BINARY DECISION DIAGRAMS

©*Giovanni De Micheli*

Stanford University

Outline

- Binary Decision Diagrams
- Operations with BDDs.
- Optimization of the BDD size:
  - Variable reordering.
- Other types of Decision Diagrams.

Binary Decision Diagrams

- Efficient representation of logic functions.
  - Proposed by Lee and Akers.
  - Popularized by Bryant (canonical form).
- Used for Boolean manipulation.
- Applicable to other domains:
  - Set and relation representation.
  - Simulation, finite-system analysis, ...

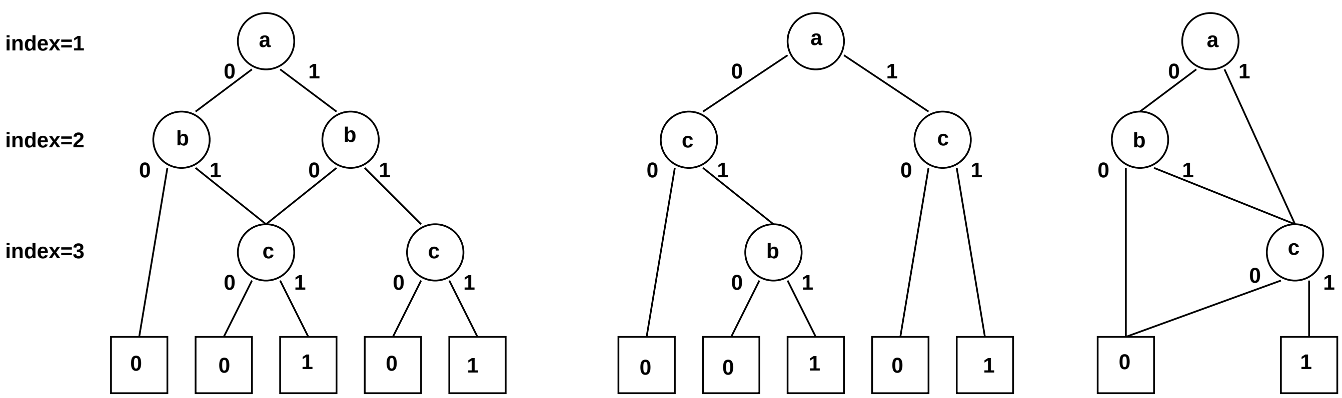
Definitions

- *Binary decision diagram* (BDD).
  - Tree or rooted dag
    with a decision at each vertex.

- *Ordered binary decision diagram* (OBDD).
  - Each decision is the evaluation of a Boolean variable.
  - The tree (or dag) can be levelized,
    so that each level corresponds to a variable.

Example

$$f = (a + b)c$$



(a) (b) (c)

Definition of OBDD

- Rooted directed acyclic graph.

- Each non-leaf vertex (v) has:
  - A pointer $index(v)$ to a variable.
  - Two children $low(v)$ and $high(v)$.

- Each leaf vertex (v) has a value (1 or 0).

- Ordering:
  - $index(v) < index(low(v))$.
  - $index(v) < index(high(v))$.

Properties

- Each OBDD with root v defines a function f^v:
  - If v is a leaf with $value(v) = 1$, then $f^v = 1$.
  - If v is a leaf with $value(v) = 0$, then $f^v = 0$.
  - If v is not a leaf and $index(v) = i$, then $f^v = x_i' \cdot f^{low(v)} + x_i \cdot f^{high(v)}$.
- A function may have different OBDDs.
- The size of the OBDD depends on the variable order.

ROBDDs

- *Reduced ordered binary decision diagrams.*

- No redundancies:

  - No vertex with $low(v) = high(v)$.

  - No pair $\{u, v\}$ with isomorphic subgraphs rooted in u and v.

- Reduction can be achieved in polynomial time.

- ROBDDs can be such by construction.

- ROBDDs are *canonical forms*.

Features

- Canonical form allows us to:
  - Verify logic equivalence in constant time.
  - Check for tautology and perform logic operations in time proportional to the graph size. (Vertex cardinality).
- Drawback:
  - Size depends on *variable order.*

ROBDD size bounds

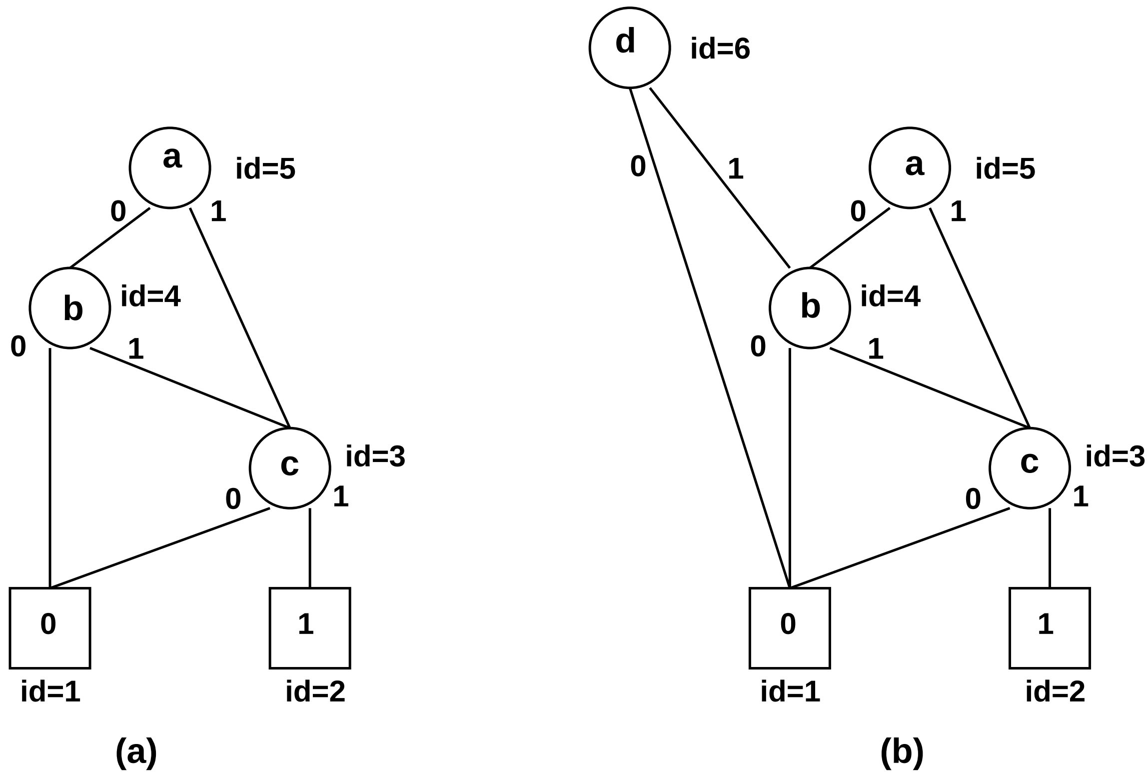
- Multiplier:
  - Exponential size.
- Adders:
  - Exponential to linear size.
- Sparse logic:
  - Good heuristics to keep size small.

Tabular representations of ROBDDs

- Represent multi-rooted graphs.
  - Multiple-output functions.
  - Multiple-level logic forms.
- Unique table:
  - One row per vertex.
    - *Identifier.*
    - *Key: (variable, left child, right child).*

Example
unique table

© GDM



Identifier	*Key*		
	Variable	*Left child*	*Right child*
6	d	1	4
5	a	4	3
4	b	1	3
3	c	1	2

The *ite* operator

- Apply operators to ROBDDs.

- Three Boolean functions: f, g, h with top variable x.

- $ite(f, g, h)$
  - if (f) then (g) else (h)
  - $fg + f'h$.

- Property:
  - $ite(f, g, h) = ite(x, ite(f_x, g_x, h_x), ite(f_{x'}, g_{x'}, h_{x'}))$

Example

- Apply and to two ROBDDs: f, g.
  - $fg = ite(f, g, 0)$
- Apply or to two ROBDDs: f, g.
  - $f + g = ite(f, 1, g)$
- Similar for other Boolean operators.

Boolean operators

Operator	*Equivalent* ite *form*
0	0
$f \cdot g$	$ite(f, g, 0)$
$f \cdot g'$	$ite(f, g', 0)$
f	f
$f'g$	$ite(f, 0, g)$
g	g
$f \oplus g$	$ite(f, g', g)$
$f + g$	$ite(f, 1, g)$
$(f + g)'$	$ite(f, 0, g')$
$f \overline{\oplus} g$	$ite(f, g, g')$
g'	$ite(g, 0, 1)$
$f + g'$	$ite(f, 1, g')$
f'	$ite(f, 0, 1)$
$f' + g$	$ite(f, g, 1)$
$(f \cdot g)'$	$ite(f, g', 1)$
1	1

The ITE algorithm

- Evaluate the $ite(f, g, h)$ operator recursively.

- Keeps OBDDs in reduced form.

- Use two tables (per function):

  - *Unique table*: represents ROBDD.

  - *Computed table*: stores previous info.

- Smart implementations of ITE have linear time complexity in the product of the ROBDD sizes.

The ITE algorithm


```
ITE(f, g, h){
    if (terminal case)
        return (r = trivial result);
    else {
        if (computed table has entry {(f, g, h), r})
            return (r from computed table);
        else {
            x = top variable of f, g, h;
            t = ITE(f_x, g_x, h_x);
            e = ITE(f_x', g_x', h_x');
            if ( t == e)
                return (t);
            r = find_or_add_unique_table(x, t, e);
            Update computed table with {(f, g, h), r};
            return (r);
        }
    }
}
```

Quantification with BDDs
Consensus and smoothing

- Quantification can be computed by ITE.

- Specialized algorithm is more efficient.
  - Structure similar to ITE.
  - Arguments:
    * Function f.
    * Variables in $varlist$.
  - Function $OP(t, e)$ returns:
    * Consensus: $AND(t, e) = ITE(t, e, 0)$.
    * Smoothing: $OR(t, e) = ITE(t, 1, e)$.

```
QUANTIFY(f, varlist){
    if (f is constant)
        return (f);
    else {
        if (comp. table has entry {(f, varlist), r})
            return (r from computed table);
        else {
            x = top variable of f;
            g = f_x;
            h = f_x';
            t = QUANTIFY(g, varlist);
            e = QUANTIFY(h, varlist);
            if (x is in varlist )
                r = OP(t, e);
            else
                r = ITE(x, t, e) ;
            Update comp. table {(f, varlist), r};
            return (r);
        }
    }
}
```

Example

- Function $f = ab + bc + ac$
- Consensus: $\mathcal{C}_a(f)$.
- $varlist = a$
- $QUANTIFY(f, a)$ with top variable a.
  - Cofactors: $g = f_a = b + c$ and $h = f_{a'} = bc$.
  - Recursion: $t = g = b + c$ and $e = h = bc$.
    - (g and h do not depend on a.)
  - $r = OP(t, e) = ITE(t, e, 0) = bc$.
- $\mathcal{C}_a(f) \ = \ bc$.

Extensions to BDDs

- *Complemented edges*
  - Reduce the size of ROBDDs.
  - Complement functions in constant time.
  - Restrictions on where the complemented edges can be placed to preserve *canonicity*.
    - Edge $\{v, high(v)\}$ not complemented.
- *Don't care leaf* to represent incompletely specified functions.

Advantages of ROBDDs

- Several algorithms for ROBDD manipulation.
  - Polynomial time.
- Most often the ROBDDS have small size.
- Software packages available.
  - Caches.
  - Garbage collection.

Variable ordering for ROBDDs

- The variable order afffets the ROBDD size.

- Problem:
  - Given a function f, find the variable order that minimizes the size.

- The optimum ordering problem is intractable.

- Exact algorithm with complexity $O(n^2 \cdot 3^n)$.

Heuristic static variable ordering

- Given a multilevel circuit.

- Order the variables according to circuit structure.

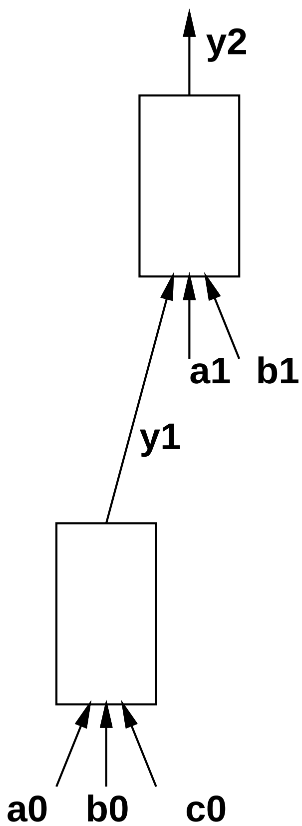
- Rationale:

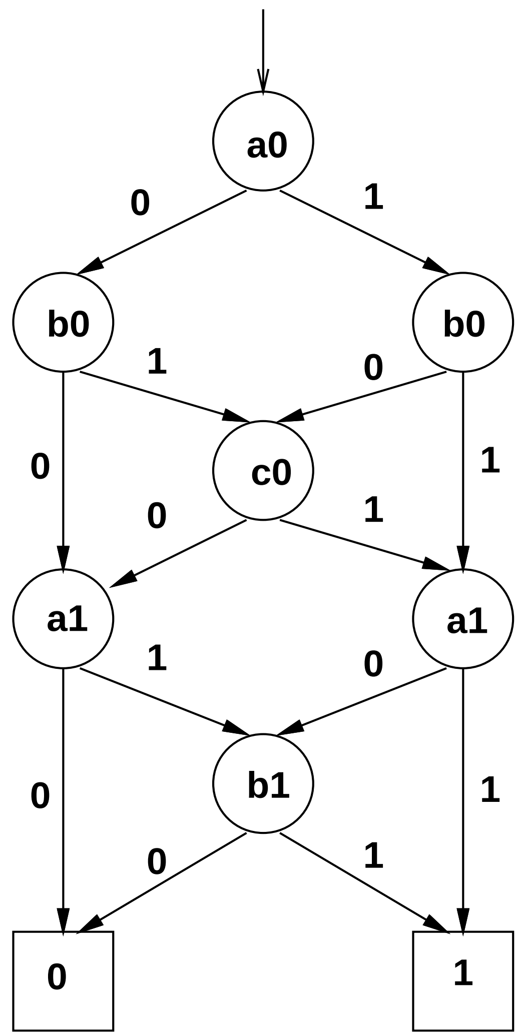
  – Variables that affect logic gates close to outputs should be at the bottom, because they affect only part of the function.

- Method:

  – Levelize variables by counting distance to output.

Example



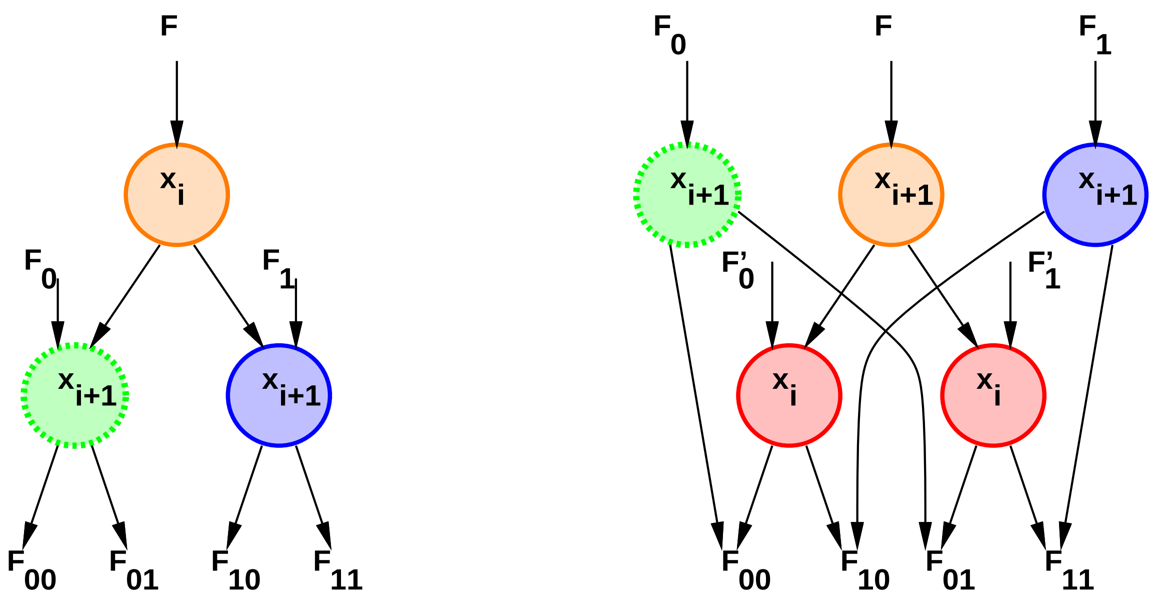


Dynamic variable reordering

- BDD sizes vary with variable ordering.
  - While manipulating logic functions, a chosen order may no longer be good.

- Software packages do variable reordering.
  - Principle: perform iterative swapping of adjacent variables.
  - Constraint: modify tables as little as possible.

Adjacent variable swapping



- $(x_i, F_1, F_0) = (x_{i+1}, (x_i, F_{11}, F_{01}), (x_i, F_{10}, F_{00}))$

Adjacent variable swapping

- The layers above and below the variables being swapped do not change.

- Two nodes are introduced
  - (May be present in unique table).

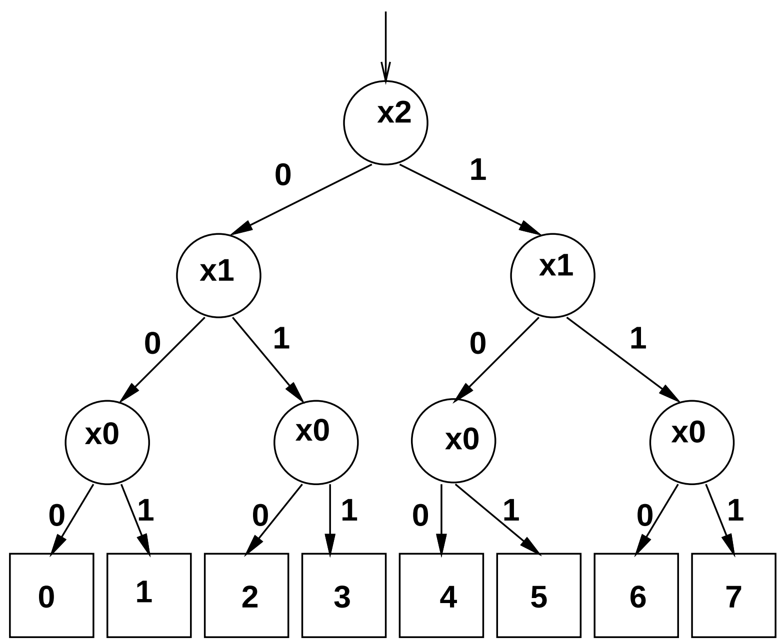
- *Sifting algorithm*.
  - Process one variable at a time.
  - Move variable to other positions in the order.
  - Repeat for all variables.

Other types of Decision Diagrams

- Decision diagrams based on other expansions:
  - OFDD - Ordered Functional Decision Diagrams
  - Based on Reed-Muller expansion:
    - $f = f_{x'} \oplus x(f_{x'} \oplus f_x)$
- Decision diagrams for discrete functions.
  - Binary inputs, outputs in finite set.
  - Examples:
    - ADD - Algebraic Decision Diagrams.
    - BMD - Binary Moment Diagrams.
- Different types of reduction rules.

Algebraic Decision Diagrams (ADDs)

- Multi-terminal ROBDDs.
- Finite number of leaves with different values.
- Good to represent discrete functions.
- Example:

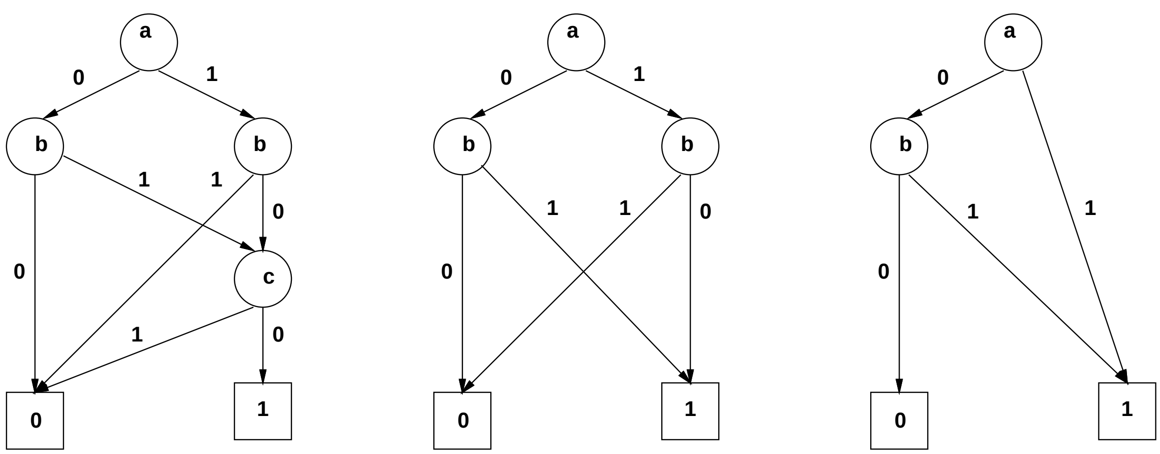


Zero-suppressed BDDs (ZBDDs)

- BDDs with different reduction rules:
  - Eliminate all nodes whose 1-edge points to the 0-leaf and redirect incoming edges to the 0-subgraph.
  - Share all equivalent subgraphs.
  - Good for representatiing ensembles of subsets.
- Rationale:
  - Most ensembles of subsets are *sparse*, i.e., subsets have few elements.

Example

f= a b' c' + a' b c' 100 + 010



BDD MODIFIED BDD ZDD

Summary

- Binary Decision Diagrams:
  - Used mainly in multi-level logic optimization.
  - Very efficient data-structure.
- Several flavors of decision diagrams address various needs.
- Efficient Boolean manipulation exploits cofactor expansion and recursion.