Check for containment, SAT, tautology, etc., is difficult.

• Factored form?

• Does a POS form satisfy these requirements?

• Does SOP form satisfy these requirements?

• Does a truth-table satisfy these requirements?

– What about Canony?

– etc.

nature; e.g., logic optimization, SAT, testing, verification;

– Versatile: Should be able to solve problems of different

– Efficiently manipulable: Should be easy to operate upon

– Compact representation: small size

Requirements for a Boolean Function Representation
Truth Table Versus Binary Decision Diagrams

## Binary Decision Diagrams (BDDs)

\[ a \ b \ c \ | \ f_0 \ 0 \ 0 \ | \ 00 \ 0 \ 1 \ | \ 00 \ 1 \ 0 \ | \ 00 \ 1 \ 1 \ | \ 11 \ 0 \ 0 \ | \ 01 \ 0 \ 1 \ | \ 11 \ 1 \ 0 \ | \ 11 \ 1 \ 1 \]

\[ a \ b \ c \]

\[ f = \overline{a} b c + \overline{a} q c + a c \]
OBDD = IR-THEN-ELSE structure, hence called IR DAG

OBDD (BDD w/ ordered variables) is a CANDY

BDD = effectively a Shannon tree

Each node represents a function (computed at that node)

\( \text{False edge} (\text{var} = 0), \text{True edge} (\text{var} = 1) \)

Solid edge = True edge

Dashed/dotted edge: Solid edges EXACTLY 2 children

Each internal node has EXACTLY 2 children

Edges \( \equiv \) decisions w.r.t. variables

Terminals have numeric values; internal nodes \( \equiv \) variables

Variables of the BDD are ordered; called OBDD

BDD is a decision tree
For our majority function: \[ f = \overline{aq + bc + ac} \]

Assign unique identifiers to each node
Assign levels to the tree; level \( \equiv \) variable order

Representing BDD on a Computer
DNode

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

Merge Terminal Nodes

\[
abc + abe + abc + ab\bar{c} + a\bar{d}c + a\bar{d}\bar{e} = f
\]

For our same majority function:

\[
\text{Reduction of an OBDD}
\]
Remove Redundant Nodes

Reduce OBDD Further
Merge Isomorphic Subgraphs

Reduce OBDD even further...
What is the effect of Variable Ordering on the size of ROBDD?

* If the variable ordering is the same, Equivalent Boolean Functions have isomorphic ROBDDs.

If $f = 1$, what does the ROBDD look like?

When you reach the root, you're done!

Applying Reduction operations from terminals to root

Reduced Ordered Binary Decision Diagram
Which var order is better? How to find a good order?

\[
abc + ab\overline{c} + a\overline{bc} = \overline{q} + \overline{ac} + \overline{a}\overline{bc} = 3f = 1f
\]

\[
\overline{q} + \overline{ac} = 3f, (\overline{q} + a) = 1f
\]
An OBDD is said to be reduced (ROBDD) if it contains no
\[ (\text{high}) \cdot \overline{x} + (\text{low}) = f \]
\[ (\text{high}) = (\text{index}) \cdot x = 0 \]
\[ (\text{low}) = (\text{index}) \cdot x = 1 \]

An OBDD with root \( \alpha \) denotes a function \( f \) such that:

An OBDD with attribute a value, value(\( \alpha \), value(\( \lambda \)), high(\( \lambda \)), low(\( \lambda \)), \( \alpha \) ∈ B.

A leaf vertex has an attribute value as an

Each non-leaf vertex has an attribute as an

An OBDD is a rooted directed graph with vertex set \( \Lambda \).

Terminology + Definitions
Given a circuit - How to Build ROBDDs?

- How do we obviate the process of first building non-reduced ROBDDs directly from a circuit?

  (function)
  
  If you get a huge OBDD, reduce operation becomes infeasible.

Recall truth-table == non-reduced OBDD?

If you can, why not just work on it, why get into BDDs?

Can you build truth-table from a huge circuit?

Reduce it - obtain ROBDD

Build truth-table -> then build non-reduced OBDD -> then

BDD and then applying reduction steps?
Operate on the graph of $a$ and $c$ and get ac!

Build RObDD for ac from RObDDs for $a$ and $c$

Build Trivial RObDDs for $a', q, c$

\[ ac + bc = f \]
Apply ITE at top node \( \text{ITE} \) to its co-factors:

\[
\begin{align*}
(7) & \quad (a y, a b, a f) \text{ITE} \cdot \alpha + (a y, a b, a f) \text{ITE} \cdot \alpha = \\
(6) & \quad ((a y, a b, a f) \text{ITE}, (a y, a b, a f) \text{ITE}) \alpha = \\
(5) & \quad ((a y, a f + a b, a f) (a y, a f + a b, a f) \text{ITE}) \alpha = \\
(4) & \quad (a y, a f + a b, a f) \alpha + (a y, a f + a b, a f) \alpha = \\
(3) & \quad a((y, b f) \text{ITE}) \alpha + a((y, b f) \text{ITE}) \alpha = \\
(2) & \quad a Z \alpha + a Z \alpha = Z
\end{align*}
\]

Apply Shannon's expansion on \( Z \) w.r.t. \( \alpha \):

\[
Z = \text{compute a function } \alpha = \text{compute top-node } \text{ITE} / \text{RDBD} \text{ w.r.t. } Z \text{ w.r.t. } \text{ITE} \text{ Operator}
\]

First learn the ITE Operator.
Compute any and all functions using ITE

\[ b \cdot I + b \cdot f = (\neg, \neg, \neg) \]

\[ b \oplus f \]

\[ b + f = (\neg, \neg, \neg) \]

\[ b + f \]

\[ 0 + b \cdot f = (0, b, f) \]

\[ 0 + b \cdot f \]

\[ \eta \cdot I + b \cdot f = (\eta, b, f) \]

\[ \eta \cdot f + b \cdot f = (\eta, b, f) \]

Boolean Computation and ITE using ITE operations
Build ROBDD using ITE operator.