

Trustworthy Boolean Reasoning

2B: Proof Generation with BDDs

Randal E. Bryant

**Carnegie
Mellon
University**

June, 2022

Important Ideas for These Lectures

- ▶ SAT solvers are useful tools
 - ▶ Many practical problems reducible to SAT
 - ▶ Need to learn effective encoding techniques
- ▶ For many applications, formulas should be unsatisfiable
 - ▶ Program should generate a checkable proof
 - ▶ There is a well-developed proof infrastructure
- ▶ **Binary Decision Diagrams (BDDs) can play important role**
 - ▶ In supplementing current SAT algorithms
 - ▶ **In proof generation**

Extended Resolution and BDDs

- ▶ Tseitin, 1967

Can introduce extension variables

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

Extension variable z becomes shorthand for formula F

- ▶ Repeated use can yield exponentially smaller proof

Extended Resolution and BDDs

- ▶ Tseitin, 1967

Can introduce extension variables

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

Extension variable z becomes shorthand for formula F

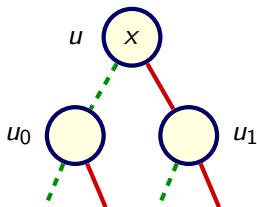
- ▶ Repeated use can yield exponentially smaller proof

Generate extension variable for every node in BDD

- ▶ Biere, Sinz, Jussila, 2006
- ▶ Each recursive step of Apply algorithm justified as proof steps
- ▶ Reducing formula to BDD \perp yields UNSAT proof

Generating Extended Resolution Proofs

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

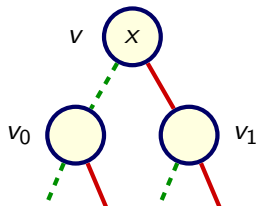
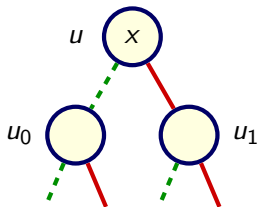


- ▶ Defining clauses encode constraint $u \leftrightarrow \text{ITE}(x, u_1, u_0)$

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

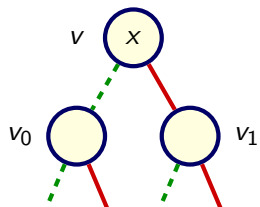
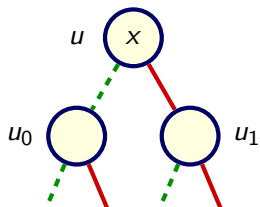
Apply Algorithm Recursion

Apply(u, v, \wedge)

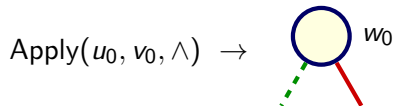
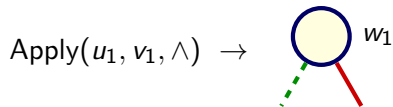


Apply Algorithm Recursion

Apply(u, v, \wedge)

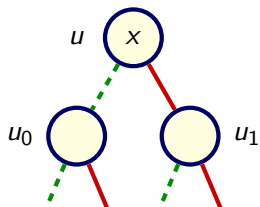


Recursion

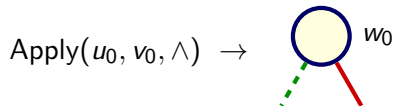
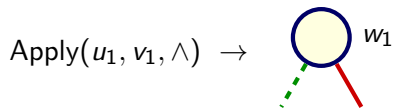


Apply Algorithm Recursion

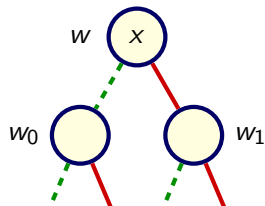
Apply(u, v, \wedge)



Recursion



Result



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

Resolution Steps

$$x \rightarrow (u \rightarrow u_1)$$

$$\bar{x} \rightarrow (u \rightarrow u_0)$$

$$x \rightarrow (v \rightarrow v_1)$$

$$\bar{x} \rightarrow (v \rightarrow v_0)$$

$$x \rightarrow (w_1 \rightarrow w) \quad u_1 \wedge v_1 \rightarrow w_1$$

$$\bar{x} \rightarrow (w_0 \rightarrow w) \quad u_0 \wedge v_0 \rightarrow w_0$$

$$x \rightarrow (u \wedge v \rightarrow w)$$

$$\bar{x} \rightarrow (u \wedge v \rightarrow w)$$

$$u \wedge v \rightarrow w$$

Can perform with 2 RUP steps

Quantification Operations

Operation $\text{EQuant}(f, X)$

- ▶ 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
 - ▶ $\text{EQuant}(u, X) \rightarrow w$
 - ▶ 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 $C_I \models \perp$

Trusted BDDs (TBDD)

Components

- ▶ BDD with root node t
- ▶ Proof step for unit clause (t)

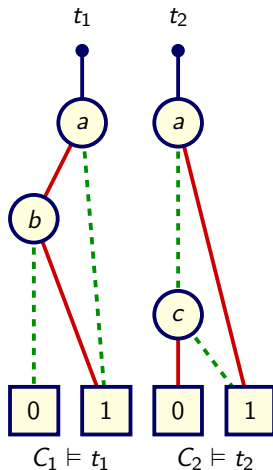
Interpretation

- ▶ $C_I \models t$
- ▶ Any variable assignment that satisfies input clauses must yield 1 for BDD with root t

TBDD Example

$$\begin{array}{l} C_1 \\ C_2 \end{array} \quad \begin{array}{l} \bar{a} \vee b \\ a \vee \bar{c} \end{array}$$

$$\begin{array}{l} t_1 \longleftarrow \text{FromClause}(C_1) \\ t_2 \longleftarrow \text{FromClause}(C_2) \end{array}$$



TBDD Example

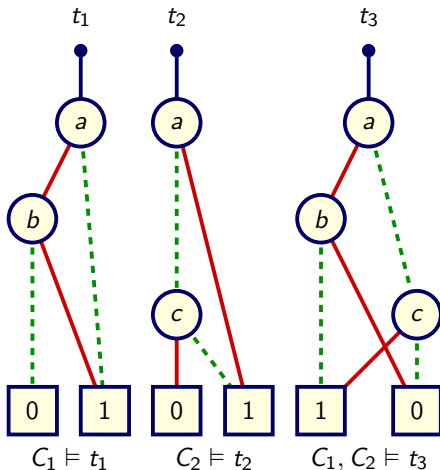
$$C_1 \quad \bar{a} \vee b$$

$$C_2 \quad a \vee \bar{c}$$

$t_1 \leftarrow \text{FromClause}(C_1)$

$t_2 \leftarrow \text{FromClause}(C_2)$

$t_3 \leftarrow \text{ApplyAnd}(t_1, t_2)$



Structure of Overall Proof

Input Variables

- ▶ BDD variable for each input variable

Structure of Overall Proof

Input Variables

- ▶ BDD variable for each input variable

Input Clauses

- ▶ For each input clause $C_i \in C_I$, generate BDD representation t_i
- ▶ Generate *validation* proof $C_i \models t_i$
 - ▶ Sequence of resolution steps based on linear structure of BDD
- ▶ Initial set of TBDDs

Structure of Overall Proof

Input Variables

- ▶ BDD variable for each input variable

Input Clauses

- ▶ For each input clause $C_i \in C_I$, generate BDD representation t_i
- ▶ Generate *validation* proof $C_i \models t_i$
 - ▶ Sequence of resolution steps based on linear structure of BDD
- ▶ Initial set of TBDDs

Combine Top-Level BDDs

- ▶ Choose TBDDs t_i, t_j . Use to generate TBDD t_k
- ▶ $t_k \leftarrow \text{ApplyAnd}(t_i, t_j)$
 - ▶ Combine proofs $C_I \models t_i, C_I \models t_j$ and $t_i \wedge t_j \rightarrow t_k$ to validate $C_I \models t_k$
- ▶ $t_k \leftarrow \text{EQuant}(t_i, X)$
 - ▶ Combine proofs $C_I \models t_i$ and $t_i \rightarrow t_k$ to validate $C_I \models t_k$

Structure of Overall Proof

Input Variables

- ▶ BDD variable for each input variable

Input Clauses

- ▶ For each input clause $C_i \in C_I$, generate BDD representation t_i
- ▶ Generate *validation* proof $C_i \models t_i$
 - ▶ Sequence of resolution steps based on linear structure of BDD
- ▶ Initial set of TBDDs

Combine Top-Level BDDs

- ▶ Choose TBDDs t_i, t_j . Use to generate TBDD t_k
- ▶ $t_k \leftarrow \text{ApplyAnd}(t_i, t_j)$
 - ▶ Combine proofs $C_I \models t_i, C_I \models t_j$ and $t_i \wedge t_j \rightarrow t_k$ to validate $C_I \models t_k$
- ▶ $t_k \leftarrow \text{EQuant}(t_i, X)$
 - ▶ Combine proofs $C_I \models t_i$ and $t_i \rightarrow t_k$ to validate $C_I \models t_k$

Completion

- ▶ When $t_k = \perp$ have proof $C_I \models \perp$

Comparing Proofs

Generated by CDCL Solver

- ▶ Resolution
- ▶ Encode conflict clauses
 - ▶ Increasingly strong constraints on set of satisfying solutions
- ▶ Reach empty clause when detect there is no solution

Generated with BDD-Based Solver

- ▶ Extended resolution
- ▶ Justify each recursive step of BDD algorithm
- ▶ Reach empty clause when reduce formula to BDD leaf \perp

Checking

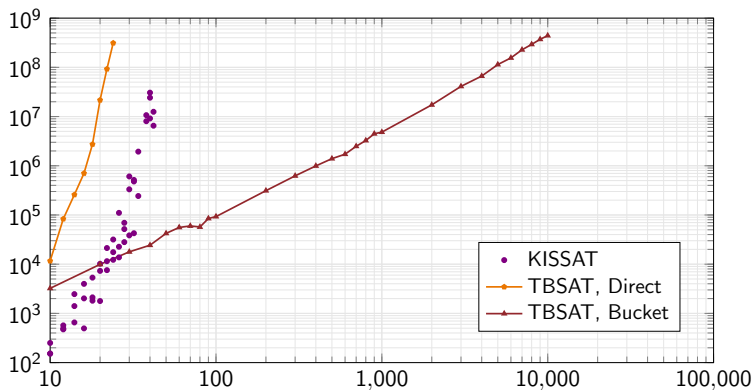
- ▶ Both checked with DRAT/LRAT checkers

TBSAT (Trusted BDD Satisfiability solver)

Implementation

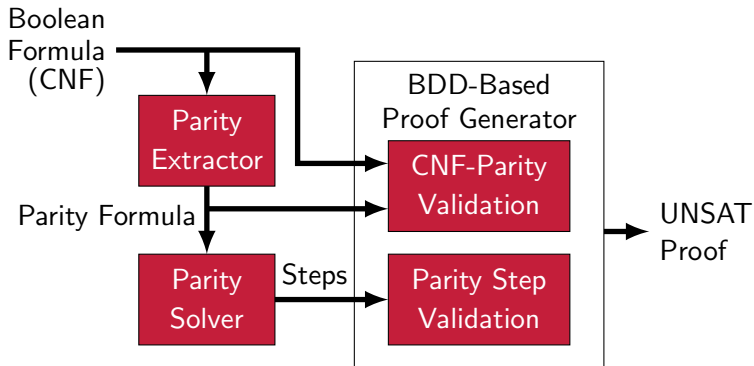
- ▶ TBUDDY: Modified version of BuDDy BDD package
 - ▶ Lind-Nielsen, ca. 1998
- ▶ Support for TBDDs and proof generation
- ▶ C/C++
- ▶ <https://github.com/rebryant/tbuddy-artifact>

Parity Benchmark Proof Complexity



- ▶ Total number of proof steps
 - ▶ Defining clauses + RUP clauses
- ▶ TBSAT with bucket elimination scales polynomially
 - ▶ Checker time \approx Solver time

Integrating Parity Reasoning into Proof-Generating SAT Solver



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

Gaussian Elimination Over GF2

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

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

Assume

- ▶ $a_{i,j}, x_j \in \{0, 1\}$
- ▶ $a + b \equiv a \oplus b$
- ▶ $a \cdot b \equiv a \wedge b$

Capability

- ▶ Can determine if there are any solutions for x_1, x_2, \dots, x_n

Gaussian Elimination Over GF2

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

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

Elimination Step

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

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

- ▶ Guarantees $a_{i,t} = 0$ for all $i \neq s$
3. Remove \mathbf{e}_s from E and repeat until single equation left

Gaussian Elimination Results

Possible Outcomes

1. If encounter degenerate equation
 - ▶ Of form $0 = 1$
 - ▶ Has no solution
2. Otherwise,
 - ▶ Can perform back substitution to find solution

CNF to Parity Constraint Validation

Clauses

- ▶ Suppose clauses $C_{i_1}, C_{i_2}, \dots, C_{i_k}$ encode parity constraint equation \mathbf{e}
- ▶ Have validated BDD representations $t_{i_1}, t_{i_2}, \dots, t_{i_k}$

Form conjunction

$$s = \bigwedge_{1 \leq j \leq k} t_{i_j}$$

- ▶ Also yields proof $C_I \models s$

Represent Constraint

- ▶ Form BDD representation t_j of \mathbf{e}

Validate

- ▶ Generate proof $s \rightarrow t_j$
- ▶ Use to validate term $C_I \models t_j$

Parity Step Validation

Assume

- ▶ Have BDDs t_i and t_j representing equations \mathbf{e}_i and \mathbf{e}_j
- ▶ Satisfying $C_I \models t_i$ and $C_I \models t_j$

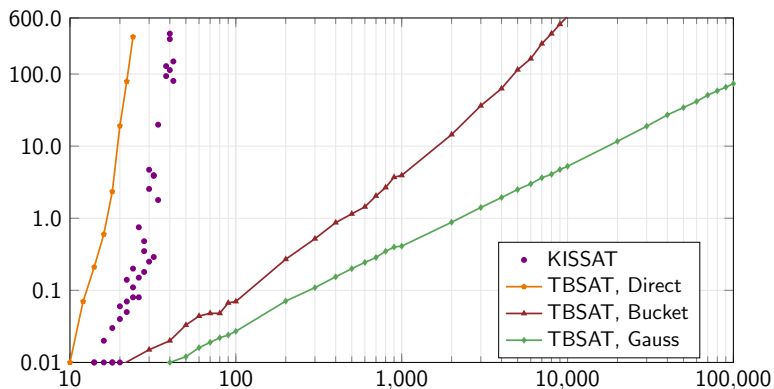
Compute

- ▶ $s \leftarrow \text{ApplyAnd}(t_i, t_j)$
 - ▶ Gives proof $t_i \wedge t_j \rightarrow s$
- ▶ Generate BDD representation t_k of equation $\mathbf{e}_k = \mathbf{e}_i + \mathbf{e}_j$

Validation

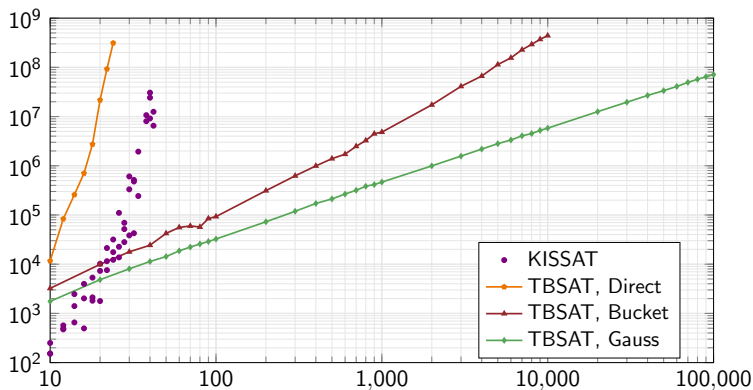
- ▶ Generate proof $s \rightarrow t_k$
- ▶ Combine with other proofs to validate $C_I \models t_k$

Parity Benchmark Runtime



- ▶ $n = 100,000$ in 74 seconds
- ▶ Upper limit: $n = 699,051$
 - ▶ BuDDy limited to $2^{21} - 1$ BDD variables

Parity Benchmark Proof Complexity



- ▶ Total number of proof steps
 - ▶ Defining clauses + RUP clauses
- ▶ Checker time \approx 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

Final Thoughts on Checkable Proofs

Important capability

- ▶ Vital to gain confidence in automated reasoning tools
- ▶ Benefits both tool developers and tool users

SAT community handled this especially well

- ▶ Started with well-established logical framework (resolution)
- ▶ Developed efficient algorithms that integrated well with solvers (RUP)
- ▶ Included more general capabilities (extended resolution)
- ▶ Formulated file formats, tool chain
- ▶ Fostered deployment through competitions

More challenging for other domains

- ▶ Beyond Boolean

Some References

BDDs

- ▶ R. E. Bryant, “[Graph-Based Algorithms for Boolean Function Manipulation](#),” *IEEE Transactions on Computers*, 1986
- ▶ R. E. Bryant, “[Binary Decision Diagrams](#),” *Handbook of Model Checking*, 2018

Proof Generation with BDDs

- ▶ R. E. Bryant and M. J. H. Heule, “[Generating Extended Resolution Proofs with a BDD-Based SAT Solver](#),” *TACAS*, 2021
- ▶ R. E. Bryant, A. Biere, and M. J. H. Heule, “[Clausal Proofs from Pseudo-Boolean Reasoning](#),” *TACAS*, 2022
- ▶ R. E. Bryant, “[TBUDDY: A Proof-Generating BDD Package](#),” *in submission*, 2022