The Power of Extended Resolution

A Practitioner's Perspective

Randal E. Bryant

Carnegie Mellon University

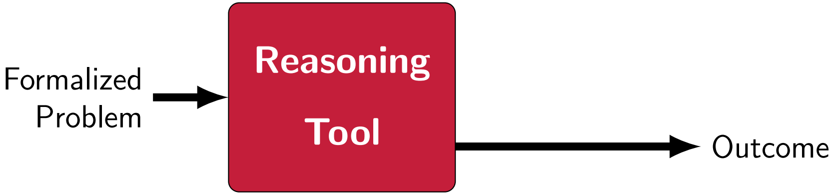
Simons Institute, 2023

http://www.cs.cmu.edu/~bryant

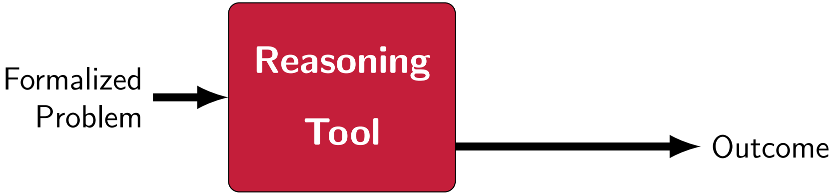
Background: My Research in “Formal” Verification

- 1990 *Formal* Verification of Digital Circuits Using Symbolic Ternary System Models
- 1991 *Formal* Hardware Verification by Symbolic Simulation
- 1991 *Formal* Verification of Memory Circuits by Switch-Level Simulation
- 1994 *Formally* Verifying a Microprocessor using a Simulation Methodology
- 1996 *Formal* Verification of PowerPC(TM) Arrays using Symbolic Trajectory Evaluation
- 1997 *Formal* Verification of a Superscalar Execution Unit
- 1998 *Formal* Verification of Pipelined Processors
- 1999 *Formal* Verification of an ARM Processor
- 2006 *Formal* Verification of Infinite State Systems Using Boolean Methods

Automated Reasoning Programs



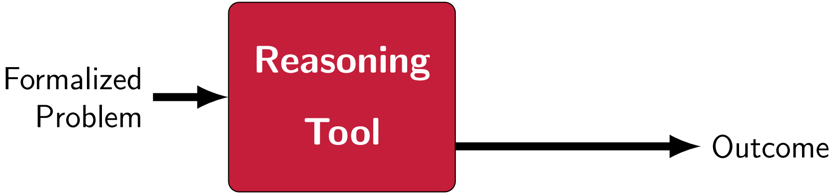
Automated Reasoning Programs



Standard Tools

- Lingering doubt about whether result can be trusted
- If find bug in tool, must rerun all prior verifications

Automated Reasoning Programs



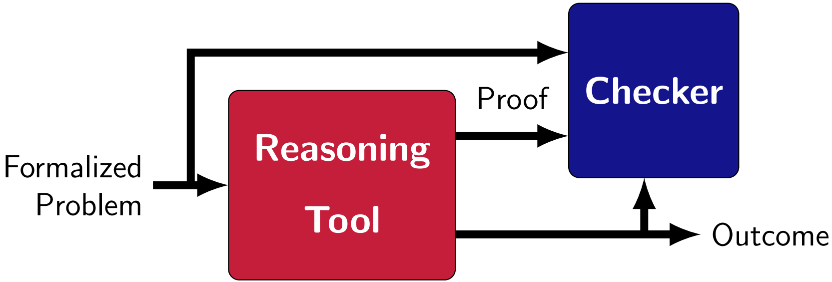
Standard Tools

- Lingering doubt about whether result can be trusted
- If find bug in tool, must rerun all prior verifications

Formally Verified Tools

- Hard to develop
- Hard to make scalable

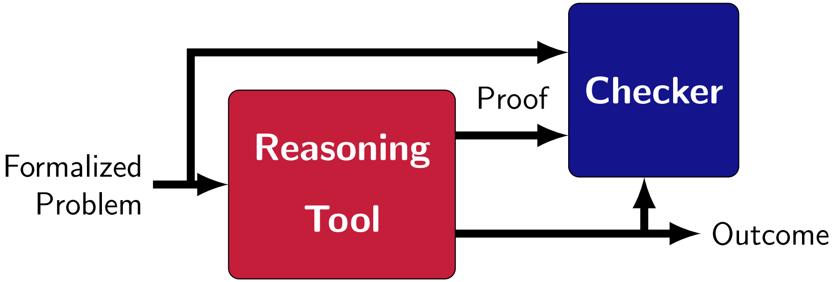
Proof-Generating Automated Reasoning Programs



Proof-Generating Tools

- Only need to prove individual executions, not entire program
- Can have bugs in tool but still trust result

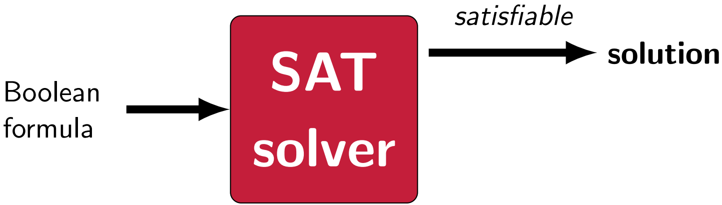
Proof-Generating Automated Reasoning Programs



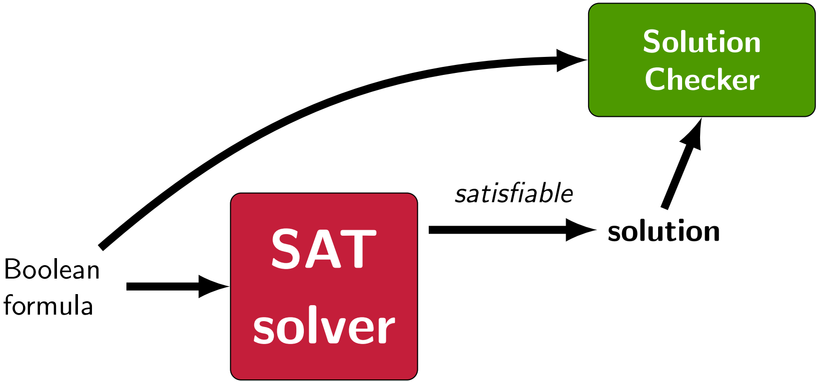
Proof-Generating Tools

- Only need to prove individual executions, not entire program
- Can have bugs in tool but still trust result
- Can we trust the checker?
  - Simple algorithms and implementation
  - Possibly formally verified

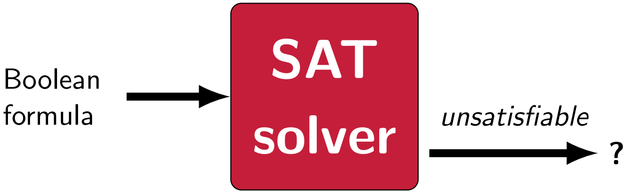
Boolean Satisfiability Solvers



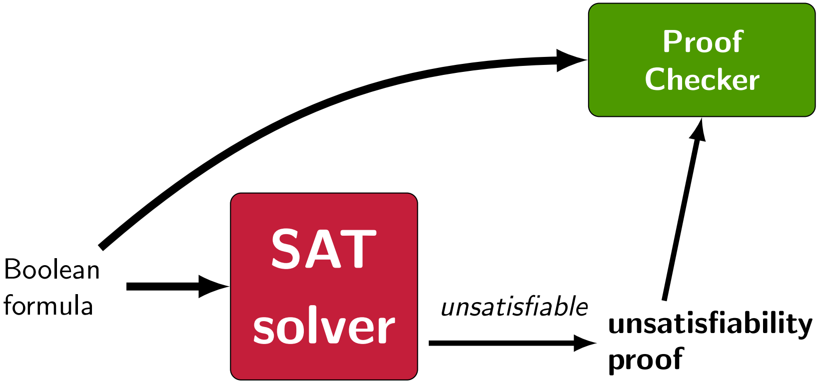
Boolean Satisfiability Solvers



Boolean Satisfiability Solvers



Proof Generating Solvers



Adoption of Proof Checking by SAT Community

Some History

2003 Proof generation added to zChaff [ZhaMal-2003] and BerkMin [GolNov-2003]

2013 Proof framework and checker well matched to CDCL solvers [HeuHunWet-2013]

2016 Proof generation mandatory for SAT competition main track

Adoption of Proof Checking by SAT Community

Some History

2003 Proof generation added to zChaff [ZhaMal-2003] and BerkMin [GolNov-2003]

2013 Proof framework and checker well matched to CDCL solvers [HeuHunWet-2013]

2016 Proof generation mandatory for SAT competition main track

Impact

- 2022 SAT Competition
  - No main track entrant reported UNSAT on satisfiable problem
  - Even on new benchmark formulas
- Tool developers alerted to bugs early in development
- Has enabled implementation of more complex & risky optimizations

Talk Overview

Add Proof Support to Other Forms of Automated Reasoning

Basics

- (Extended) resolution
- Clausal proofs

Binary Decision Diagrams (BDDs) and Proof Generation

- BDDs and extended resolution
- Supporting other Boolean reasoning methods

Certified Knowledge Compilation

- Partitioned-Operation Graphs (POGs)
- Equivalence proofs

Basics

Clauses

- $[\overline{u} \vee v \vee w]$ Disjunction of literals
- $\perp$ Empty clause (False)

Resolution Principle

- Robinson, 1965

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

Basics

Clauses

- $[\overline{u} \vee v \vee w]$ Disjunction of literals
- $\perp$ Empty clause (False)

Resolution Principle

- Robinson, 1965

$$(u \wedge \overline{v}) \rightarrow w \qquad\qquad w \rightarrow (x \vee \overline{z})$$

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

Basics

Clauses

- $[\overline{u} \vee v \vee w]$ Disjunction of literals
- $\bot$ Empty clause (False)

Resolution Principle

- Robinson, 1965

$$(u \wedge \overline{v}) \rightarrow w \qquad\qquad w \rightarrow (x \vee \overline{z})$$

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

$$(u \wedge \overline{v}) \rightarrow (x \vee \overline{z})$$

Clausal Proof Systems

Input Formula: Set of Clauses

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

Clausal Proof (of Unsatisfiability):

$$C_{m+1}, C_{m+2}, \ldots, C_t$$

- Preserves satisfiability: For $i \geq m$:

  If $\{C_1, C_2, \ldots, C_i\}$ is satisfiable

  then $\{C_1, C_2, \ldots, C_i, C_{i+1}\}$ is satisfiable
- $C_t = \bot$

Clausal Proof Systems

Input Formula: Set of Clauses

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

Clausal Proof (of Unsatisfiability):

$$C_{m+1}, C_{m+2}, \ldots, C_t$$

- Preserves satisfiability: For $i \geq m$:

If	$\{C_1, C_2, \ldots, C_i\}$	is satisfiable
then	$\{C_1, C_2, \ldots, C_i, C_{i+1}\}$	is satisfiable

- $C_t = \bot$
- Resolution rule preserves solutions:

$$\bigwedge_{1 \leq j \leq i} C_j \quad \Longrightarrow \quad C_{i+1}$$

Clausal Proof Example

Step	Clause	Antecedents	Formula	
1	$[\overline{v} \vee w]$		$v \rightarrow w$	Input clauses
2	$[\overline{v} \vee \overline{w}]$		$v \rightarrow \overline{w}$	
3	$[v]$		v	
4	$[\overline{v}]$	$1, 2$	$\overline{v}$	Derived clauses
5	$\perp$	$3, 4$	$v \wedge \overline{v}$	

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

Extended Resolution

- Tseitin, 1967

Can introduce extension variables

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

Extension variable becomes shorthand for larger formula

- Through repeated application, can have exponentially smaller proof

Extended Resolution Example

Example: Prove following set of constraints unsatisfiable

Constraint	Clauses
$u \wedge v \rightarrow w$	$[\overline{u} \vee \overline{v} \vee w]$
$u \wedge v \rightarrow \overline{w}$	$[\overline{u} \vee \overline{v} \vee \overline{w}]$
$u \wedge v$	$[u]$
	$[v]$

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

Constraint	Clauses
$u \wedge v \rightarrow e$	$[e \vee \overline{u} \vee \overline{v}]$
$e \rightarrow u$	$[\overline{e} \vee u]$
$e \rightarrow v$	$[\overline{e} \vee v]$

ER Proof

Step	Clause	Antecedents	Formula	
1	$[\overline{u} \lor \overline{v} \lor w]$		$u \land v \rightarrow w$	Input clauses
2	$[\overline{u} \lor \overline{v} \lor \overline{w}]$		$u \land v \rightarrow \overline{w}$	
3	$[u]$		u	
4	$[v]$		v	
5	$[e \lor \overline{u} \lor \overline{v}]$		$u \land v \rightarrow e$	Defining clauses
6	$[\overline{e} \lor u]$		$e \rightarrow u$	
7	$[\overline{e} \lor v]$		$e \rightarrow v$	
8	$[\overline{e} \lor \overline{v} \lor w]$	1, 6	$e \land v \rightarrow w$	Derived clauses
9	$[\overline{e} \lor w]$	7, 8	$e \rightarrow w$	
10	$[\overline{e} \lor \overline{v} \lor \overline{w}]$	2, 6	$e \land v \rightarrow \overline{w}$	
11	$[\overline{e} \lor \overline{w}]$	7, 10	$e \rightarrow \overline{w}$	
12	$[e \lor \overline{v}]$	3, 5	$v \rightarrow e$	
13	$[e]$	4, 12	e	
14	$[\overline{e}]$	9, 11	$\overline{e}$	
15	$\bot$	13, 14	$e \land \overline{e}$	

ER Proof

Step	Clause	Antecedents	Formula
1	$[\overline{u} \vee \overline{v} \vee w]$		$u \wedge v \rightarrow w$
2	$[\overline{u} \vee \overline{v} \vee \overline{w}]$		$u \wedge v \rightarrow \overline{w}$
3	$[u]$		u
4	$[v]$		v
5	$[e \vee \overline{u} \vee \overline{v}]$		$u \wedge v \rightarrow e$
6	$[\overline{e} \vee u]$		$e \rightarrow u$
7	$[\overline{e} \vee v]$		$e \rightarrow v$
8	$[\overline{e} \vee \overline{v} \vee w]$	1, 6	$e \wedge v \rightarrow w$
9	$[\overline{e} \vee w]$	7, 8	$e \rightarrow w$
10	$[\overline{e} \vee \overline{v} \vee \overline{w}]$	2, 6	$e \wedge v \rightarrow \overline{w}$
11	$[\overline{e} \vee \overline{w}]$	7, 10	$e \rightarrow \overline{w}$
12	$[e \vee \overline{v}]$	3, 5	$v \rightarrow e$
13	$[e]$	4, 12	e
14	$[\overline{e}]$	9, 11	$\overline{e}$
15	$\bot$	13, 14	$e \wedge \overline{e}$

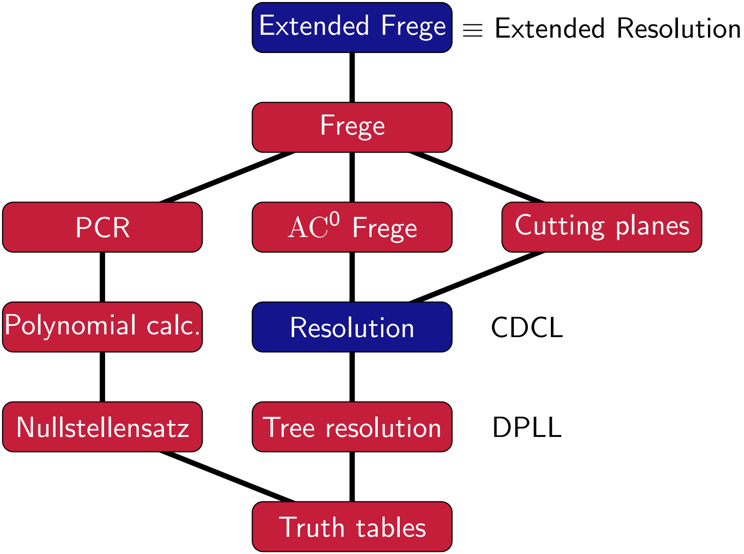
Input clauses (steps 1–4)

Defining clauses (steps 5–7)

Derived clauses (steps 8–15)

$u \wedge v$ replaced by e (steps 9, 11, 13)

Proof Complexity Hierarchy



The Power of (Extended) Resolution

Resolution

- Very weak

Implications for CDCL Solvers

- (Almost) every inference step can be expressed as polynomial number of resolution proof steps
- Exception: Bounded variable addition requires extension variables

Extended Resolution

- Can simulate all other known propositional proof systems
- No known class of formulas with superpolynomial lower bound

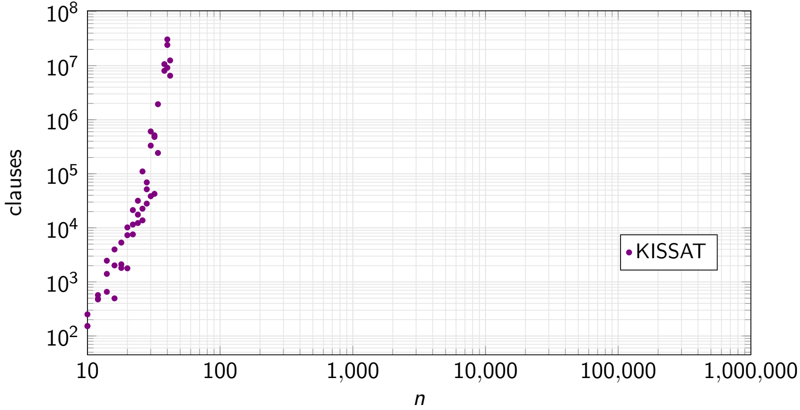
Parity Benchmark

- Chew and Heule, SAT 2020
- For random permtuation π:

$$\begin{array}{ccccccccc} x_1 & \oplus & x_2 & \oplus & \cdots & \oplus & x_n & = & 1 & \text{Odd parity} \\ x_{\pi(1)} & \oplus & x_{\pi(2)} & \oplus & \cdots & \oplus & x_{\pi(n)} & = & 0 & \text{Even parity} \end{array}$$

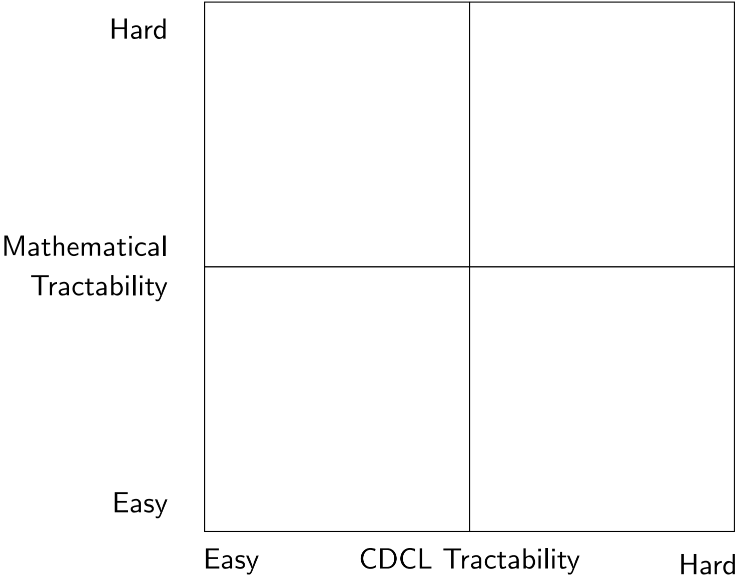
- Encode each equation in CNF
  - $n - 3$ auxiliary variables
  - Linear sequence of 3-argument parity constraints
- Conjunction unsatisfiable
- Very challenging for CDCL solvers

Chew-Heule Parity Benchmark Proof Sizes

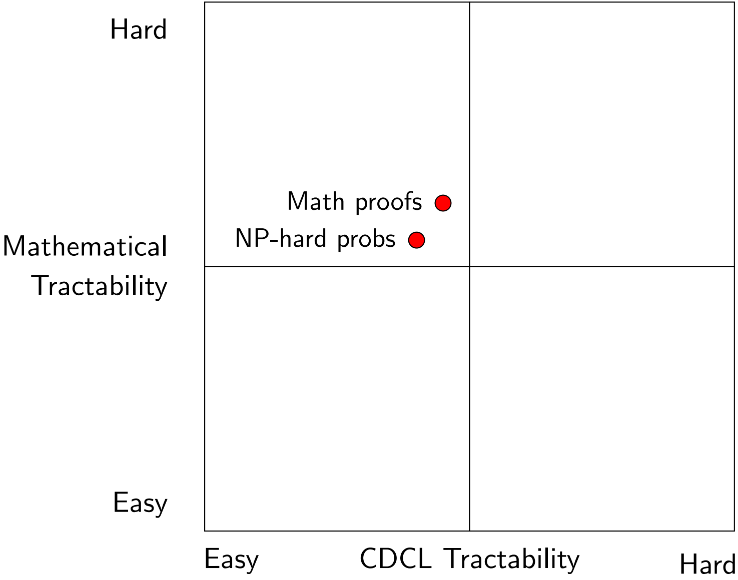


- KISSAT: State-of-the-art CDCL solver
- 3 different seeds for each value of n
- Cannot get beyond $n = 42$ within 600 seconds

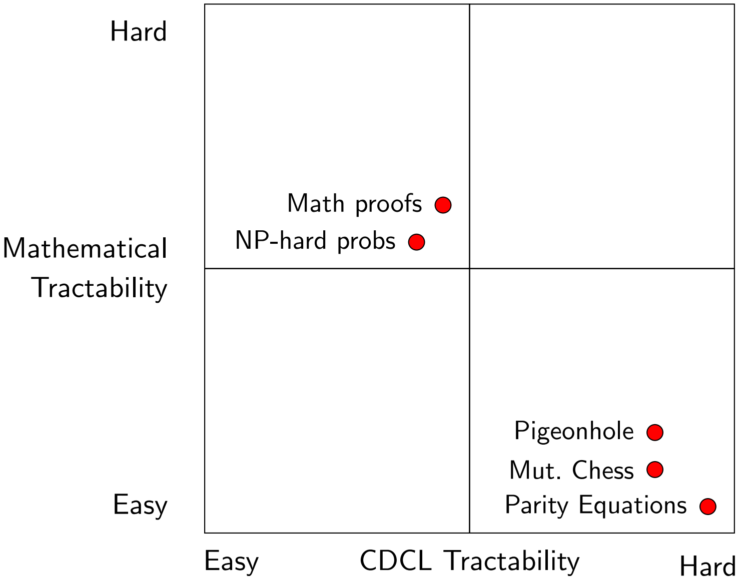
A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving

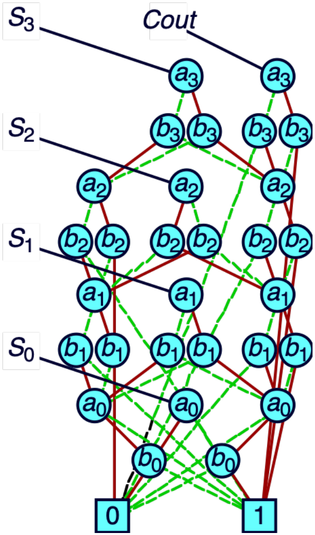


Reduced, Ordered Binary Decision Diagrams (BDDs)

- Bryant [Bry-1986]

Representation

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



Proof-Generating SAT Solvers Based on BDDs

Implementations

- EBDDRES: Sinz, Biere, Jussila, 2006

  [SinBie-2006, JusSinBie-2006]
- PGBDD: Bryant, Heule, 2021 [BryHeu-2021]
- TBUDDY: Bryant [Bry-2022]

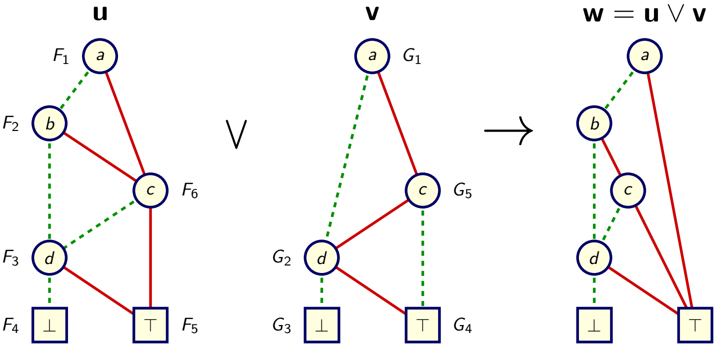
Extended-Resolution Proof Generation

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

BDD Apply Algorithm

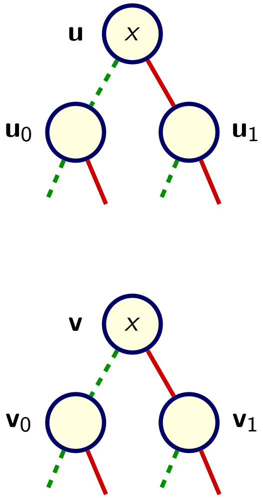
$$\mathbf{w} \quad \leftarrow \quad \mathbf{u} \odot \mathbf{v}$$

- **u**, **v**, **w** BDD root nodes representing Boolean functions
- $\odot$ binary Boolean operator
  - E.g., $\wedge$, $\vee$, $\oplus$



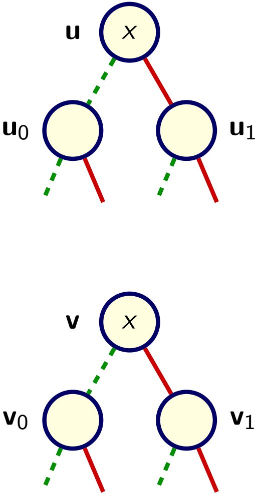
Apply Algorithm Recursion

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



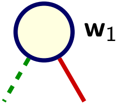
Apply Algorithm Recursion

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

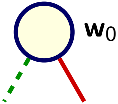


Recursion

$\text{Apply}(\mathbf{u}_1, \mathbf{v}_1, \wedge) \rightarrow$

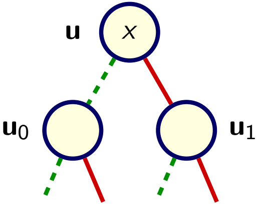


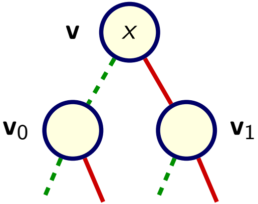
$\text{Apply}(\mathbf{u}_0, \mathbf{v}_0, \wedge) \rightarrow$



Apply Algorithm Recursion

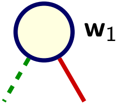
$\text{Apply}(\mathbf{u}, \mathbf{v}, \wedge)$



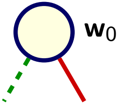


Recursion

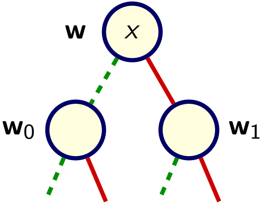
$\text{Apply}(\mathbf{u}_1, \mathbf{v}_1, \wedge) \;\rightarrow$



$\text{Apply}(\mathbf{u}_0, \mathbf{v}_0, \wedge) \;\rightarrow$

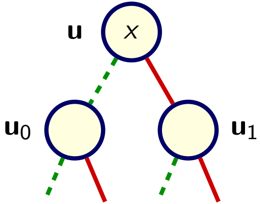


Result



Generating Extended Resolution Proofs

- Extension variable u for each node $\mathbf{u}$ in BDD



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

Clause name	Formula	Clausal form
HD($\mathbf{u}$)	$x \rightarrow (u \rightarrow u_1)$	$[\overline{x} \vee \overline{u} \vee u_1]$
LD($\mathbf{u}$)	$\overline{x} \rightarrow (u \rightarrow u_0)$	$[x \vee \overline{u} \vee u_0]$
HU($\mathbf{u}$)	$x \rightarrow (u_1 \rightarrow u)$	$[\overline{x} \vee \overline{u}_1 \vee u]$
LU($\mathbf{u}$)	$\overline{x} \rightarrow (u_0 \rightarrow u)$	$[x \vee \overline{u}_0 \vee u]$

Proof-Generating Apply Operation

Integrate Proof Generation into Apply Operation

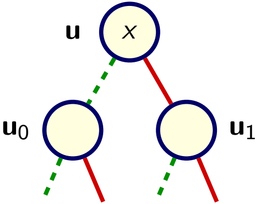
- Apply($\mathbf{u}, \mathbf{v}, \wedge$) returns $\mathbf{w}$
- Also generate proof $u \wedge v \rightarrow w$

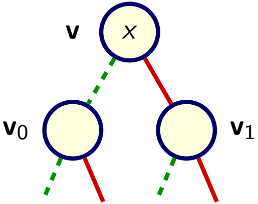
Key Idea:

Proof follows recursion of the Apply *algorithm*

Apply Algorithm Recursion

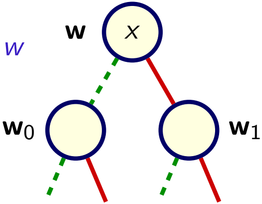
$\text{Apply}(\mathbf{u}, \mathbf{v}, \wedge)$





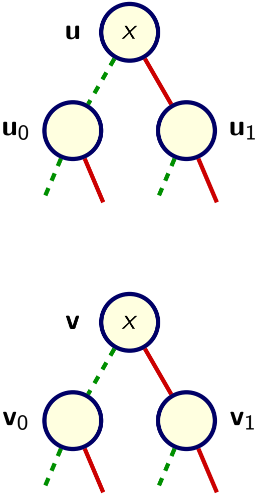
Result

$u \wedge v \rightarrow w$



Apply Algorithm Recursion

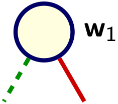
Apply($\mathbf{u}, \mathbf{v}, \wedge$)



Recursion

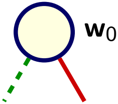
Apply($\mathbf{u}_1, \mathbf{v}_1, \wedge$) $\rightarrow$

$u_1 \wedge v_1 \rightarrow w_1$

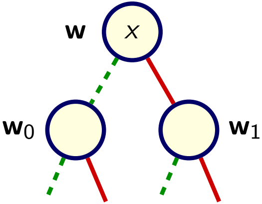


Apply($\mathbf{u}_0, \mathbf{v}_0, \wedge$) $\rightarrow$

$u_0 \wedge v_0 \rightarrow w_0$



Result



Apply Proof Structure

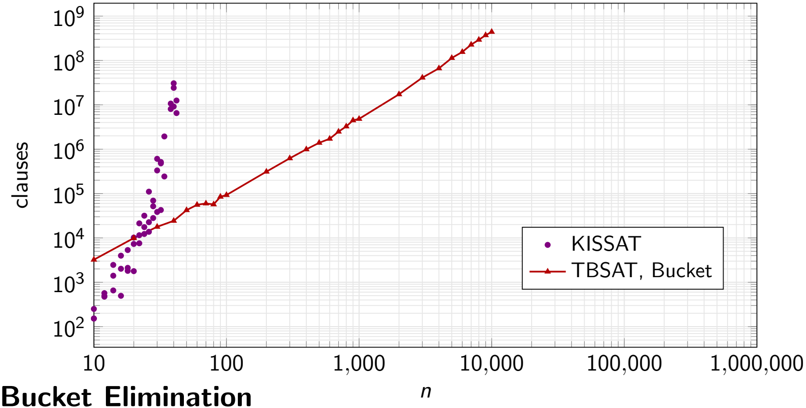
Defining Clauses

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

Resolution Steps

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

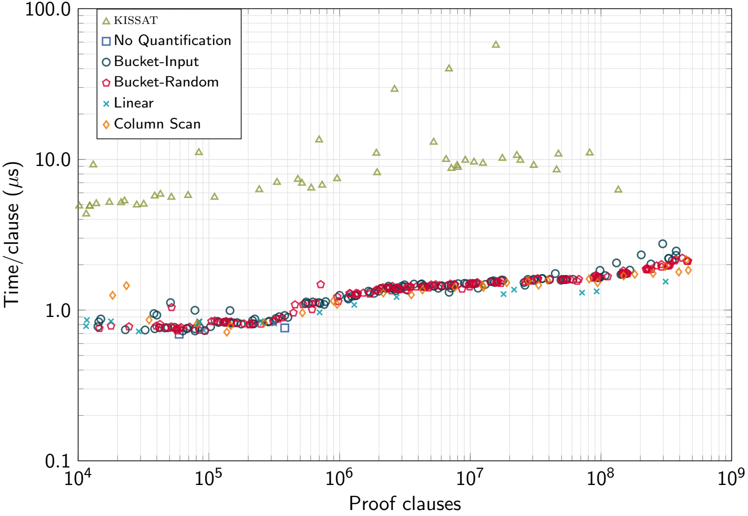
Chew-Heule Parity Benchmark Proof Sizes



Bucket Elimination

- Generate BDD representations of clauses
- Systematically form conjunctions and quantify out variables
  - Each recursive step generates up to 6 proof clauses
- Unsatisfiable formula generates BDD leaf node ⊥

CDCL Proofs vs. BDD Proofs



- CDCL proof step indicates reduction in search space
- BDD proof steps justify algorithmic steps

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

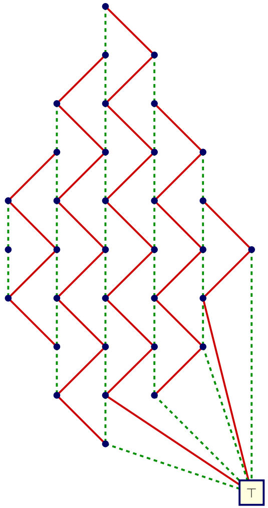
$$\sum_{1 \leq i \leq n} a_i \, x_i \quad \geq \quad b$$

- Modular Equations

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

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

Representing PB Ordering Constraints with BDDs



- Example constraint:

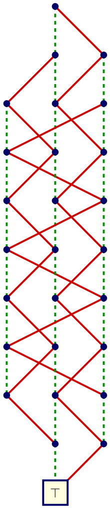
$$\begin{array}{l} +x_1 + x_3 + x_5 + x_7 + x_9 \\ -x_2 - x_4 - x_6 - x_8 - x_{10} \end{array} \geq 0$$

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

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

  - Independent of variable ordering

Representing PB Modular Equations with BDDs

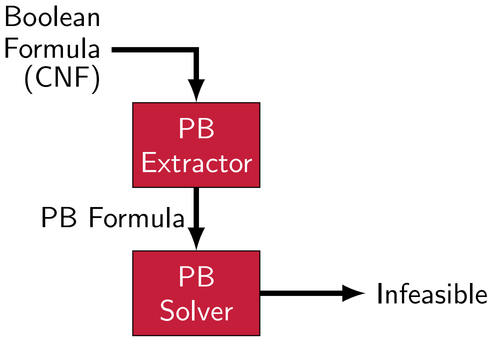


- Example equation:

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

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

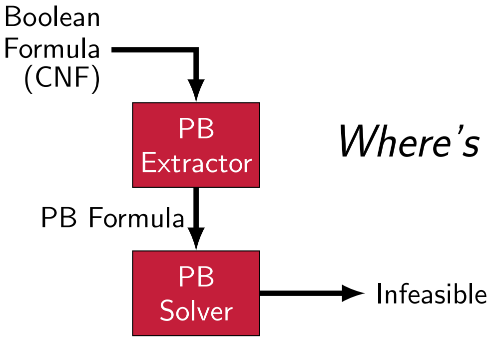
(Un)satisfiability with a Pseudo-Boolean Sover



Pseudo-Boolean Reasoning Methods

- (Modular) Equations
  - Gaussian elimination
- Ordering Constraints
  - Cutting planes
  - Fourier-Motzkin elimination

(Un)satisfiability with a Pseudo-Boolean Sover

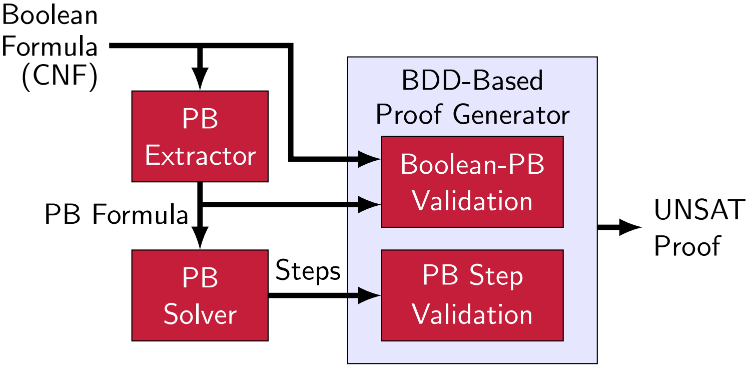


Where's the proof?

Pseudo-Boolean Reasoning Methods

- (Modular) Equations
  - Gaussian elimination
- Ordering Constraints
  - Cutting planes
  - Fourier-Motzkin elimination

Integrating Pseudo-Boolean Reasoning into Proof-Generating SAT Solver [BryBieHeu-2022]



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

Validating Solver Steps

Individual Solver Step

- Given constraints p_i and p_j, compute new constraint p_k:

$$p_k \quad \leftarrow \quad p_i \odot p_j$$

- E.g., $\odot \ = \ +$

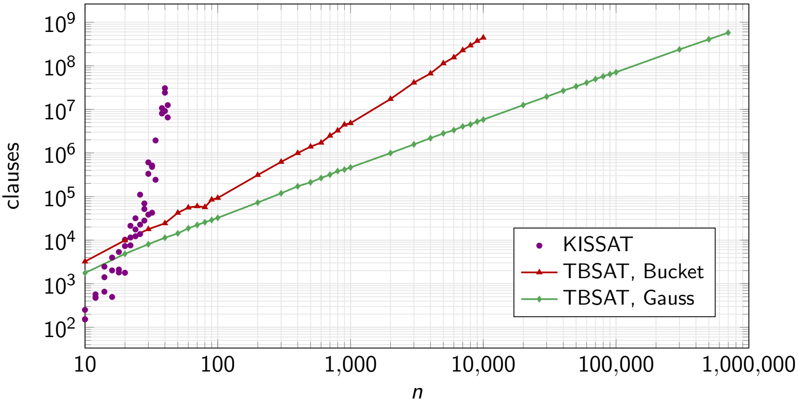
Validation

- Maintain BDD(p) for each constraint p
- When generate p_k, also generate proof:

$$\text{BDD}(p_i) \wedge \text{BDD}(p_j) \quad \Longrightarrow \quad \text{BDD}(p_k)$$

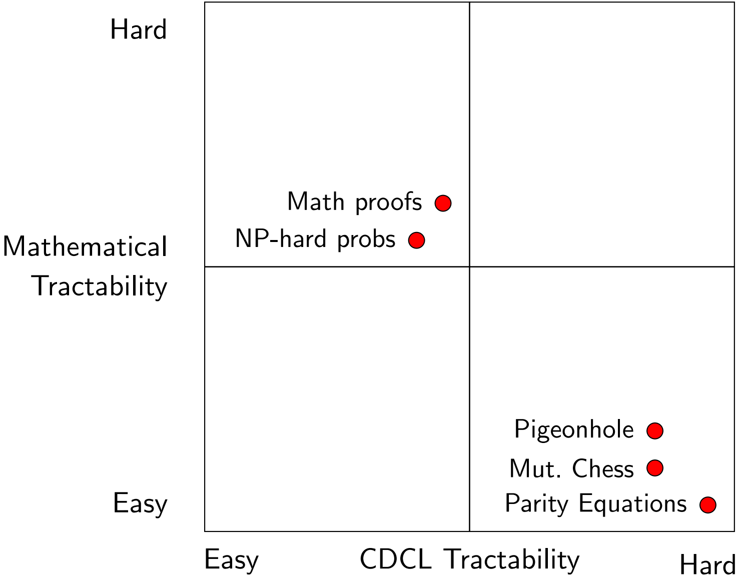
- Complexity $O(m_i \cdot m_j \cdot m_k)$
  - for BDDs of size m_i, m_j, and m_k

Chew-Heule Parity Benchmark Proof Sizes

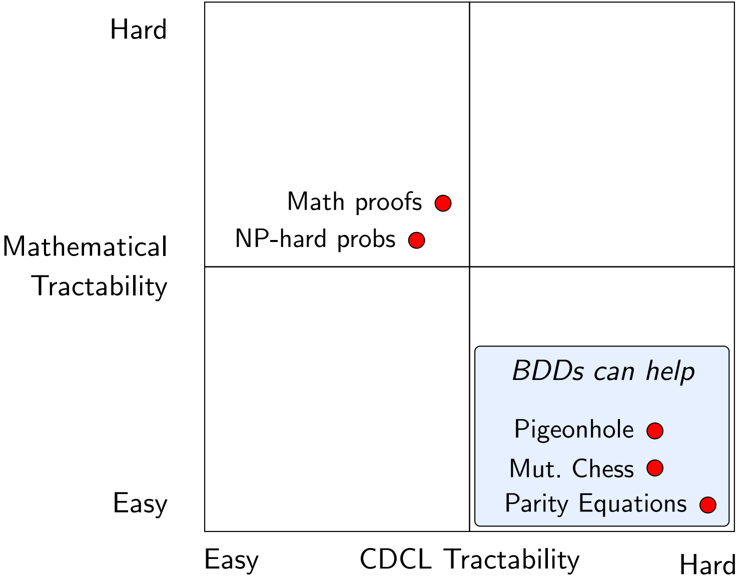


- Upper limit: $n = 699{,}051$
  - Node data structure sets limit of $2^{21} - 1$ BDD variables
  - CNF file has 2,097,147 variables and 5,592,392 clauses
- Some failures for large values of n due to poor pivot selection

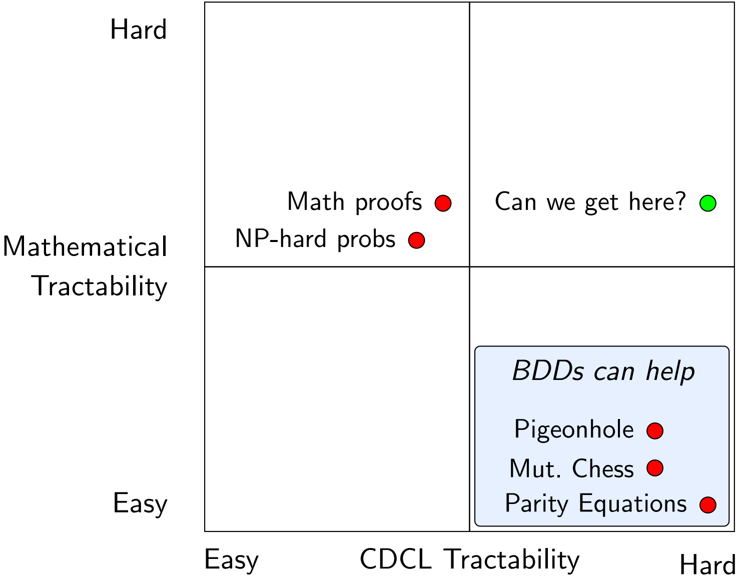
A Perspective on the State of SAT Solving



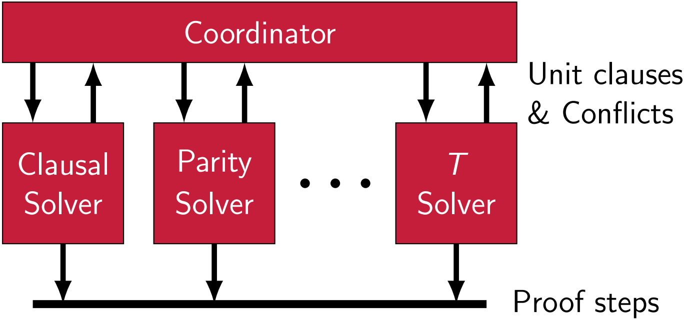
A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



Proof Generation for CDCL(T)



- Solvers coordinate in unit propagation and conflict detection
- Proof generation: Each solver justifies its propagations & conflicts
- CryptoMiniSAT
  - Gauss-Jordan elimination for parity constraints
  - Can use BDDs to justify each elimination step [SooBry-22]

Knowledge Compilation

- Darwiche [DarMar-2002]

Convert CNF Formula into More Tractable Representation

Sample Query

- *Model Counting:* How many satisfying assignments does the formula have?

Challenging Problem

- #SAT more difficult than SAT

Knowledge Compilation

- Darwiche [DarMar-2002]

Convert CNF Formula into More Tractable Representation

Sample Query

- *Model Counting:* How many satisfying assignments does the formula have?

Challenging Problem

- #SAT more difficult than SAT

Questions

- *How do I know the generated representation is logically equivalent to the input formula?*
- *How do I know if the computed query values are correct?*

Algebraic Formulation

- Kimmig et al. [KimVdbDra-2017]

Definitions

- Input variables $x_1, x_2, \ldots, x_n$
- *Assignment*: $\alpha = \{\ell_1, \ell_2, \ldots, \ell_n\}$ with each $\ell_i \in \{x_i, \overline{x}_i\}$
- *Models*: $\mathcal{M}(\phi)$ is set of satisfying assignments for formula ϕ

Ring Evaluation

- Commutative ring $\mathcal{R}$
- Assign weight $w(x_i) \in \mathcal{R}$ to each input variable x_i
- Define $w(\overline{x}_i) = 1 - w(x_i)$
- Ring evaluation $\mathbf{R}(\phi, w)$ of formula ϕ:

$$\mathbf{R}(\phi, w) \quad = \quad \sum_{\alpha \in \mathcal{M}(\phi)} \prod_{\ell_i \in \alpha} w(\ell_i)$$

Ring Evaluation Examples

Model Counting

- Let $w(x_i) = w(\overline{x}_i) = 1/2$ for all i
- $\mathbf{R}(\phi, w)$ gives *density* of function
  - Fraction of assignments that satisfy ϕ
- Scale by 2^n to get model count

Probabilistic Inference

- Each input variable x_i is true with probability $p(x_i)$.
- $\mathbf{R}(\phi, p)$ is probability that formula is true

Partitioned-Operation Formulas

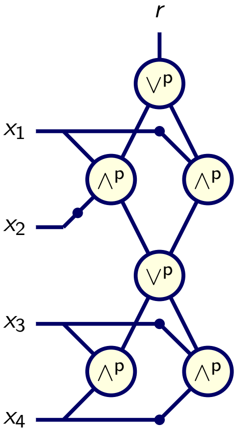
Allowed Operations

- **Product:** $\phi_1 \wedge^{\mathsf{p}} \phi_2$, where $\mathcal{D}(\phi_1) \cap \mathcal{D}(\phi_2) = \emptyset$
  - $\mathcal{D}(\phi)$: Set of all variables occuring in ϕ
- **Sum:** $\phi_1 \vee^{\mathsf{p}} \phi_2$, where $\mathcal{M}(\phi_1) \cap \mathcal{M}(\phi_2) = \emptyset$
- **Negation:** $\neg\phi$

Ring Evaluation of Partitioned Formula

$$\begin{aligned}
\mathbf{R}(\phi_1 \wedge^{\mathsf{p}} \phi_2,\, w) &= \mathbf{R}(\phi_1, w) \cdot \mathbf{R}(\phi_2, w) \\
\mathbf{R}(\phi_1 \vee^{\mathsf{p}} \phi_2,\, w) &= \mathbf{R}(\phi_1, w) + \mathbf{R}(\phi_2, w) \\
\mathbf{R}(\neg\phi,\, w) &= 1 - \mathbf{R}(\phi, w)
\end{aligned}$$

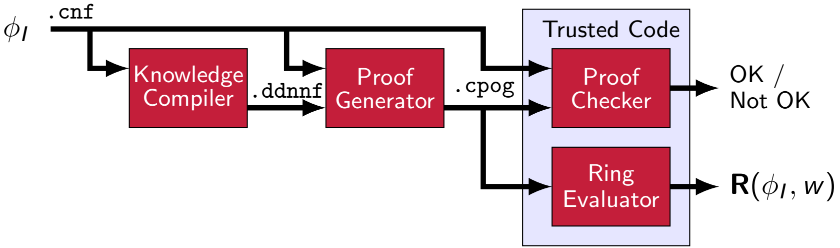
Partitioned-Operation Graphs (POGs)



- Directed graph representation of formula
  - Leaf nodes: Input variables
  - Operation nodes: Partitioned product and sum
  - Each edge can be negated
- Can encode other compiled representations

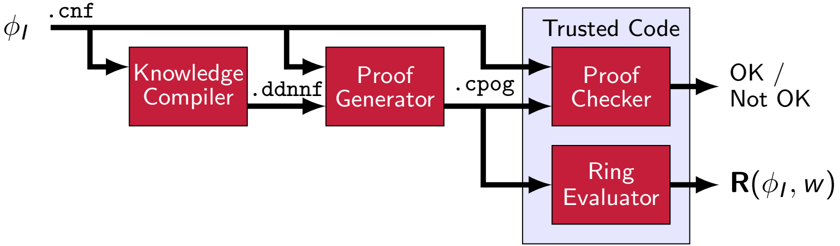
Certifying Toolchain

- Joint work with Wojciech Nawrocki, Jeremy Avigad, and Marijn Heule



- *Knowledge Compiler* (D4 [LagMar-2017]): Convert CNF into representation using only partitioned operations
- *Proof Generator*: Generate file combining POG definition + equivalence proof
- *Proof Checker*: Validate proof file
- *Ring Evaluator*: Compute standard or weighted model count

Trusting the Trusted Code



Within Lean 4 Proof Framework [DemUlr-2021]

- Soundness of proof system
  - Helped us identify some weaknesses in our proof rules
- Verified checker
  - Around 6× slower than one implemented in C
- Ring Evaluator: Over rationals

CPOG Declaration + Proof

Input Formula ϕ_I

POG Declaration θ_P

- Extension variable u for each operation node $\mathbf{u}$
- Node $\mathbf{u}$ with k children characterized by $k + 1$ defining clauses
- Children indicated by literals
  - Positive or negated arguments
  - Input variables or results from other operation
- Unit clause $[r]$ for root node $\mathbf{r}$

Proof Objective

$$\phi_I \iff \theta_P$$

CPOG Proof Structure

Forward Implication Proof

$$\phi_I \quad \Longrightarrow \quad \theta_P$$

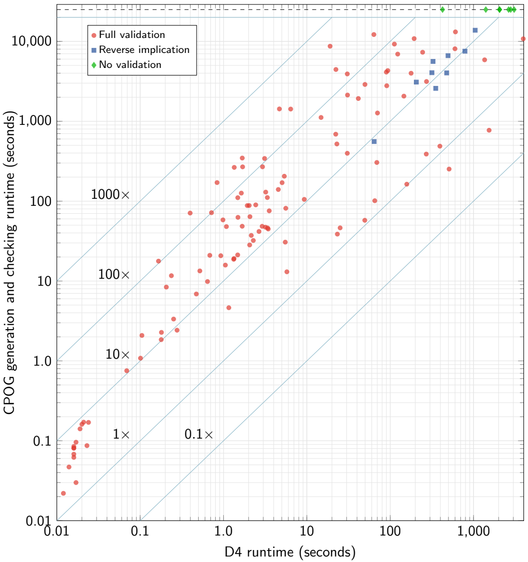
- Add clauses by resolution
- Terminating with unit clause $[r]$
- *Any assignment satisfying* ϕ_I *causes the POG formula to evaluate to true*

Reverse Implication Proof

$$\theta_P \quad \Longrightarrow \quad \phi_I$$

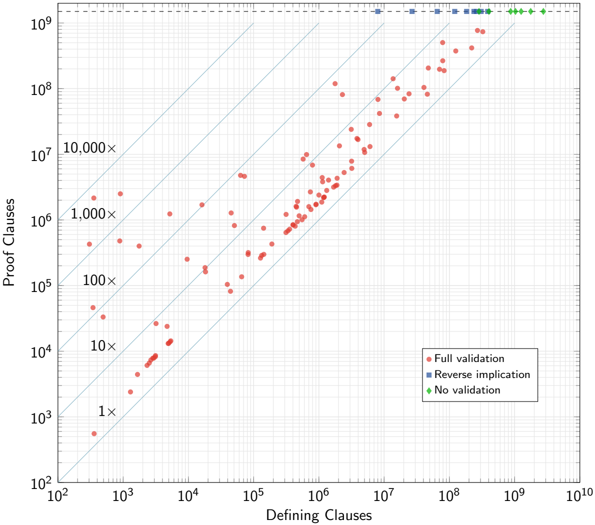
- Delete clauses by resolution
  - Deleted clause implied by remaining ones
- Including each of the input clauses
- *Any assignment falsifying the input clause causes the POG formula to evaluate to false*

Experimental Results: CPOG Generation and Checking



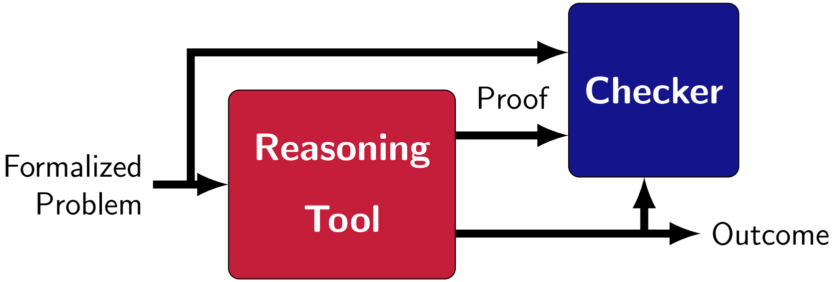
- 180 benchmark files from 2022 model checking competitions
- D4 completed 124 with 4000-second time limit
- Generated complete proofs for 108 with 10,000-second time limit
- Reverse implications for 9
- No proofs for 7

Experimental Results: CPOG Sizes



- 108 problems fully verified
- CPOG files up to 160 GB
- Reverse implications for 9
- No proofs for 7

Recap: Important Principle



Proof-Generating Tools

- Formally verifying a large, complex program is impractical
- Instead, certify individual executions of the program

Important Concepts

Checkable Proofs of Program Executions

- Very habit forming
- Many research possibilities

Clausal Proof Frameworks

- Well understood set of principles
- Can build on existing infrastructure
  - E.g., our CPOG proof generator uses CaDiCal and Drat-trim
- Not just for refutation proofs

Extended Resolution

- Can reason about other representations of Boolean formulas
- Can introduce intermediate proof structures
  - E.g., CPOG proof generator uses lemmas to control recursion
  - Validation for each shared POG node generated once and used multiple times

References #1

Bry-1986 R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, 1986

BryHeu-2021 R. E. Bryant and M. J. H. Heule, "Generating extended resolution proofs with a BDD-based SAT solver," *TACAS*, 2021. Extended version on ArXiv

Bry-2022 R. E. Bryant, "TBUDDY: A Proof-generating BDD package," *FMCAD*, 2022

BryBieHeu-2022 R. E. Bryant, A. Biere, and M. J. H. Heule, "Clausal proofs from pseudo-Boolean reasoning," *TACAS*, 2022

SooBry-2022 M. Soos and R. E. Bryant, "Proof generation for CDCL solvers using Gauss-Jordan elimination" *Pragmatics of SAT Workshop*, 2022.

References #2

CheHeu-2020 L. Chew and M. J. H. Heule, "Sorting parity encodings by reusing variables," *SAT*, 2020.

DarMar-2002 A. Darwiche and P. Marquis, "A Knowledge Compilation Map," *JAIR*, 2002

DemUlr-2021 L. de Moura and S. Ulrich, "The Lean 4 theorem prover and programming language," *CADE*, 2021

GolNov-2003 E. I. Goldberg and Y. Novikov, "Verification of proofs of unsatisfiability for CNF formulas," *DATE*, 2003

JusSinBie-2006 T. Jussila, C. Sinz and A. Biere, "Extended resolution proofs for symbolic SAT solving with quantification," *SAT*, 2006

KimVdbDra-2017 Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt, "Algebraic model counting," *J. Applied Logic*, 2017

LagMar-2017 J.-M. Lagniez and P. Marquis, "An improved decision-DNNF compiler," *IJCAI*, 2017

SinBie-2006 C. Sinz and A. Biere, "Extended resolution proofs for conjoining BDDs," *Constraint Programming*, 2006.

HeuHunWet-2013 M. J. H. Heule, W. A. Hunt, Jr., N. Wetzler, "Trimming while checking clausal proofs," *FMCAD*, 2013.

ZhaMal-2003 L. Zhang and S. Malik, "Validating SAT solvers using an independent resolution-based checker,"' *DATE*, 2003