - But, critical for obtaining good performance

Proof Generation

- Do not attempt to follow recursive structure of algorithm
- Instead, follow with separate implication proof generation
  - EQuant$(u, X) \rightarrow w$
  - Generate proof $u \rightarrow w$
  - Algorithm similar to proof-generating Apply operation

Overall Proof Task

Input Variables

Input Clauses

- Set of input clauses C_I over the input variables

Completion

- Generate Proof $C_I \vdash \bot$

Structure of Overall Proof

Input Variables

- Generate BDD variable for each input variable

Input Clauses

- For each input clause $C \in C_I$, generate BDD representation u
  - Using Apply with $\vee$ operation
- Generate proof $C \vdash u$
  - Sequence of resolution steps based on linear structure of BDD

Combine Top-Level BDDs

- Apply$(u, v, \wedge) \rightarrow w$
  - Combine proofs $C_I \vdash u$, $C_I \vdash v$ and $u \wedge v \rightarrow w$ to get $C_I \vdash w$
- EQuant$(u, X) \rightarrow w$
  - Combine proofs $C_I \vdash u$ and $u \rightarrow w$ to get $C_I \vdash w$

Completion

- When Apply$(u, v, \wedge) \rightarrow 0$ have proof $C_I \vdash \bot$

Implementation

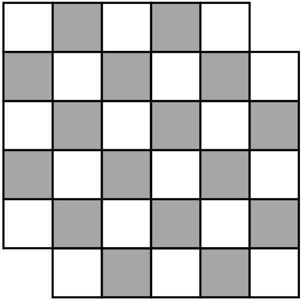
Package

- 2000 lines Python code (slow!)
- BDD package + proof generator
- `https://github.com/rebryant/pgbdd-artifact`

Benchmark Generators

- CNF file
- File specifying ordering of variables
- File specifying schedule:
  - Defines sequence of conjunctions and quantifications

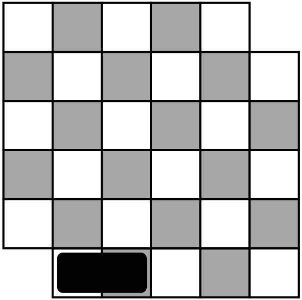
Mutilated Chessboard Problem



Definition

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

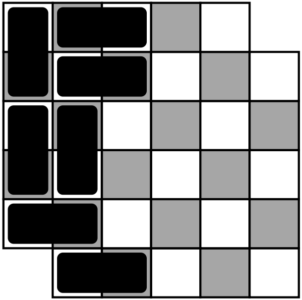
Mutilated Chessboard Problem



Definition

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

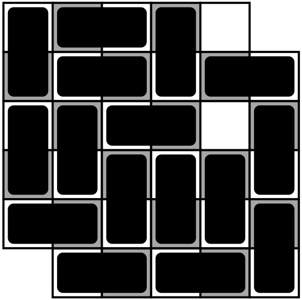
Mutilated Chessboard Problem



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

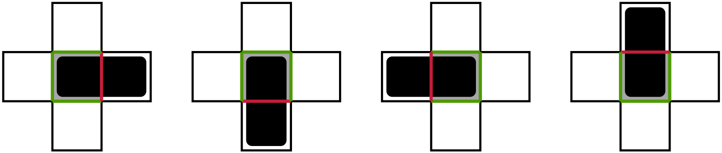
Solutions

- 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



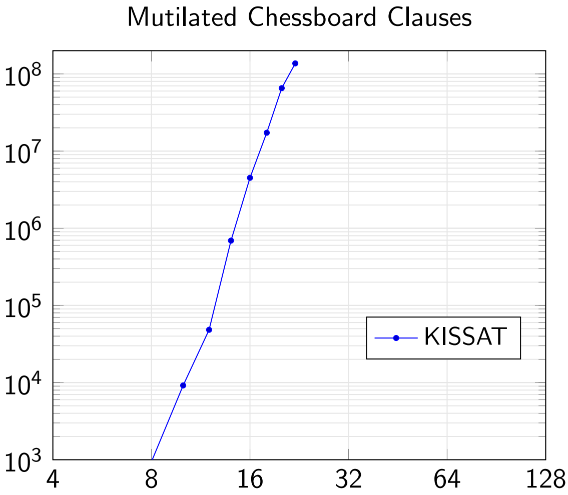
Boolean variable for each boundary between two squares

- $(N-1) \cdot N - 2$ vertical boundaries $x_{i,j}$
- $(N-1) \cdot N - 2$ horizontal boundaries $y_{i,j}$

Constraints

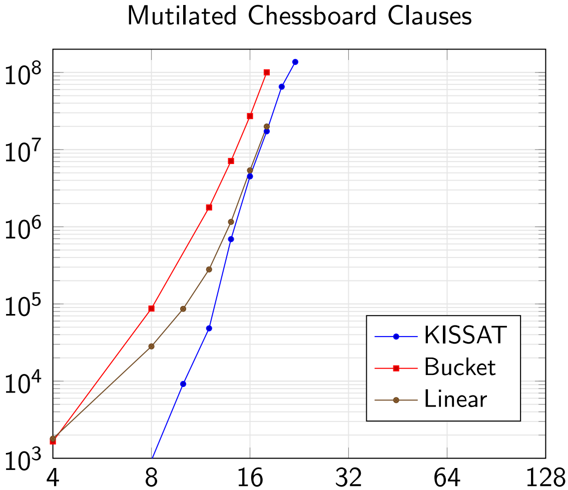
- For each square, exactly one of its boundary variables $= 1$

Chess Proof Complexity: KISSAT



- Winner of 2020 SAT competition
- Requires 12.6 hours for $N = 22$.

Chess Proof Complexity: Earlier BDD-Based Approaches



- Linear: No quantification (Sinz & Biere, 2006)
- Bucket: Eliminate variables from top of BDD downward (Jussila, Sinz, & Biere, 2006)

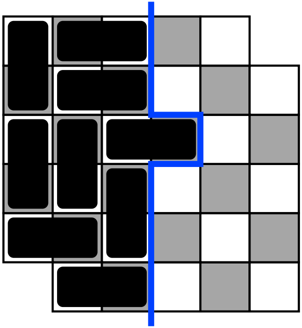
Column Scanning

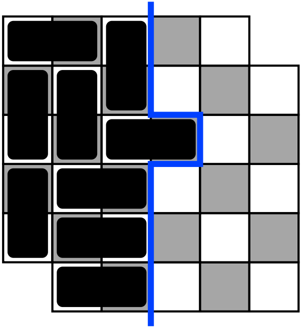
Scanning

- Add tiles for each column from left to right

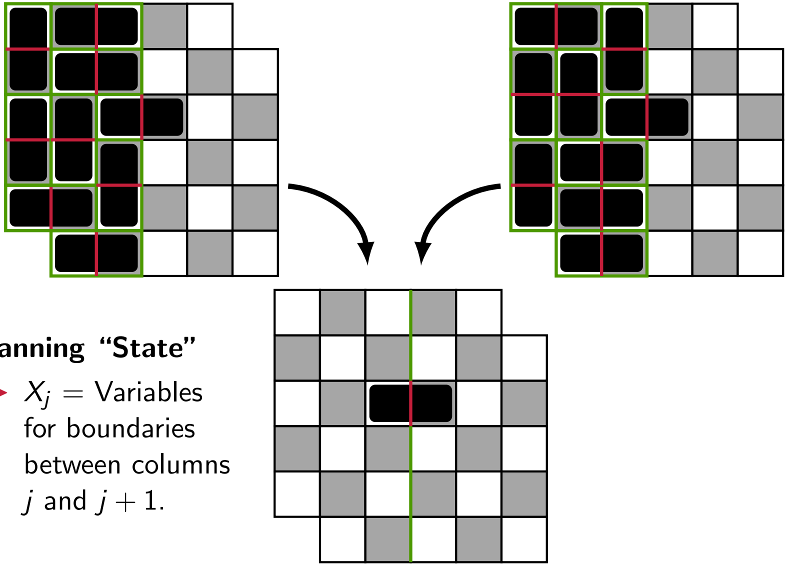
Observation

- When placing tiles in column, only need to know which squares are already occupied





Abstraction Via Quantification

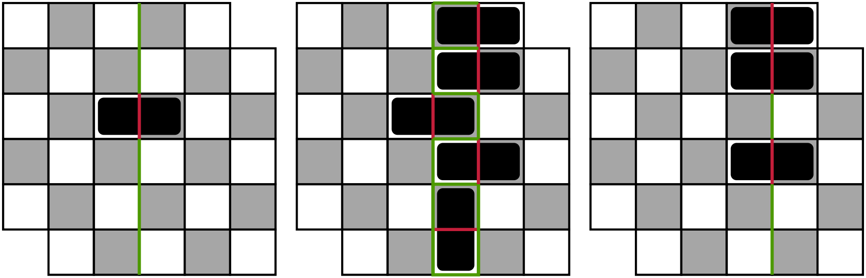


Scanning "State"

- X_j = Variables for boundaries between columns j and $j+1$.

Symbolic Computation of State Sets

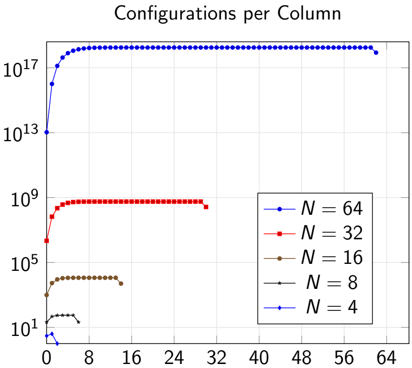
State at column j-1	Column j transition	State at column j
$\sigma_{j-1}(X_{j-1})$	$T_j(X_{j-1}, Y_j, X_j)$	$\sigma_j(X_j)$

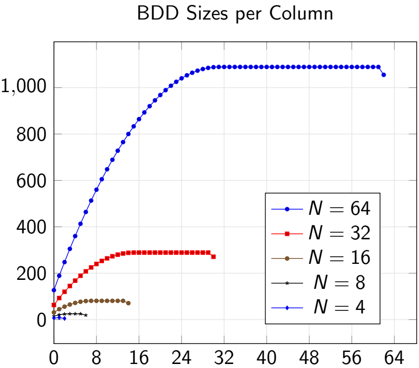


$$\sigma_j(X_j) = \exists X_{j-1}[\sigma_{j-1}(X_{j-1}) \wedge \exists Y_j \, T_j(X_{j-1}, Y_j, X_j)]$$

- Does not redefine underlying problem
- Way to order conjunctions and quantifications
- Requires quantification ordering to differ from BDD variable ordering

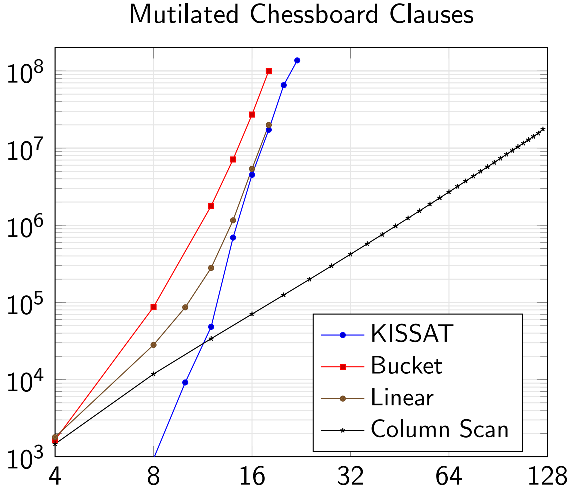
Representing State Sets





- Number of configurations $\sim 2^N$
- BDD representation $\sim N^2$

Chess Proof Complexity: Column Scanning



- Problem size $\sim N^2$
- Proof size $\sim N^{2.7}$

Observations

Key Insight

- Sinz, Biere, and Jussila
- Capture underlying logic of BDD algorithms as ER proofs

Our Contributions

- Handle arbitrary existential quantification
  - Required for column scanning
- Demonstrate on key benchmarks
  - Mutilated chessboard
    - Complexity $O(N^{2.7})$
  - Pigeonhole principle
    - Complexity $O(N^{3})$
    - Also based on column scanning

Further Work

Higher Performance Implementation

- Extend existing BDD package

More Automation

- Variable ordering
- Conjunction and quantification scheduling

Apply to Other Problems

- QBF
- Model checking
- Model counting