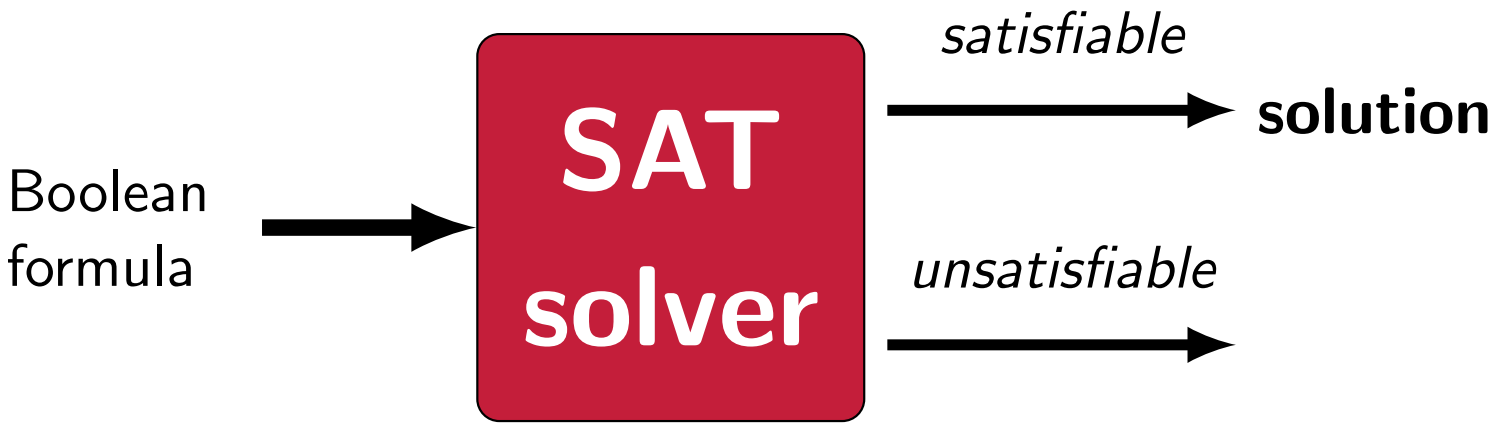
TBUDDY: a Proof-Generating BDD Package

Randal E. Bryant

Carnegie Mellon University

October, 2022

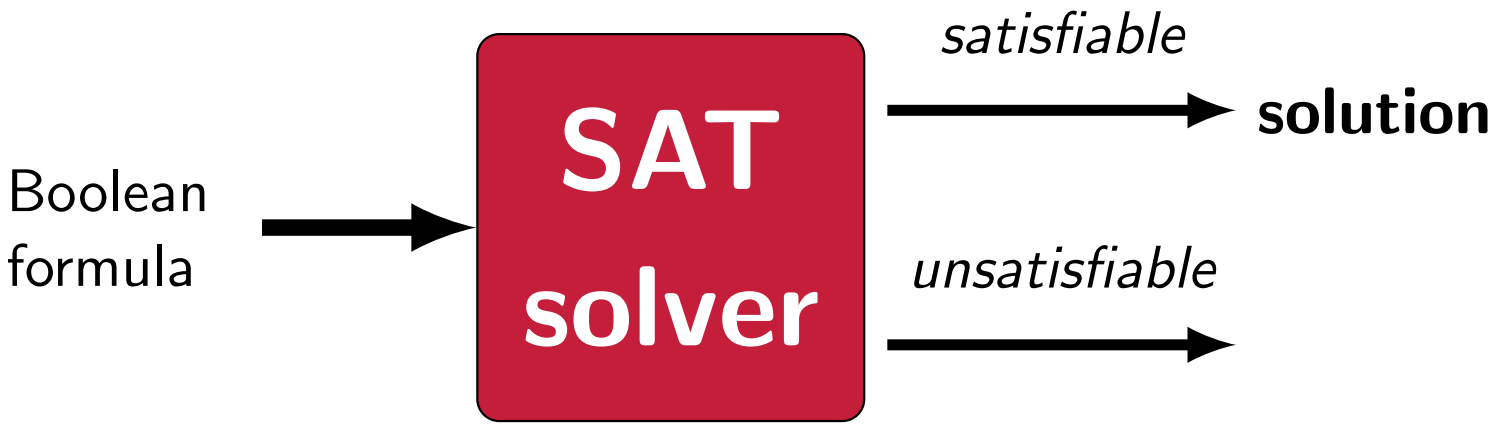
Context: Boolean Satisfiability Solvers



SAT Solvers Useful & Powerful

- Mathematical proofs
- Formal verification
- Optimization

Context: Boolean Satisfiability Solvers



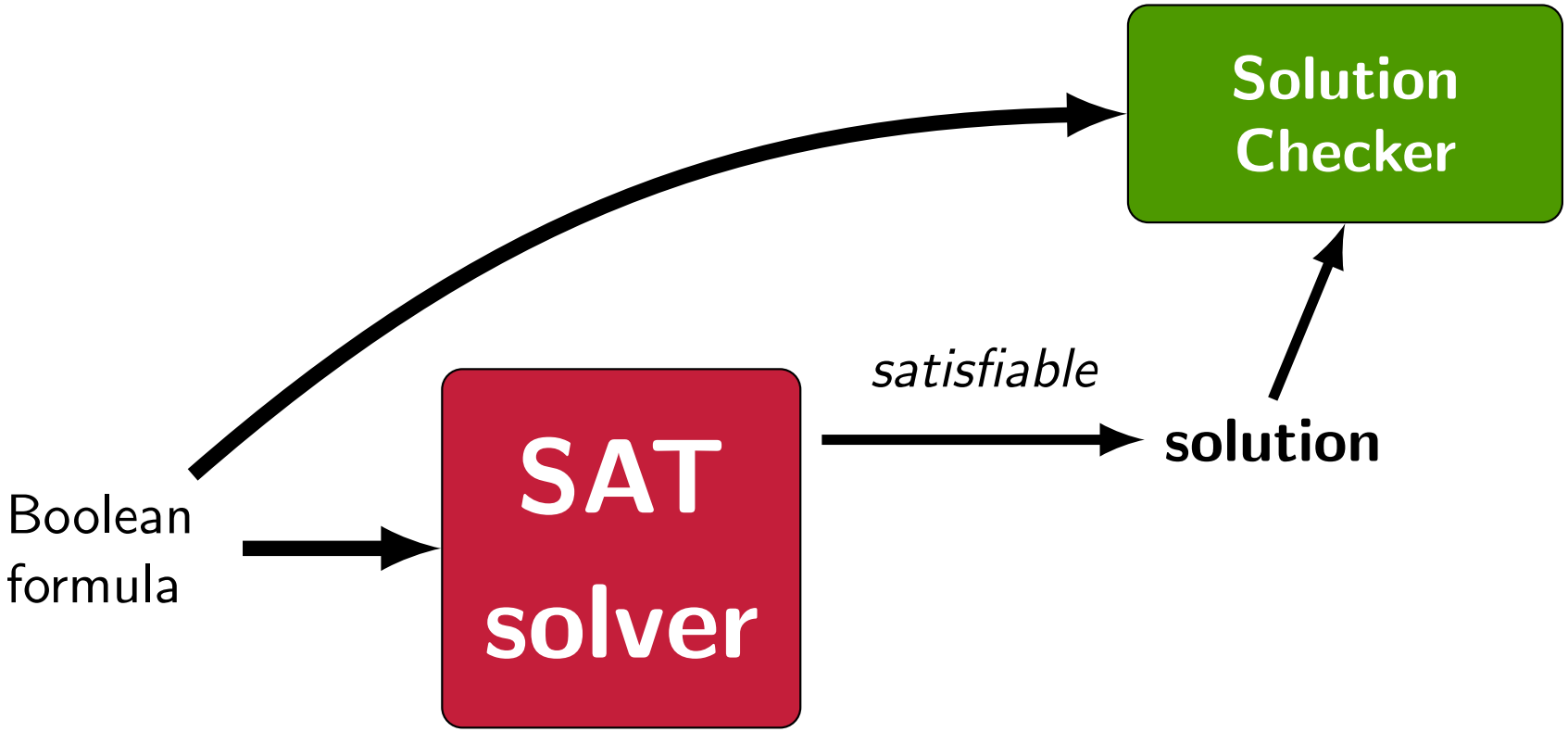
SAT Solvers Useful & Powerful

- Mathematical proofs
- Formal verification
- Optimization

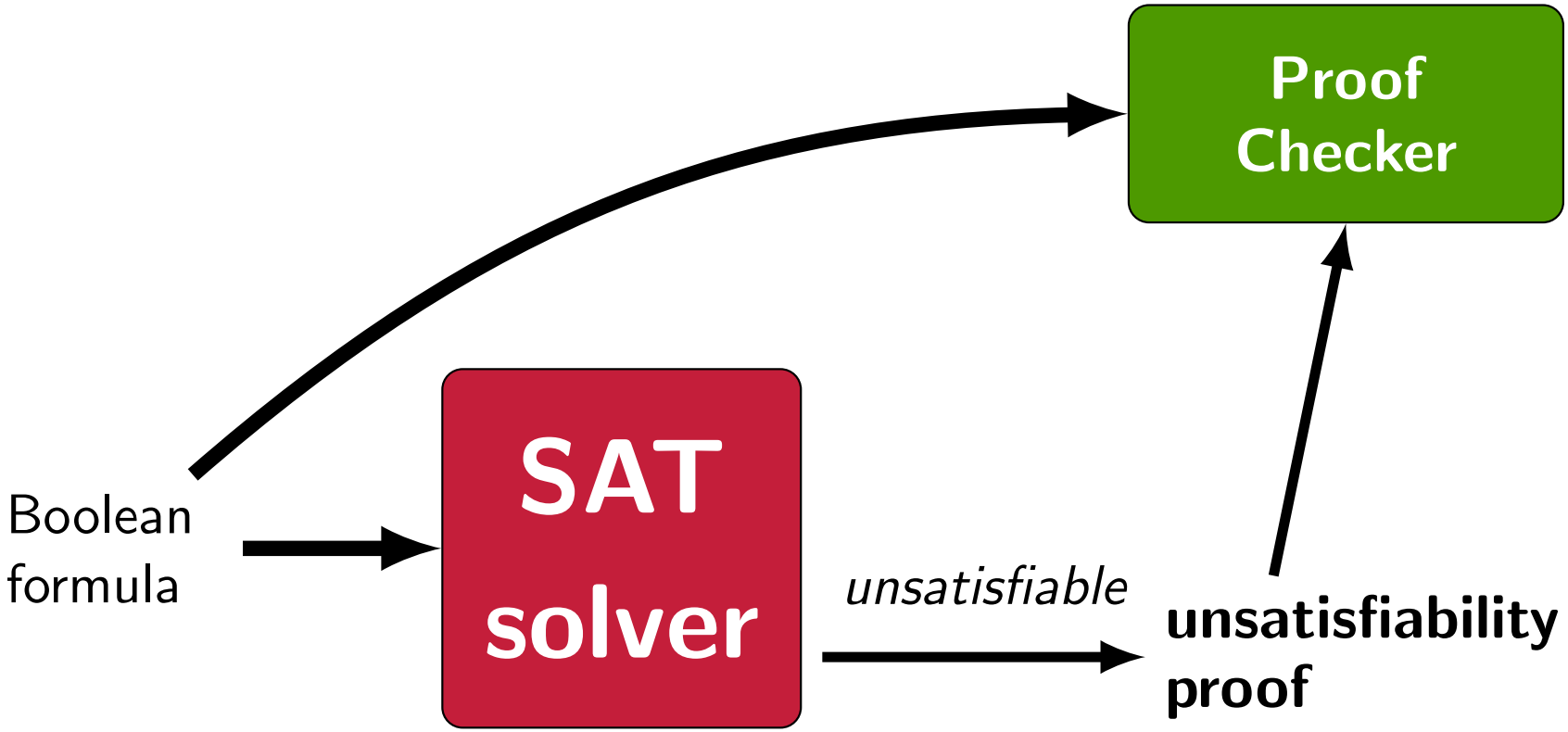
Can We Trust Them?

- 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

- Proof-generating BDD package to support other approaches to SAT

Motivating Example: Parity Benchmark

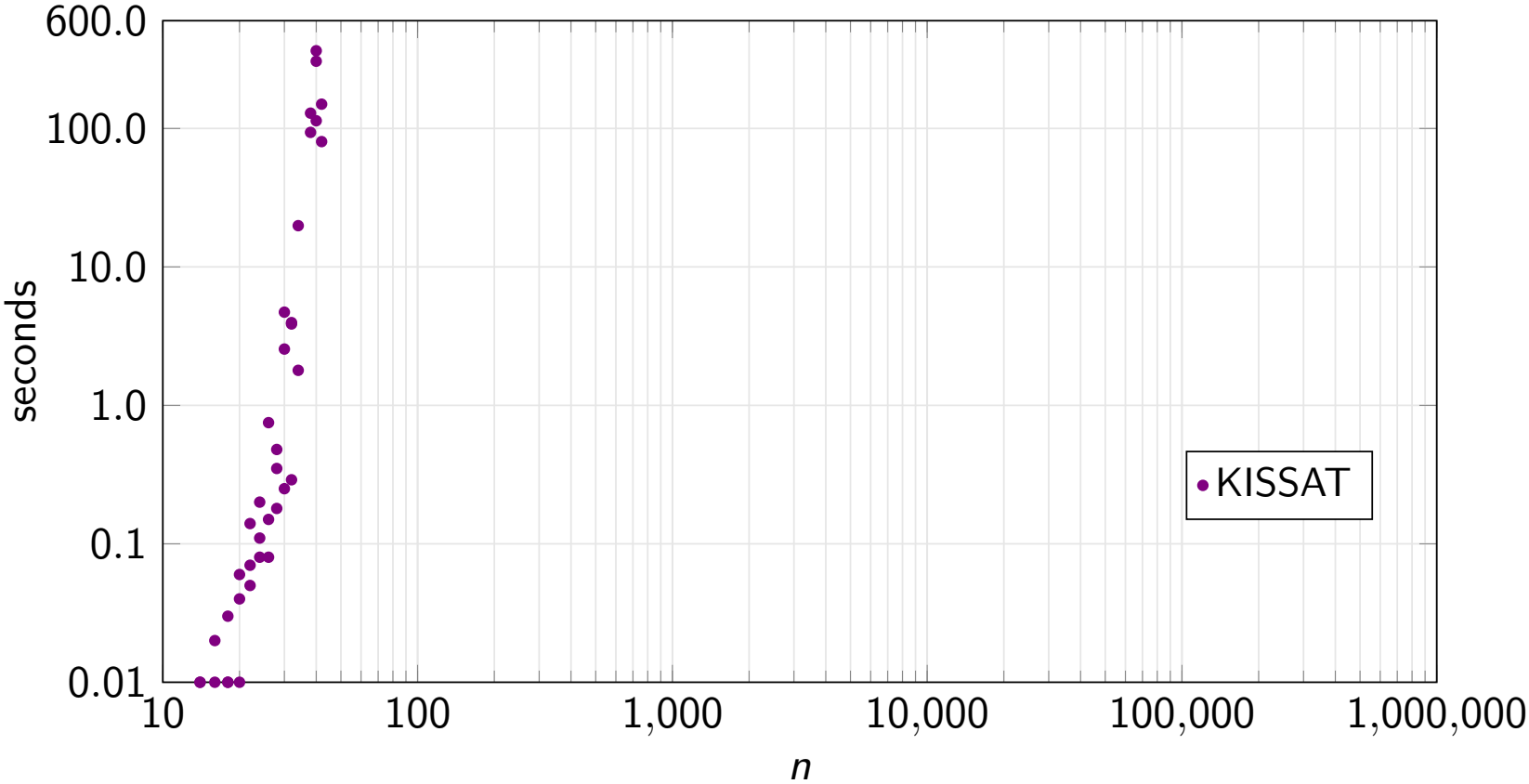
- Chew and Heule, SAT 2020

For n Boolean variables and random permutation π:

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

- Each expressed with $n-1$ three-way XOR constraints
- Conjunction unsatisfiable

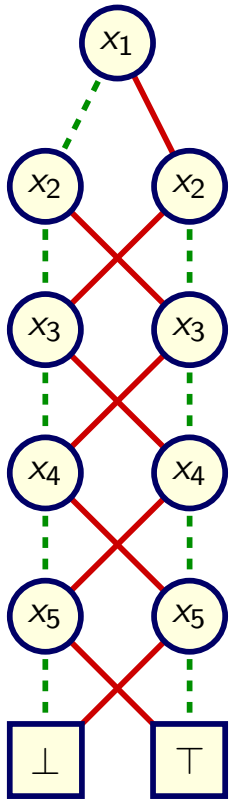
Parity Benchmark Runtime



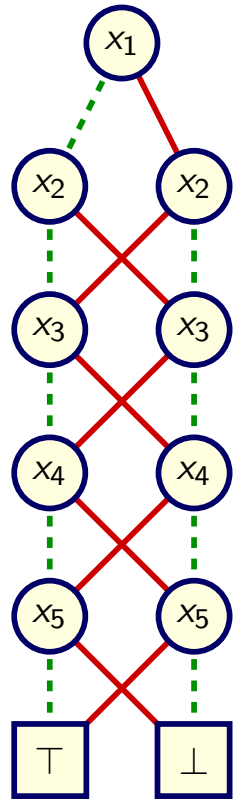
- KISSAT: State-of-the-art CDCL solver
- 3 different seeds for each value of n
- Cannot get beyond $n = 42$ within 600 seconds

BDD Representation of Parity Constraints

Odd Parity



Even Parity



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

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 $C_1, \ldots, C_{i-1}, C_i$.

- $C_t = [\,]$
  - Empty clause unsatisfiable
  - $\Rightarrow$ Original formula 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
- PGPBS: Bryant, Biere, Heule, 2022
  - Supports pseudo-Boolean reasoning

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 Comparison

UNSAT Proof from CDCL Solver

- Clauses describe detected conflicts
- Keep narrowing search space until it becomes empty

UNSAT Proof from BDD-Based Solver

- Step by step justification for each node generated
- Sequence of BDD operations leading to leaf node $\bot$.

Both within DRAT Framework

- Proof system can accomodate variety of styles

TBUDDY Trusted BDD Package

Concept

- BDD package with built-in support for proof generation
- Generate clausal proof as BDD operations proceed

Applications

- Implement standalone solver TBSAT
- Incorporate into other solvers

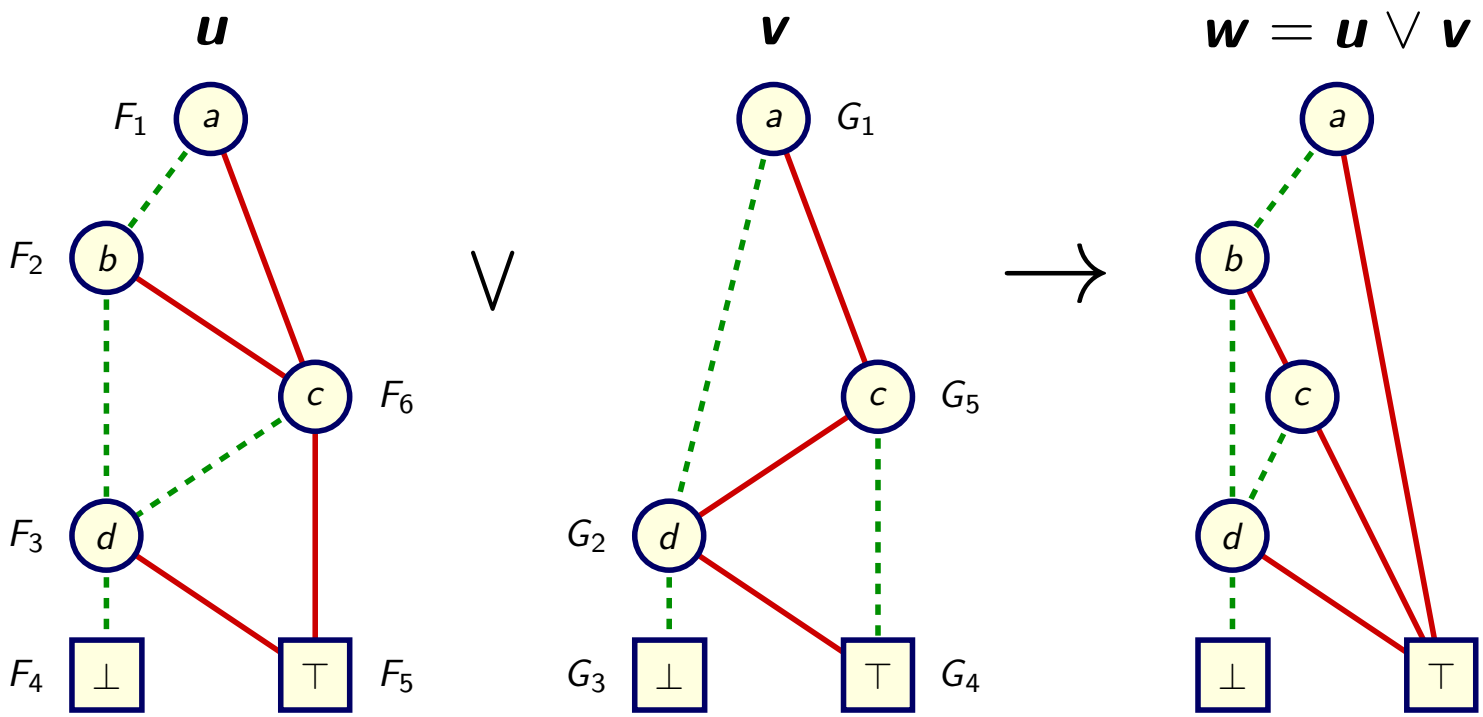
Implementation

- Build on BUDDY BDD package
- Also support parity reasoning

BDD Apply Algorithm

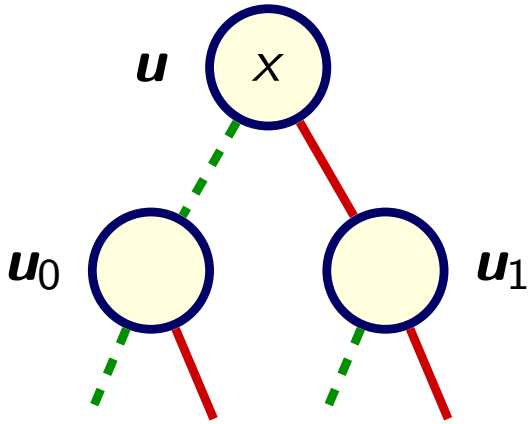
$$\boldsymbol{w} \leftarrow \boldsymbol{u} \odot \boldsymbol{v}$$

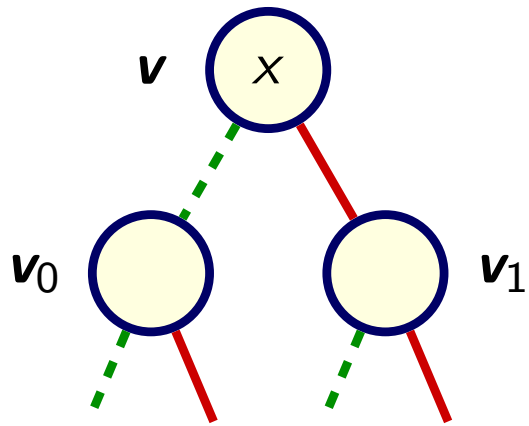
- $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$ BDD root nodes representing Boolean functions
- $\odot$ binary Boolean operator
  - E.g., $\wedge$, $\vee$, $\oplus$



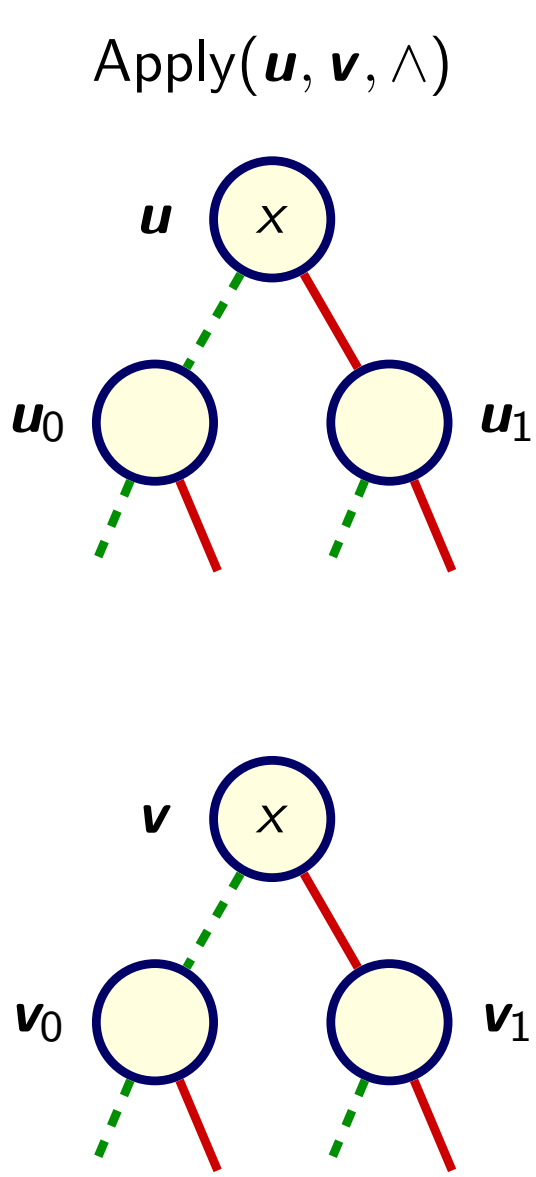
Apply Algorithm Recursion

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





Apply Algorithm Recursion

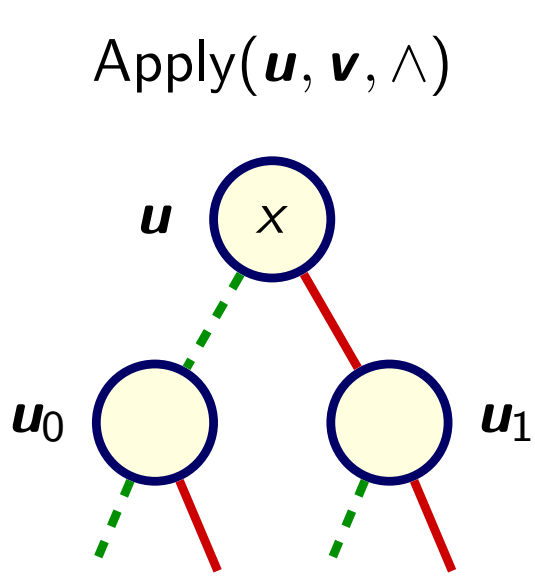


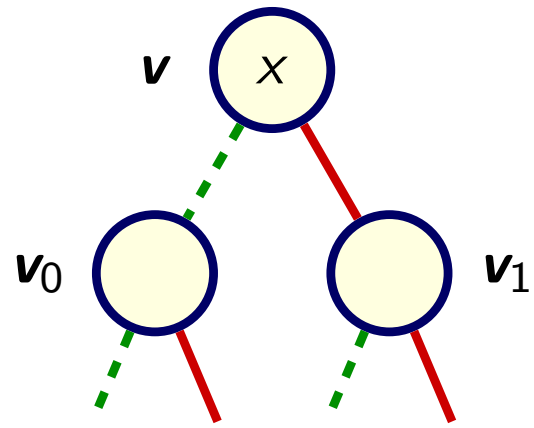
Recursion

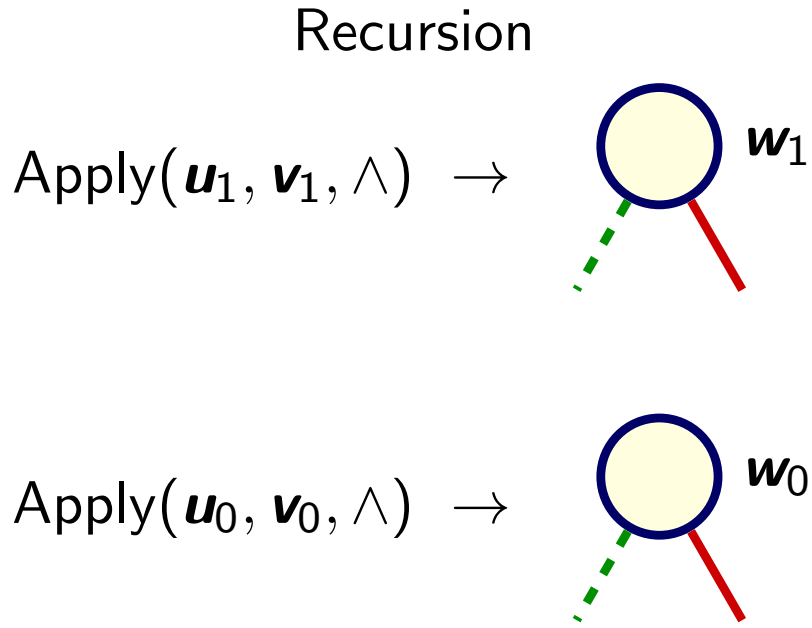
Apply(u_1, v_1, $\wedge$) $\rightarrow$ w_1

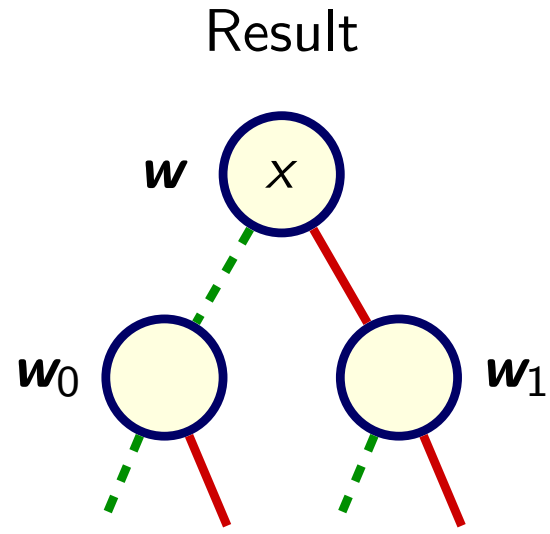
Apply(u_0, v_0, $\wedge$) $\rightarrow$ w_0

Apply Algorithm Recursion



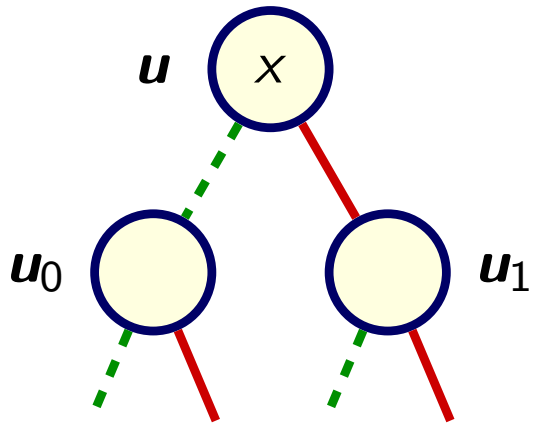






Generating Extended Resolution Proofs

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



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

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($\boldsymbol{u}$)	$\overline{x} \rightarrow (u_0 \rightarrow u)$	$[x \vee \overline{u}_0 \vee u]$

Proof-Generating Apply Operation

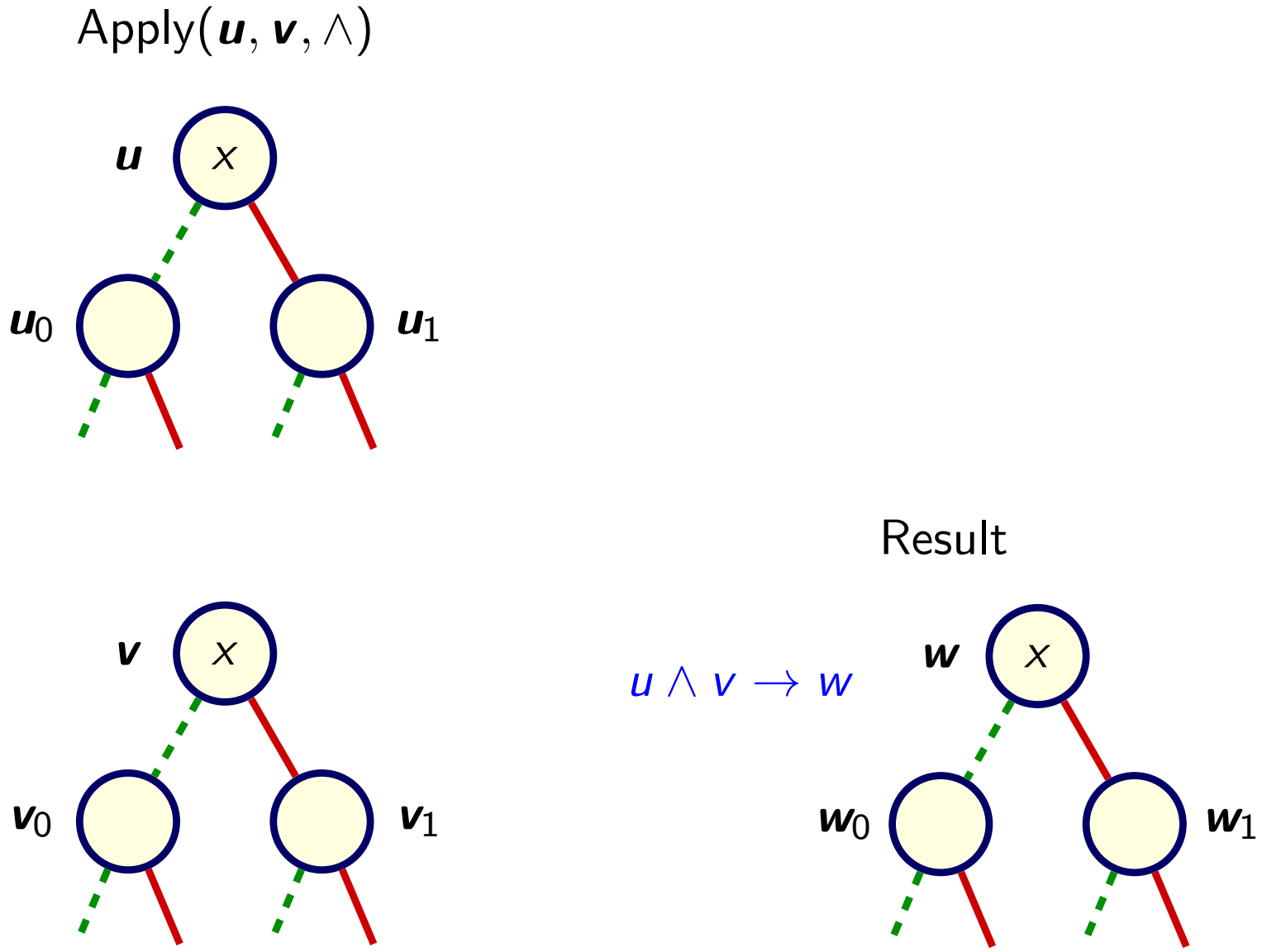
Integrate Proof Generation into Apply Operation

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

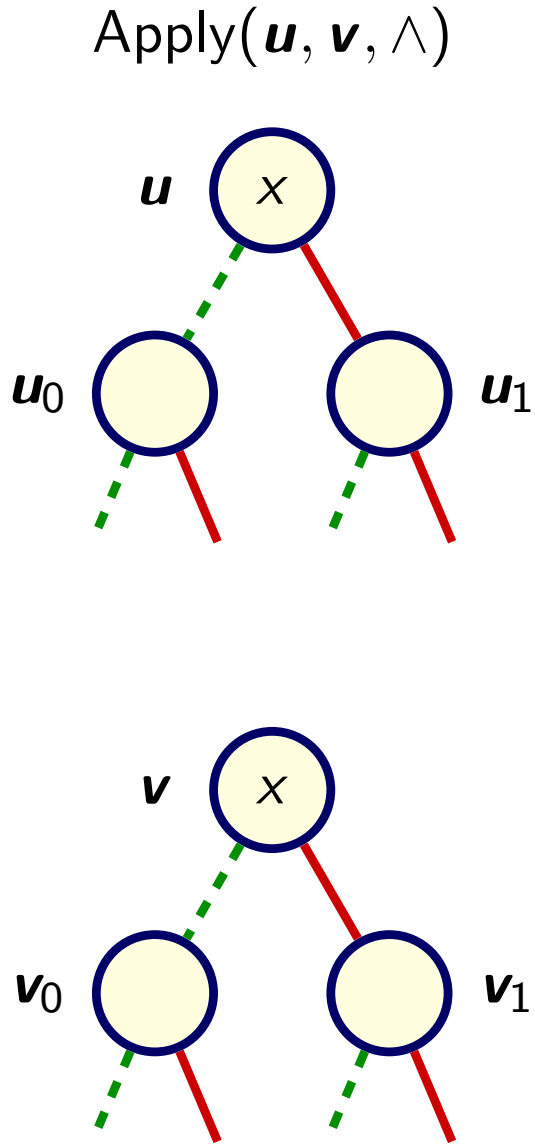
Key Idea:

Proof follows recursion of the Apply *algorithm*

Apply Algorithm Recursion



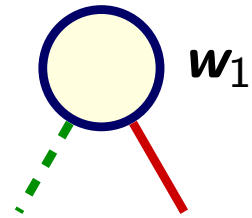
Apply Algorithm Recursion



Recursion

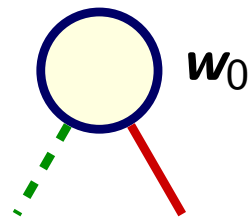
Apply($u_1, v_1, \wedge$) $\rightarrow$

$u_1 \wedge v_1 \rightarrow w_1$

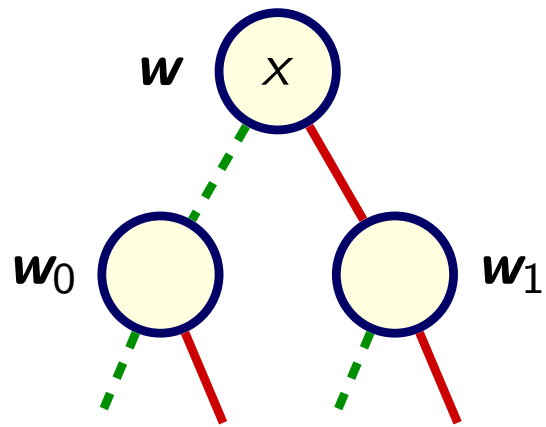


Apply($u_0, v_0, \wedge$) $\rightarrow$

$u_0 \wedge v_0 \rightarrow w_0$



Result



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

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

Quantification Operation

Operation EQuant(f, 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

- EQuant$(\boldsymbol{u}, x) \rightarrow \boldsymbol{v}$
- Separately run ProveImplication$(\boldsymbol{u}, \boldsymbol{v})$
  - Generates proof $u \rightarrow v$
  - Algorithm similar to proof-generating Apply operation

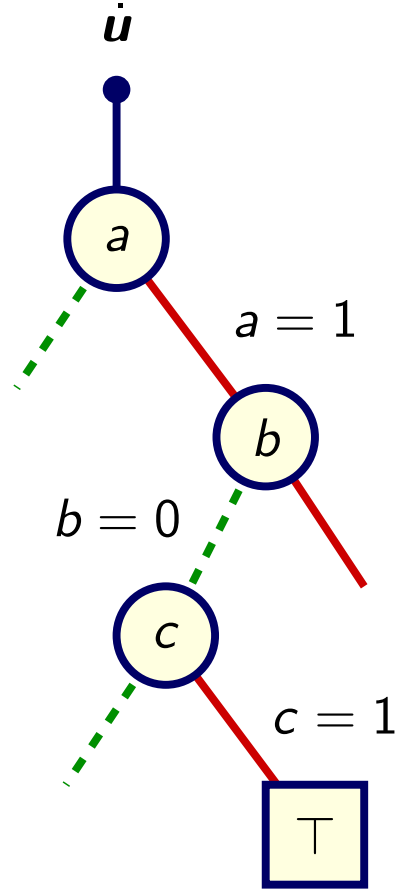
Trusted BDDs (TBDDs)

Components of TBDD $\dot{u}$

- BDD with root node u
- Associated extension variable u
- Proof step for unit clause $[u]$

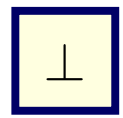
Interpretation. For input formula ϕ:

- $\phi \models u$
- Any variable assignment that satisfies ϕ traces path in BDD from u to leaf node $\top$



Terminal Case

- $\dot{u} = \dot{\bot}$
- ϕ has no satisfying assignments



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 $[w]$

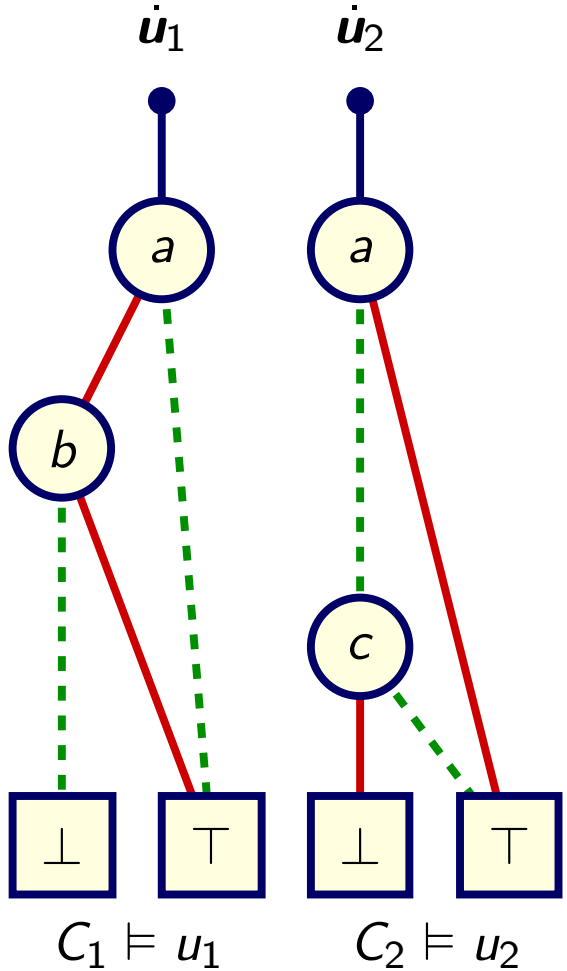
`tbdd tbdd_validate(bdd` $\boldsymbol{v}$`, tbdd` $\dot{\boldsymbol{u}}$`);`

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

TBDD Execution Example

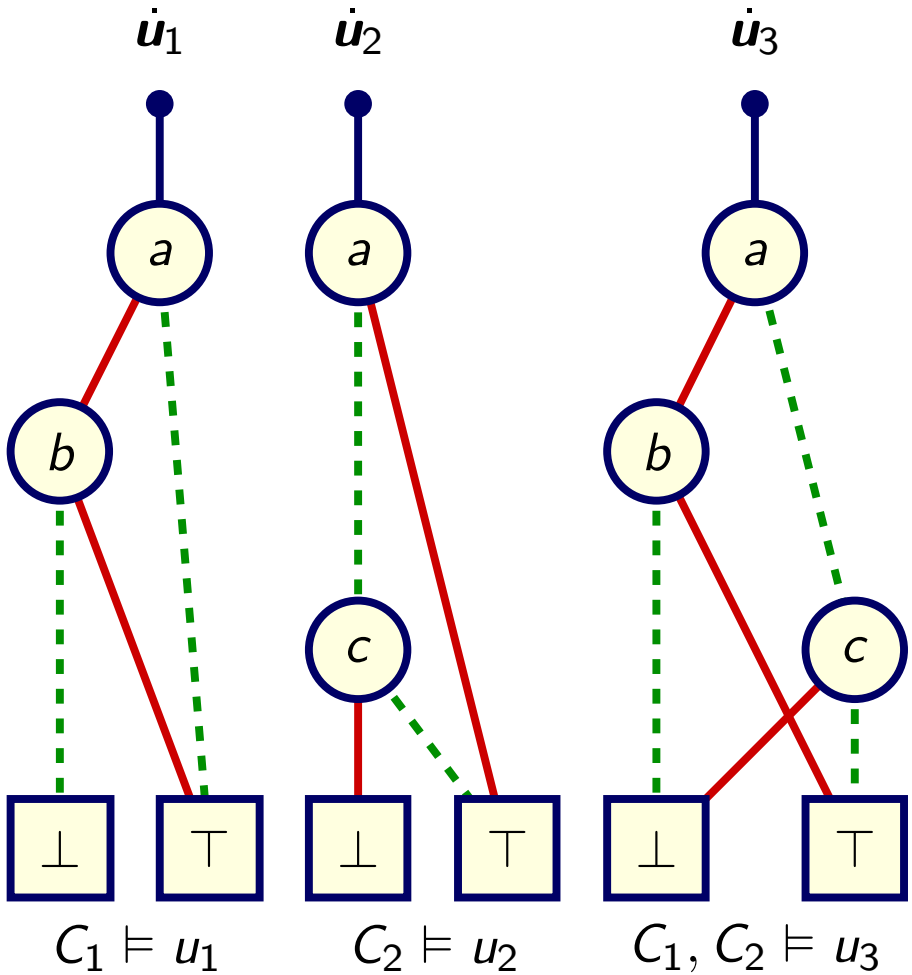
$\dot{u}_1 \longleftarrow$ `tbdd_from_clause`(C_1)

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



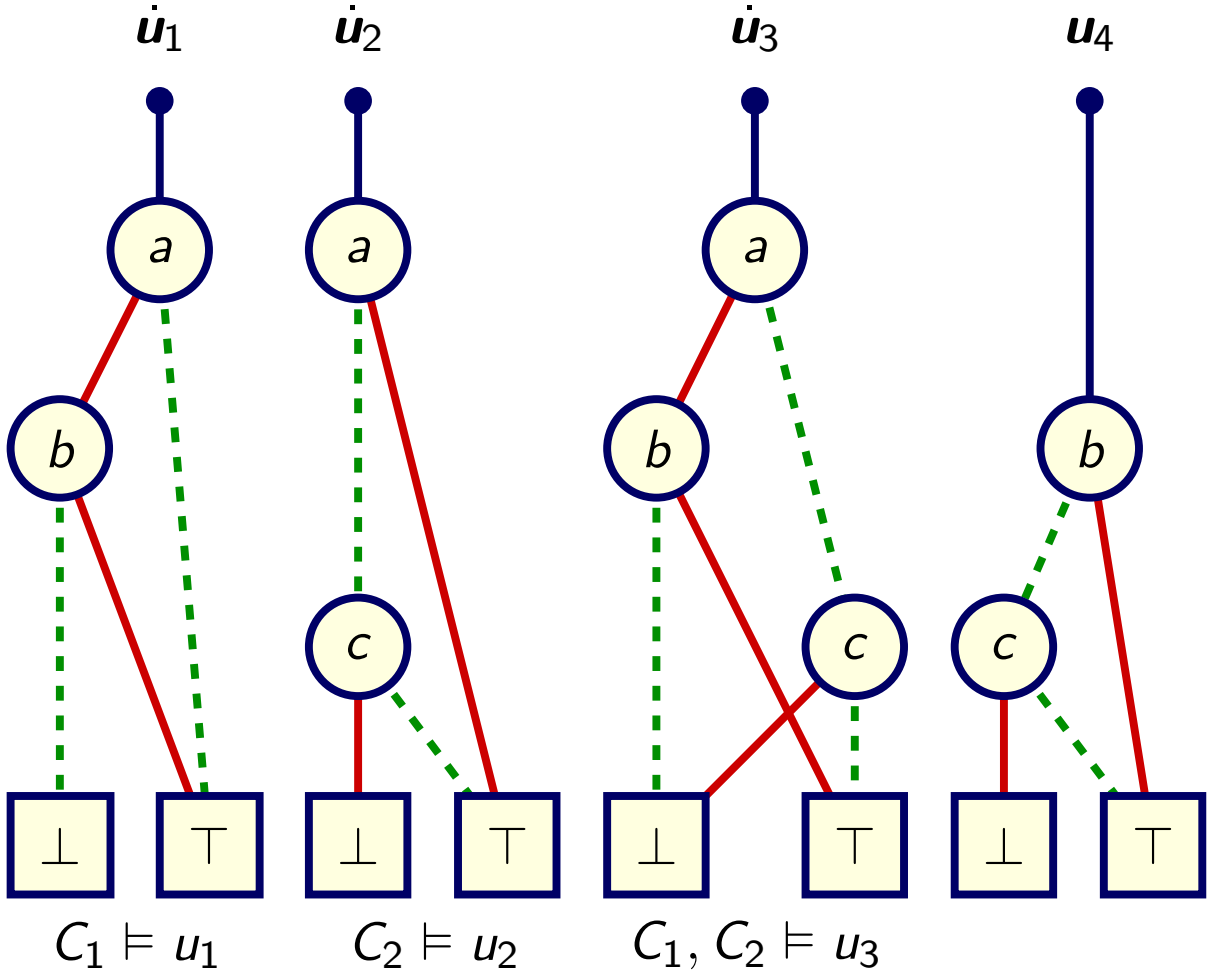
TBDD Execution Example

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



TBDD Execution Example

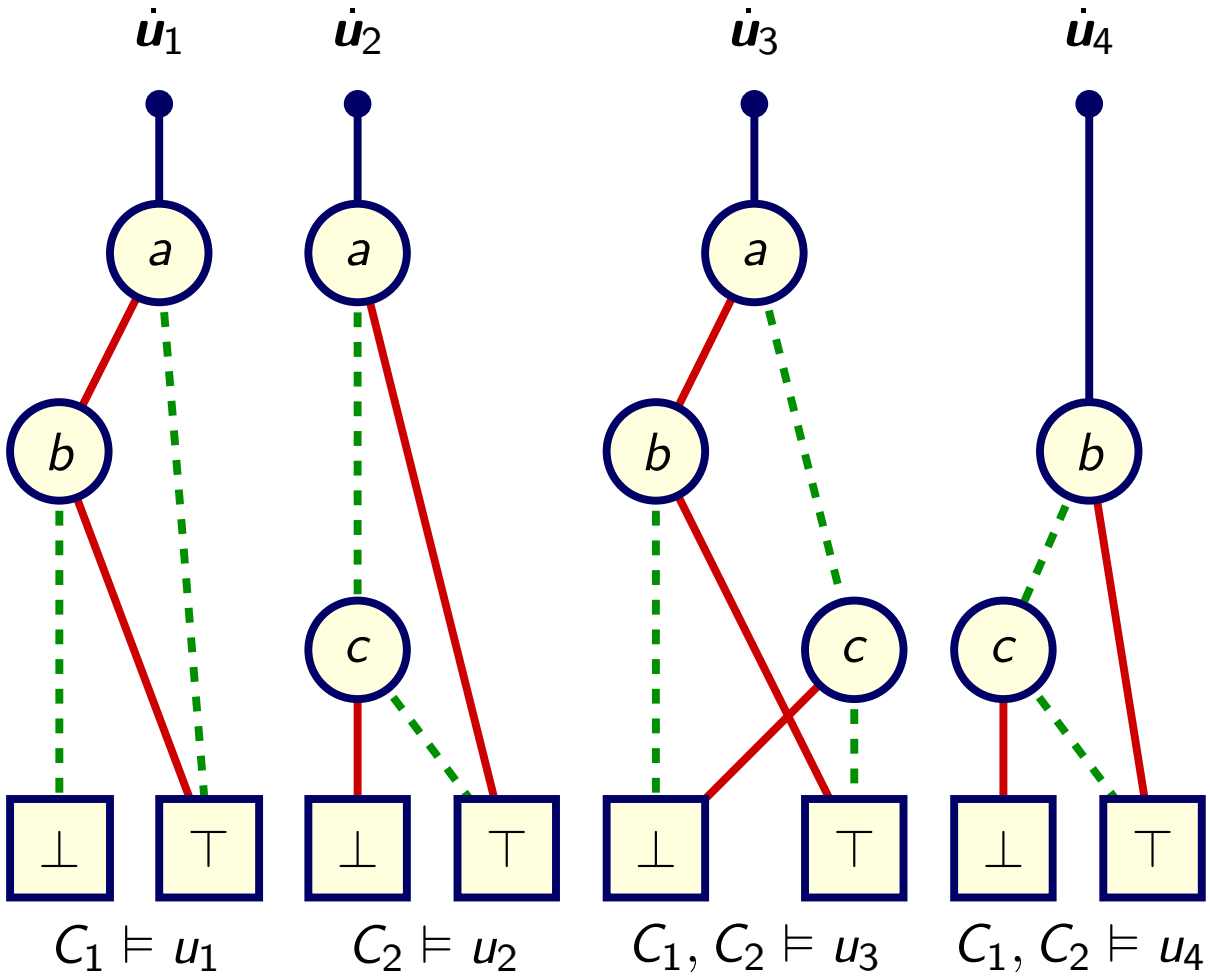
$\boldsymbol{u}_4 \longleftarrow$ `bdd_exists`$(\boldsymbol{u}_3, a)$



TBDD Execution Example

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

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



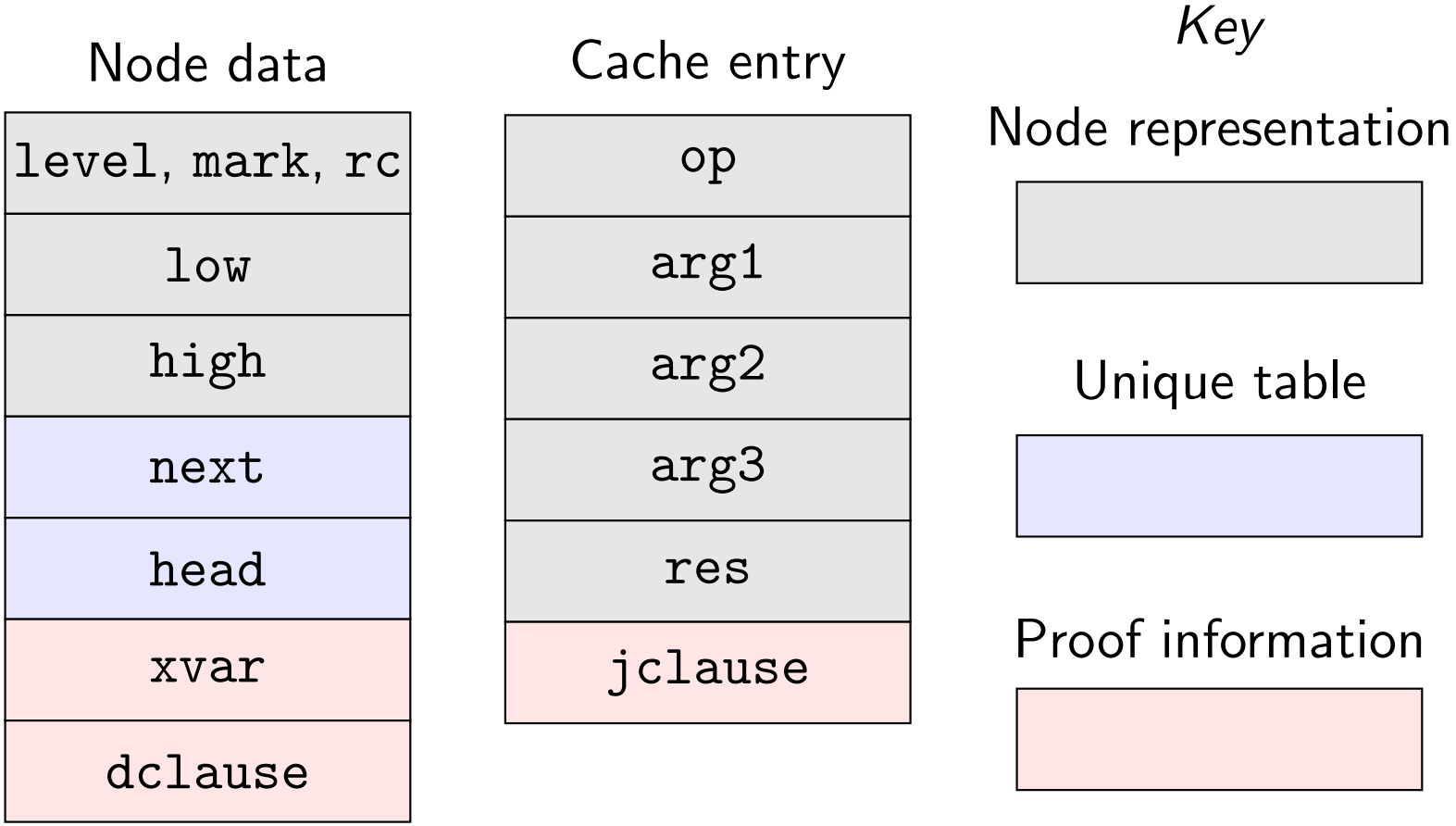
BuDDy BDD Package

BuDDy: Binary Decision Diagram package
Release 2.2

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

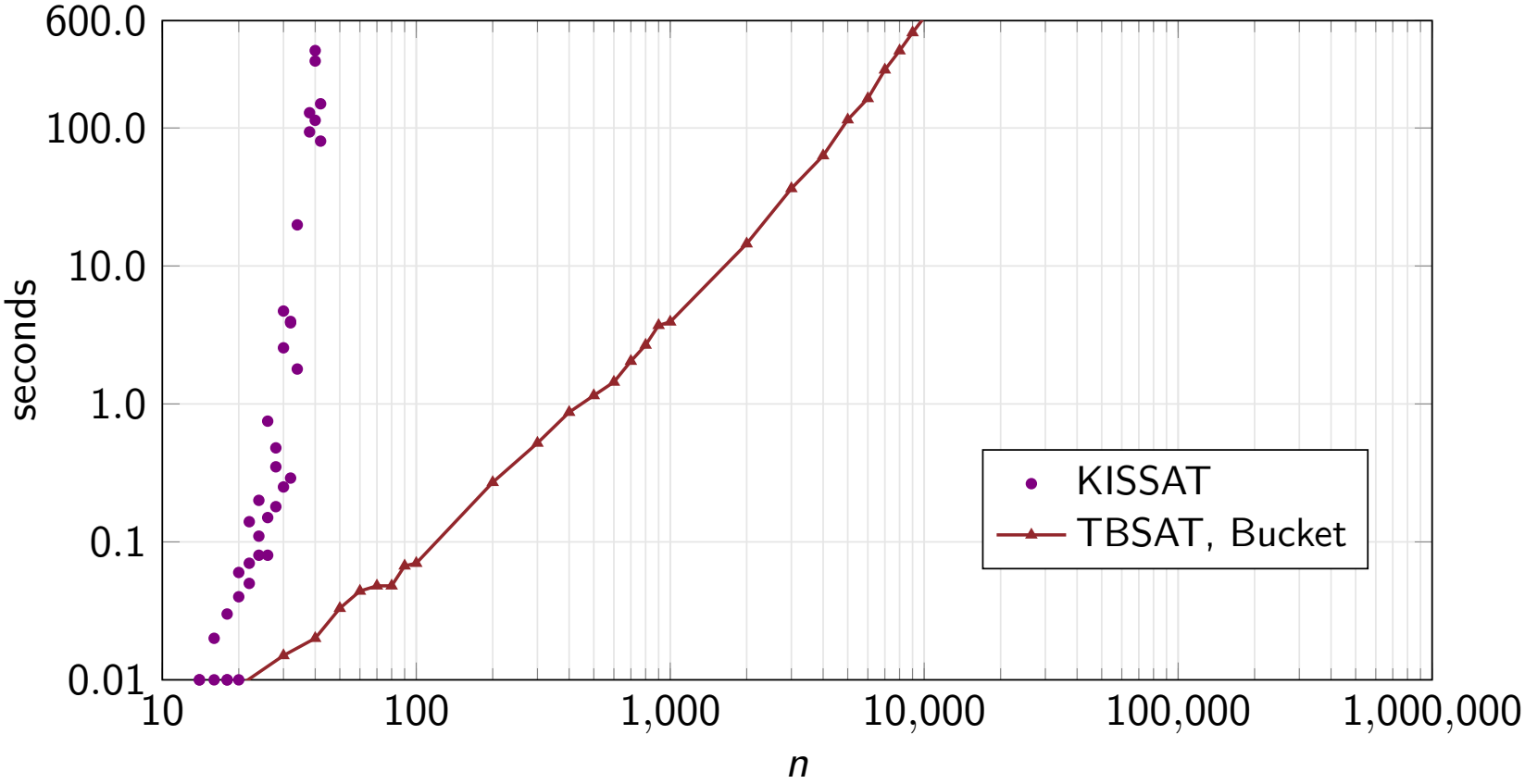
- ∼12K lines of code
- Clean, robust, and well documented
- Benchmark comparisons demonstrate good performance
- Node identified by 32-bit index into table
  - Rather than as 64-bit pointer

TBUDDY Data Structures



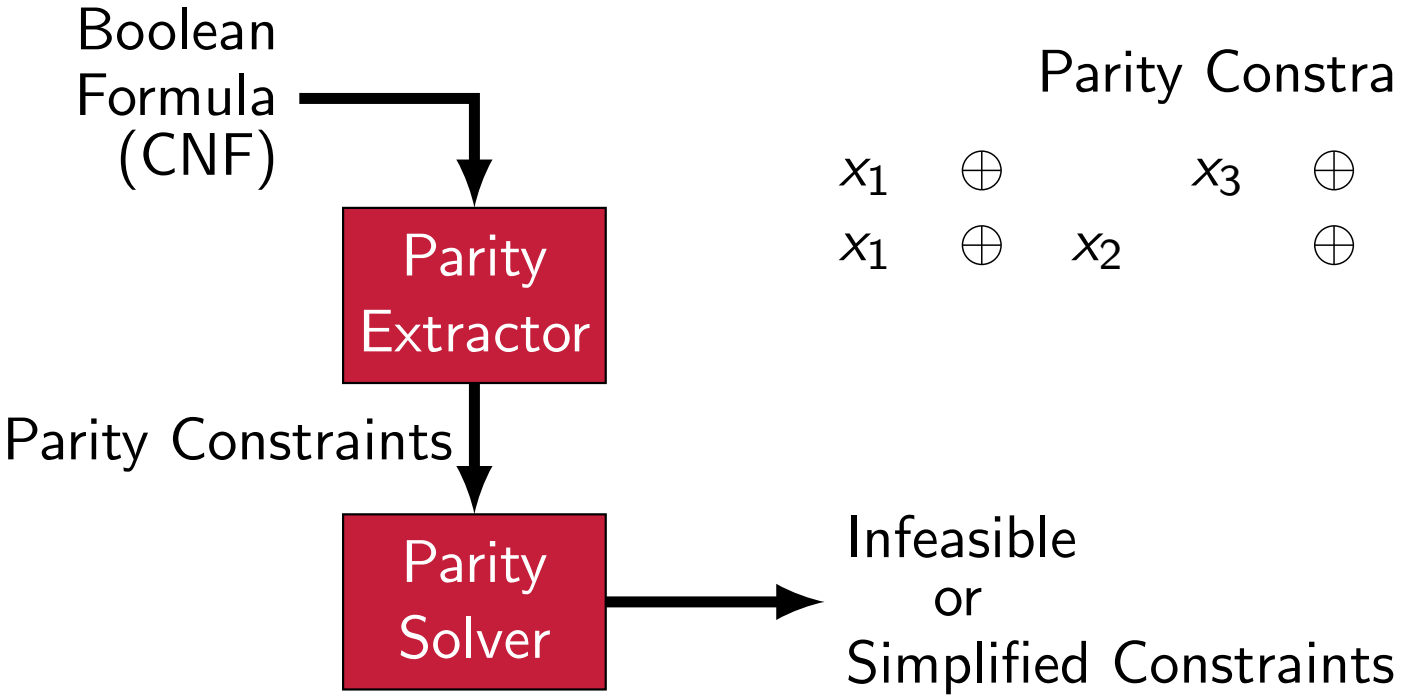
- Node entries: 20 bytes → 28 to store proof information
- Cache entry: Existing 24 bytes can also hold proof information
- Total memory overhead 1.35×

Parity Benchmark Runtime



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

Exploiting Parity Reasoning

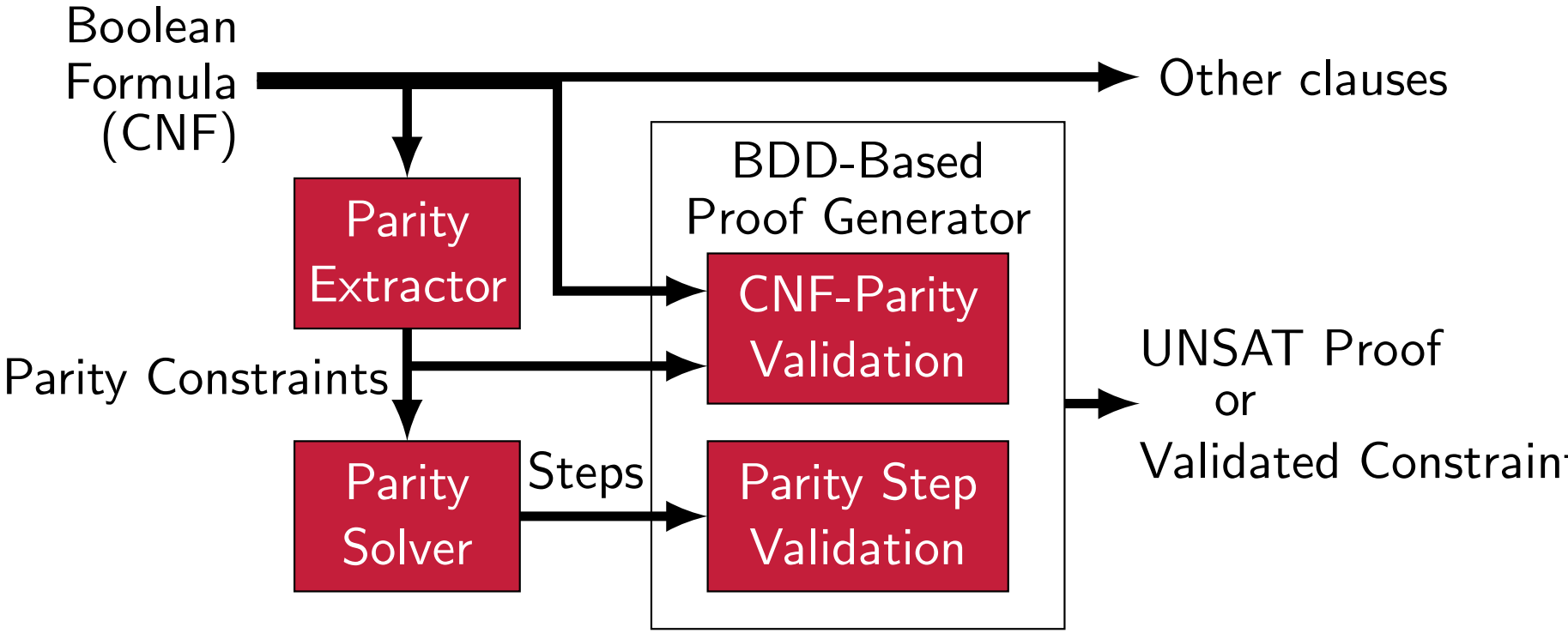


Parity Constraints

$$x_1 \oplus x_3 \oplus x_4 = 0$$
$$x_1 \oplus x_2 \oplus x_4 = 1$$

- View parity constraints as system of linear equations
- Reduce via Gaussian elimination
  - If get constraint $0 = 1$, then infeasible
  - Otherwise, can easily generate solutions
- **Challenge:** Certifying results

Integrating Parity Reasoning



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

Gaussian Elimination Over GF2

Parity Constraints $P_1, P_2, \ldots, P_m$, each of form

$$a_1 \cdot x_1 \oplus a_2 \cdot x_2 \oplus \cdots \oplus a_n \cdot x_n \quad = \quad p$$

- Coefficients $a_i \in \{0, 1\}$
- Phase $p \in \{0, 1\}$

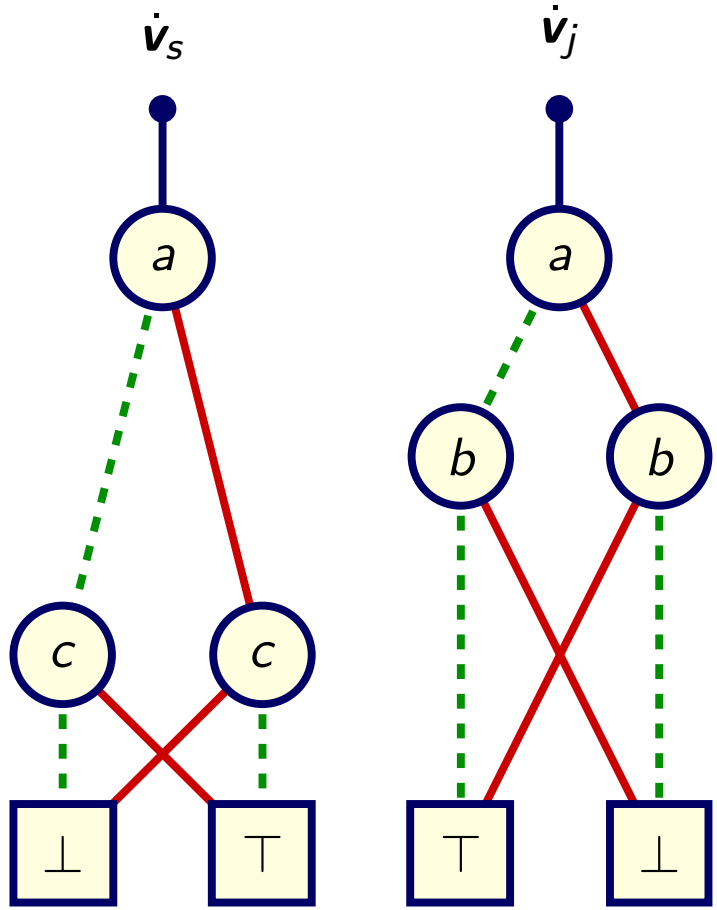
Elimination Step

- For pivot P_s, replace each constraint P_j by $P'_j = P_j \oplus P_s$

	a_1	a_2	a_3	a_4	p
P_s	1	0	1	1	0
P_j	1	1	0	1	1
P'_j	0	1	1	0	1

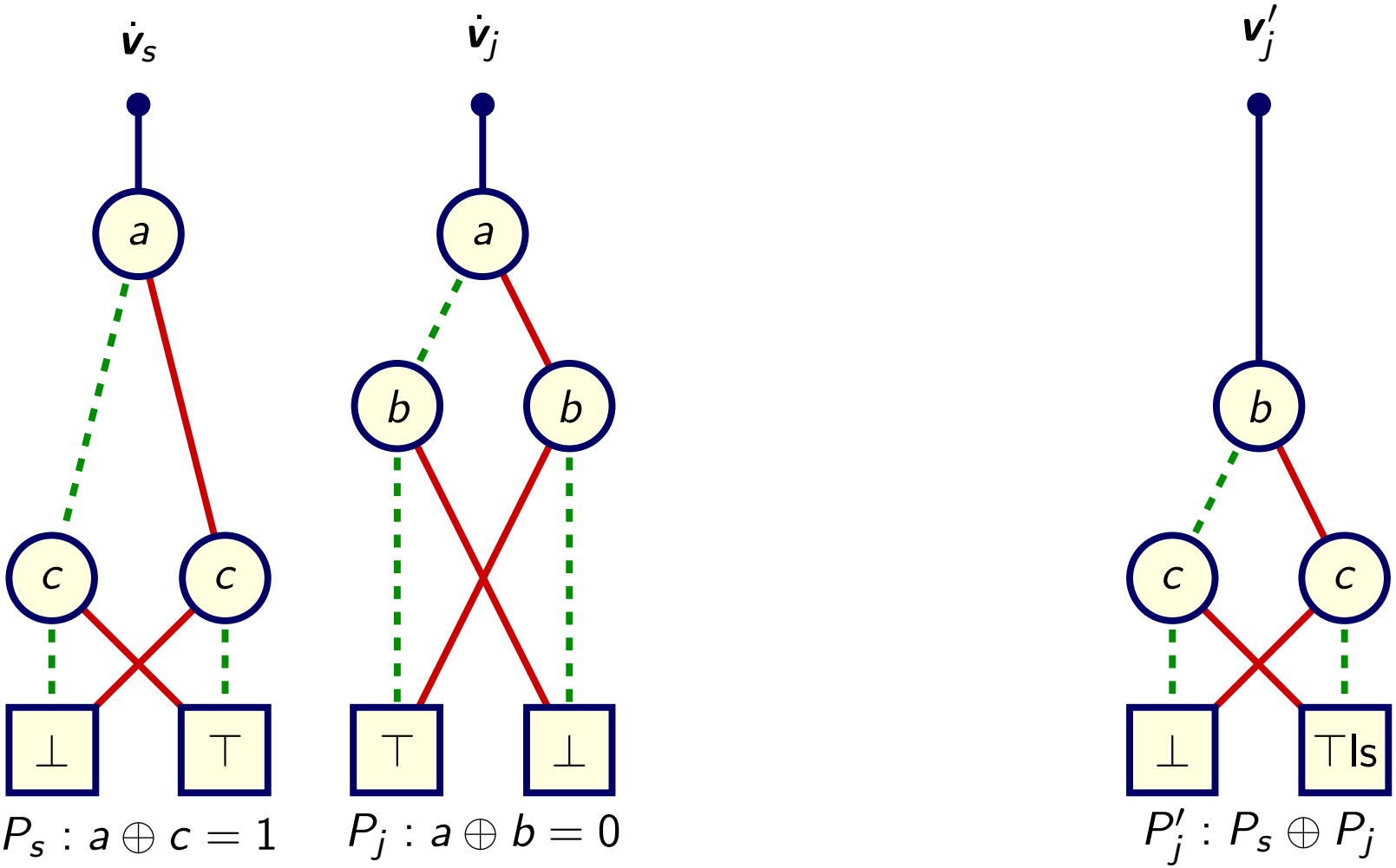
TBDD-Based Parity Reasoning Example

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



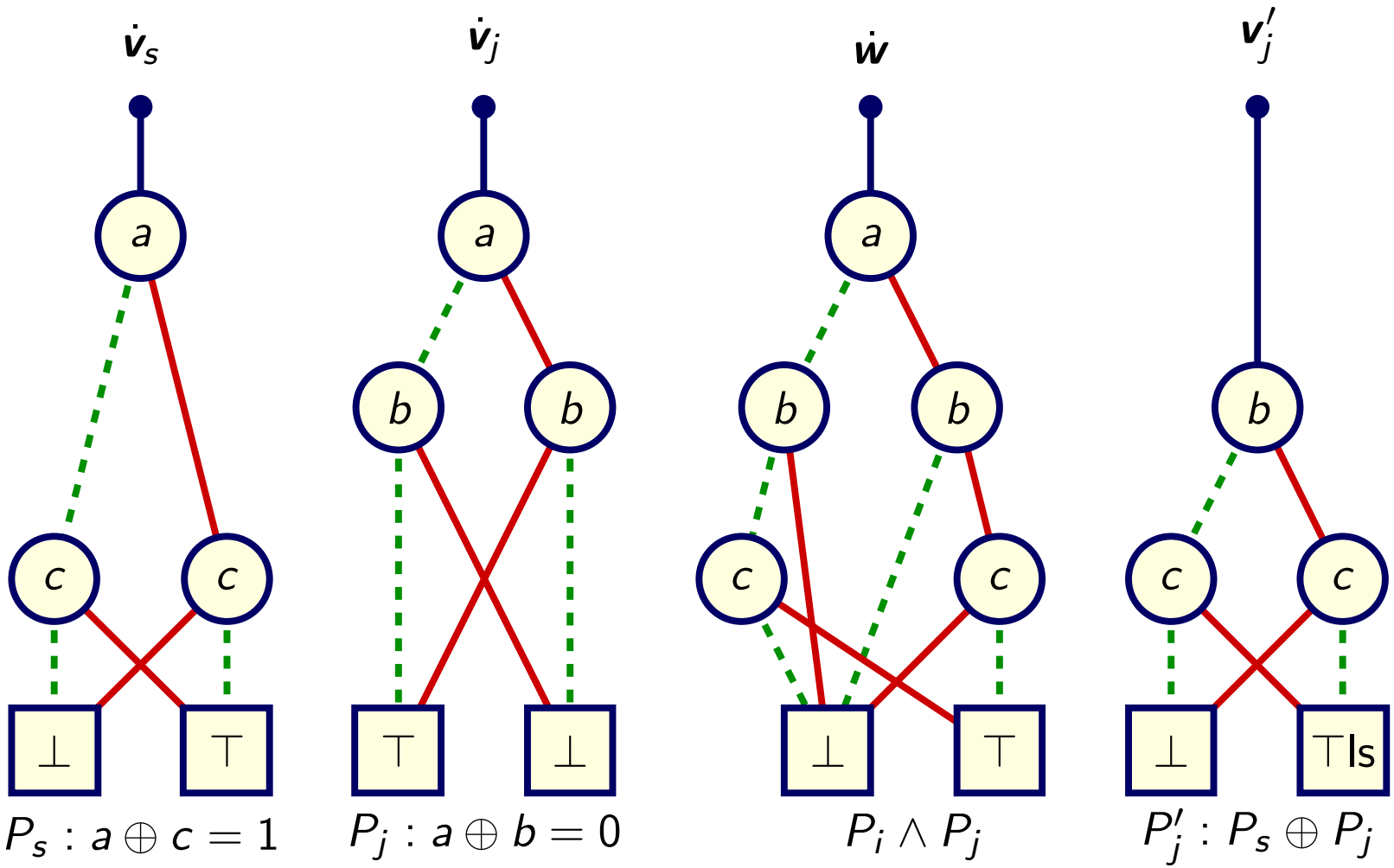
TBDD-Based Parity Reasoning Example

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



TBDD-Based Parity Reasoning Example

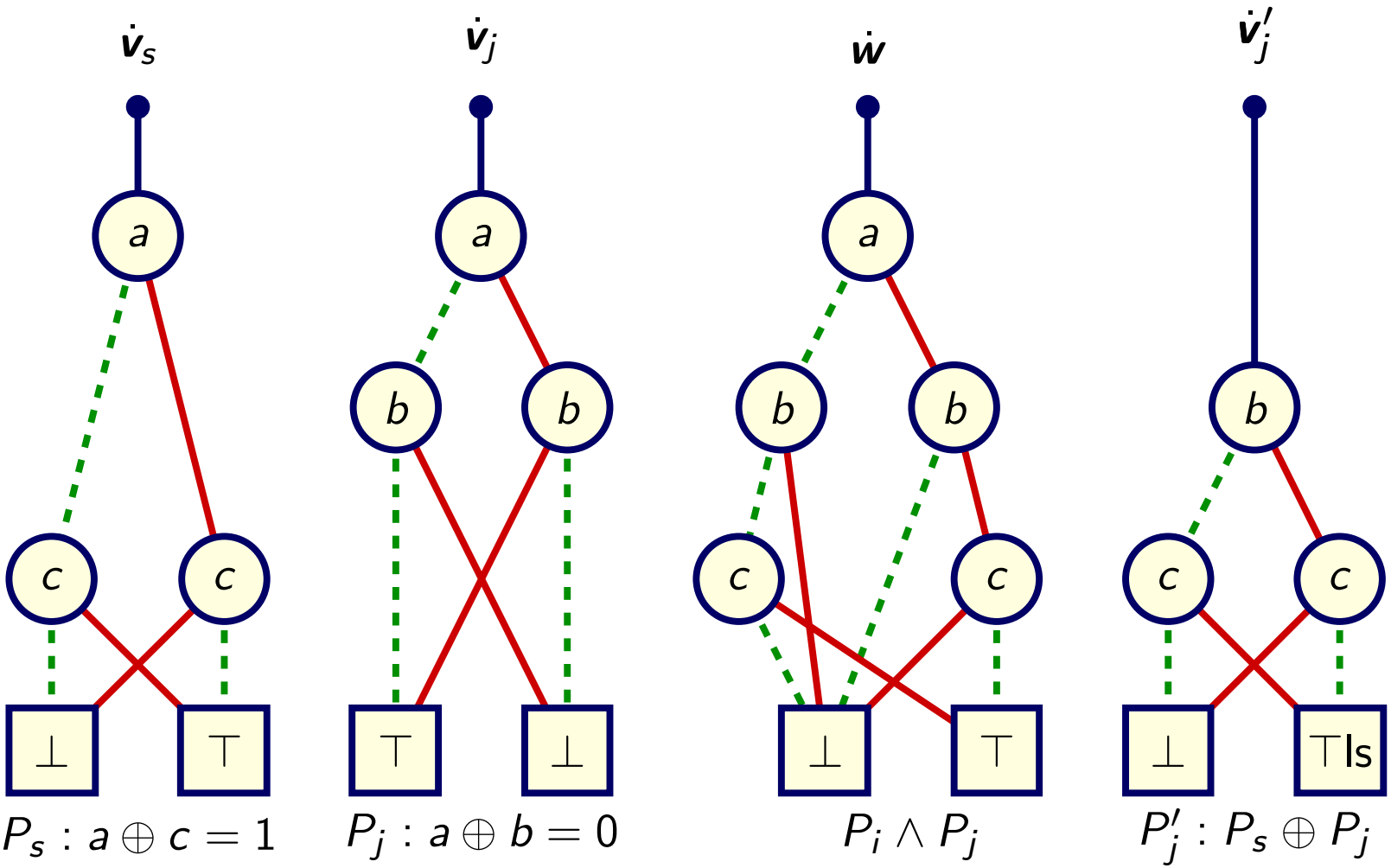
$\dot{w} \longleftarrow$ tbdd_and($\dot{v}_s$, $\dot{v}_j$)



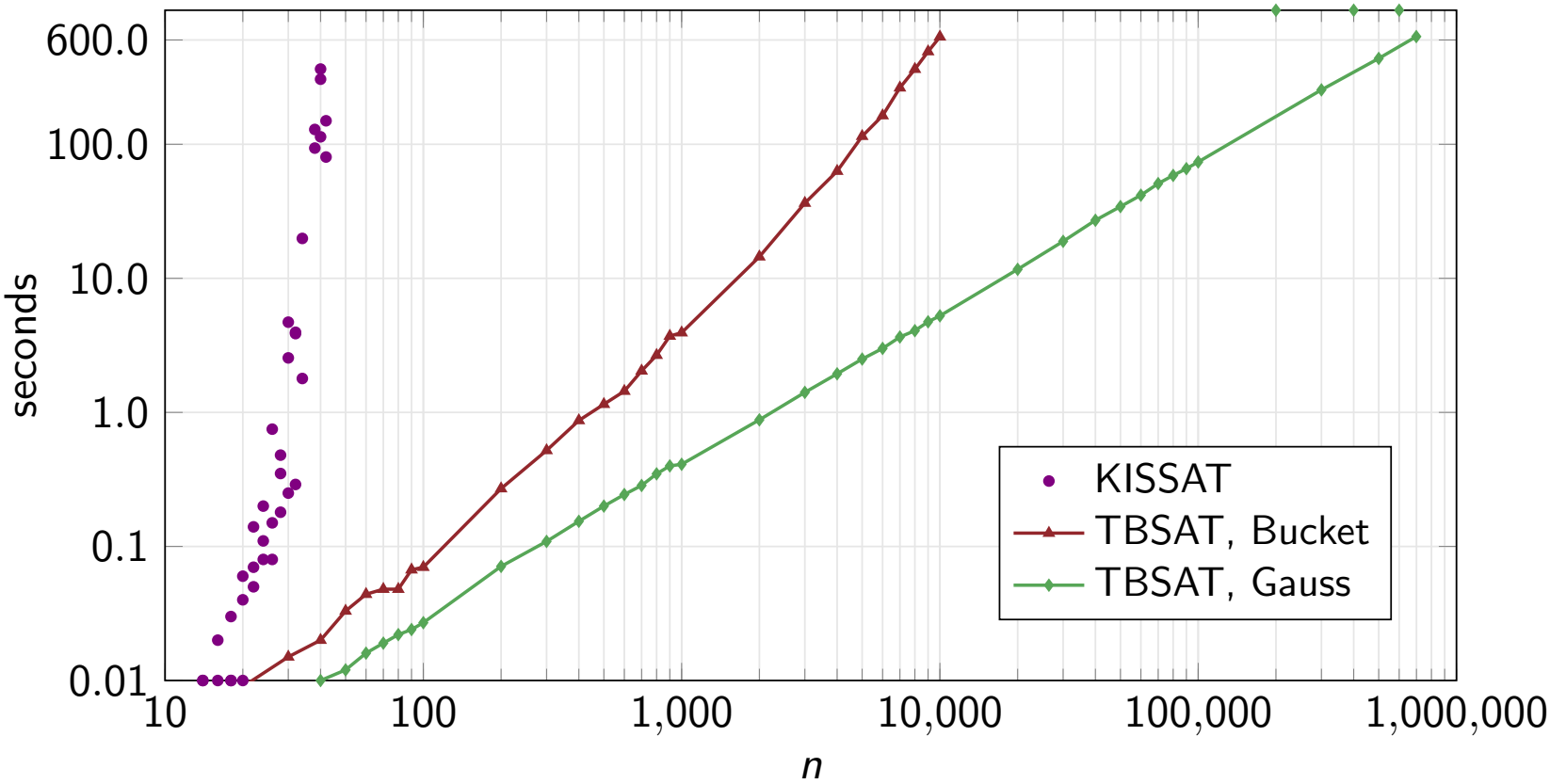
TBDD-Based Parity Reasoning Example

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

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

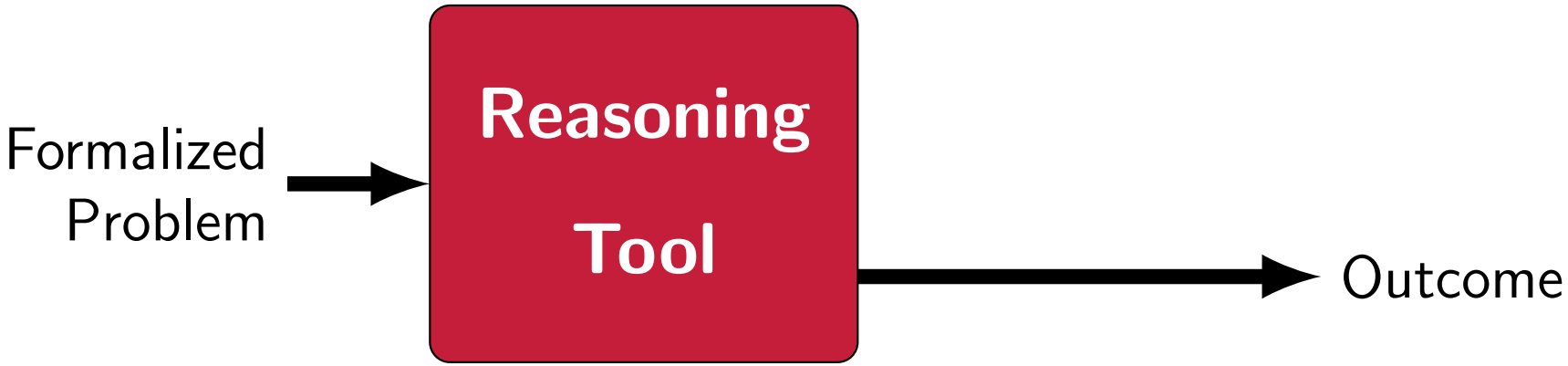


Parity Benchmark Runtime



- 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
- Some failures for large values of n due to poor pivot selection

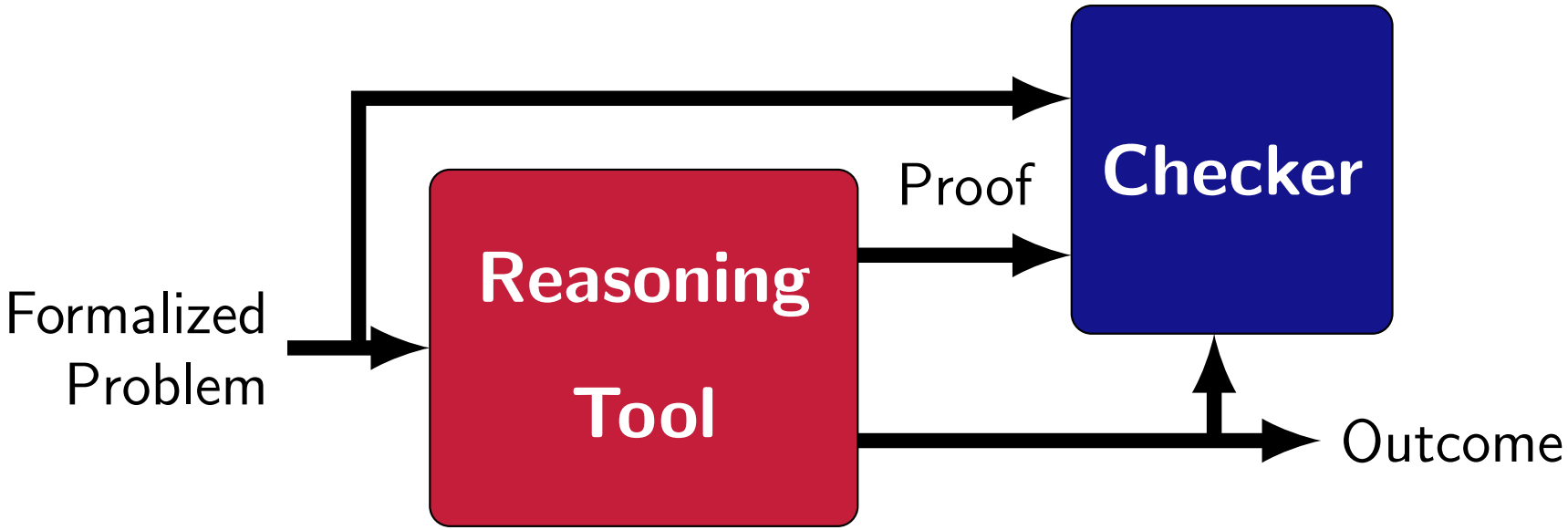
Trustworthy Automated Reasoning Programs



Standard Reasoning Tools

- Lingering doubt about whether result can be trusted

Trustworthy Automated Reasoning Programs



Standard Reasoning Tools

- Lingering doubt about whether result can be trusted

Proof-Generating Tools

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

Supporting other Reasoning Tools

CryptoMiniSAT Solver

- Mate Soos
- State-of-the-art CDCL solver
- Integrated Gauss-Jordan (G-J) solver for parity reasoning
- Solvers exchange unit propagations and conflicts

Previous Limitation

- Could not generate UNSAT proof when G-J enabled

Supporting other Reasoning Tools

CryptoMiniSAT Solver

- Mate Soos
- State-of-the-art CDCL solver
- Integrated Gauss-Jordan (G-J) solver for parity reasoning
- Solvers exchange unit propagations and conflicts

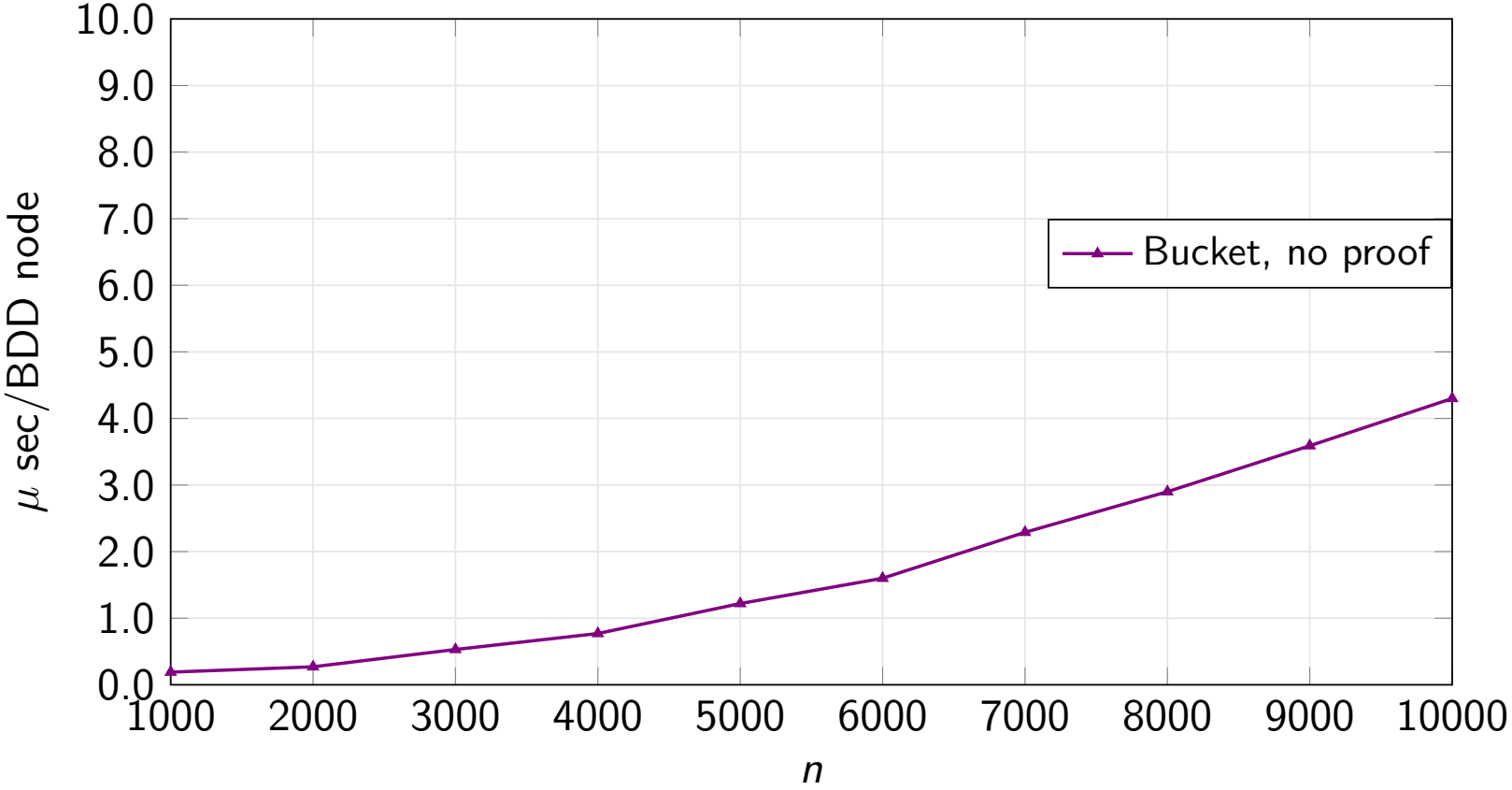
Previous Limitation

- Could not generate UNSAT proof when G-J enabled

Integrating Tbuddy

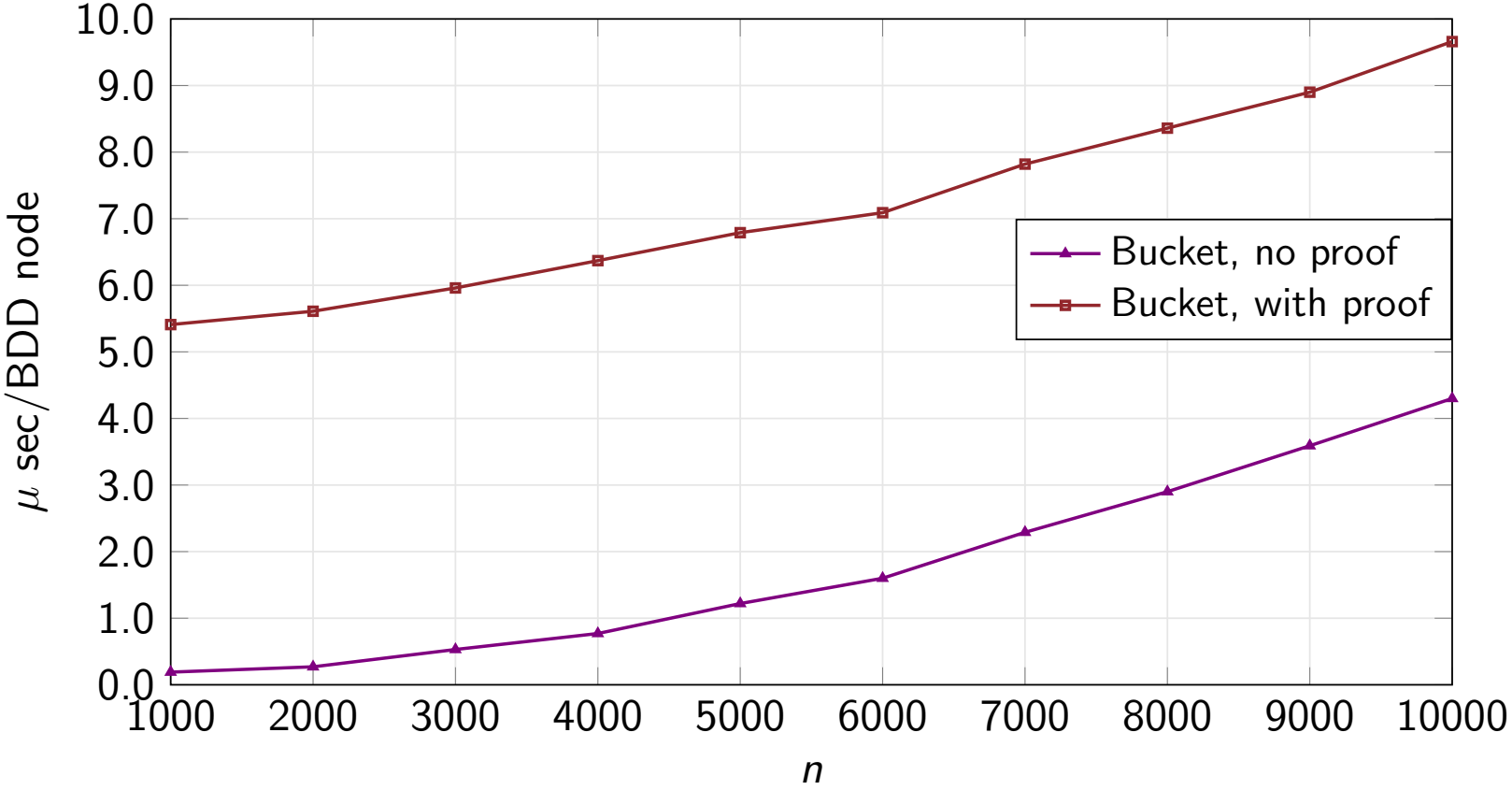
- Bryant & Soos, 2022
- Represent parity constraints with TBDDs
- Use TBDD operations to justify unit propagation steps

Parity Benchmark: Time Overhead



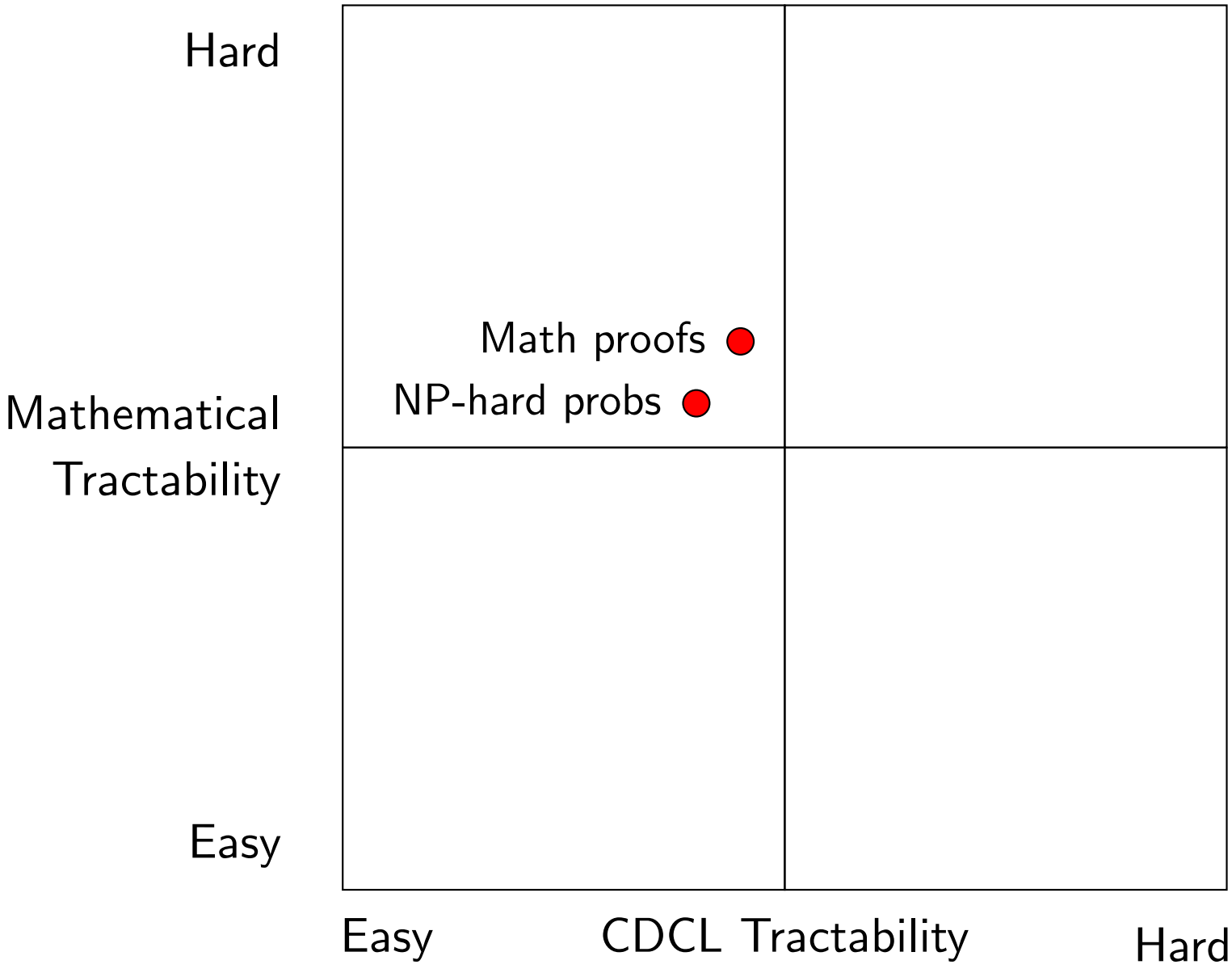
- Normalize time as μ seconds per BDD node generated
- BDD management costs increase as BDDs grow larger

Parity Benchmark: Time Overhead

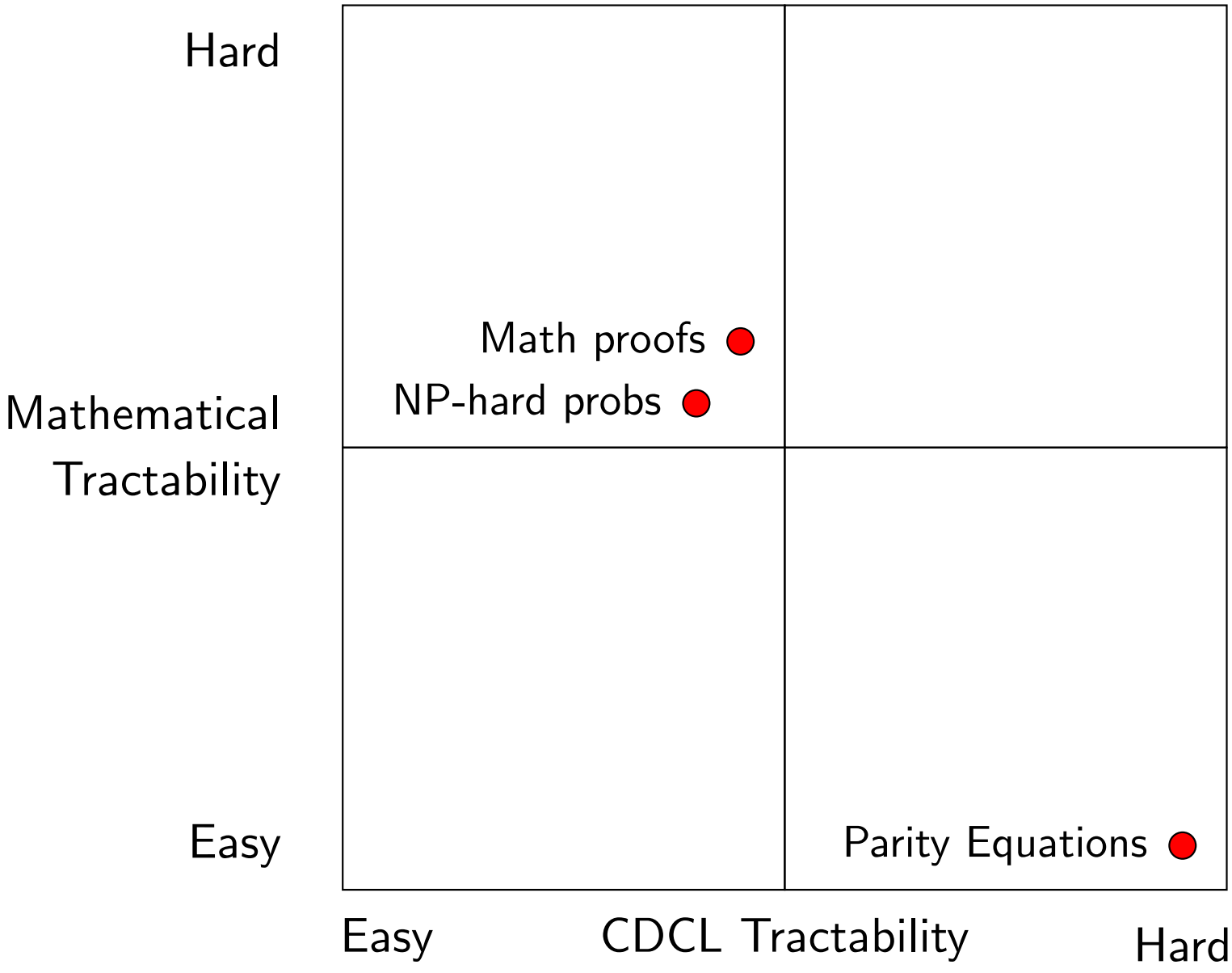


- Normalize time as μ seconds per BDD node generated
- BDD management costs increase as BDDs grow larger
- Proof generation adds fixed time of ≈ 5 μ seconds/node

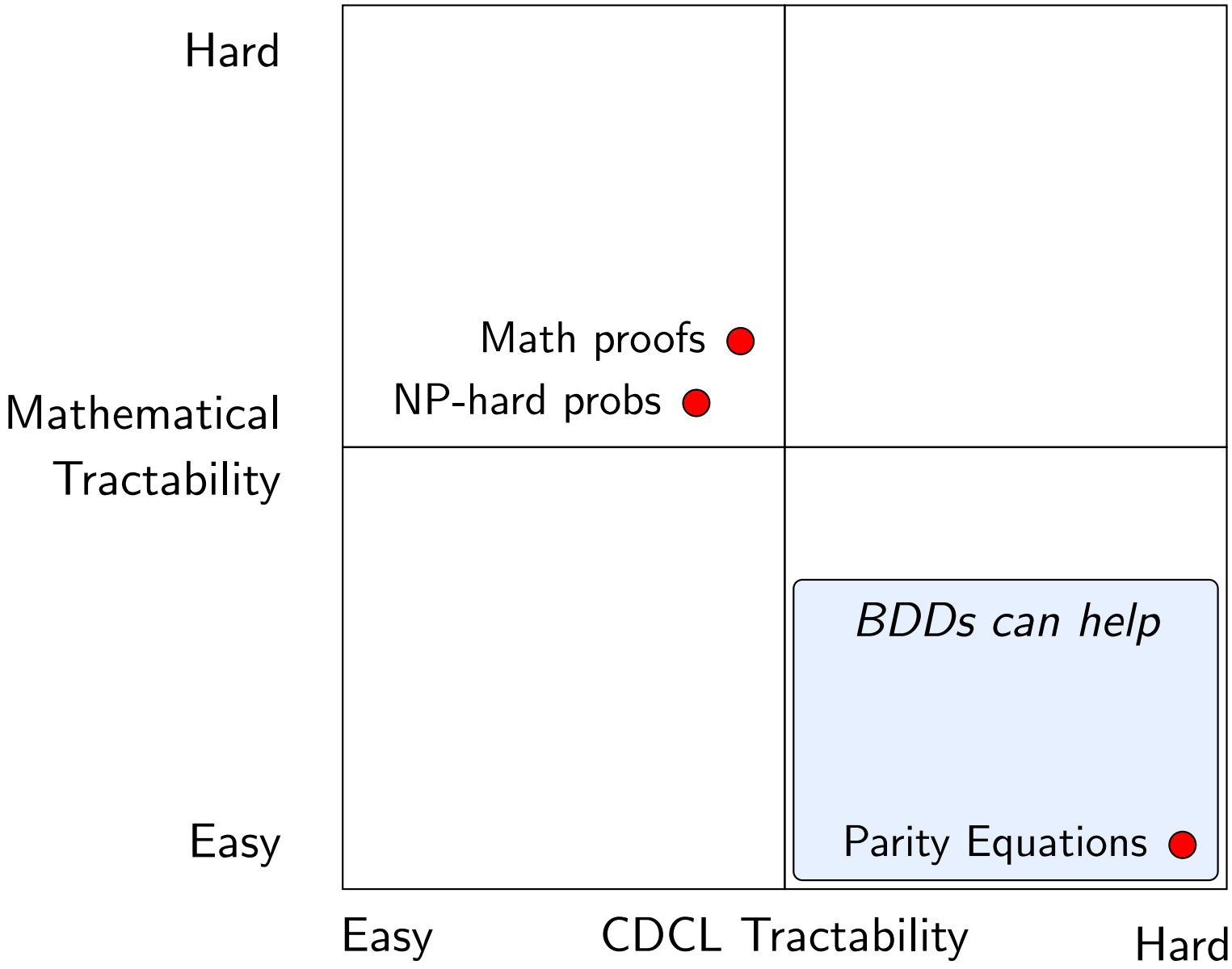
A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving



A Perspective on the State of SAT Solving