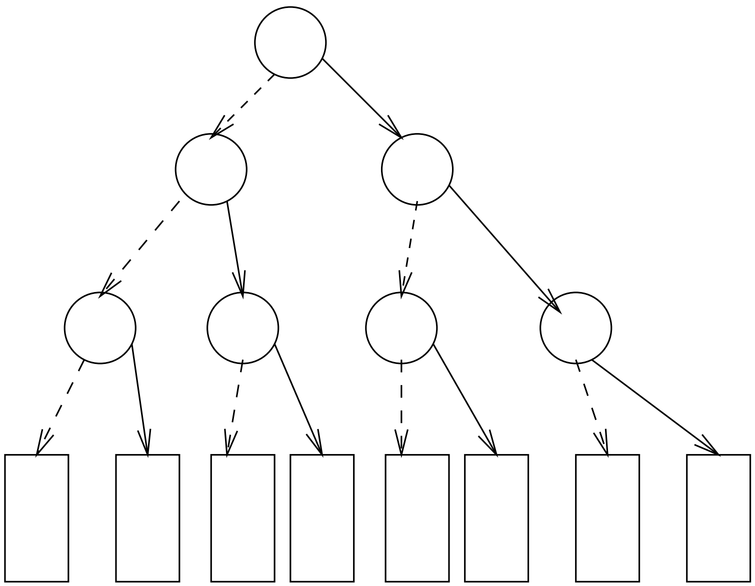
Boolean Function Representation

- Requirements for a Boolean Function Representation?
  - Compact representation: small size
  - Efficiently manipulable: should be easy to operate upon
  - Versatile: Should be able to solve problems of different nature; *e.g.* logic optimization, SAT, testing, verification, etc.
  - What about Canony?
- Does a truth-table satisfy these requirements?
- Does SOP form satisfy these requirements?
- Does a POS form satisfy these requirements?
- Factored form?
- Check for containment, SAT, tautology, etc., is difficult

Binary Decision Diagrams (BDDs)

- Truth Table versus Binary Decision Diagrams
- $f = ab + bc + ac$

a	b	c	|	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	|	1
1	1	1	|	1

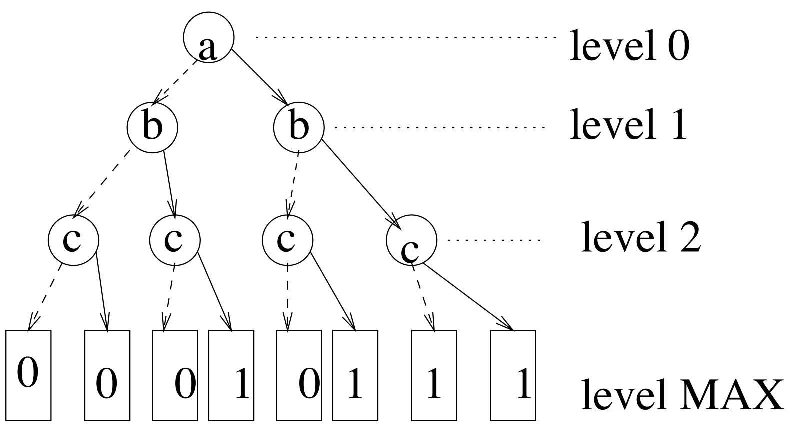


Salient Features of a BDD

- BDD is a Decision Tree
- Variables of the BDD are ordered: called OBDD
- Terminals have numeric values; internal nodes $\equiv$ variables
- Edges $\equiv$ decisions w.r.t variables
- Each internal node has EXACTLY 2 children
- Solid Edge = TRUE edge ($var = 1$), Dashed/dotted edge: FALSE edge ($var = 0$)
- Each node represents a function (computed at that node)
- BDD = effectively a Shannon tree
- OBDD (BDD w/ ordered variables) is a CANONY!
- OBDD = IF-THEN-ELSE structure, hence called ITE DAG

Representing BDD on a Computer

- Assign *levels* to the tree; level $\equiv$ variable order
- Assign *unique* identifiers to each node
- For our majority function: $f = ab + bc + ac$



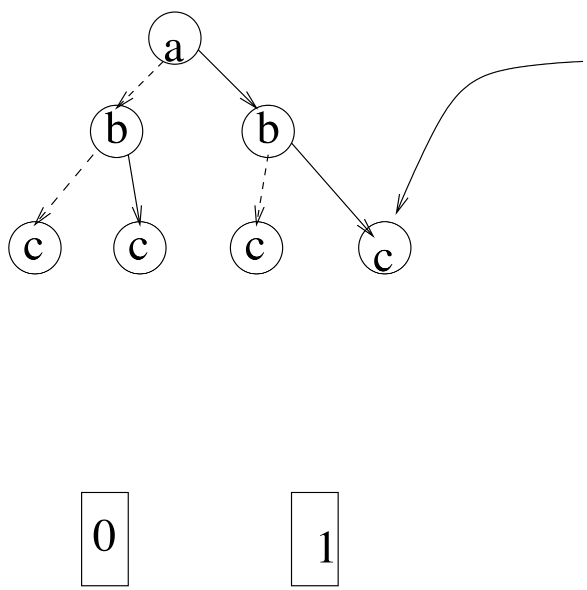

```
DdNode{
  int level /* index or variable order */
  int id    /* unique identifier */
  int value /* for terminal nodes */
  DdNode * T /* PTR to T-child */
  DdNode * E /* PTR to E-child */
}
```

Reduction of an OBDD

- For our same majority function:

  $f = ab + bc + ac = a'bc + ab'c + abc' + abc$

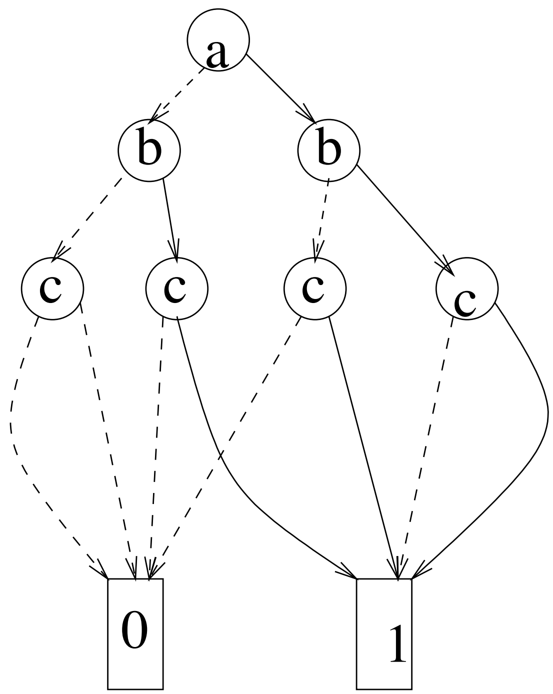
- Merge Terminal Nodes

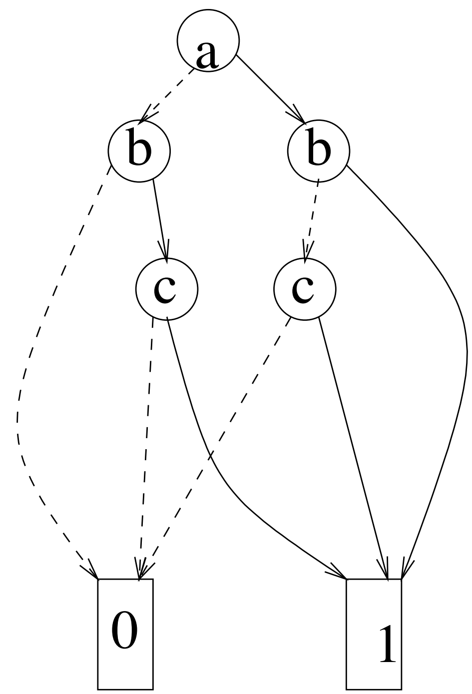



```
DdNode{
  level = 2;
  unique id = 6;
  value = -1 /* non-terminal */
  T-child PTR =
  E-child PTR =
}
```

Reduce OBDD Further...

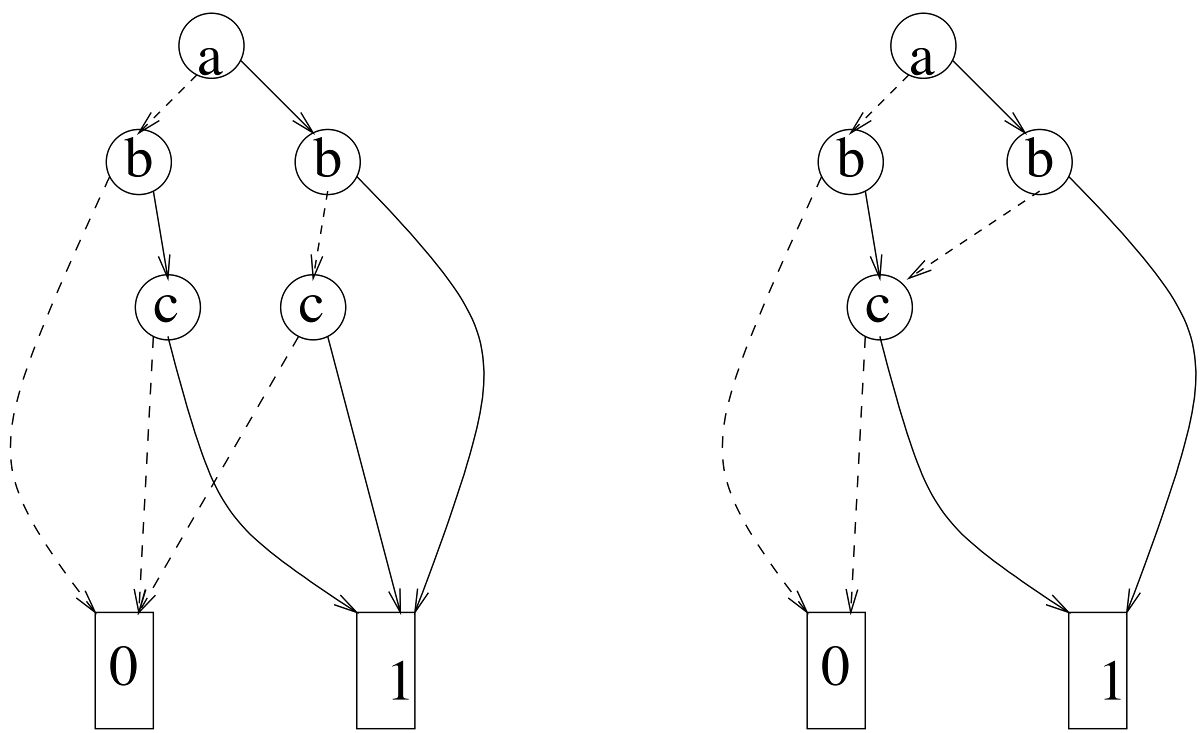
- Remove Redundant Nodes





Reduce OBDD even Further...

- Merge Isomorphic Subgraphs

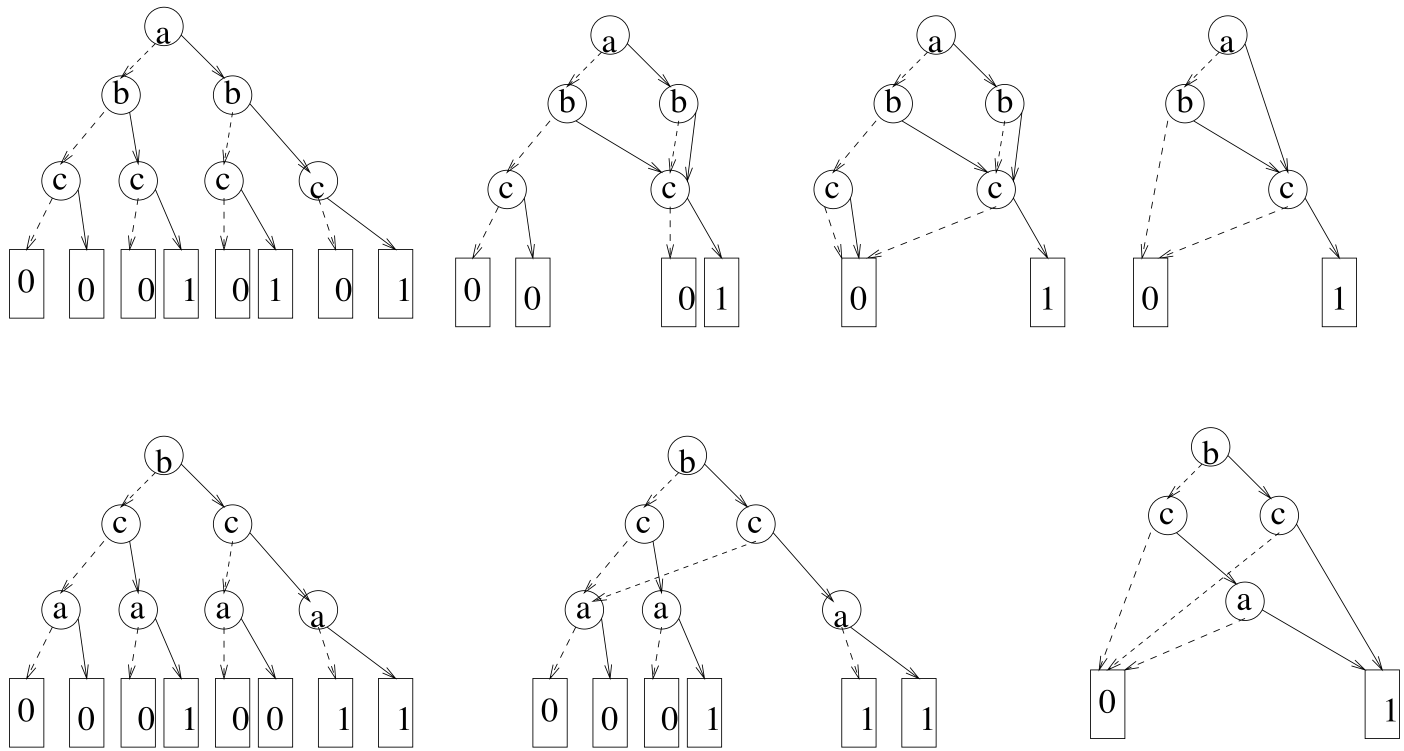


Reduced Ordered Binary Decision Diagram

- Apply reduction operations from terminals to root
- Reduction = remove redundant nodes and merge isomorphic subgraphs
- When you reach the root, you're done!
- ROBDD: subject to a variable order, it is a canony
- If $f = 1$ what does the ROBDD look like?
- Equivalent Boolean Functions have **isomorphic ROBDDs**, if the variable ordering is the same
- What is the effect of Variable Ordering on the size of ROBDD?

Variable Ordering and ROBDD Size

- $f1 = (a + b)c; \;\; f2 = ac + bc$
- $f1 = f2 = ac + bc = ab'c + abc + a'bc$
- Which var order is better? How to find a good order?



Terminology + Definitions

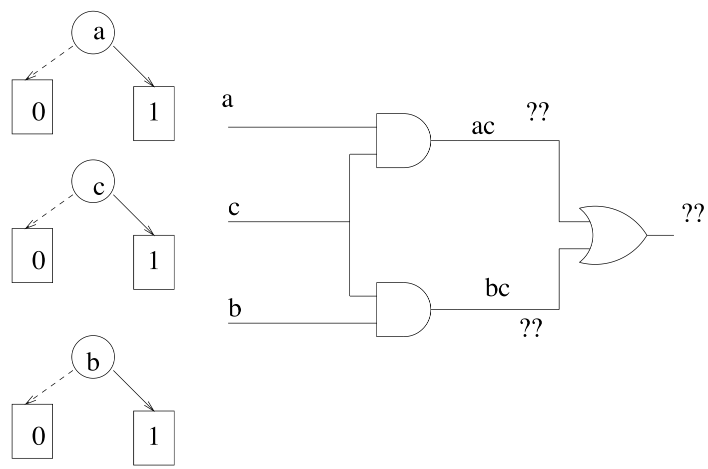
- An **OBDD** is a rooted directed graph with vertex set V. Each non-leaf vertex has as attributes a pointer index(v) $\in \{1, 2, \ldots, n\}$ to an input variable in the set $\{x_1, x_2, \ldots, x_n\}$, and two children low(v), high(v) $\in V$. A leaf vertex v has as an attribute a value, value(v) $\in \mathbf{B}$.
- An OBDD with root v denotes a function f^v such that:
  - If v is a leaf with $value(v) = 1$, $f^v = 1$.
  - If v is a leaf with $value(v) = 0$, $f^v = 0$.
  - If v is a non-leaf node with $index(v) = i$, $f^v = x_i' \cdot f^{low(v)} + x_i \cdot f^{high(v)}$.
- An OBDD is said to be **reduced** (ROBDD) if it contains no vertex v with $low(v) = high(v)$, nor any vertex pair $\{u, v\}$ such that subgraphs rooted at u and v are isomorphic.

Given a Circuit - How to Build ROBDDs?

- Build truth-table $\rightarrow$ then build non-reduced OBDD $\rightarrow$ then reduce it $\rightarrow$ obtain ROBDD
- Can you build truth-table from a huge circuit?
- If you can, why not just work on it, why get into BDDs?
- Recall Truth-table == non-reduced OBDD!
- If you get a HUGE OBDD, Reduce operation becomes infeasible
- How do we "efficiently" build ROBDDs directly from a circuit (function)?
  - How do we obviate the process of first building non-reduced BDD and then applying reduction steps?

Build ROBDD for a Circuit

- $f = ac + bc$
- Build Trivial ROBDDs for a, b, c
- Build ROBDD for ac from ROBDDs for a and c
- Operate on the GRAPHs of a and c and get ac!



First Learn the ITE Operator

- Let $Z = ITE(f, g, h) = f \cdot g + f' \cdot h$
- Let an ROBDD w/ top-node = v compute a function = Z
- Apply Shannon's expansion on Z w.r.t. v

$$\begin{aligned}
Z &= vZ_v + v'Z_{v'} & (1)\\
&= v(ite(f, g, h))_v + v'(ite(f, g, h))_{v'} & (2)\\
&= v(fg + f'h)_v + v'(fg + f'h)_{v'} & (3)\\
&= v(f_v g_v + f'_v h_v) + v'(f_{v'} g_{v'} + f'_{v'} h_{v'}) & (4)\\
&= ite(v, (f_v g_v + f'_v h_v), (f_{v'} g_{v'} + f'_{v'} h_{v'})) & (5)\\
&= ite(v, ite(f_v, g_v, h_v), ite(f_{v'}, g_{v'}, h_{v'})) & (6)\\
&= v \cdot ite(f_v, g_v, h_v) + v' \cdot ite(f_{v'}, g_{v'}, h_{v'}) & (7)
\end{aligned}$$

- Apply ITE at top node → Apply ITE to its co-factors!

Boolean Computations and ITE

- Compute $f \cdot g$ using ITE operation
- $ITE(f, g, h) = f \cdot g + f' \cdot h$
- $ITE(f, g, 0) = f \cdot g + 0$
- Compute $f + g$: $ITE(\_\_, \_\_, \_\_) = f + g$
- Compute $f \oplus g = ITE(\_\_, \_\_, \_\_) = f \cdot g' + f' \cdot g$
- Compute any and all functions using ITE