Binary Decision Diagrams

Hao Zheng

Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
Email: zheng@cse.usf.edu
Phone: (813)974-4757
Fax: (813)974-5456

Binary Decision Diagrams

- Binary decision diagrams (BDDs) are graphs representing Boolean functions.
- They can be made canonical.
- They can be very compact for many applications.
- They are important since many applications can be converted to sequences of Boolean operations.
- References:
  - R. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, 1986.
  - R. Bryant *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*, ACM Computing Surveys, 1992.
  - Textbook, $6.7.3 - 6.7.4$

Representing Switching Functions

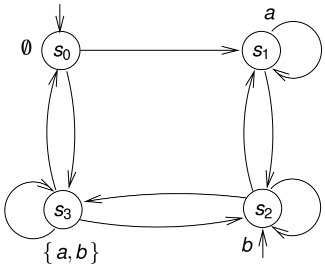
- Truth Tables
  - Satisfiability and equivalence check: easy; boolean operations also easy.
  - Very space inefficient: 2^n entries for n variables.
- Disjunctive Normal Form (DNF)
  - Satisfiability is easy: find a disjunct without complementary literals.
  - Negation and conjunction complicated.
- Conjunctive Normal Form (CNF)
  - Satisfiability problem is NP-complete (Cook's theorem).
  - Negation and disjunction complicated.

Representing Switching Functions

representation	*compact?*	*sat*	*equ*	$\wedge$	$\vee$	$\neg$
truth table	never	hard	hard	hard	hard	hard
DNF	sometimes	easy	hard	hard	easy	hard
CNF	sometimes	hard	hard	easy	hard	hard
propositional formula	often	hard	hard	easy	easy	easy
reduced ordered binary decision diagram	often	easy	easy*	medium	medium	easy

* Provided appropriate implementation techniques are used.

Symbolic Encoding: An Example

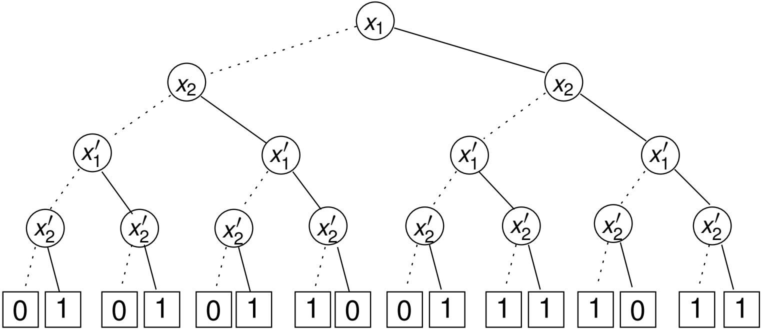


Switching function: $\Delta(\underbrace{x_1, x_2}_{s}, \underbrace{x_1', x_2'}_{s'}) = 1$ if and only if $s \to s'$

$$\begin{array}{rcl} \Delta(x_1, x_2, x_1', x_2') = & & (\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge x_2') \\ & \vee & (\neg x_1 \wedge \neg x_2 \wedge x_1' \wedge x_2') \\ & \vee & (\neg x_1 \wedge x_2 \wedge x_1' \wedge \neg x_2') \\ & \vee & \ldots \\ & \vee & (x_1 \wedge x_2 \wedge x_1' \wedge x_2') \end{array}$$

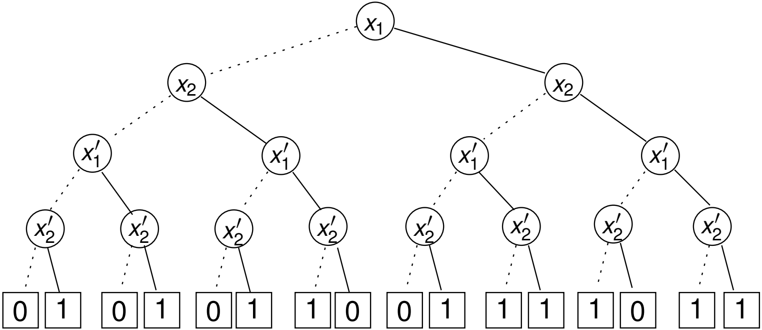
Contents

Binary Decision Tree: Example



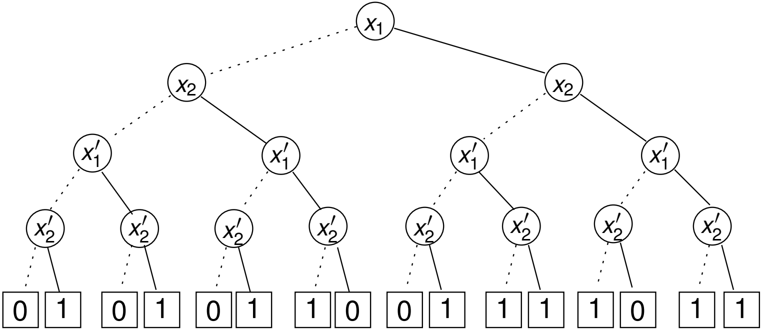
- The BDT for function f on $\vec{x} = \{ x_1, \dots, x_n \}$ has depth n.
- Every node is labeled with a variable.
- Every node labeled with x_i has two outgoing edges.
  - **0**-edge: $x_i = 0$ (dashed) and,
  - **1**-edge: $x_i = 1$ (solid).

Binary Decision Tree: Example



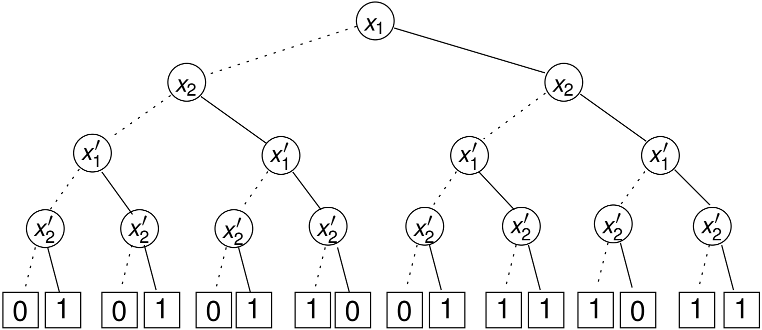
- Every non-terminal node n has two successor nodes.
  - low(n): the node at the end of the $\mathbf{0}$-edge of node n.
  - high(n): the node at the end of the $\mathbf{1}$-edge of node n.

Binary Decision Tree: Example



- The edge labelings of a path from the root to a terminal is an evaluation $s = [x_1 = b_1, \ldots, x_m = b_m]$, where $b_i \in \{0, 1\}$.
- The labeling of the terminal node is the output of $f(s)$.

Binary Decision Tree: Example



The subtree of node v at level i for variable ordering $x_1 < \ldots < x_n$ represents:

$$f_v = f|_{x_1=b_1,\ldots,x_{i-1}=b_{i-1}}$$

which is a switching function over $\{ x_i, \ldots, x_n \}$ and where $x_1 = b_1$, ..., $x_{i-1} = b_{i-1}$ are the decisions made along the path from root to node v.

Binary Decision Tree

- The BDT for function f on $Var = \{z_1, \ldots, z_m\}$ has depth m with outgoing edges for node at level i stand for $z_i = 0$ (dashed) and $z_i = 1$ (solid).
- For evaluation $s = [z_1 = b_1, \ldots, z_m = b_m]$, $f(s)$ is the value of the leaf reached by traversing the BDT from the root using branch $z_i = b_i$.
- The subtree of node v at level i for variable ordering $z_1 < \ldots < z_m$ represents:

$$f_v \;=\; f|_{z_1=b_1,\ldots,z_{i-1}=b_{i-1}}$$

which is a switching function over $\{z_i, \ldots, z_m\}$ and where $z_1 = b_1, \ldots, z_{i-1} = b_{i-1}$ are the decisions made along the path from root to node v.

Considerations on BDTs

- BDTs are a different form of truth tables.
- BDTs are not compact:
  - A BDT for switching function f on n variables has 2^n leafs.
  - The size of a BDT does not change if the variable order changes.

  $\Rightarrow$ They are as space inefficient as truth tables!
- BDTs contain a lot of redundancy:
  - All leafs with value one (zero) could be collapsed into a single leaf.
  - A similar scheme could be adopted for *isomorphic* subtrees.

Two graphs rooted at nodes u and v are **isomorphic**, denoted as $u \equiv v$ when both following conditions hold.

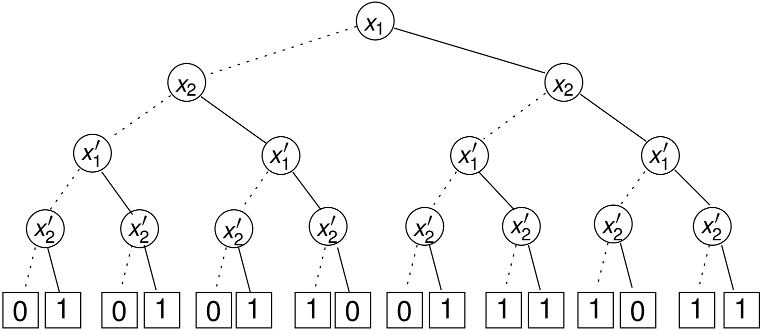
- $value(u) = value(v)$ if u and v are terminals.
- $\mathsf{low}(u) \equiv \mathsf{low}(v) \ \wedge \ \mathsf{high}(u) \equiv \mathsf{high}(v)$, otherwise.

Contents

Ordered Binary Decision Diagram (OBDD)

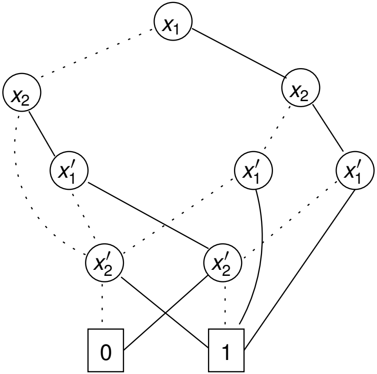
- OBDDs rely on compactions of BDT representations.
- Idea: skip redundant fragments of BDT representations.
- Collapse subtrees with all terminals having same value.
- Identify nodes with isomorphic subtrees.
- This yields directed acyclic graphs with outdegree two.
- Inner nodes are labeled with variables.
- Leafs are labeled with function values (zero and one).
- A unique variable ordering is followed by every path.
- Each variable is assigned an unique index.
- Each BDD node v has an index $index(v)$ which is the index of the variable labeled in v.
  - $index(v) < index(\text{low}(v))$ if low(v) is a non-terminal,
  - $index(v) < index(\text{high}(v))$ if high(v) is a non-terminal.

Transition Relation as a BDT



A BDT representing Δ for our example using ordering $x_1 < x_2 < x_1' < x_2'$.

Transition Relation as a BDD



A BDT representing Δ for our example using ordering $x_1 < x_2 < x_1' < x_2'$.

OBBDs and Boolean Functions

- Let $\wp$ be a variable ordering for Var where $\wp = (z_1, \ldots, z_m)$.
- Every OBDD is defined wrt a given variable ordering.
  - The nodes in every path are labeled with variables in the order as in $\wp$.
- A terminal node represents a constant Boolean function either $\mathbf{1}$ or $\mathbf{0}$.
- For a non-terminal node n labeled with z representing a Boolean function f_n, its two successor nodes represent Boolean functions:
  - Node at the end of the $\mathbf{0}$-edge (low(n)): $f_n|_{z=0}$.
  - Node at the end of the $\mathbf{1}$-edge (high(n)): $f_n|_{z=1}$.

  Therefore,

$$f_n = \neg z \wedge f_n|_{z=0} \;\vee\; z \wedge f_n|_{z=1}$$

- A OBDD is *reduced* (i.e. ROBDD) if for every pair (v, w) of nodes,

$$v \neq w \quad \text{implies} \quad f_v \neq f_w.$$

Universality and Canonicity Theorem

[Fortune, Hopcroft & Schmidt, 1978]

Let $\vec{x}$ be a finite set of Boolean variables and $\wp$ a variable ordering for $\vec{x}$.

(a) For each switching function f for $\vec{x}$ there exists a $\wp$-ROBDD $OBDD$ with $f_{OBDD} = f$.

(b) For any $\wp$-ROBDDs G and H with $f_G = f_H$, G and H are isomorphic, i.e., agree up to renaming of the nodes.

- ROBDDs are canonical for a fixed variable ordering.

The Importance of Canonicity

- Absence of redundant vertices:

  If f_B does not depend on x_i, ROBDD B does not contain an x_i node.
- Test for equivalence: $f(x_1, \ldots, x_n) \equiv g(x_1, \ldots, x_n)$?

  Generate ROBDDs B_f and B_g, and check isomorphism.
- Test for validity: $f(x_1, \ldots, x_n) = 1$?

  Generate ROBDD B_f and check whether it only consists of a 1-leaf.
- Test for implication: $f(x_1, \ldots, x_n) \rightarrow g(x_1, \ldots, x_n)$?

  Generate ROBDD $B_{f \ \wedge \ \neg g}$ and check if it just consists of a 0-leaf.
- Test for satisfiability:

  f is satisfiable if and only if B_f has a reachable 1-leaf.

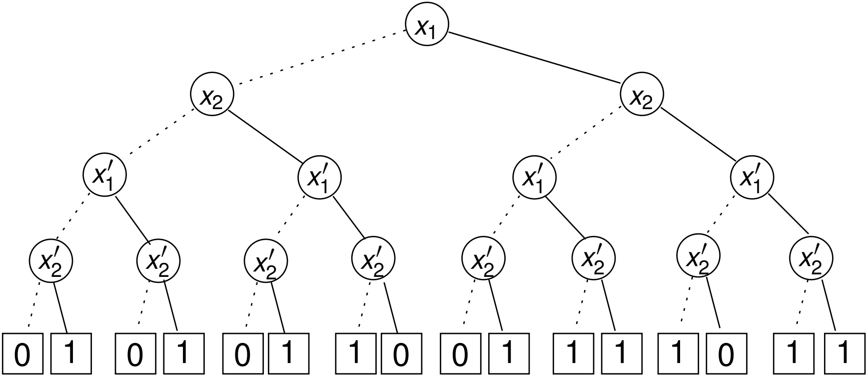
Contents

Reducing OBDDs

- Generate an OBDD (or BDT) for a boolean expression, then reduce by means of a recursive descent over the OBDD.
- Elimination rule:

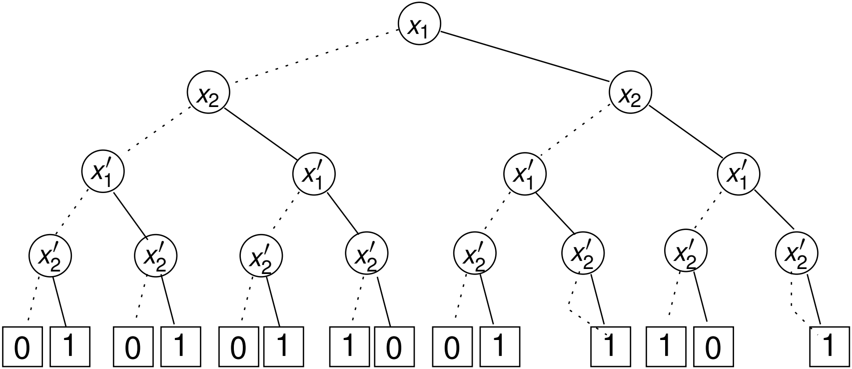
  If $\text{low}(v) = \text{high}(v) = w$, eliminate v and redirect all incoming edges to v to node w.
- Isomorphism rule:
  - If $v \neq w$ are roots of isomorphic subtrees, remove v, and redirect all incoming edges to v to node w.
  - (Special case) Combine all 0/1-leaves, redirect all incoming edges.

From BDT to ROBDD



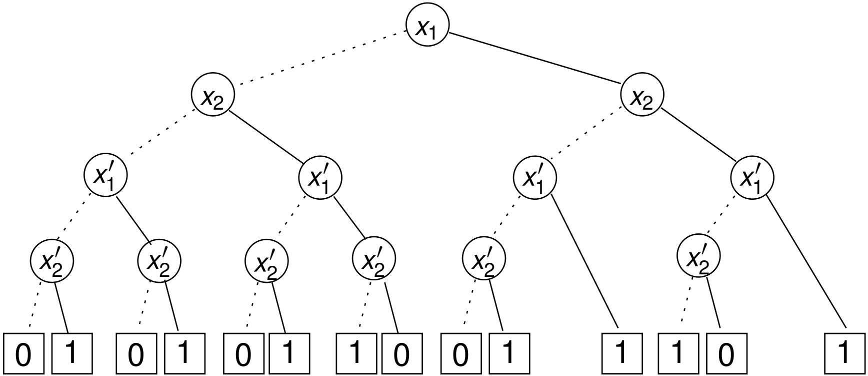
An OBDD representing Δ for our example using ordering $x_1 < x_2 < x_1' < x_2'$.

From BDT to ROBDD



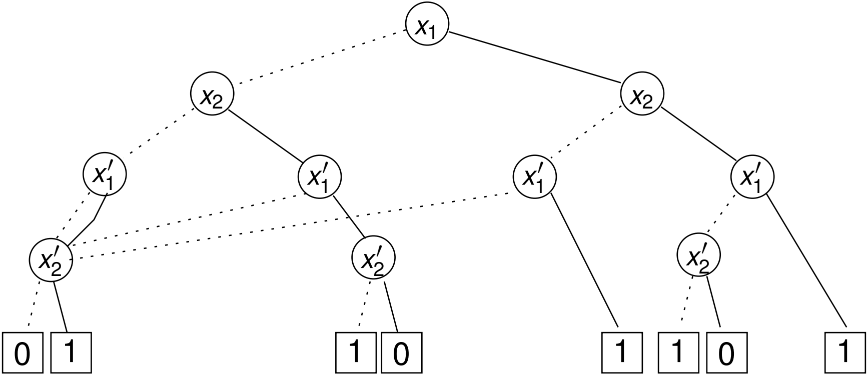
After isomorphism rule
Next, elimination rule

From BDT to ROBDD



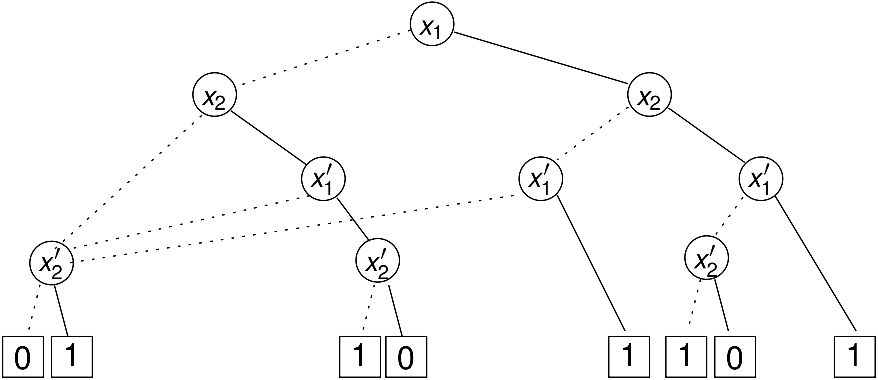
Next, isomorphism rule

From BDT to ROBDD



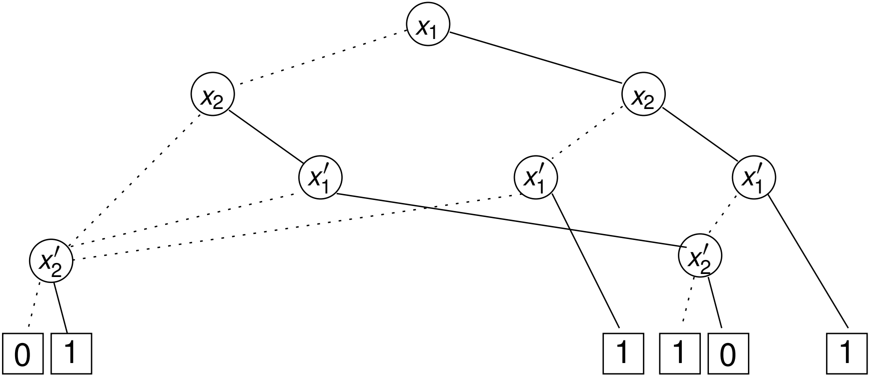
Next, elimination rule

From BDT to ROBDD



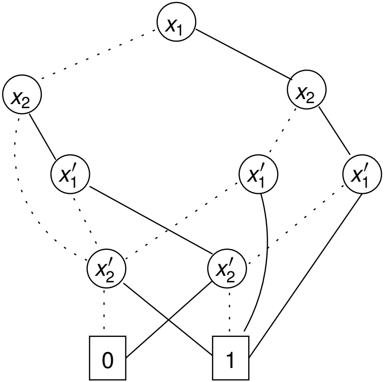
Next, isomorphism rule

From BDT to ROBDD



Next, isomorphism rule

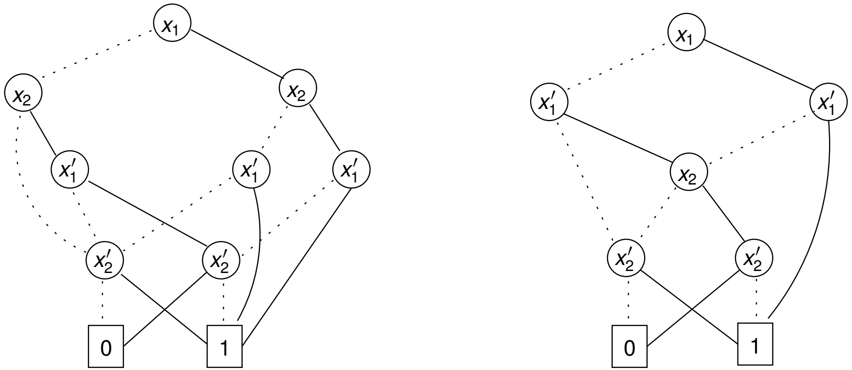
From BDT to ROBDD



Final reduced BDD

Contents

Variable Ordering



(a) ordering $x_1 < x_2 < x_1' < x_2'$

(b) ordering $x_1 <' x_1' <' x_2 <' x_2'$

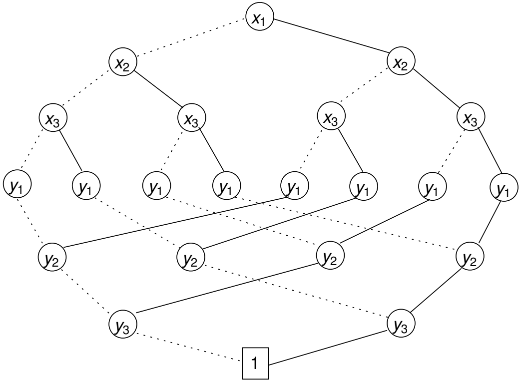
Variable Ordering and Size of BDDs

- The size of the ROBDD crucially depends on the variable ordering.
- $\#$ nodes in ROBDD $OBDD = \#$ of $\wp$-consistent co-factors of f.
- Some switching functions have linear and exponential ROBDDs.

  e.g., the addition function, or the stable function.
- Some switching functions only have polynomial ROBDDs.
  - This holds, e.g., for symmetric functions.
  - Examples $f(\ldots) = x_1 \oplus \ldots \oplus x_n$, or $f(\ldots) = 1$ iff $\geq k$ variables x_i are true.
- Some switching functions only have exponential ROBDDs.

  This holds, e.g., for the middle bit of the multiplication function.

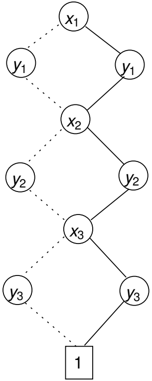
The Function Stable with Exponential ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \;\wedge\; \ldots \;\wedge\; (x_n \leftrightarrow y_n)$

has $3{\cdot}2^n - 1$ vertices under ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$.

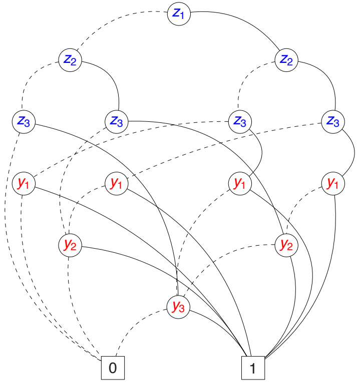
The Function Stable with Linear ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \;\wedge\; \ldots \;\wedge\; (x_n \leftrightarrow y_n)$

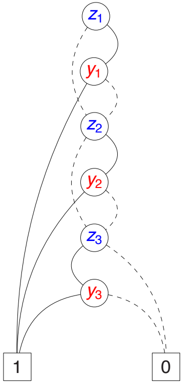
has $3{\cdot}n + 2$ vertices under ordering $x_1 < y_1 < \ldots < x_n < y_n$.

Another Function with an Exponential ROBDD



ROBDD for $f_3(\overline{\vec{z}}, \overline{\vec{y}}) = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$
for the variable ordering $z_1 < z_2 < z_3 < y_1 < y_2 < y_3$.

An Optimal Linear ROBDD



- ROBDD for $f_3(\cdot) = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$.
- For ordering $z_1 < y_1 < z_2 < y_2 < z_3 < y_3$.
- As all variables are essential, this ROBDD is optimal.
- For no variable ordering, a smaller ROBDD exists.

The Multiplication Function

- Consider two n-bit integers:

  Let $b_{n-1}b_{n-2}\ldots b_0$ and $c_{n-1}c_{n-2}\ldots c_0$ where b_{n-1} is the most significant bit, and b_0 the least significant bit.
- Multiplication yields a $2n$-bit integer:

  The ROBDD $OBDD_{f_{n-1}}$ has at least 1.09^n vertices where f_{n-1} denotes the $(n-1)$-st output bit of the multiplication.

Optimal Variable Ordering

- The size of ROBDDs is dependent on the variable ordering.
- Is it possible to determine $\wp$ such that the ROBDD has minimal size?
  - To check whether a variable ordering is optimal is NP-hard.
  - Polynomial reduction from the 3SAT problem. [Bollig & Wegener, 1996]
- There are many switching functions with large ROBDDs:

  For almost all switching functions the minimal size is in $\Omega(\frac{2^n}{n})$.
- How to deal with this problem in practice?
  - Guess a variable ordering in advance.
  - Rearrange the variable ordering during the ROBDD manipulations.
  - Not necessary to test all $n!$ orderings, best known algorithm in $\mathcal{O}(3^n \cdot n^2)$.

Dynamic Re-ordering

- Finding an optimal ordering is NP-hard.
- Static ordering does not work well across different applications, or for BDDs that need to be transformed during different stages of an application.
- Automated dynamic re-ordering rearranges the variable orders periodically to reduce the size of BDDs.
- Rudell's "sifting" is widely used.
  - Try moving a variable to all other positions, leaving the others fixed. Then place variable in the position that minimizes BDD size.
  - Do this for all variables.

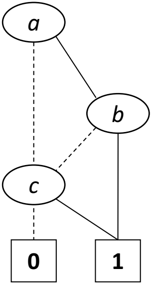
Dynamic Re-ordering

- Greatly improved effectiveness of BDDs.
- It is usually performed in the background.
- It may slow down the performance.
  - BDD operations stop when re-ordering is activated.
- It makes a difference between success and failure in complete an application.
- Some functions are inherently hard, e.g. outputs of integer multiplier.

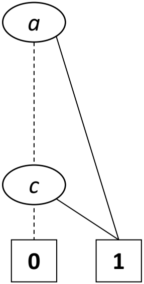
Contents

Logic Operations on BDDs

- **Restriction** $f[b/x]$: replacing variable x with a value 0 or 1.
  - if $b = 0$, direct all incoming edges of node labeled with x to low(v), or
  - if $b = 1$, direct all incoming edges of node labeled with x to high(v),
  - remove node x and its outgoing edges.
- The result of restriction is a cofactor of f.



f



f[*1/b*]

Logical Operations on BDDs (cont'd)

- **Negation** is a constant time operation with OBDDs.
  - Swap terminal nodes.
- The binary operations are based on the Shannon expansion.

$$f \circ g = \overline{x} \cdot (f[0/x] \circ g[0/x]) \vee x \cdot (f[1/x] \circ g[1/x]).$$

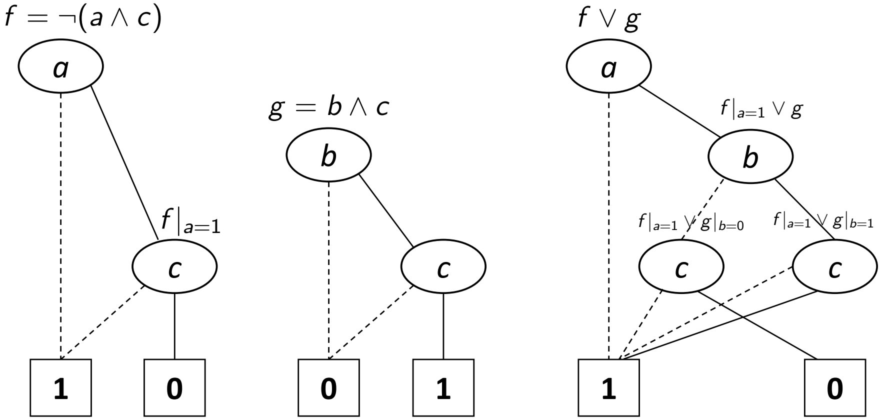
where $\circ$ is some binary logic operator.

- Both BDDs must have the same variable ordering.
- The new BDD for $f \circ g$ is constructed as follows.
  - The root of $f \circ g$ is the root of f or g with the smaller index.
  - For any node in $f \circ g$,

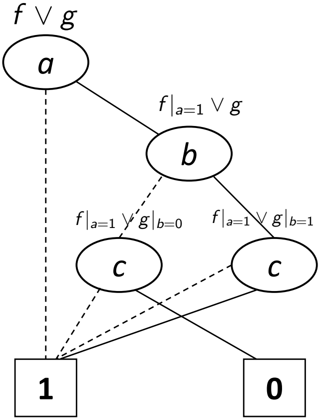
$$\mathsf{low}(v) = f[0/x] \circ g[0/x], \text{ and } \mathsf{high}(v) = f[1/x] \circ g[1/x]$$

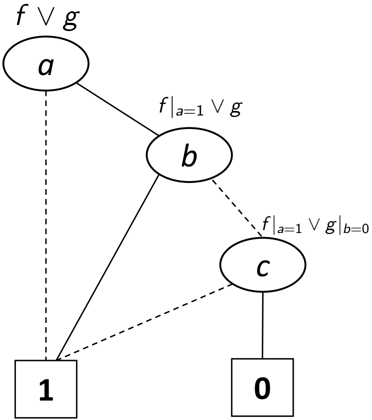
  - Repeat the above step for low(v) and high(v) until either of them becomes terminal.
  - Reduce the constructed BDD to make it canonical.

Logical Operations on BDDs — Example



Logical Operations on BDDs — Example



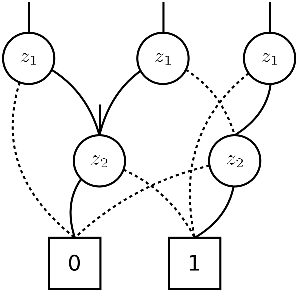


Variants of BDDs

- Various kinds of BDDs for compactness or different applications.
- Compactness
  - Multi-rooted BDDs, free BDDs, partitioned OBDDs, etc
- Arithmetic operations
  - Multi-terminal BDDs (ADDs), edge-valued BDDs, binary moment diagrams (BMDs), etc.

Implementation: Shared OBDDs

A shared $\wp$-OBDD is an OBDD with multiple roots.



Shared OBDD representing $\underbrace{z_1 \wedge \neg z_2}_{f_1}$, $\underbrace{\neg z_2}_{f_2}$, $\underbrace{z_1 \oplus z_2}_{f_3}$ and $\underbrace{\neg z_1 \vee z_2}_{f_4}$.

Main underlying idea: combine several OBDDs with same variable ordering such that common $\wp$-consistent co-factors are shared.