Lecture 11
Binary Decision Diagrams (BDDs)

Boolean Logic Functions Representations
- Function can be represented in different ways
  - Truth table, equation, K-map, circuit, etc...
  - Some representations not unique (not canonical)

Why BDDs
An Efficient Representation
- Synthesis, optimization, verification, and testing algorithms/tools manipulate large Boolean functions
  - Important to have efficient way to represent these functions
- Binary Decision Diagrams (BDDs) have emerged as a popular choice for representing these functions
- BDDs
  - Graph representation similar to a binary tree (i.e. decision trees from previous lectures)
  - Able to efficiently represent large functions
  - Some representations are canonical (unique)
Mux Representation of Boolean Functions

- MUX circuit to implement logic function \( S \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( S(x_1, x_2, x_3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Mux Representation of Boolean Functions
Relation to BDDs

- Corresponding BDD to implement function \( S \)
- One-to-one correspondence to the MUX gates in the flipped circuit

Binary Decision Diagram (BDD)
Example 1

- How does it work?
  - Line with bubble represent value = 0
  - Lines without bubble represent value = 1
Binary Decision Diagram (BDD)

Edge Notations

- Several ways to represent value = 1 and value = 0
  - Bubble vs. Non-bubble line
  - Dashed vs. Solid line
  - T (then) vs. E (else) labels
- We will adopt T vs. E labels - consistent with most of the book (Hatchel) examples

Example 2

Let's consider another function:
\[ f(a,b,c,d) = abc + b'd + c'd \]

What is the value of \( f(1,0,1,0) \)?

Notice that if \( a = 1 \) and \( b = 0 \), the function does not depend on a value for \( c \).

Ordered Binary Decision Diagram (OBDD)

What is a OBDD?

- Ordered binary decision diagrams ensure the variables appear in the same order along all paths from the root to the leaves

Ordering: \( a \leq c \leq b \) Not ordered
Ordered Binary Decision Diagram (OBDD)
Different Ordering Lead to Different Complexity – Example 1

- Variable ordering important, may result in a more complex (or simple) BDD
- All three BDDs below represent the same function
- Third ordering \((b \leq c \leq a \leq d)\) optimal because there is exactly one node for each variable

\[
\begin{align*}
\text{Order: } & a \leq b \leq c \leq d \\
\text{Order: } & a \leq d \leq b \leq c \\
\text{Order: } & b \leq c \leq a \leq d
\end{align*}
\]

\[
\begin{align*}
\text{Order: } & a \leq b \leq c \leq d \\
\text{Order: } & a \leq d \leq b \leq c \\
\text{Order: } & b \leq c \leq a \leq d
\end{align*}
\]

Ordered Binary Decision Diagram (OBDD)
Different Ordering Lead to Different Complexity – Example 2

- Consider \(F = ab + cd + ef\), again both BDDs represent same function
- Variable order has a large impact on resulting BDD; first variable ordering \((a \leq b \leq c \leq d \leq e \leq f)\) yields a much simpler BDD

\[
\begin{align*}
\text{Order: } & a \leq b \leq c \leq d \leq e \leq f \\
\text{Order: } & a \leq c \leq e \leq b \leq d \leq f
\end{align*}
\]

BDDs for Basic Logic Functions
Formal Definition of BDDs

- A BDD is a direct acyclic graph (DAG) representing a multiple-output switching function $F$.

Nodes are partitioned into three subsets:

- **Function node**
  - Represents the function symbol (f)
  - Indegree = 0
  - Outdegree = 1

- **Internal node**
  - Represents variable in function (a, b, c, d)
  - Indegree $\geq 1$
  - Outdegree = 2

- **Terminal node**
  - Represents a value (1 or 0)
  - Indegree $\geq 1$
  - Outdegree = 0

Three types of edges:

- **Incoming edge**
  - From the function node and defines function $f$.

- **T edge**
  - From an internal node and represents when the corresponding variable is 1.

- **E edge**
  - From an internal node and represents when the corresponding variable is 0.

The function $f$ represented by a BDD is defined as follows:

- The function of the terminal node is a constant value (1 or 0).
- The function of a T edge is the function of the head node.
- The function of an E edge is the complement of the function of the node.
- The function of a node is given by $f = f_T + f' E$ where $f_T$ is the function of the T edge and $f'$ is the function of the E edge.
- The function of the function node is the function of its outgoing edge.
**BDD Canonical Form**

- BDDs are canonical (unique) for a representation of \( F \) given a variable ordering \( \pi \).
- All internal nodes are descendants of some node.
- There are no isomorphic subgraphs.
- For every node \( f_T \neq f_E \).

**Isomorphic**

Two graphs are isomorphic if there is a one-to-one correspondence between their vertices and there is an edge between two vertices of one graph if and only if there is an edge between the two corresponding vertices in the other graph.

**Building BDDs For a Function F**

- How do I build a BDD given a function \( F \)?
- Recursive use of Shannon’s Expansion Theorem.
  - \( F = aF_a + a'F_a' \)
  - We can keep applying expansion theorem, eventually we reach the unique canonical form, which uses only minterms.

\[ F = a'b + abc' + a'b'c \]
\[ F = cF_c + c'F_c' \]
\[ F = b(b+b'c) \]
\[ F = (bc') + a'(b+b'c) \]

**Building BDDs - Exercise 1**

- Build a BDD for \( f = abc + ab'c + a'bc' + a'b'c' \).
- Use the variable ordering \( a \leq b \leq c \).

Compute cofactors of \( f \) with respect to \( a \) (first variable in ordering):

\[ f = abc + ab'c + a'bc' + a'b'c' \]
\[ f_a = bc + b'c' \]
\[ f_a' = (b+b'c) \]
Building BDDs - Exercise 1

• Build a BDD for $f = abc + ab'c + a'bc' + a'b'c'$
• Use the variable ordering $a \leq b \leq c$

Compute cofactors of $f_b$ and $f'_b$ with respect to $b$ (second variable in ordering)

- $f_b = bc + b'c$
- $f'_b = bc' + b'c'$

Compute cofactors of $f_{ab}$, $f_{ab'}$, $f_{a'b}$, $f_{a'b'}$ with respect to $c$ (third variable in ordering)

- $f_{abc} = 0$
- $f_{abc'} = 1$
- $f_{ab'c} = 0$
- $f_{ab'c'} = 1$
- $f_{a'bc} = 0$
- $f_{a'bc'} = 1$
- $f_{a'b'c} = 0$
- $f_{a'b'c'} = 1$

Does it work?

---

Building BDDs - Exercise 1

• Build a BDD for $f = abc + ab'c + a'bc' + a'b'c'$
• Use the variable ordering $a \leq b \leq c$

Compute cofactors of $f_{ab}$, $f_{ab'}$, $f_{a'b}$, $f_{a'b'}$ with respect to $c$ (third variable in ordering)

- $f_{abc} = 0$
- $f_{abc'} = 1$
- $f_{ab'c} = 0$
- $f_{ab'c'} = 1$
- $f_{a'bc} = 0$
- $f_{a'bc'} = 1$
- $f_{a'b'c} = 0$
- $f_{a'b'c'} = 1$

Does it work?

---

Building BDDs - Exercise 1

• Build a BDD for $f = abc + ab'c + a'bc' + a'b'c'$
• Use the variable ordering $a \leq b \leq c$

Compute cofactors of $f_{ab}$, $f_{ab'}$, $f_{a'b}$, $f_{a'b'}$ with respect to $c$ (third variable in ordering)

- $f_{abc} = 0$
- $f_{abc'} = 1$
- $f_{ab'c} = 0$
- $f_{ab'c'} = 1$
- $f_{a'bc} = 0$
- $f_{a'bc'} = 1$
- $f_{a'b'c} = 0$
- $f_{a'b'c'} = 1$

Does it work?
Building BDDs - Exercise 2

- Build a BDD for \( f = abc + b'd + c'd \)
- Use the variable ordering \( b \leq c \leq d \leq a \)

Compute cofactors of \( f \) with respect to \( b \) (first variable in ordering):

\[ f = abc + b'd + c'd \]

Partial expansion with respect to \( b \):

\[ f_b' = d + c'd \]

Compute cofactors of \( f_b' \) and \( f_b' \) with respect to \( c \) (second variable in ordering):

\[ f_b = ac + c'd \]

Partial expansion with respect to \( c \):

\[ f_{bc} = a \]

\[ f_{bc'} = d \]

Equivalent cofactors, we can create a single node (reduced):

\[ f_{bc'} = d \]

Compute cofactors of \( f_{bc'} \), \( f_{bc'} \), and \( f_{bc'} \) with respect to \( d \) (third variable in ordering):

\[ f_{bc'd} = f_{bc'd} = f_{bc'd} = 1 \]

\[ f_{bc'd'} = f_{bc'd'} = f_{bc'd'} = 0 \]
**Building BDDs - Exercise 2**

- Build a BDD for \( f = abc + b'd + c'd \)
- Use the variable ordering \( b \leq c \leq d \leq a \)

Compute cofactors of \( f_{bc} \) with respect to \( a \) (fourth variable in ordering):

\[
\begin{align*}
F &= bca + bc'd + b'd \\
&= aF_a + a'F_{a'} \\
&= a(1) + a(0) \\
&= a + 0 \\
&= a \\
\end{align*}
\]

\[
\begin{align*}
C &= cFa + c'Fa' \\
&= c(a) + c'(d) \\
&= ca + c'd \\
\end{align*}
\]

\[
\begin{align*}
B &= bFb + b'F_{b'} \\
&= b(ca + c'd) + b'(d) \\
&= bca + bc'd + b'd \\
\end{align*}
\]

**BDD to Boolean Function – Exercise 1**

- Can we go from a BDD to a Boolean function?
- Sum all paths from function node to terminal nodes:

\[
F = bca + bc'd + b'd
\]
BDD to Boolean Function – Exercise 2

- Another Example

\[ F = abc' + ab' + a' \]

Reducing BDDs

- When building BDDs, result not always reduced (Example 1 - slide 19)
  - We have isomorphic subgraphs, potential for reduction
  - Transform a non-reduced BDD into a reduced BDD by iteratively applying
    - Identify isomorphic subgraphs
    - Remove redundant nodes
  - A ordered reduced BDD (ROBDD) is unique so we have a canonical form

Reducing BDDs

Example 2

- Let’s try to reduce and OBDD
  - Iterative apply
    - Identify isomorphic subgraphs
    - Remove redundant nodes
Summary, and then some...

- Binary Decision Diagrams (BDDs)
  - Efficient mechanism to representation of Boolean functions in terms of memory and CPU
  - Translating function—BDD and BDD—function
  - Importance of variable ordering
  - Method to reduced OBDDs, getting a ROBDD

- We said that BDDs can be efficiently stored and manipulated – HOW?
  - Refer to handwritten notes
  - Supplemental materials attached

ITE Operator

Two argument operators expressed in terms of ITE

<table>
<thead>
<tr>
<th>Table</th>
<th>Name</th>
<th>Expression</th>
<th>Equivalent Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>F 0</td>
<td>F 0</td>
<td>F 0</td>
</tr>
<tr>
<td>0001</td>
<td>F 1</td>
<td>F 0</td>
<td>F 0</td>
</tr>
<tr>
<td>0010</td>
<td>F 0</td>
<td>F 0</td>
<td>F 0</td>
</tr>
<tr>
<td>0011</td>
<td>F 1</td>
<td>F 1</td>
<td>F 1</td>
</tr>
<tr>
<td>0100</td>
<td>G 0</td>
<td>G 0</td>
<td>G 0</td>
</tr>
<tr>
<td>0101</td>
<td>G 0</td>
<td>G 0</td>
<td>G 0</td>
</tr>
<tr>
<td>0110</td>
<td>G 0</td>
<td>G 0</td>
<td>G 0</td>
</tr>
<tr>
<td>0111</td>
<td>G 0</td>
<td>G 0</td>
<td>G 0</td>
</tr>
<tr>
<td>1000</td>
<td>H 0</td>
<td>H 0</td>
<td>H 0</td>
</tr>
<tr>
<td>1001</td>
<td>H 0</td>
<td>H 0</td>
<td>H 0</td>
</tr>
<tr>
<td>1010</td>
<td>H 0</td>
<td>H 0</td>
<td>H 0</td>
</tr>
<tr>
<td>1011</td>
<td>H 0</td>
<td>H 0</td>
<td>H 0</td>
</tr>
<tr>
<td>1100</td>
<td>F 1</td>
<td>F 1</td>
<td>F 1</td>
</tr>
<tr>
<td>1101</td>
<td>F 1</td>
<td>F 1</td>
<td>F 1</td>
</tr>
<tr>
<td>1110</td>
<td>F 1</td>
<td>F 1</td>
<td>F 1</td>
</tr>
<tr>
<td>1111</td>
<td>F 1</td>
<td>F 1</td>
<td>F 1</td>
</tr>
</tbody>
</table>

ITE Algorithm

- Most standard manipulation of BDDs can be done with ITE
- Algorithm recursive, based on formulation where v is the top variable of F, G, H

\[
\text{ITE}_v(F, G, H) = FG + F'G' = v(FG + F'G') + v'(FG + F'G') = \text{ITE}_v \text{ITE}_v(G, H, F) \text{ITE}_v(G, H, F)
\]

- Terminal cases are as follows

\[
\text{ITE}_v(F, G, H) = \text{ITE}_v(F, G, H) = \text{ITE}_v(F, G, H) = \text{ITE}_v(F, G, H) = \text{ITE}_v(F, G, H)
\]
Pseduo-code of the ITE Algorithm

ITE(F, G, H)

(result, terminal_case) = TERMINAL_CASE(F, G, H)  // did we find a terminal case?
if (terminal_case) return (result)

(result, in_computed_table) = COMPUTED_TABLE_HAS_ENTRY(F, G, H)  // have we already calculated this value?
if (in_computed_table) return (result)

v = TOP_VARIABLE(F, G, H)  // recursively calculate this value
T = ITE(Fv, Gv, Hv)
E = ITE(Fv', Gv', Hv')
R = FIND_OR_ADD_UNIQUE_TABLE(v, T, E)  // see if subtree already present
INSERT_COMPUTED_TABLE((F, G, H), R)  // record the calculated value
return (R)
