Binary Decision Diagrams
Part 2

15-414 Bug Catching: Automated Program Verification and Testing

Sagar Chaki
September 14, 2011
BDDs Recap

Typically mean Reduced Ordered Binary Decision Diagrams (ROBDDs)
- Can be viewed as reduced forms of Ordered Binary Decision Trees
  - Obtained by eliminating duplicate nodes and redundant nodes
  - Often substantially smaller than the OBDDT

Canonical representation of Boolean formulas
- Unlike other normal forms like CNF and DNF

Size of BDD depends critically on variable ordering

In practice, BDDs are built up from their components
- Via (efficient) Boolean operations
- Dynamic variable ordering used to manage BDD size
Running Example: Comparator

(A) \( f = 1 \iff a_1 = b_1 \land a_2 = b_2 \)

(B) \( f = 1 \iff a_1 = b_1 \lor a_2 = b_2 \)
Conjunctive Normal Form

\[ (b_1 \lor a_1) \land (\neg a_1 \lor \neg b_1) \land (\neg a_2 \lor b_2) \land (\neg b_2 \lor a_2) \]

\[ (\neg b_1 \lor a_1) \land (\neg a_1 \lor b_1) \land (\neg a_2 \lor b_2) \land (\neg b_2 \lor a_2) \]

\[ (\neg a_1 \lor b_1) \land (\neg b_1 \lor a_1) \land (\neg a_2 \lor b_2) \land (\neg b_2 \lor a_2) \]

\[ (\neg a_1 \lor b_1) \land (\neg b_1 \lor a_1) \land (\neg a_2 \lor \neg b_2) \land (b_2 \lor a_2) \]
Truth Table

<table>
<thead>
<tr>
<th>Row#</th>
<th>( a_1 )</th>
<th>( b_1 )</th>
<th>( a_2 )</th>
<th>( b_2 )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Representing a Truth Table using a Graph

Binary Decision Diagrams – Part 2
Sagar Chaki, Sep 14, 2011
© 2011 Carnegie Mellon University
Representing a Truth Table using a Graph
Which pairs are isomorphic?
(1) \{A,B\} and \{C,D\}
(2) \{A,C\} and \{B,D\}
(3) \{A,D\} and \{B,C\}
OBDD to ROBDD

Is this a ROBDD?

(1) YES ✔
(2) NO
What function does X represent?

1. \( a_2 = b_2 \)
2. \( a_2 = (\neg b_2) \)
3. \( a_2 \Rightarrow b_2 \)
4. \( a_2 \oplus b_2 \)
5. \( \neg (a_2 \oplus b_2) \)
6. \( (a_2 \land b_2) \lor (\neg a_2 \land \neg b_2) \)
ROBDD (a.k.a. BDD) Summary

If BDD(f₁) and BDD(f₂) are isomorphic then:

1. \( f₁ = f₂ \)
2. \( f₁ \) and \( f₂ \) have the same variables
3. BDD(f₁) and BDD(f₂) have the same variable ordering

If BDD(f) is the leaf node “1” then f is:

1. Satisfiable
2. Unsatisfiable
3. Valid
ROBDD and variable ordering

Is this a ROBDD?
(1) YES
(2) NO

12

Binary Decision Diagrams – Part 2
Sagar Chaki, Sep 14, 2011
© 2011 Carnegie Mellon University
ROBDD and variable ordering

Is this a ROBDD?

(1) YES ✓
(2) NO
ROBDD and variable ordering

There exists a function whose BDD grows polynomially in the number of variables for some ordering and exponentially for others?

• TRUE

There exists a function whose BDD grows exponentially for all variable orderings?

• TRUE

There exists a function whose BDD grows linearly for all variable orderings?

• TRUE
BDD Operations

True: BDD(TRUE)

False: BDD(FALSE)

Var: v \mapsto BDD(v)

Not: BDD(f) \mapsto BDD(\neg f)

And: BDD(f_1) \times BDD(f_2) \mapsto BDD(f_1 \land f_2)

Or: BDD(f_1) \times BDD(f_2) \mapsto BDD(f_1 \lor f_2)

Exist: BDD(f) \times v \mapsto BDD(\exists v \cdot f)
Basic BDD Operations

Var(v)

True
1

False
0

Binary Decision Diagrams – Part 2
Sagar Chaki, Sep 14, 2011
© 2011 Carnegie Mellon University
BDD Operations: Not

\[ 1 \xrightarrow{} 0 \quad 0 \xrightarrow{} 1 \]

\[ O(1) \quad O(1) \]
BDD Operations: Not

Swap “0” and “1”
BDD Operations: And

Suppose this is the BDD for $f$

What formula does this represent?

What formula does this represent?
BDD Operations: And

\[ f_{v=0} \text{ and } f_{v=1} \text{ are known as the co-factors of } f \text{ w.r.t. } v \]

\[ f = (X \land f_{v=0}) \lor (Y \land f_{v=1}) \]
BDD Operations: And

Suppose this is the BDD for $f$

$f_{v=0}$ and $f_{v=1}$ are known as the co-factors of $f$ w.r.t. $v$

$$f = (\neg v \land f_{v=0}) \lor (v \land f_{v=1})$$
BDD Operations: And (Simple Cases)

\[
\text{And } (f, 0) = 0
\]

\[
\text{And } (f, 1) = f
\]

\[
\text{And } (1, f) = f
\]

\[
\text{And } (0, f) = 0
\]
BDD Operations: And (Complex Case)

\((\neg v_1 \land f_1) \lor (v_1 \land g_1) \land (\neg v_2 \land f_2) \lor (v_2 \land g_2)\)
BDD Operations: And (Complex Case 1)

\[ v_1 = v_2 \]

\[ (\neg v_1 \land f_1) \lor (v_1 \land g_1) \]

\[ (\neg v_1 \land f_2) \lor (v_1 \land g_2) \]
BDD Operations: And (Complex Case 1)

\[ \neg v_1 = v_2 \]

\[ (\neg v_1 \land X) \lor (v_1 \land Y) \]

\[ (\neg v_1 \land f_1) \lor (v_1 \land g_1) \land (\neg v_1 \land f_2) \lor (v_1 \land g_2) \]
BDD Operations: And (Complex Case 1)

\[ v_1 = v_2 \]

Compute recursively

\[ (\neg v_1 \land (f_1 \land f_2)) \lor (v_1 \land (g_1 \land g_2)) \]

\[ (\neg v_1 \land f_1) \lor (v_1 \land g_1) \land (\neg v_1 \land f_2) \lor (v_1 \land g_2) \]
BDD Operations: And (Complex Case 1)

What if \( f_1 \land f_2 = g_1 \land g_2 \)?

Return \( f_1 \land f_2 \)

\[
(\neg v_1 \land (f_1 \land f_2)) \lor (v_1 \land (g_1 \land g_2))
\]

\[
(\neg v_1 \land f_1) \lor (v_1 \land g_1) \land (\neg v_1 \land f_2) \lor (v_1 \land g_2)
\]
BDD Operations: And (Complex Case 2)

\[ (\neg v_1 \land f_1) \lor (v_1 \land g_1) \land (\neg v_2 \land f_2) \lor (v_2 \land g_2) \]

\[ v_1 \text{ appears before } v_2 \text{ in the variable ordering} \]
BDD Operations: And (Complex Case 2)

What if $f_1 \land d_2 = g_1 \land d_2$?

Return $f_1 \land d_2$

Complex Case 2:

$(\neg v_1 \land (f_1 \land d_2)) \lor (v_1 \land (g_1 \land d_2))$

$(\neg v_1 \land f_1) \lor (v_1 \land g_1) \land d_2$

$O(n_1 \times n_2)$
BDD Operations: Or

\[ \text{Or}(d_1, d_2) = \neg (\neg d_1 \land \neg d_2) \]

\[ O(n_1 \times n_2) \]
BDD Operations: Exist

\[
\text{Exist(“0”,v) = ?}
\]
BDD Operations: Exist

Exist(“0”, v) = “0”

Exist(“1”, v) = ?
BDD Operations: Exist

Exist("0", v) = "0"
Exist("1", v) = "1"

Exist((¬ v ∧ f) ∨ (v ∧ g), v) = ?
BDD Operations: Exist

Exist(“0”, v) = “0”

Exist(“1”, v) = “1”

Exist((¬ v ∧ f) ∨ (v ∧ g), v) = Or(f, g)

Exist((¬ v’ ∧ f) ∨ (v’ ∧ g), v) = ?
BDD Operations: Exist

\[ \text{Exist}("0", v) = "0" \]
\[ \text{Exist}("1", v) = "1" \]
\[ \text{Exist}(\neg v \land f) \lor (v \land g), v) = \text{Or}(f, g) \]
\[ \text{Exist}(\neg v' \land f) \lor (v' \land g), v) = (\neg v' \land \text{Exist}(f, v)) \lor (v' \land \text{Exist}(g, v)) \]

But f is SAT iff \( \exists V. f \text{ is not } "0" \). So why doesn’t this imply \( P = NP \)?

Because the BDD size changes!
BDD Applications

SAT is great if you are interested to know if a solution exists

BDDs are great if you are interested in the set of all solutions
  • How many solutions are there?
  • How do you do this on a BDD?

Or if your problem involves computing a fixed point
  • Set of nodes reachable from a given node in a graph
BDD Application: Counting Sudoku Solutions

How many ways can you solve this puzzle?
BDD Application: Counting Sudoku Solutions

How many ways can you solve this puzzle? At least 2.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
BDD Application: Counting Sudoku Solutions

How many ways can you solve this puzzle? At least 2.
BDD Application: Counting Sudoku Solutions

\[ a_{11}, b_{11} \]

\[ a_{44}, b_{44} \]
BDD Application: Counting Sudoku Solutions

\[ \neg a_{13} \land b_{13} \]

\[ a_{31} \land b_{31} \]

\[ a_{33} \land \neg b_{33} \]
BDD Application: Counting Sudoku Solutions

Repeat for each row, column and sub-square

Construct BDD

Count number of solutions
Which nodes are reachable from "7"?

\{2, 3, 5, 6, 7\}

But what if the graph has trillions of nodes?
Use three Boolean variables (a,b,c) to encode each node?
Use three Boolean variables (a, b, c) to encode each node?
Graph Reachability

Use three Boolean variables (a,b,c) to encode each node?
Key Idea 1: Every Boolean formula represents a set of nodes!

The nodes whose encodings satisfy the formula.
Key Idea 1: Every Boolean formula represents a set of nodes!
Graph Reachability

Key Idea 1: Every Boolean formula represents a set of nodes!
Key Idea 1: Every Boolean formula represents a set of nodes!
Key Idea 1: Every Boolean formula represents a set of nodes!
Graph Reachability

Key Idea 1: Every Boolean formula represents a set of nodes!

\[ a \oplus b = \{2,3,4,5\} \]
Key Idea 2: Edges can also be represented by Boolean formulas

- An edge is just a pair of nodes
- Introduce three new variables: $a'$, $b'$, $c'$
- Formula $Φ$ represents all pairs of nodes $(n,n')$ that satisfy $Φ$ when $n$ is encoded using $(a,b,c)$ and $n'$ is encoded using $(a',b',c')$
Graph Reachability

Key Idea 2: Edges can also be represented by Boolean formulas

¬a ∧ ¬b ∧ ¬c ∧ ¬a’ ∧ ¬b’ ∧ c’
Graph Reachability

Key Idea 2: Edges can also be represented by Boolean formulas

\[ a \land \neg b \land c \land \neg a' \land b' \land \neg c' \]
Graph Reachability

Key Idea 2: Edges can also be represented by Boolean formulas.
Graph Reachability

Key Idea 3: Given the BDD for a set of nodes $S$, and the BDD for the set of all edges $R$, the BDD for all the nodes that are adjacent to $S$ can be computed using the BDD operations.

\[
\text{Image}(S,R) = (\exists a,b,c . (S \land R)) \ [ a \ \backslash \ a', b \ \backslash \ b', c \ \backslash \ c']
\]
Graph Reachability Algorithm

S = BDD for initial set of nodes;
R = BDD for all the edges of the graph;

while (true) {
    I = Image(S,R); //compute adjacent nodes to S
    if (And(Not(S),I) == False) //no new nodes found
        break;
    S = Or(S,I); //add newly discovered nodes to result
}

return S;

Symbolic Model Checking. Has been done for graphs with $10^{20}$ nodes.