Trustworthy Boolean Reasoning
2A: Introduction to 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

Reduced Ordered Binary Decision Diagrams (BDDs)

- Bryant, 1986
- Based on earlier work by Lee (1959) and Akers (1978)

Graph Representation of Boolean Functions

- Canonical Form
- Compact for many useful problems
- Simple algorithms to construct & manipulate

Used in SAT, Model Checking, . . .

- Bottom-up approach
  - Construct canonical representation of problem
  - Generate solutions
- Compare to search-based methods
  - E.g., CDCL
  - Top-down approaches
  - Keep branching on variables until find solution

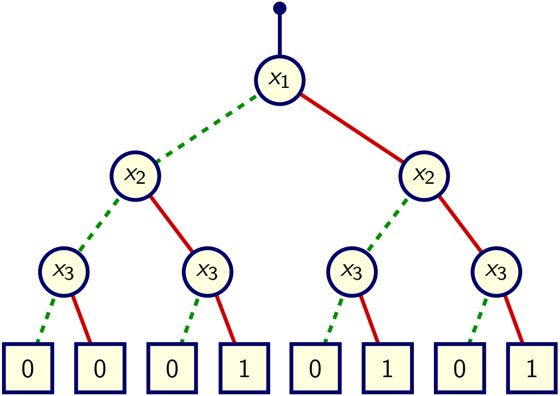
Boolean Function Representations

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Size $= O(2^n)$

Decision Tree

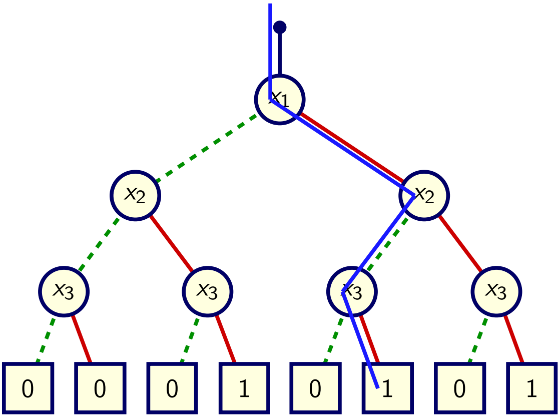


Boolean Function Representations

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Decision Tree



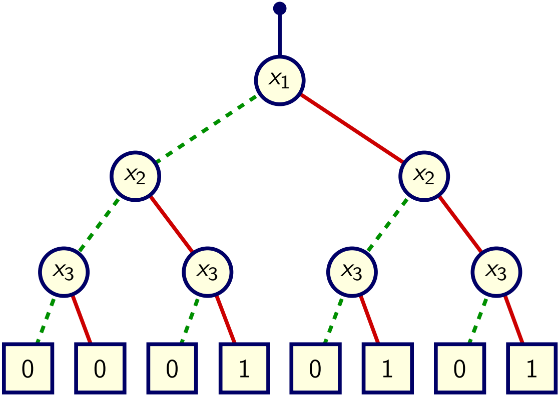
- Size $= O(2^n)$
- Assignment defines path from root to leaf

Reducing to Canonical Form

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Graph Representation



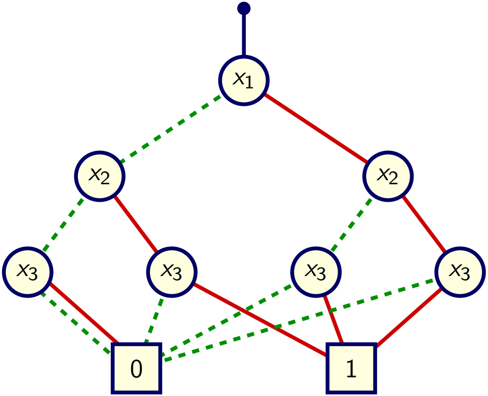
- Merge isomorphic nodes
- Eliminate redundant tests

Reducing to Canonical Form

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Graph Representation



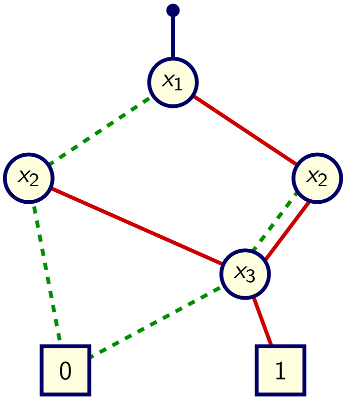
- Merge isomorphic nodes
- Eliminate redundant tests

Reducing to Canonical Form

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Graph Representation



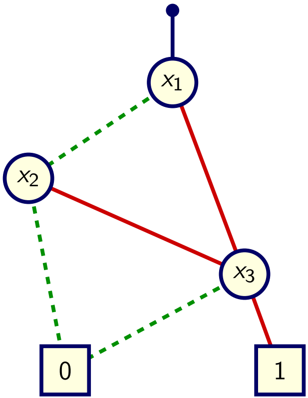
- Merge isomorphic nodes
- Eliminate redundant tests

Reducing to Canonical Form

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Graph Representation



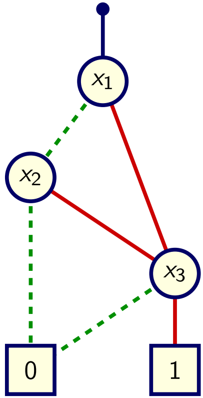
- Merge isomorphic nodes
- Eliminate redundant tests

Canonical Form

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Reduced Ordered Binary Decision Diagram



- Canonical representation of Boolean function
- No further simplifications possible

BDD Representation of Unsatisfiable Formula

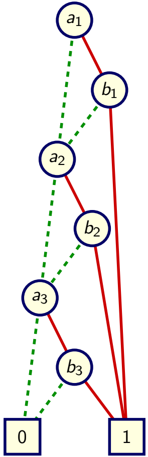
$$\boxed{0}$$

- Refer to this as $\perp$
- Unique
- Converting from CNF to BDD may require exponential number of steps

Effect of Variable Ordering

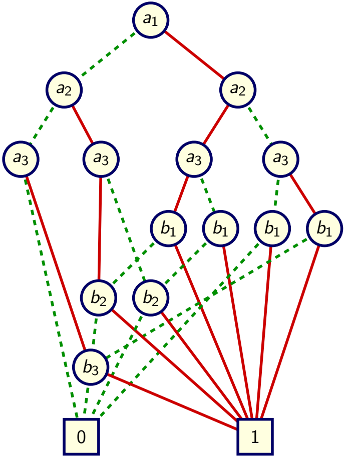
$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$$

Good Ordering



- Linear growth

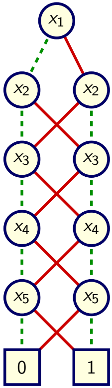
Bad Ordering



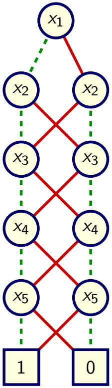
- Exponential growth

BDD Representation of Parity Constraints

Odd Parity



Even Parity



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

Symbolic Manipulation with BDDs

Strategy

- Represent data as set of BDDs
  - All with same variable ordering
- Express method as sequence of symbolic operations
  - Generate new BDDs. Test properties of BDDs
- Implement each operation via BDD manipulation
  - Never enumerate individual cases
  - Efficient, as long as BDDs stay small

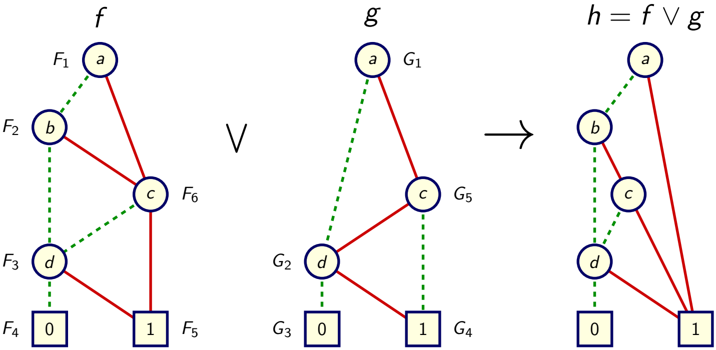
Key Algorithmic Properties

- Arguments at each step are BDDs with same variable ordering
- Result is BDD with same ordering
- Each step has polymomial complexity

Apply Algorithm

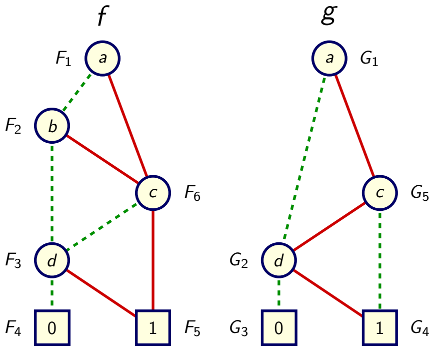
$$h \leftarrow f \odot g$$

- f, g, h functions represented as BDDs
- $\odot$ binary Boolean operator
  - E.g., $\wedge$, $\vee$, $\oplus$



Apply Algorithm Recursion

- Recurse through argument graphs
- Stop when hit terminal case
- Save results in cache to reuse when hit same arguments

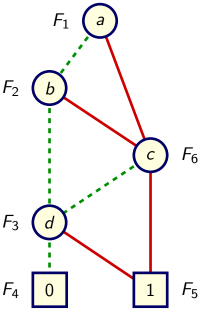


Recursive Calls

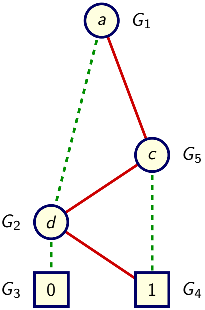
Apply Algorithm Recursion

- Recurse through argument graphs
- Stop when hit terminal case
- Save results in cache to reuse when hit same arguments

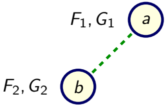
f



g

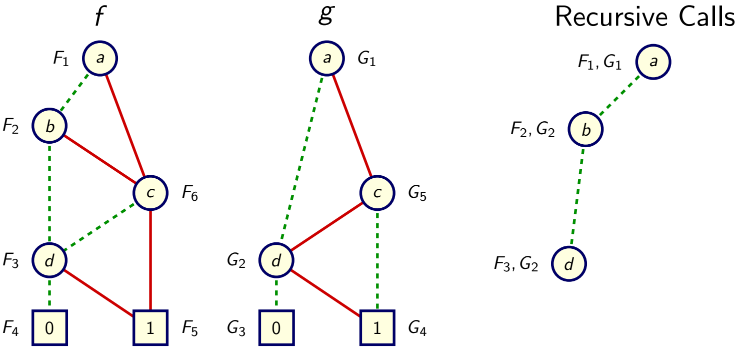


Recursive Calls



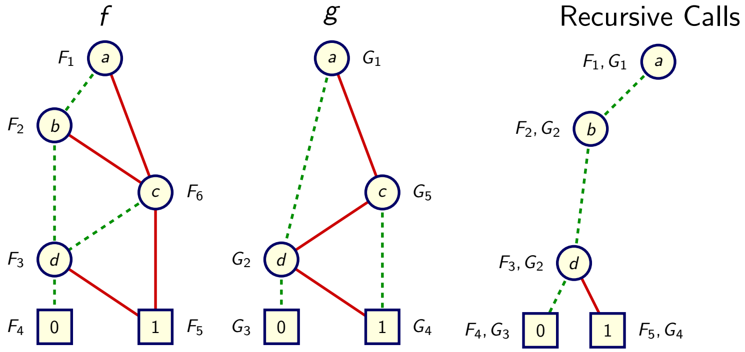
Apply Algorithm Recursion

- Recurse through argument graphs
- Stop when hit terminal case
- Save results in cache to reuse when hit same arguments



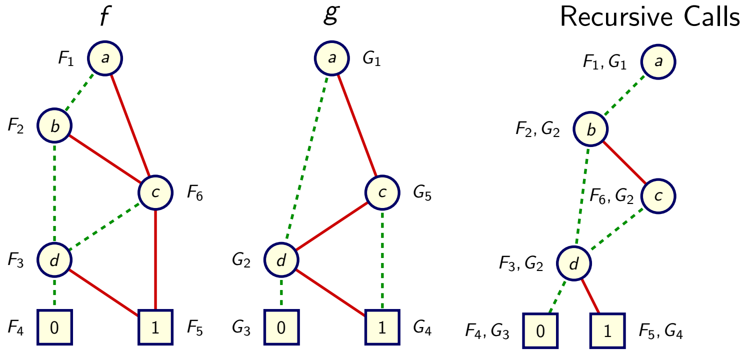
Apply Algorithm Recursion

- Recurse through argument graphs
- Stop when hit terminal case
- Save results in cache to reuse when hit same arguments



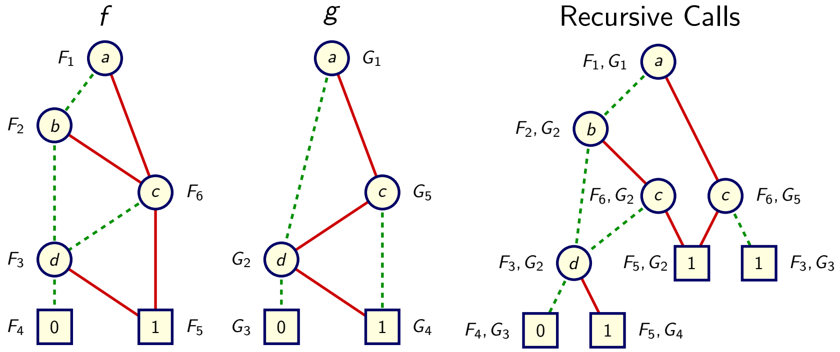
Apply Algorithm Recursion

- Recurse through argument graphs
- Stop when hit terminal case
- Save results in cache to reuse when hit same arguments

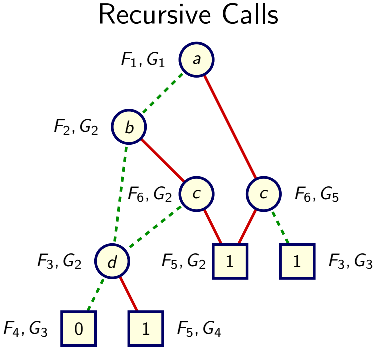


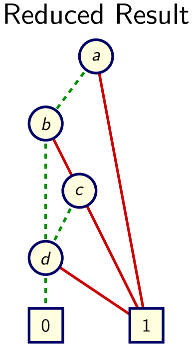
Apply Algorithm Recursion

- Recurse through argument graphs
- Stop when hit terminal case
- Save results in cache to reuse when hit same arguments



Apply Algorithm Result





BDD-Based SAT Solving: Direct Evaluation

Algorithm

1. Compute BDD t_i for each input clause C_i
2. Form conjunction

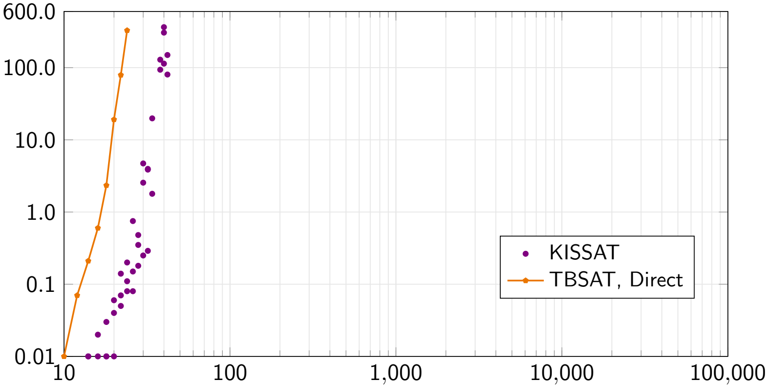
$$s = \bigwedge_{1 \leq i \leq m} t_i$$

   - E.g., with linear or tree evaluation
3. Return UNSAT ($s = \bot$) or SAT ($s \neq \bot$)

Practicality

- Only for small problems
- Resulting BDD s represents *all* solutions

Parity Benchmark Runtime



- TBSAT: BDD-Based SAT Solver
- In direct mode, even worse than KISSAT
- Limited to $n \leq 24$ within 600 seconds

BDD-Based SAT Solving: Bucket Evaluation

- Maintain list ("bucket") B_j for each variable x_j
- Each BDD stored in bucket according to root node variable

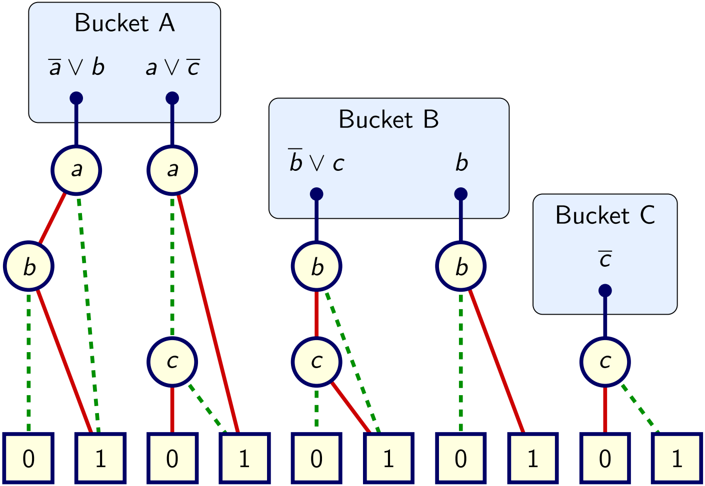
Algorithm

Initialization:
Form BDD t_i for each input clause C_i
Place each t_i in bucket according to $Var(t_i)$

For each bucket B_j:
Form conjunction s_j of all BDDs in bucket B_j
If $s_j = \perp$ then return UNSAT
Compute $r_j = \exists x_j s_j$
Place r_j in bucket according to $Var(r_j)$

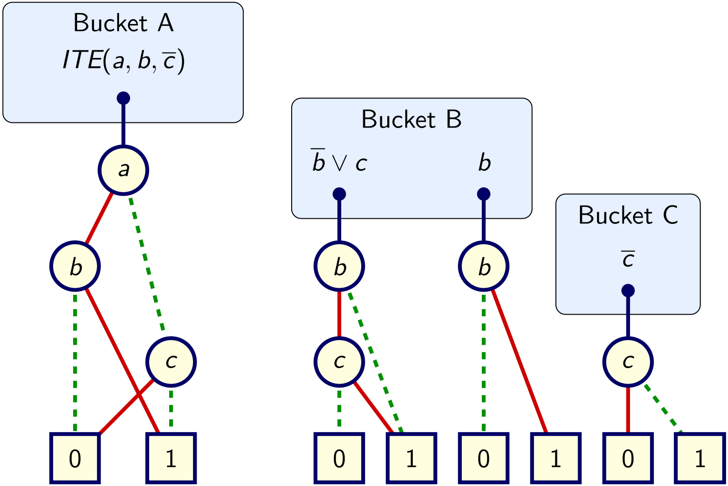
return SAT

Bucket Evaluation Example



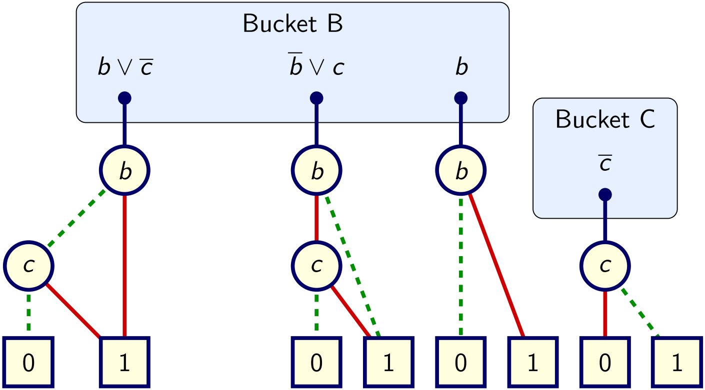
- Initially: BDD for each input clause
- In bucket according to root variable

Bucket Evaluation Example



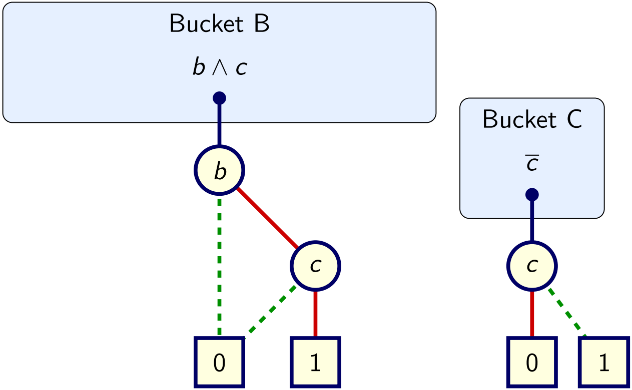
- Conjunct BDDs in topmost bucket A

Bucket Evaluation Example



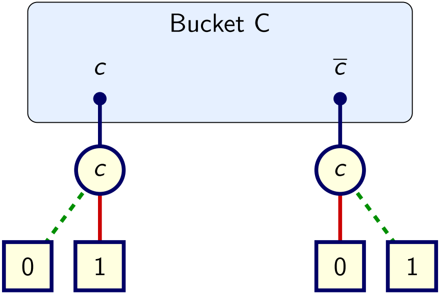
- Existentially quantify variable a
- Place result in appropriate bucket

Bucket Evaluation Example



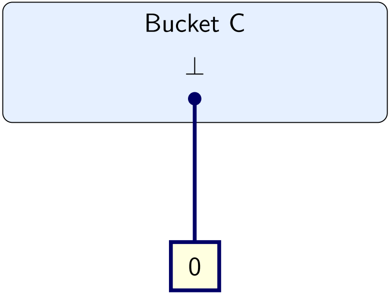
- Conjunct BDDs in bucket B

Bucket Evaluation Example



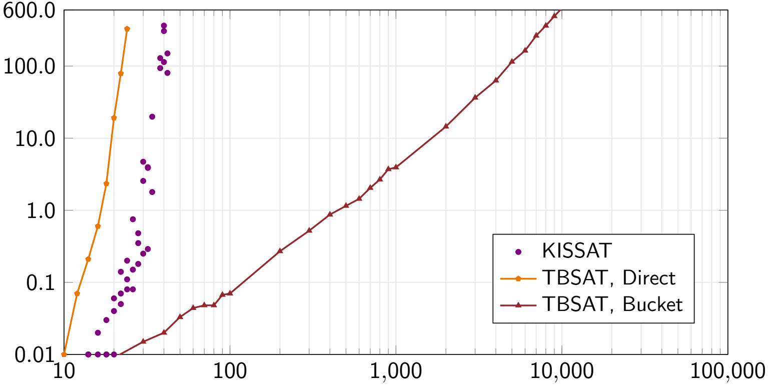
- Existentially quantify variable b
- Place result in appropriate bucket

Bucket Evaluation Example



- Conjunct BDDs in bucket C
- Final result will be $\perp$ or $\top$

Parity Benchmark Runtime



- $n = 10{,}000$ in 633 seconds
- Large benefit from quantification
  - abstracts away intermediate variables

What Parity Benchmark Demonstrates

- Binary Decision Diagrams (BDDs) can play important role in SAT
  - In supplementing current SAT algorithms
  - But, *what about proof generation?*