BDDs in a nutshell

Typically mean Reduced Ordered Binary Decision Diagrams (ROBDDs)

Canonical representation of Boolean formulas

Often substantially more compact than a traditional normal form

Can be manipulated very efficiently
  • Conjunction, Disjunction, Negation, Existential Quantification

Running Example: Comparator

Comparator

\[ f = 1 \ , \ a_1 = b_1 \ \&\& \ a_2 = b_2 \]
Conjunctive Normal Form

\[
\begin{align*}
\text{Not Canonical}
\end{align*}
\]
## Truth Table (1)

<table>
<thead>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Still Not Canonical
Truth Table (2)

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$b_1$</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>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Canonical if you fix variable order.

But always exponential in # of variables. Let’s try to fix this.
Shannon’s / Boole’s Expansion

Every Boolean formula $f(a_0, a_1, ..., a_n)$ can be written as

$$(a_0 \land f(\text{true}, a_1, ..., a_n)) \lor (a_0 \land f(\text{false}, a_1, ..., a_n))$$

or, simply,

$$\text{ITE} (a_0, f(\text{true}, a_1, ..., a_n), f(\text{false}, a_1, ..., a_n))$$

where ITE stands for If-Then-Else

The formula $f(\text{true}, a_1, ..., a_n)$ is called the *cofactor* of $f$ w.r.t. $a_0$

The formula $f(\text{false}, a_1, ..., a_n)$ is called the *cofactor* of $f$ w.r.t. $a_0^\neg$
Representing a Truth Table using a Graph

Binary Decision Tree (in this case ordered)
Binary Decision Tree: Formal Definition

Balanced binary tree. Length of each path = # of variables

Leaf nodes labeled with either 0 or 1

Internal node v labeled with a Boolean variable var(v)
  • Every node on a path labeled with a different variable

Internal node v has two children: low(v) and high(v)

Each path corresponds to a (partial) truth assignment to variables
  • Assign 0 to var(v) if low(v) is in the path, and 1 if high(v) is in the path

Value of a leaf is determined by:
  • Constructing the truth assignment for the path leading to it from the root
  • Looking up the truth table with this truth assignment
Binary Decision Tree

\[ \text{low}(v) \]

\[ \text{high}(v) \]

\[ \text{var}(v) = a_1 \]
The truth assignment corresponding to the path to this leaf is:

\[a_1 = ? \quad b_1 = ? \quad a_2 = ? \quad b_2 = ?\]
The truth assignment corresponding to the path to this leaf is:

\[ a_1 = 0 \quad b_1 = 0 \quad a_2 = 1 \quad b_2 = 0 \]
Binary Decision Tree

The truth assignment corresponding to the path to this leaf is:

\[ a_1 = 0 \quad b_1 = 0 \quad a_2 = 1 \quad b_2 = 0 \]
The truth assignment corresponding to the path to this leaf is:

\[ a_1 = 0 \quad b_1 = 0 \quad a_2 = 1 \quad b_2 = 0 \]
Binary Decision Tree (BDT)

Canonical if you fix variable order (i.e., use ordered BDT)

But still exponential in # of variables. Let’s try to fix this.
Reduced Ordered BDD

Conceptually, a ROBDD is obtained from an ordered BDT (OBDT) by eliminating redundant sub-diagrams and nodes.

Start with OBDT and repeatedly apply the following two operations as long as possible:

1. Eliminate duplicate sub-diagrams. Keep a single copy. Redirect edges into the eliminated duplicates into this single copy.
2. Eliminate redundant nodes. Whenever low(v) = high(v), remove v and redirect edges into v to low(v).
   - Why does this terminate?

ROBDD is often exponentially smaller than the corresponding OBDT.
OBDD to ROBDD
OBDD to ROBDD
OBDT to ROBDD
OBDT to ROBDD
OBDT to ROBDD

Binary Decision Diagrams
Arie Gurfinkel, March 2014
© 2014 Carnegie Mellon University
OBDT to ROBDD

Diagram of OBDT to ROBDD transformation with nodes labeled as follows:
- $a_1$
- $b_1$
- $a_2$
- $b_2$
- 0
- 1

The diagram illustrates the mapping of OBDT nodes to ROBDD nodes, showing the relationships and connections between them.
OBDT to ROBDD

Redundant node
OBDD to ROBDD
OBDT to ROBDD
OBDT to ROBDD
OBDT to ROBDD
If $a_1 = 0$ and $b_1 = 1$ then $f = 0$ irrespective of the values of $a_2$ and $b_2$. 
OBDT to ROBDD
OBDT to ROBDD
OBDT to ROBDD
OBDT to ROBDD
OBDD to ROBDD
OBDT to ROBDD
Let’s move things around a little bit so that the BDD looks nicer.
Bryant gave a linear-time algorithm (called Reduce) to convert OBDT to ROBDD.

In practice, BDD packages don’t use Reduce directly. They apply the two reductions on-the-fly as new BDDs are constructed from existing ones. Why?
ROBDD (a.k.a. BDD) Summary

BDDs are canonical representations of Boolean formulas

• $f_1 = f_2$ , ?
ROBDD (a.k.a. BDD) Summary

BDDs are canonical representations of Boolean formulas

• $f_1 = f_2$, $\text{BDD}(f_1)$ and $\text{BDD}(f_2)$ are isomorphic
• $f$ is unsatisfiable, ?
ROBDD (a.k.a. BDD) Summary

BDDs are canonical representations of Boolean formulas

- $f_1 = f_2$, BDD($f_1$) and BDD($f_2$) are isomorphic
- $f$ is unsatisfiable, BDD($f$) is the leaf node “0”
- $f$ is valid, ?
ROBDD (a.k.a. BDD) Summary

BDDs are canonical representations of Boolean formulas
- $f_1 = f_2$, BDD($f_1$) and BDD($f_2$) are isomorphic
- $f$ is unsatisfiable, BDD($f$) is the leaf node “0”
- $f$ is valid, BDD($f$) is the leaf node “1”
- BDD packages do these operations in constant time

Logical operations can be performed efficiently on BDDs
- Polynomial in argument size

BDD size depends critically on the variable ordering
- Some formulas have exponentially large sizes for all ordering
- Others are polynomial for some ordering and exponential for others
ROBDD and variable ordering
ROBDD and variable ordering
ROBDD and variable ordering
ROBDD and variable ordering
ROBDD and variable ordering
ROBDD and variable ordering
ROBDD and variable ordering

Let’s move things around a little bit so that the BDD looks nicer.
ROBDD and variable ordering

a_1 < b_1 < a_2 < b_2

a_1 < a_2 < b_1 < b_2
ROBDD and variable ordering

\[ a_1 < b_1 < \ldots < a_n < b_n \]

\[ a_1 < \ldots < a_n < b_1 < \ldots < b_n \]
ROBDD and variable ordering

\[ a_1 < b_1 < \ldots < a_n < b_n \]

\[ a_1 < \ldots < a_n < b_1 < \ldots < b_n \]
BDD Operations

True : BDD(TRUE)

False: BDD(FALSE)

Var : v ∈ BDD(v)

Not : BDD(f) ∈ BDD(:f)

And : BDD(f₁) £ BDD(f₂) ∈ BDD(f₁ ∨ f₂)

Or : BDD(f₁) £ BDD(f₂) ∈ BDD(f₁ ⊕ f₂)

Exists : BDD(f) £ v ∈ BDD(∃ v. f)
Basic BDD Operations

True

False

Var(v)

1

0

v

0

1
BDD Operations: Not

\[
\begin{align*}
1 &\rightarrow 0 \\
0 &\rightarrow 1
\end{align*}
\]

\[
\begin{align*}
0 &\rightarrow 1 \\
1 &\rightarrow 0
\end{align*}
\]

\[
\begin{align*}
O(1) &\quad O(1) \quad O(1)
\end{align*}
\]
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

Suppose this is the BDD for $f$

$f_{v=0}$

$f_{v=1}$

$f = (X \land f_{v=0}) \lor (Y \land f_{v=1})$

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

Suppose this is the BDD for $f$

- $f_{v=0}$
- $f_{v=1}$

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

$$f = (v \not\in E f_{v=0}) \lor (v \in E f_{v=1})$$
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)

\[
\begin{align*}
(f_0 \land f_1) \land (g_0 \land g_1) &= (v_1 \land f_0) \land (v_1 \land f_1) \land (v_2 \land g_0) \land (v_2 \land g_1)
\end{align*}
\]
BDD Operations: And (Complex Case 1)

\[ v_1 = v_2 \]

\[ (v_1 \& f_0) \land (v_1 \& f_1) \]

\[ (v_1 \& g_0) \land (v_1 \& g_1) \]
BDD Operations: And (Complex Case 1)

\[ v_1 = v_2 \]

\[ \left( v_1 \forall X \right) \land \left( v_1 \forall Y \right) \]

\[ \left( v_1 \forall f_0 \right) \land \left( v_1 \forall f_1 \right) \]

\[ \left( v_1 \forall g_0 \right) \land \left( v_1 \forall g_1 \right) \]
BDD Operations: And (Complex Case 1)

\[ v_1 = v_2 \]

Compute recursively

\[ (: v_1 \land (f_0 \land g_0)) \lor (v_1 \land (f_1 \land g_1)) \]

\[ (: v_1 \land f_0) \lor (v_1 \land f_1) \]

\[ (: v_1 \land g_0) \lor (v_1 \land g_1) \]
BDD Operations: And (Complex Case 1)

\[ v_1 = v_2 \]

What if \( f_0 \land g_0 = f_1 \land g_1 \)?

Return \( f_0 \land g_0 \)

\[ (\land v_1 (f_0 \land g_0)) \lor (\land v_1 (f_1 \land g_1)) \]

\[ \land (v_1 \land f_0) \lor (v_1 \land f_1) \]

\[ \land (v_1 \land g_0) \lor (v_1 \land g_1) \]
BDD Operations: And (Complex Case 2)

\[ (\text{v}_1 \triangleleft \text{f}_0) \land (\text{v}_1 \triangleleft \text{f}_1) \land (\text{v}_2 \triangleleft \text{g}_0) \land (\text{v}_2 \triangleleft \text{g}_1) \]

\[ (\text{v}_1 < \text{v}_2) \land (\text{v}_2 \text{ appears before } \text{v}_1 \text{ in the variable ordering}) \]
BDD Operations: And (Complex Case 2)

What if \( f_0 \bowtie g = f_1 \bowtie g \)?

Return \( f_0 \bowtie g \)

\[
\begin{align*}
(v_1 < v_2) & \quad \text{What if } f_0 \bowtie g = f_1 \bowtie g \\
\text{(Complex Case 2)} & \quad \text{Return } f_0 \bowtie g
\end{align*}
\]
BDD Operations: And

BDD bddAnd (BDD f, BDD g)
  if (f == g || f == True) return g
  if (g == True) return f
  if (f == False || g == False) return False

v = (var(f) < var(g)) ? var(f) : var(g)
f0 = (v == var(f)) ? low(f) : f
f1 = (v == var(f)) ? high(f) : f

g0 = (v == var(g)) ? low(g) : g
g1 = (v == var(g)) ? high(g) : g

T = bddAnd (f1, g1); E = bddAnd (f0, g0)
if (T == E) return T

return mkUnique (v, T, E)

returns unique BDD for ite(v,T,E)
BDD Operations: Or

\[ \text{Or}(f, g) = \text{Not} \left( \text{And} \left( \text{Not}(f), \text{Not}(g) \right) \right) \]

\[ O(n_1 \leq n_2) \]
BDD Operations: Exists

Exists("0",\nu) = ?
BDD Operations: Exists

\[
\text{Exists(“0”,v) = “0”}
\]

\[
\text{Exists(“1”,v) = ?}
\]
BDD Operations: Exists

\[
\begin{align*}
&\text{Exists(“0”, } v \text{) = “0”} \\
&\text{Exists(“1”, } v \text{) = “1”} \\
&\text{Exists((: v \land f) \lor (v \land g) , v) = ?}
\end{align*}
\]
BDD Operations: Exists

 Exists(“0”,v) = “0”

 Exists(“1”,v) = “1”

 Exists((: v Æ f) Ç (v Æ g) , v) = Or(f,g)

 Exists((: v’ Æ f) Ç (v’ Æ g) , v) = ?
BDD Operations: Exists

\[ \text{Exists}("0", v) = "0" \]
\[ \text{Exists}("1", v) = "1" \]

\[ \text{Exists}(\langle v \not\rightarrow f \rangle \sqcap (v \not\rightarrow g) , v) = \text{Or}(f, g) \]
\[ \text{ Exists}(\langle v' \not\rightarrow f \rangle \sqcap (v' \not\rightarrow g) , v) = \]
\[ \langle v' \not\rightarrow \text{Exists}(f,v) \rangle \sqcap (v' \not\rightarrow \text{Exists}(g,v)) \]

But f is SAT iff \( 9 V. f \text{ is not } "0" \). 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?

BDDs are great for computing a fixed points:
  • Set of nodes reachable from a given node in a graph.
Graph Reachability

Which nodes are reachable from “7”?

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

But what if the graph has trillions of nodes?
Graph Reachability

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?
Graph Reachability

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

The nodes whose encodings satisfy the formula.

\[ a \lor b \land : c = ? \]
Key Idea 1: Every Boolean formula represents a set of nodes!
Graph Reachability

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

$\exists \, a \land \exists \, b = \ ?$
Graph Reachability

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

\[ a \land b = \{6,7\} \]
Graph Reachability

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

\[ a \text{ xor } b = ? \]
Key Idea 1: Every Boolean formula represents a set of nodes!

\[ a \text{ xor } b = \{2,3,4,5\} \]
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 © represents all pairs of nodes (n,n’) that satisfy © 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 \land: b \land: c \land: a' \land: b' \land c'
Graph Reachability

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

\[
\begin{align*}
  &a \land \neg b \land c \land \neg a' \land b' \land \neg c' \\
\end{align*}
\]
Graph Reachability

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

\[ a \land b' = c \land a' \land b \land c' \]

\[ \land \]

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

Variable renaming: replace $a'$ with $a$

Image($S$, $R$) =

$\{ (a, b, c . (S \not\in R)) [ a \backslash a', b \backslash b', c \backslash c] \}$

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.
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.