TBUDDY: a Proof-Generating BDD Package

Randal E. Bryant

Carnegie Mellon University

August, 2022

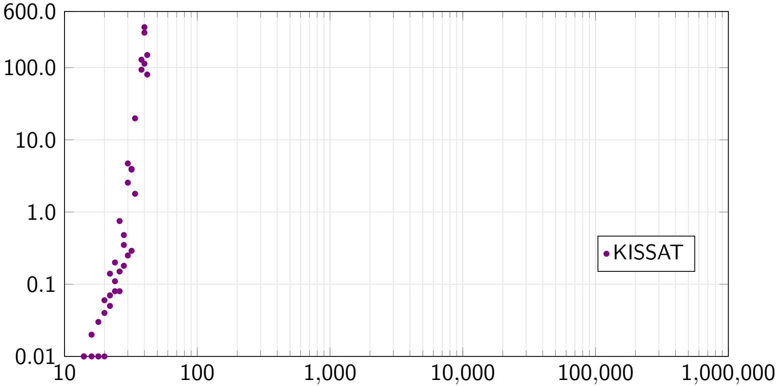
Motivation: Parity Benchmark

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

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

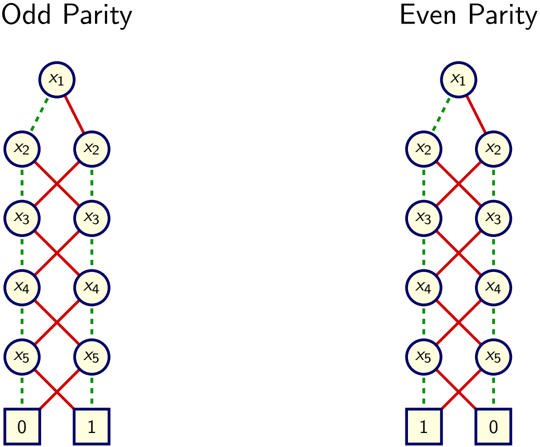
- Conjunction unsatisfiable

Motivation: Parity Benchmark Runtime



- KISSAT: State-of-the-art CDCL solver
- 3 different seeds for each value of *n*
- Limited to $n \leq 42$ within 600 seconds

BDD Representation of Parity Constraints



- Linear complexity
- Insensitive to variable order
- Potential major advantage over CDCL

Trusted Binary Decision Diagrams (TBDDs)

Motivation

- BDDs can outperform CDCL on some classes of problems
- Need to be able to generate proofs of unsatisfiability

Concept

- Generate clausal proof as BDD operations proceed
- Standalone solver, plus can incorporate into other solvers

Implementation

- Build on BuDDy BDD package
- Also support parity reasoning

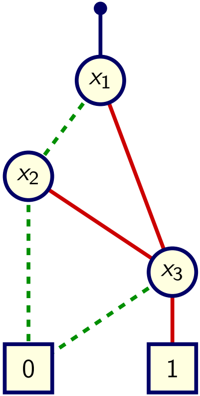
Reduced Ordered Binary Decision Diagrams (BDDs)

Represent Boolean Function as Graph

- Canonical form
- Simple algorithms to construct & manipulate

Used in SAT, Model Checking, . . .

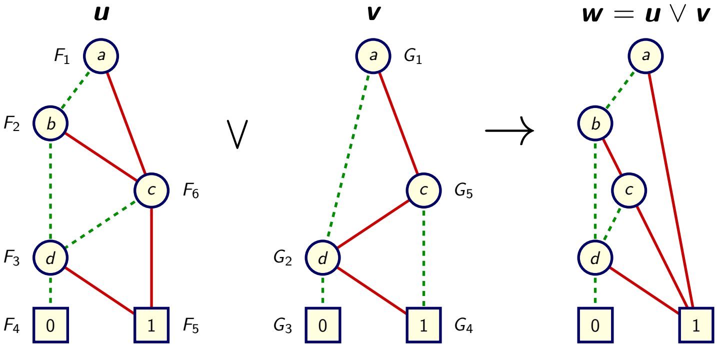
- Bottom-up approach
  - Construct canonical representation
  - Generate solutions
- Compare to CDCL
  - Top-down approach
  - Keep branching on variables until find solution



Apply Algorithm

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

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



Extended Resolution and BDDs

Extended Resolution

- Tseitin, 1967
- *Extension variable* z becomes shorthand for formula F
    - F: Boolean formula over input and earlier extension variables
- Add *defining* clauses
    - Encode constraint of form $z \leftrightarrow F$
- Repeated use can yield exponentially smaller proof
- Supported by DRAT proof framework

Extended Resolution and BDDs

Extended Resolution

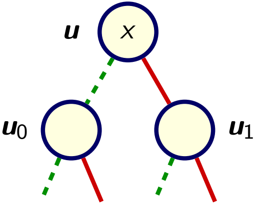
- Tseitin, 1967
- *Extension variable z* becomes shorthand for formula F
  - F: Boolean formula over input and earlier extension variables
- Add *defining* clauses
  - Encode constraint of form $z \leftrightarrow F$
- Repeated use can yield exponentially smaller proof
- Supported by DRAT proof framework

Proof-Generating BDD Operations

- Biere, Sinz, Jussila, 2006
- Each node $\boldsymbol{u}$ has associated extension variable u
- Each recursive step of Apply algorithm justified as proof steps

Generating Extended Resolution Proofs

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

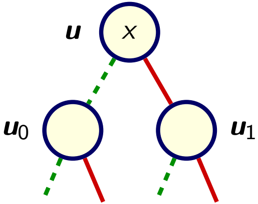


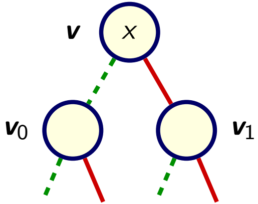
- Defining clauses encode constraint $u \leftrightarrow ITE(x, u_1, u_0)$

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

Apply Algorithm Recursion

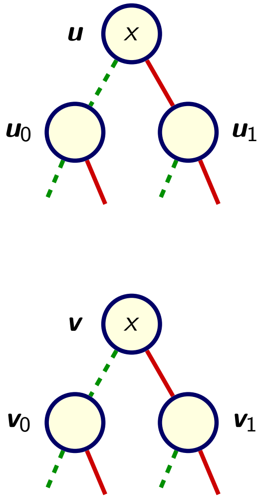
$\text{Apply}(\boldsymbol{u}, \boldsymbol{v}, \wedge)$





Apply Algorithm Recursion

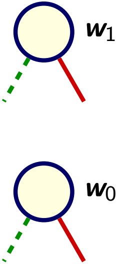
Apply($\boldsymbol{u}, \boldsymbol{v}, \wedge$)



Recursion

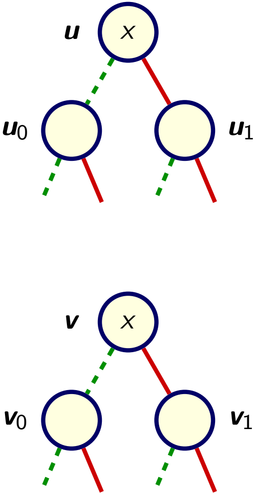
Apply($\boldsymbol{u}_1, \boldsymbol{v}_1, \wedge$) $\rightarrow$

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



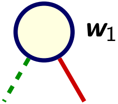
Apply Algorithm Recursion

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

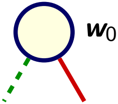


Recursion

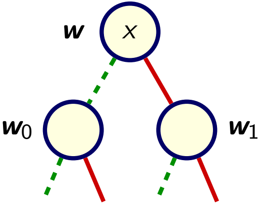
Apply($\boldsymbol{u}_1, \boldsymbol{v}_1, \wedge$) $\rightarrow$



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



Result



Proof-Generating Apply Operation

Integrate Proof Generation into Apply Operation

- When Apply($\boldsymbol{u}, \boldsymbol{v}, \wedge$) returns $\boldsymbol{w}$, 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 $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$

Apply Proof Structure

Defining Clauses

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

Resolution Steps

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

Can express as two reverse unit propagation (RUP) proof steps

Quantification Operation

Operation EQuant($\boldsymbol{u}, x$)

$$\exists x\, f \quad = \quad f|_{x=0} \vee f|_{x=1}$$

- Abstract away details of satisfying 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($\boldsymbol{u}, x$) $\rightarrow \boldsymbol{w}$
  - Generate proof $u \rightarrow w$
  - Algorithm similar to proof-generating Apply operation

Trusted BDDs (TBDDs)

Components of TBDD $\dot{\boldsymbol{u}}$

- BDD with root node $\boldsymbol{u}$.
- Associated extension variable u
- Proof step for unit clause $[u]$

Interpretation. For input formula ϕ:

- $\phi \models u$
- Any variable assignment that satisfies ϕ must yield 1 for BDD with root $\boldsymbol{u}$

TBDD API

`tbdd tbdd_from_clause_id(int` i`);`

- Create TBDD representation $\dot{\boldsymbol{u}}_i$ of input clause C_i
    - Add proof step for $C_i \models u_i$

`tbdd tbdd_and(tbdd` $\dot{\boldsymbol{u}}$`, tbdd` $\dot{\boldsymbol{v}}$`);`

- Form conjunction $\dot{\boldsymbol{w}}$ of TBDDs $\dot{\boldsymbol{u}}$ and $\dot{\boldsymbol{v}}$.
    - Apply operation generates proof $u \wedge v \rightarrow w$
    - Resolution with unit clauses $[u]$ and $[v]$ yields unit clause $[w]$

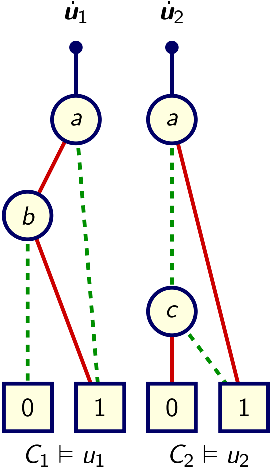
`tbdd tbdd_validate(bdd` $\boldsymbol{v}$`, tbdd` $\dot{\boldsymbol{u}}$`);`

- Upgrade BDD $\boldsymbol{v}$ to TBDD $\dot{\boldsymbol{v}}$
    - Apply operation generates proof $u \rightarrow v$
    - Resolution with unit clause $[u]$ yields unit clause $[v]$

TBDD Execution Example

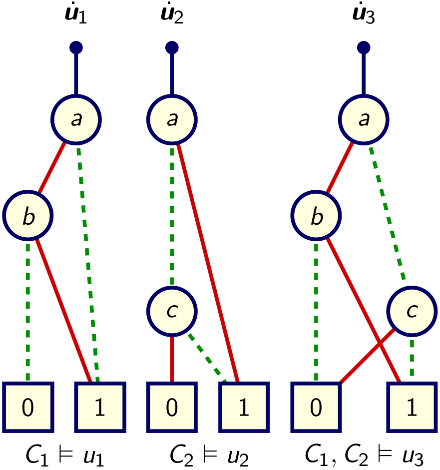
$\dot{\boldsymbol{u}}_1 \longleftarrow$ `tbdd_from_clause`(C_1)

$\dot{\boldsymbol{u}}_2 \longleftarrow$ `tbdd_from_clause`(C_2)



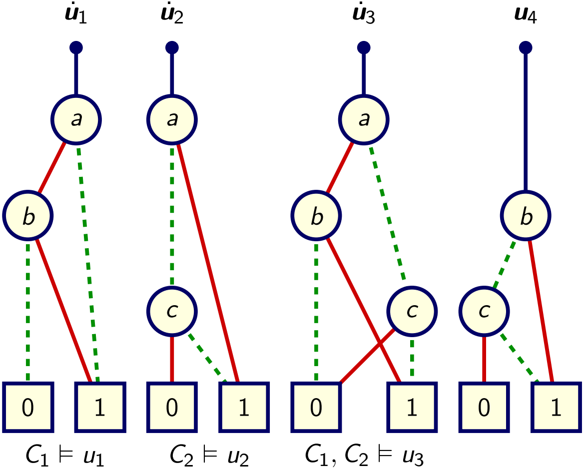
TBDD Execution Example

$\dot{\boldsymbol{u}}_3 \longleftarrow \texttt{tbdd\_and}(\dot{\boldsymbol{u}}_1,\ \dot{\boldsymbol{u}}_2)$



TBDD Execution Example

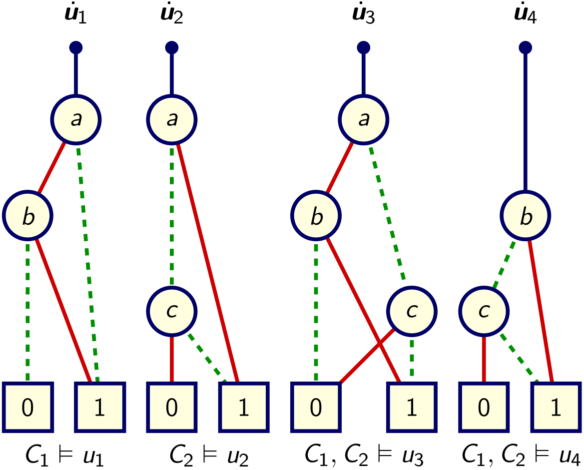
$\boldsymbol{u}_4 \longleftarrow$ `bdd_exists`$(\boldsymbol{u}_3, a)$



TBDD Execution Example

$\boldsymbol{u}_4 \longleftarrow$ `bdd_exists`$(\boldsymbol{u}_3,\, a)$

$\dot{\boldsymbol{u}}_4 \longleftarrow$ `tbdd_validate`$(\boldsymbol{u}_4,\, \dot{\boldsymbol{u}}_3)$



Clausal Proof (LRAT Format)

ID	Clause	Hints
Defining clauses for node $\boldsymbol{u}_{17} = ITE(x_2, \boldsymbol{u}_9, \boldsymbol{u}_8)$		
68	`17 -9 -2 0`	`0`
69	`17 -8 2 0`	`0`
70	`-17 9 -2 0`	`-68 -69 0`
71	`-17 8 2 0`	`-68 -69 0`

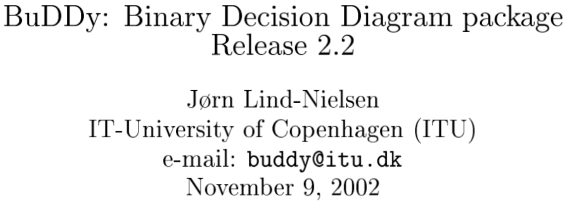
- Variables denoted by signed integers
  - $x_i \rightarrow \quad i$
  - $\overline{x}_i \rightarrow -i$
- Each clause identified by numerical ID
- Clause addition justified by list of hints
  - For defining clause, list of clauses for which extension variable has opposite polarity

Clausal Proof (LRAT Format)

ID	Clause	Hints
Proof that $\boldsymbol{u}_{12} \wedge \boldsymbol{u}_{13} \rightarrow \boldsymbol{u}_{17}$		
`72`	`17 -13 -12 -2 0`	`68 48 0`
`73`	`17 -13 -12 0`	`72 69 44 0`
c Validate unit clause for node $\boldsymbol{u}_{17}$		
`74`	`17 0`	`45 50 73 0`

- Each clause identified by numerical ID
- Clause addition justified by list of hints
  - For RUP clause, sequence of clauses for resolution operations

BuDDy BDD Package



BuDDy: Binary Decision Diagram package
Release 2.2

Jørn Lind-Nielsen
IT-University of Copenhagen (ITU)
e-mail: buddy@itu.dk
November 9, 2002

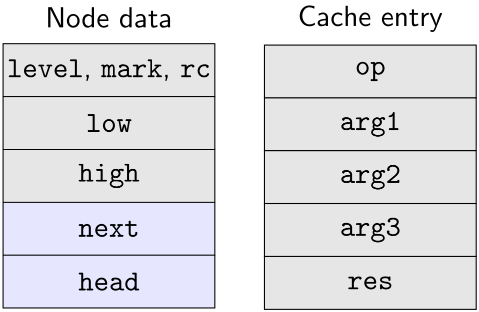
- ~12K lines of code
- Clean, robust, and well documented
- Benchmark comparisons demonstrate good performance
- Node identified by 32-bit index into table
  - Rather than as 64-bit pointer

Tracking Proof Information in TBUDDY

	ID	Clause	Hints
	Defining clauses for node $\boldsymbol{u}_{17} = ITE(x_2, \boldsymbol{u}_9, \boldsymbol{u}_8)$		
`dclause`	68	`17 -9 -2 0`	`0`
`xvar`	69	`17 -8 2 0`	`0`
	70	`-17 9 -2 0`	`-68 -69 0`
	71	`-17 8 2 0`	`-68 -69 0`
	Proof that $\boldsymbol{u}_{12} \wedge \boldsymbol{u}_{13} \rightarrow \boldsymbol{u}_{17}$		
	72	`17 -13 -12 -2 0`	`68 48 0`
`jclause`	73	`17 -13 -12 0`	`72 69 44 0`
	c Validate unit clause for node $\boldsymbol{u}_{17}$		
`vclause`	74	`17 0`	`45 50 73 0`

- Information tracked with nodes, cache entries, and TBDDs

BuDDy Data Structures



Node data

`level`, `mark`, `rc`

`low`

`high`

`next`

`head`

Cache entry

`op`

`arg1`

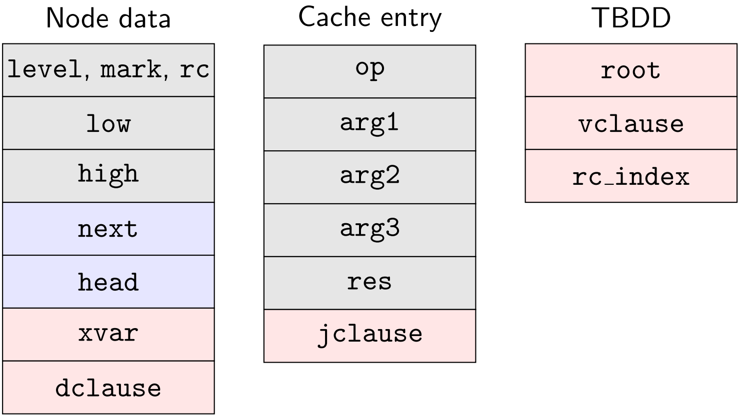
`arg2`

`arg3`

`res`

- Four byte fields
- Node table integrates node data structures + unique table
- Memory management
  - Reference counting for external references
  - Mark-sweep to detect internal references

TBUDDY Data Structures



Node data

- `level, mark, rc`
- `low`
- `high`
- `next`
- `head`
- `xvar`
- `dclause`

Cache entry

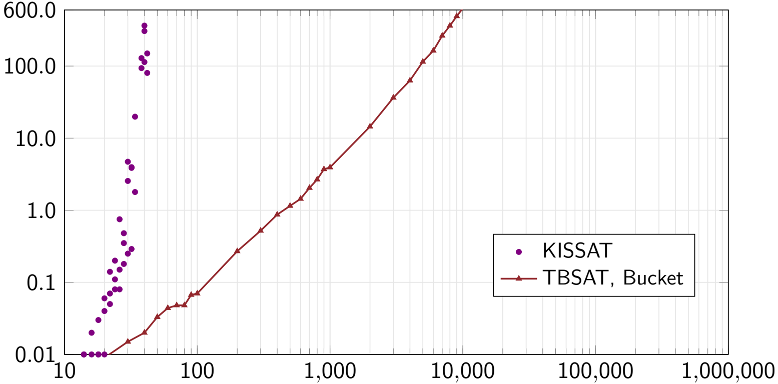
- `op`
- `arg1`
- `arg2`
- `arg3`
- `res`
- `jclause`

TBDD

- `root`
- `vclause`
- `rc_index`

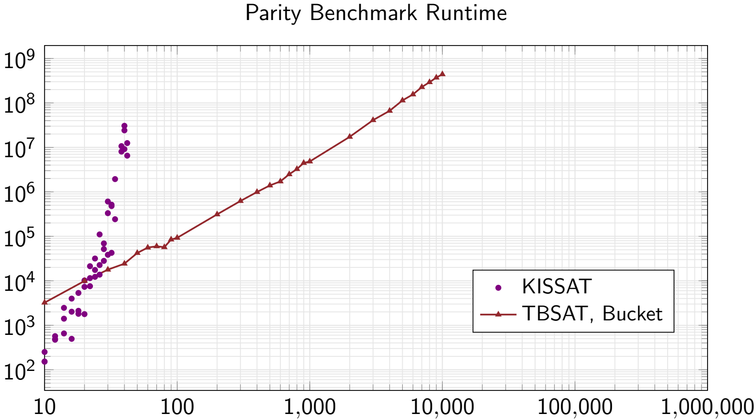
- Node entry includes extension variable, defining clause ID
- Cache entry includes justifying clause ID
- TBDDD includes root node, validating clause ID

Parity Benchmark Runtime



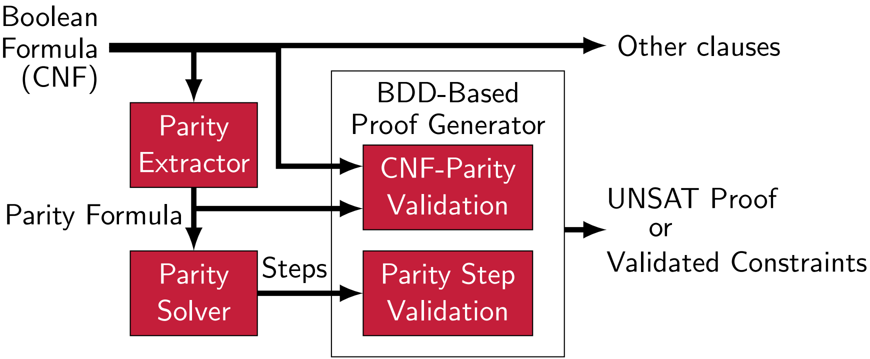
- Bucket elimination
  - Systematic way to perform conjunctions and quantifications
- Random variable ordering
- No guidance from user

Parity Benchmark Proof Complexity



- Total number of proof steps
- TBSAT with bucket elimination scales polynomially
    - Checker time ≈ solver time
    - Large proofs, but efficiently checkable

Integrating Parity Reasoning



- Fully automated
- UNSAT if constraints infeasible
- Otherwise, supply validated constraints to BDD-based solver

Gaussian Elimination Over GF2

Parity Constraints $\mathcal{P} = P_1, P_2, \ldots, P_m$, each of form

$$x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus x_{i_k} \quad = \quad p$$

with phase $p \in \{0, 1\}$

Elimination Step

1. Choose pivot constraint P_s and variable x_t such that $x_t \in P_s$
2. For each $j \neq s$:

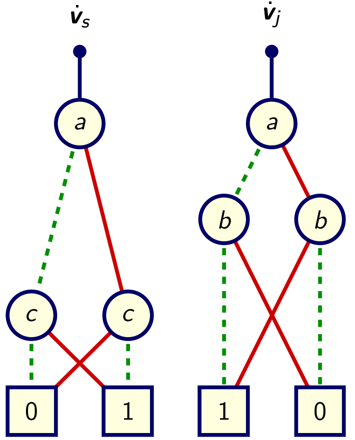
$$P_j \quad \leftarrow \quad \begin{cases} P_j & x_t \notin P_j \\ P_s \oplus P_j, & x_t \in P_j \end{cases}$$

   - Removes x_t from all other constraints

3. Remove P_s from $\mathcal{P}$ and repeat
4. Stop with infeasible constraint $0 = 1$ or have $|\mathcal{P}| = 1$.

TBDD-Based Parity Reasoning Example

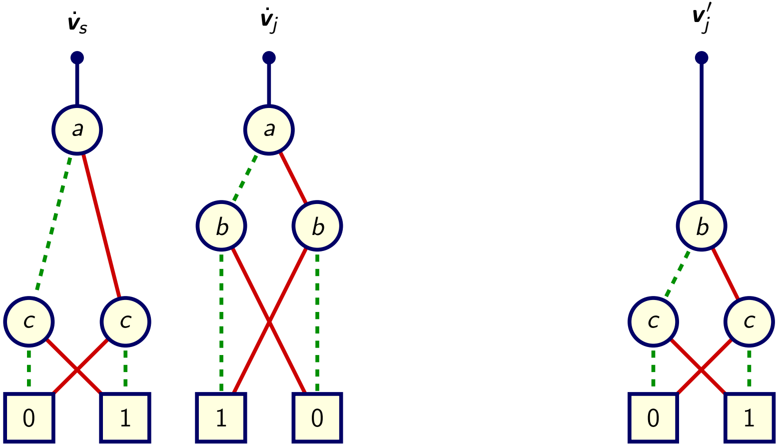
Goal: Compute $P'_j \longleftarrow P_s \oplus P_j$



$P_s : a \oplus c = 1$ $P_j : a \oplus b = 0$

TBDD-Based Parity Reasoning Example

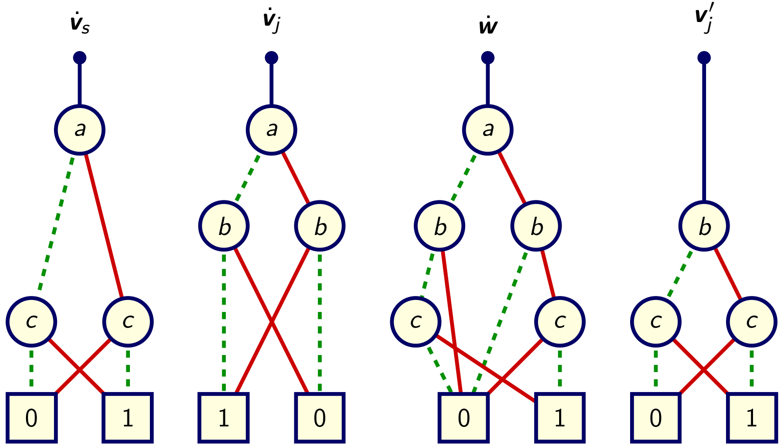
$\mathbf{v}'_j \longleftarrow \texttt{bdd\_xnor}(\mathbf{v}_s,\ \mathbf{v}_j)$



$P_s : a \oplus c = 1$ $\quad P_j : a \oplus b = 0$ $\quad P'_j : P_s \oplus P_j$

TBDD-Based Parity Reasoning Example

$$\dot{\boldsymbol{w}} \longleftarrow \texttt{tbdd\_and}(\dot{\boldsymbol{v}}_s,\ \dot{\boldsymbol{v}}_j)$$



$P_s : a \oplus c = 1$

$P_j : a \oplus b = 0$

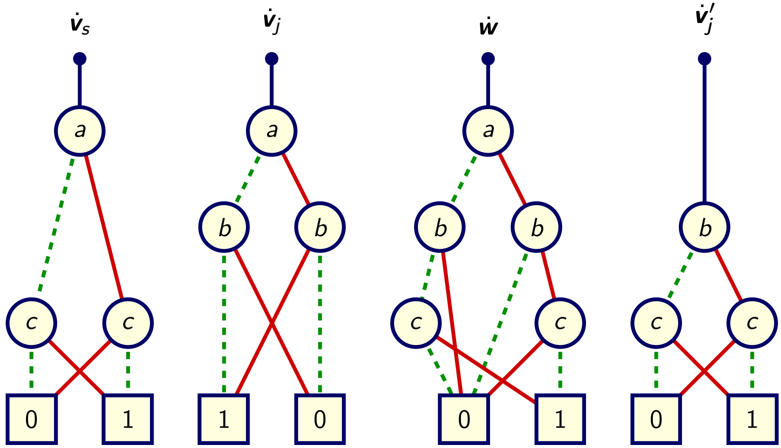
$P_i \wedge P_j$

$P'_j : P_s \oplus P_j$

TBDD-Based Parity Reasoning Example

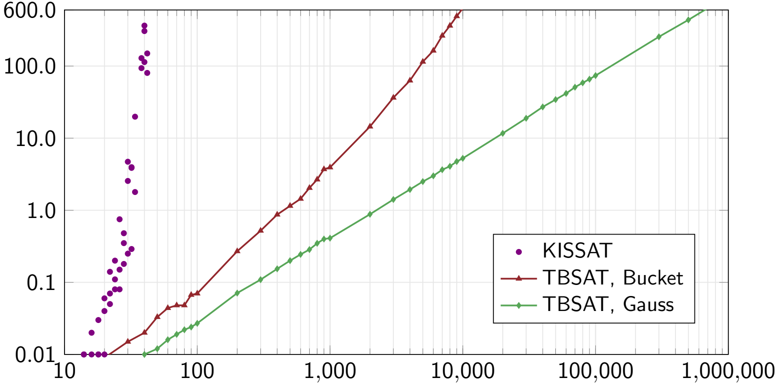
$\dot{\boldsymbol{w}} \longleftarrow \texttt{tbdd\_and}(\dot{\boldsymbol{v}}_s,\ \dot{\boldsymbol{v}}_j)$

$\dot{\boldsymbol{v}}'_j \longleftarrow \texttt{tbdd\_validate}(\boldsymbol{v}'_j,\ \dot{\boldsymbol{w}})$



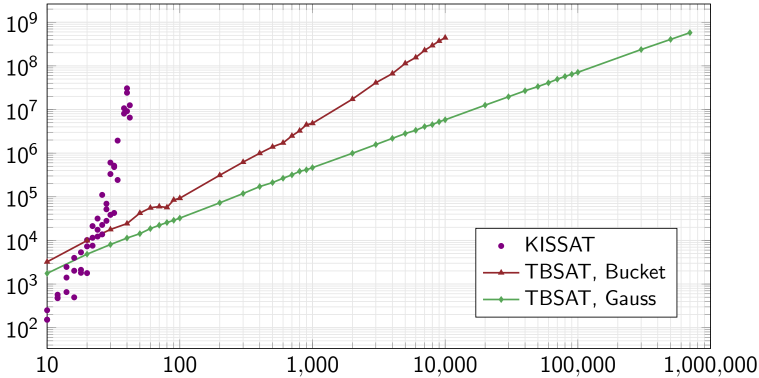
$P_s : a \oplus c = 1$ $P_j : a \oplus b = 0$ $P_i \wedge P_j$ $P'_j : P_s \oplus P_j$

Parity Benchmark Runtime



- Upper limit: $n = 699{,}051$
  - BuDDy limited to $2^{21} - 1$ BDD variables
  - CNF file has 2,097,147 variables and 5,592,392 clauses
  - Parity extractor finds 1,398,098 euqations

Parity Benchmark Proof Complexity



- Checker time ≈ solver time

Final Thoughts on SAT Solvers

CDCL is the best overall approach

- Readily generates resolution proofs
- But, very weak for parity and cardinality constraints

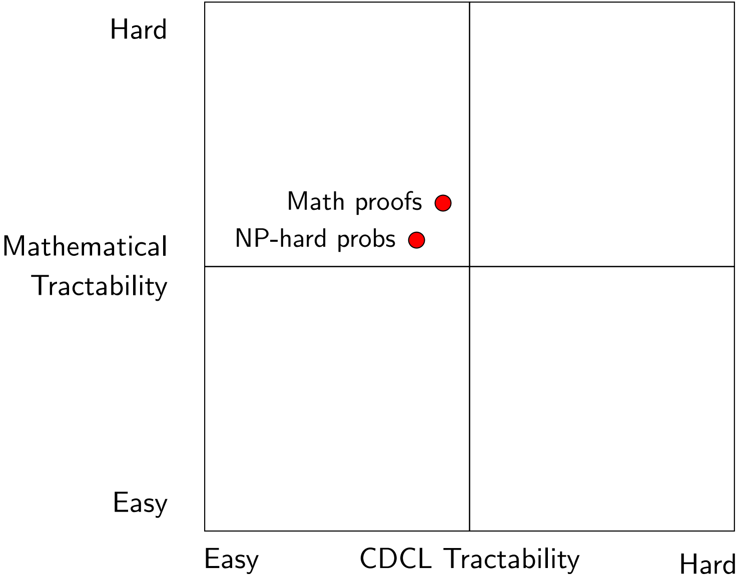
BDDs provide complementary strengths

- Can generate extended resolution proofs
- Very strong for parity constraints
- Some success with cardinality constraints

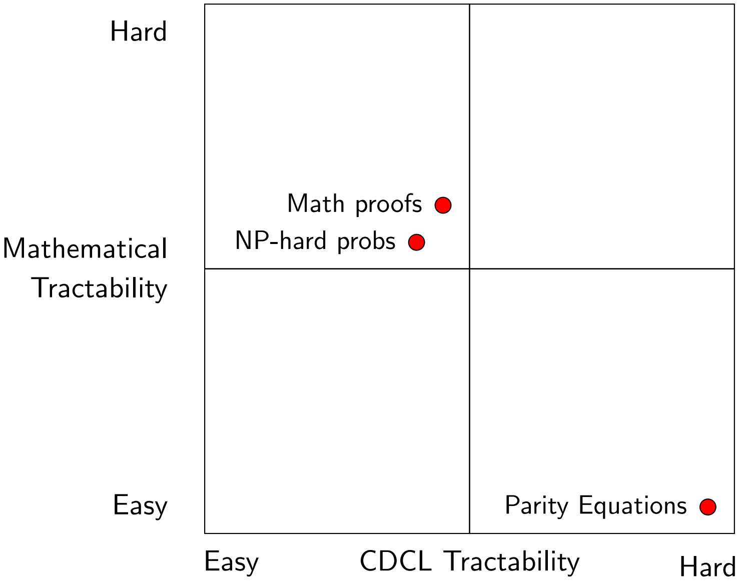
Future solvers should use combination of methods

- With unified proof framework
- Clausal reasoning
- Constraint reasoning
- Boolean reasoning

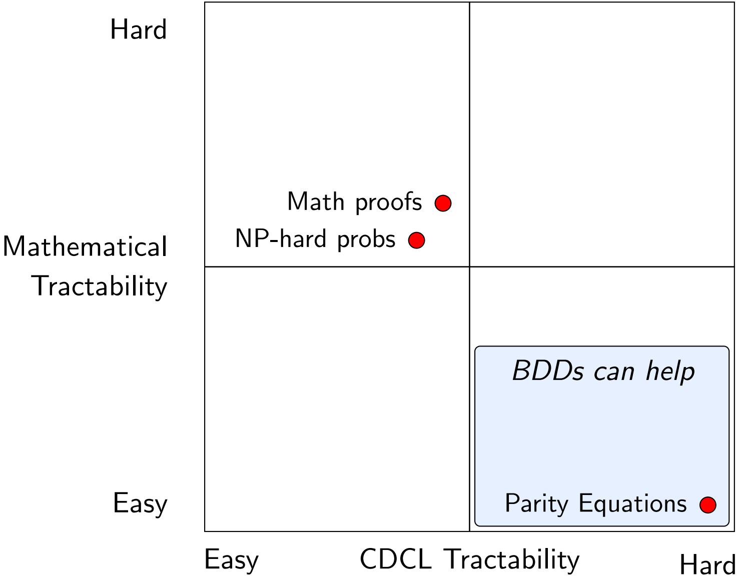
A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



Hard

Mathematical Tractability

Easy

Math proofs ●

NP-hard probs ●

BDDs can help

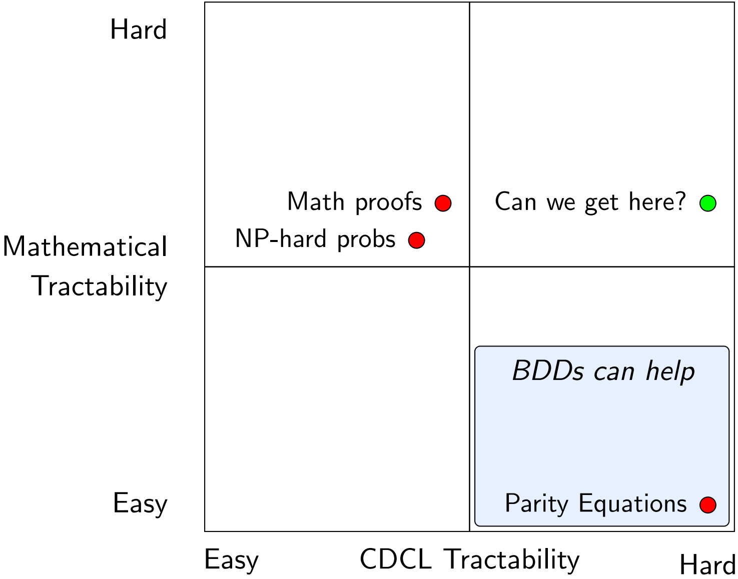
Parity Equations ●

Easy

CDCL Tractability

Hard

A Perspective on the State of SAT Solving



Parity Benchmark Runtime: Proof Generation Overhead