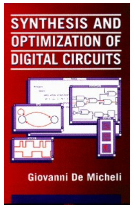
Binary Decision Diagrams

Giovanni De Micheli
Integrated Systems Centre
EPF Lausanne



This presentation can be used for non-commercial purposes as long as this note and the copyright footers are not removed

© Giovanni De Micheli – All rights reserved

Module 1

- **Objectives:**
  - **Definitions of BDDs, OBDDs and ROBDDs**
  - **Logic operations on BDDs**
  - **The ITE operator**

Why ?

- Efficient way to represent logic functions
- History
  - Original idea for BDD due to Lee (1959) and Akers (1978)
  - Refined and popularized by Bryant (1986)
    - Smaller structure
    - Canonical form – each distinct function correspond to a unique distinct diagram

Canonical forms - review

- **Each logic function has a unique representation**

- **Truth table**

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	0
1	1	1	1

- **Sum of minterms**

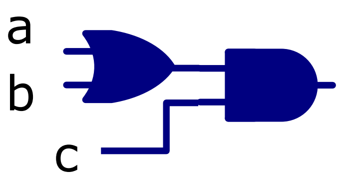
a'bc+ab'c+abc

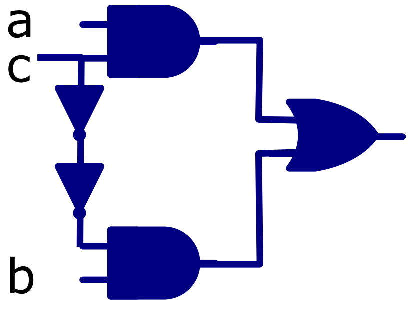
Non canonical forms - review

- Each function has also multiple representations
- Factored form

  (a+b)c ac+bc

- Logic gate representation





Terminology

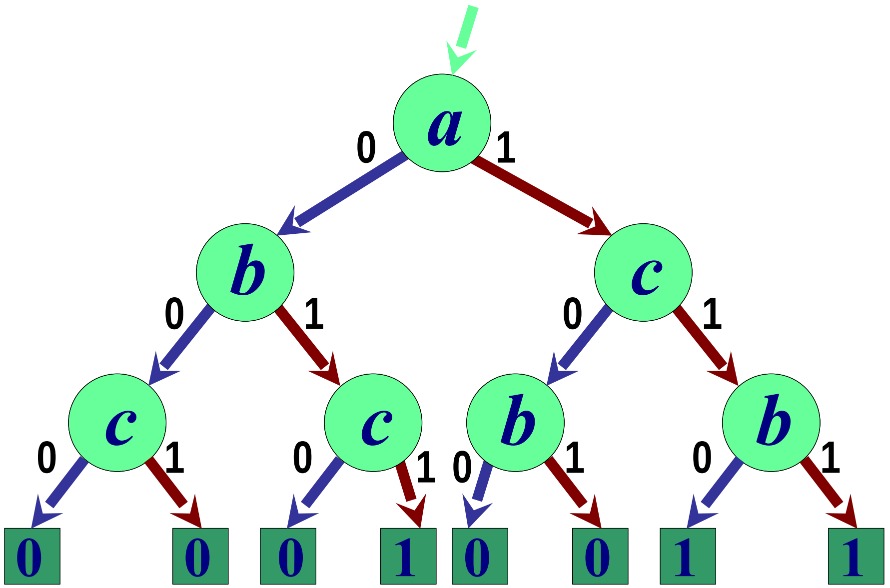
- A Binary Decision Diagram (BDD) is a *directed acyclic graph*
  - Graph: set of vertices connected byedges
  - Directed: edges have direction
  - Acyclic: no path in the graph can lead to a cycle

  - Often abbreviated as DAG

BDD - Example

- *F = (a + b) c*

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	0
1	1	1	1



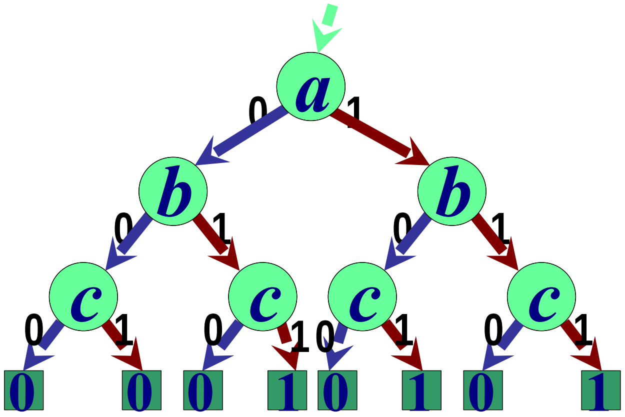
1. Each vertex represents a decision on a variable
2. The value of the function is found at the leaves
3. Each path from root to leaf corresponds to a row in the truth table

BDD - observations

- **The size of a BDD is as big as a truth table:**
  - **1 leaf per row**

- **Each path from root to leaf evaluates variables in some order**

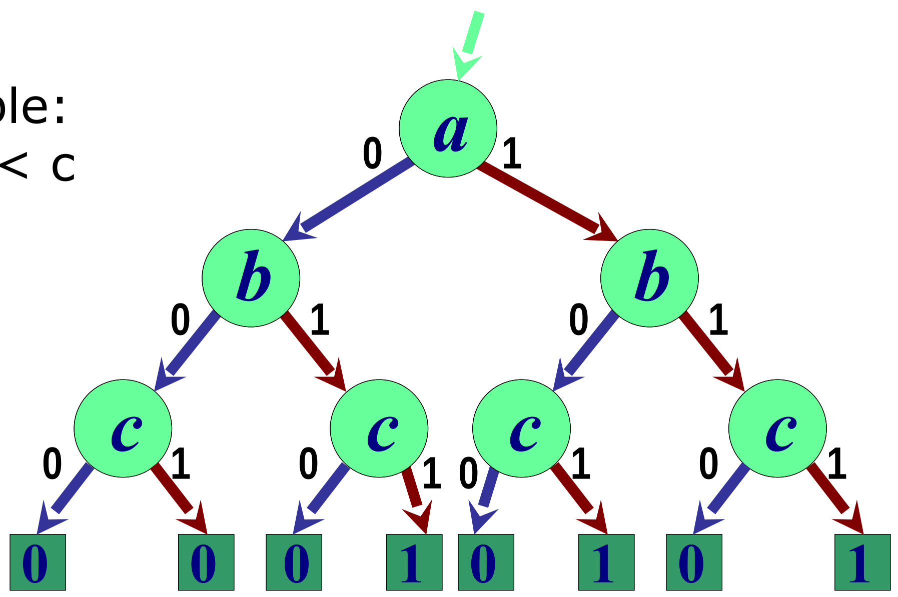
  - but the order is not fixed:



1st idea: Ordered BDD (OBDD)

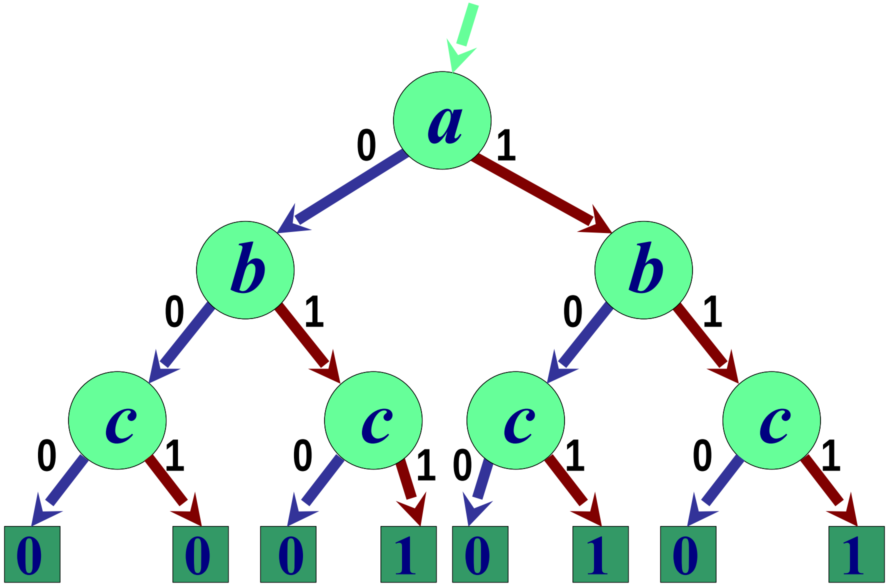
- Choose arbitrary total ordering on the variables
- Variables must appear in the same order along each path from root to leaves
- Each variable can appear at most once on a path

example:
a < b < c

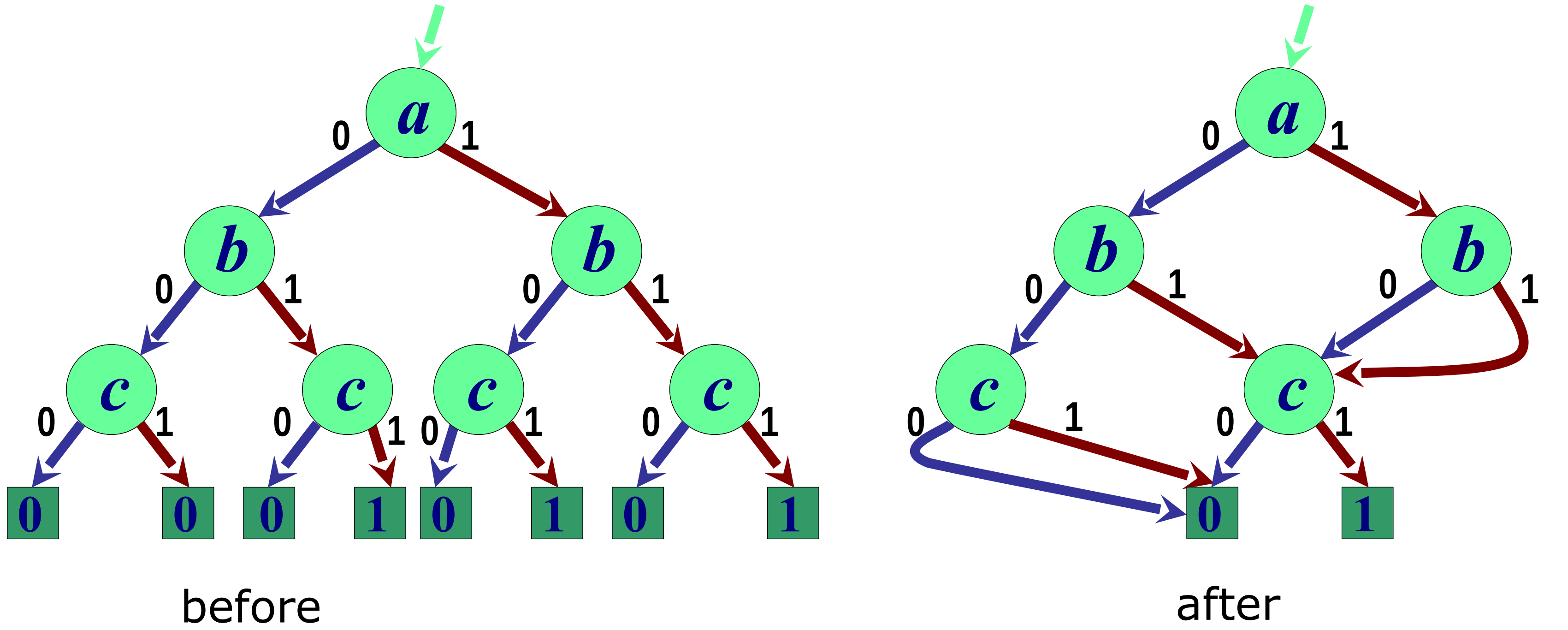


2nd idea: Reduced OBDD (ROBDD)

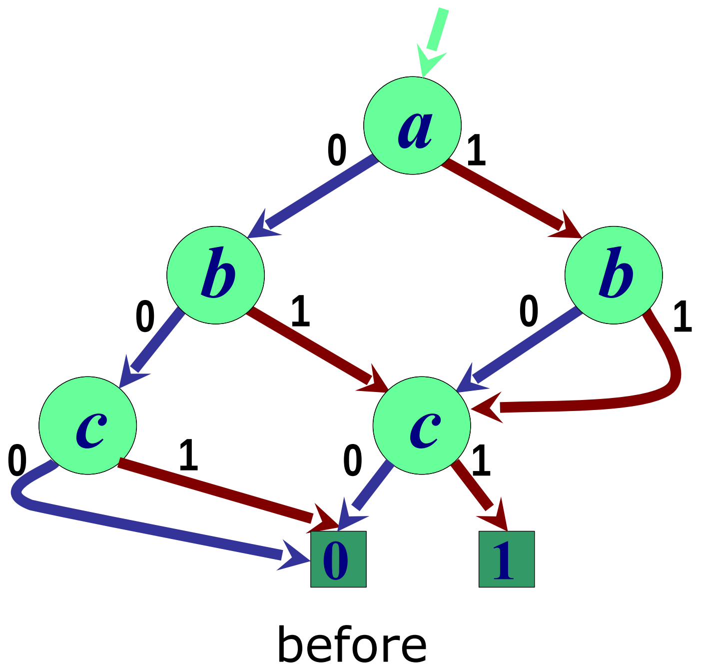
- **Two reduction rules:**
  1. Merge equivalent sub-trees
  2. Remove nodes with identical children



1. Merge equivalent sub-trees



2. Remove node with identical children

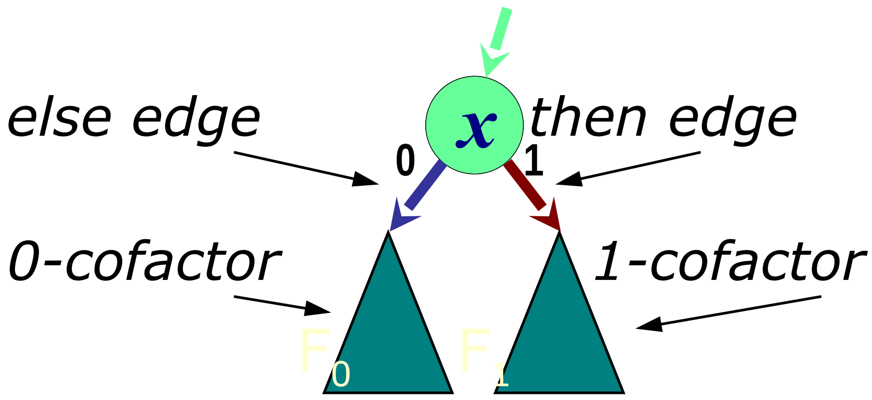


BDD semantics

Constant nodes

0 1

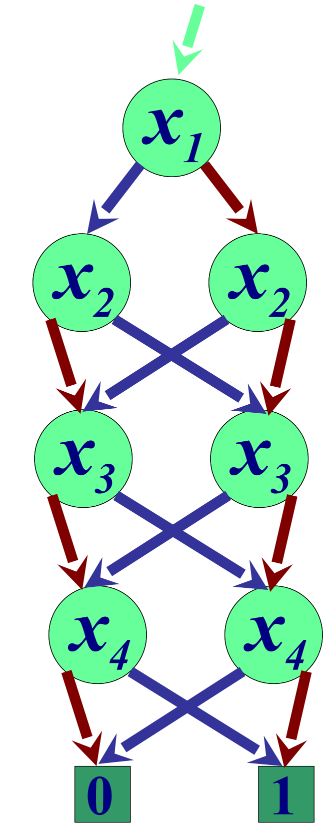
ITE(x,F1,F0)



Cofactor(F,x): the function you obtain when you substitute 1 for x in F

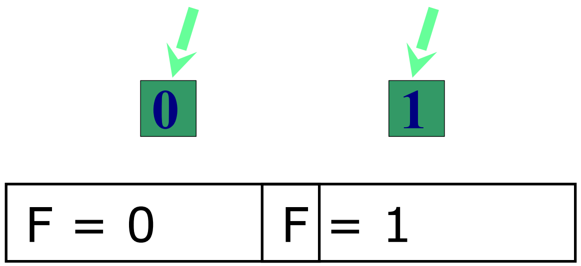
ROBDDs

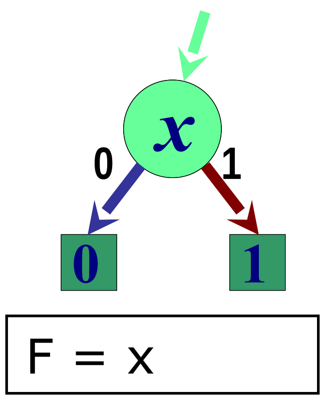
- ROBDDs are canonical
  - for a given variable order
- ROBDD are more compact than other canonical forms
- ROBDD size depends on the variable order
  - many useful function have linear-space representation

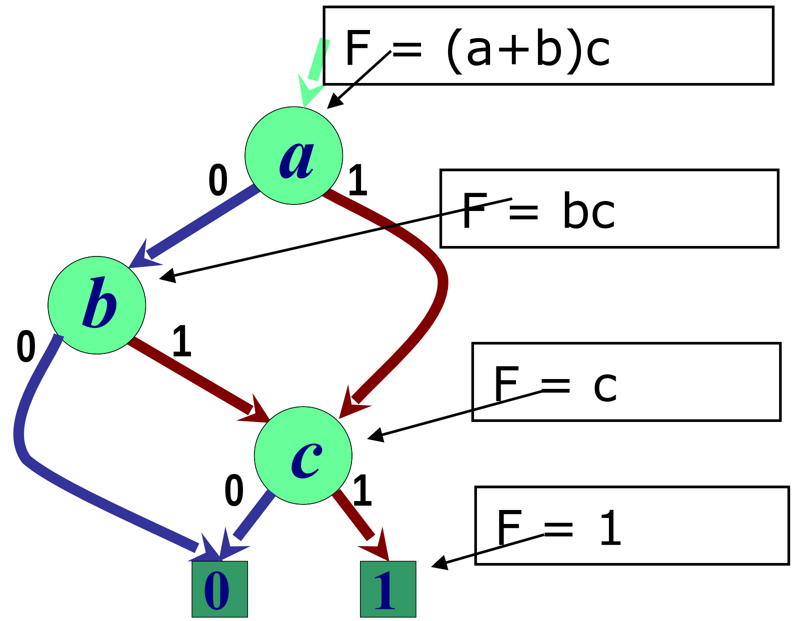


$$F = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$

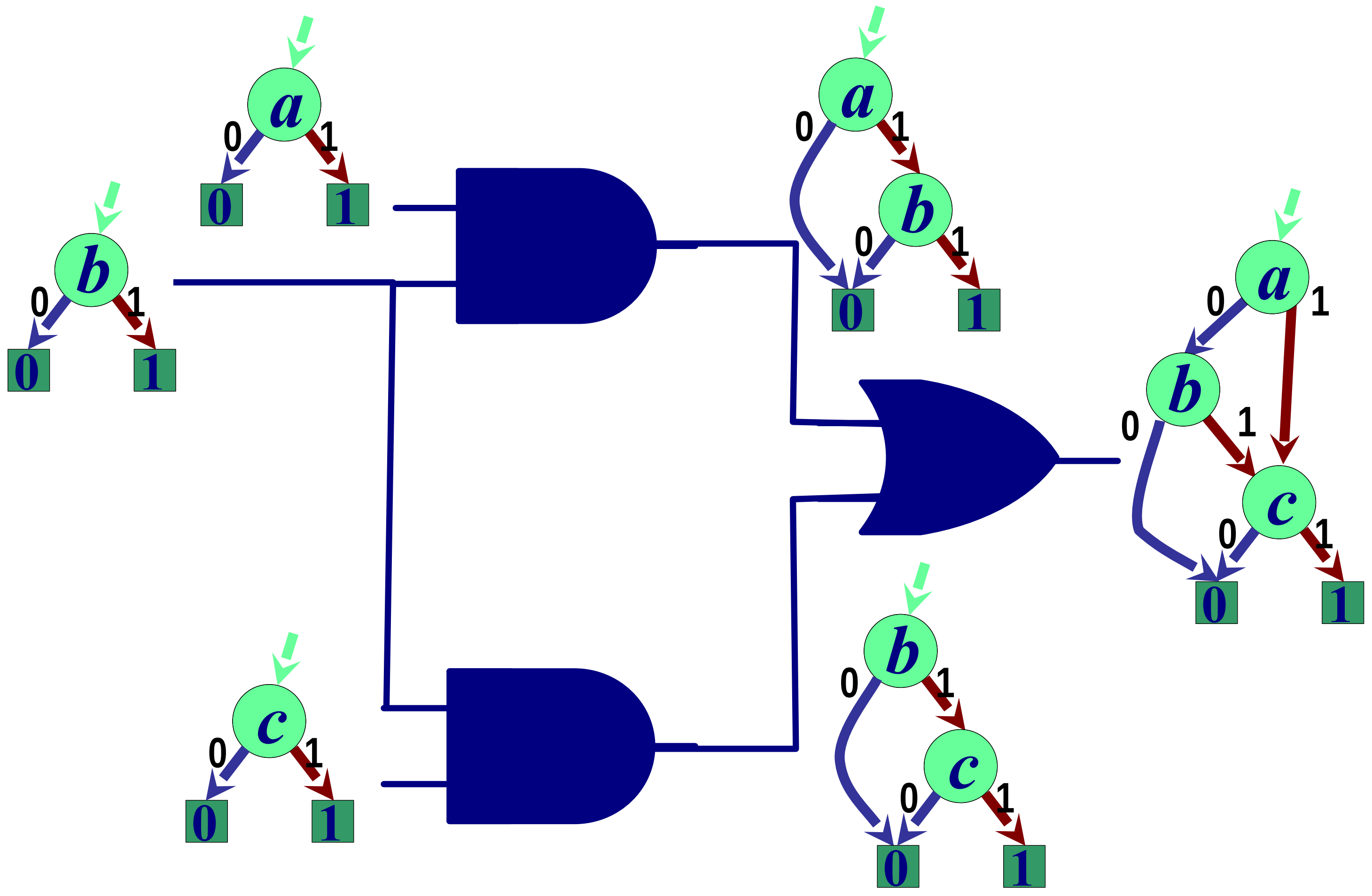
A few simple functions







A network example



ROBDDs- why do we care ?

- **Easy to solve some important problems:**
  1. **Tautology checking**

     just check if BDD is identical to function $\boxed{1}$

  2. **Identity checking**

  3. **Satisfiability**

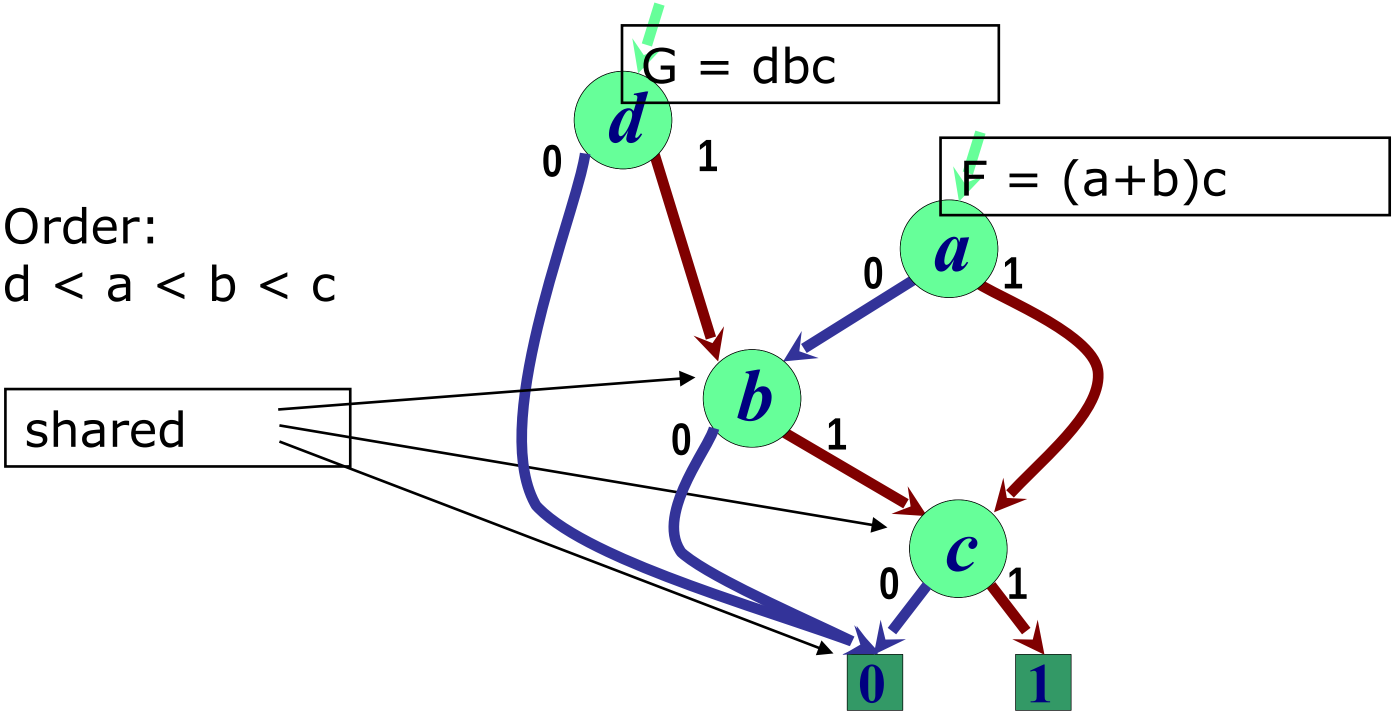
     look for a path from root to leaf

- **All while having a compact representation**
  - **Use small memory footprint**

ROBDD- sharing

We already share subtrees within a ROBDD

…but we can share also among multiple ROBDDS



Logic operations with ROBDDs

- *Problem*: given two functions G and H, represented by their ROBDDs, compute the ROBDD of a function of (G,H)
- ite operator:
  - ite(f,g,h)
  - If (f) then (g) else (h)
- Recursive paradigm
  - exploit the generalized expansion of G and H

$$\text{ite}(f,g,h) = \text{ite}(x,\text{ite}(x,g_x,h_x),\text{ite}(x',g_{x'},h_{x'}))$$

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

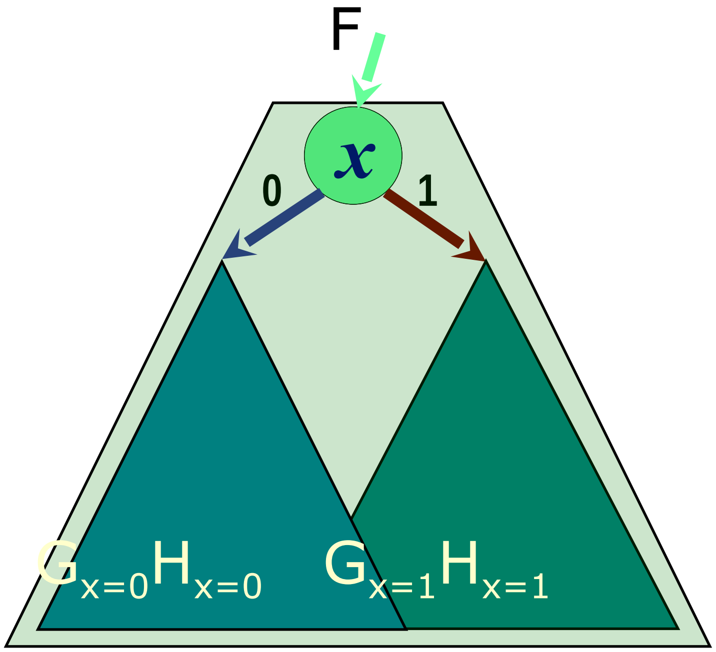
Example

- Compute AND of two ROBDDs
- Terminal cases:
  - AND (0,H) = 0
  - AND (1,H) = H
  - AND (G,0) = 0
  - AND (G,1) = G

Recursive step

- $G(x,\ldots) = x' G_{x=0} + x\, G_{x=1}$
- $H(x,\ldots) = x' H_{x=0} + x\, H_{x=1}$
- $F = GH = x' G_{x=0} H_{x=0} + x\, G_{x=1} H_{x=1}$

Now we have reduced the problem to computing 2 ANDs of smaller functions



One last problem

- Suppose, we have computed
  $G_{x=0}\, H_{x=0}$ and $G_{x=1} H_{x=1}$

- We need to construct a new node,
  - label: x
  - 0-cofactor($F_{x=0}$): ROBDD of $G_{x=0}\, H_{x=0}$
  - 1-cofactor($F_{x=1}$): ROBDD of $G_{x=1}\, H_{x=1}$

- BUT, first we need to make that we don’t violate the reduction rules!

The unique table

To obey reduction rule #1:

- if $F_{x=0} == F_{x=1}$, the result if just $F_{x=0}$

To obey reduction rule #2:

- We keep a *unique table* of all the BDD nodes and check first if there is already a node

  $(x, F_{x=0}, F_{x=1})$

Otherwise, we build the new node

- and add it to the unique table

Putting all together

```
AND(G,H) {
        if (G==0) || (H==0) return 0;
        if (G==1) return H;
        if (H==1) return G;
        cmp = computed_table_lookup(G,H);
        if (cmp != NULL) return cmp;

        x = top_variable(G,H);
        G1 = G.then; H1 = H.then;
        G0 = G.else; H0 = H.else;
        F0 = AND(G0,H0);
        F1 = AND(G1,H1);
        if (F0 == F1) return F0;
        F = find_or_add_unique_table(x,F0,F1);
        computed_table_insert(G,H,F);
        return F;
}
```

Logic operations - summary

- Recursive routines – traverse the DAGs depth first
- Two tables:
  - Unique table – hash table with and entry for each BDD node
  - Computed table – store previously computed partial results
- To perform other operations, just change the terminal cases

Some algorithmic complexities

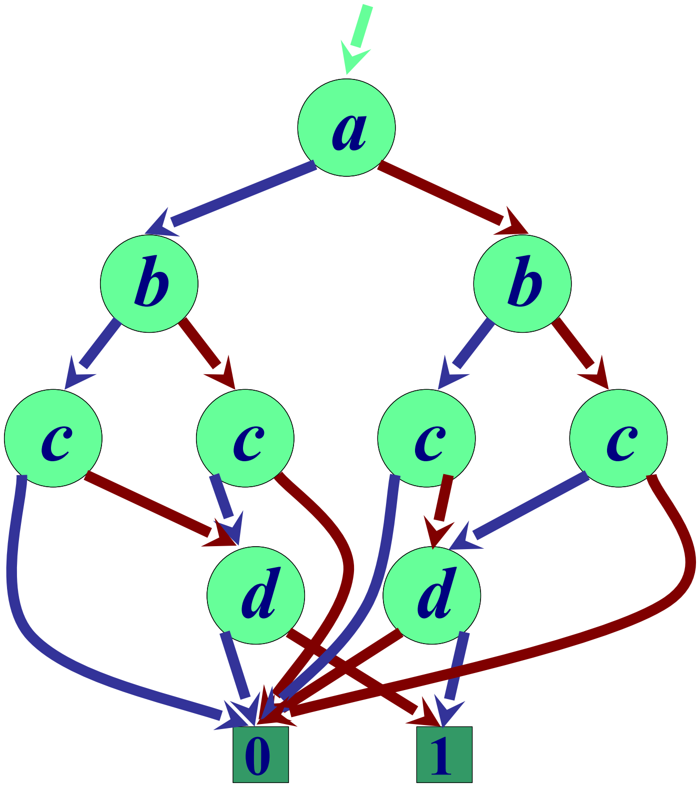
Checking identity	K time
Checking tautology	K time
Satisfiability	linear (#vars)
Binary operators: AND, OR	quadratic
Smoothing, Consensus	quadratic

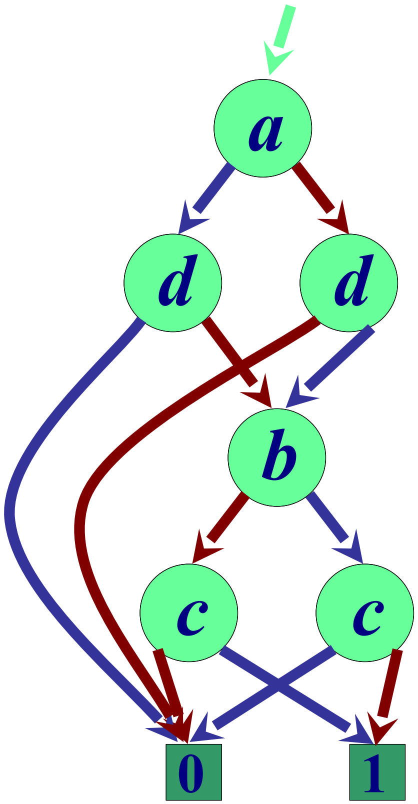
Module 2

- **Objectives:**
  - **Variable ordering (static and dynamic)**
  - **Other diagrams and applications**

The importance of variable order

$$F = (a \oplus d)(b \oplus c)$$





Ordering results

Function type	*Best order*	*Worst order*
addition	**linear**	**exponential**
symmetric	**linear**	**quadratic**
multiplication	**exponential**	**exponential**

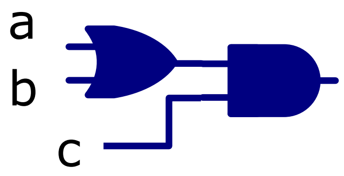
- **In practice:**
  - **Many common functions have reasonable size**
  - **Can build ROBDDs with millions of nodes**
  - **Algorithms to find good variables ordering**

Variable ordering algorithms

- *Problem*: given a function F, find the variable order that minimizes the size of its ROBBDs
- *Answer*: problem is intractable
- Two heuristics
  - Static variable ordering (1988)
  - Dynamic variable ordering (1993)

Static variable ordering

- **Variables are ordered based on the network topology**
  - ***How*: put at the bottom the variables that are closer to circuit's outputs**
  - ***Why*: because those variables only affect a small part of the circuit**



good order: a < b < c

  - ***Disclaimer*: it's a heuristic, results are not guaranteed**

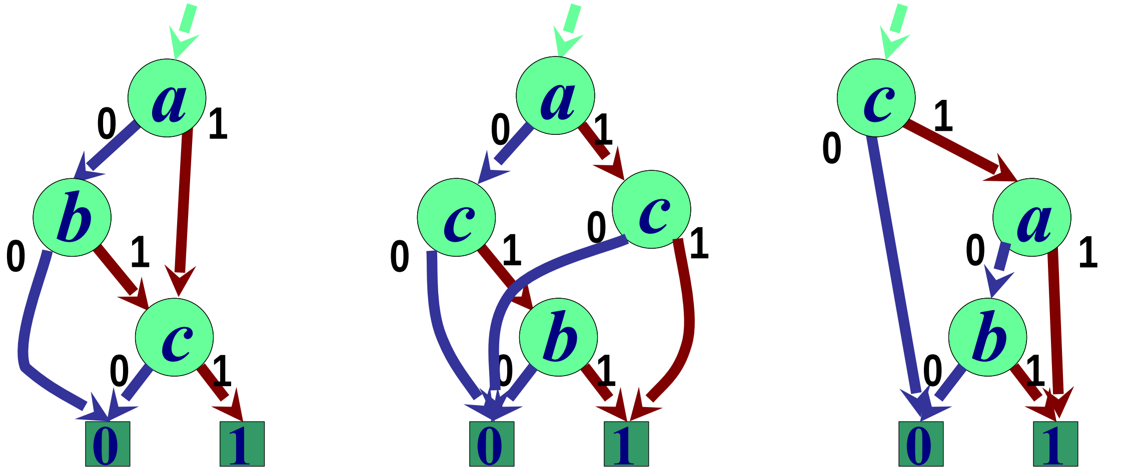
Dynamic variable ordering

- Changes the variable order on the fly whenever ROBDDs become too big
- *How*: trial and error – SIFTING ALGORITHM
  1. Choose a variable
  2. Move it in all possible positions of the variable order
  3. Pick the position that leaves you with the smallest ROBDDs
  4. Choose another variable …

Dynamic variable ordering

- **Tiny example: F=(a+b)c**

  we want to find the optimal position for variable c



Final order:
c<a<b

initial order:
a < b < c

Swap (b, c):
a < c < b

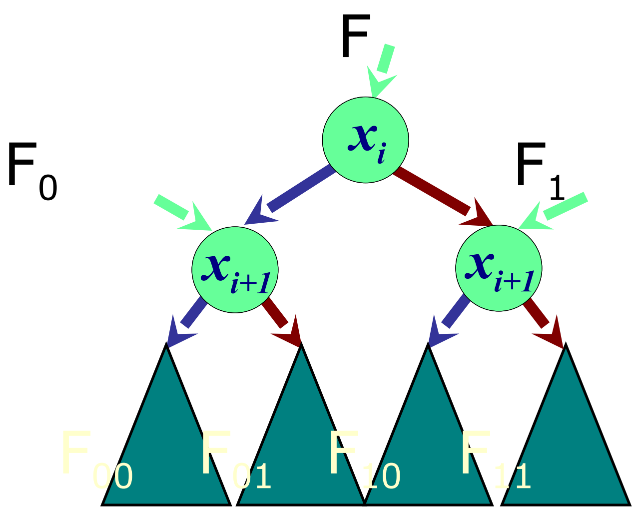
Swap (a, c):
c < a < b

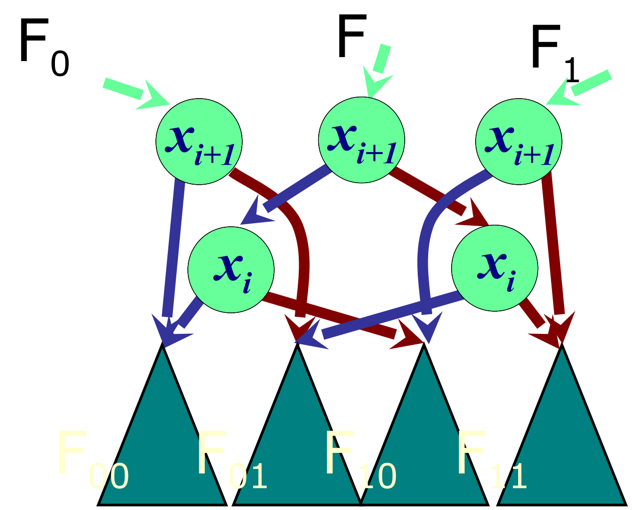
Variable swapping

$$ITE(x_i, F_1, F_0) =$$

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

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





Dynamic variable ordering

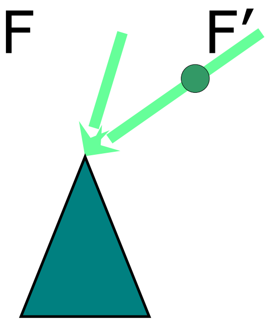
- ***Key idea*: swapping two variables can be done locally**
  - ***Efficient:***
    - **can be done just by sweeping the unique table**
  - ***Robust*:**
    - **works well on many more circuits**
  - ***Warning*:**
    - **It's still non optimal.**
    - **At convergence, you most probably have found only a local minimum.**

Improvements of BDDs

- **Complement edges** (1990)
  - Creates more opportunities for sharing

    -> fewer nodes
  - For every pair (F,F'), we
    - only construct the ROBDD for F
    - F' is given by using a complement edge to F
  - Which do you pick ?
    - THEN edge can never be complemented
    - Only constant value

1

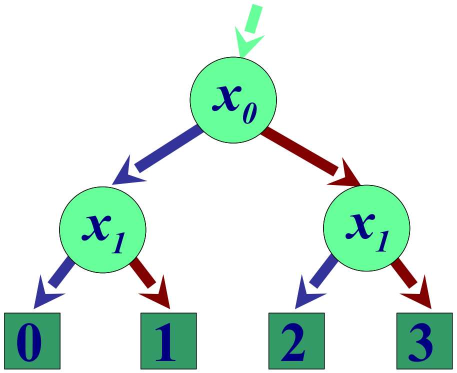


Other types of Decision Diagram

- **Based on different expansion**
  - **OFDD**
  - **Ordered functional decision diagrams**

$$\Phi = \Phi_{\xi=0} \oplus \xi(\Phi_{\xi=0} \oplus \Phi_{\xi=1})$$

- **For discrete functions:**
  - **ADD**
  - **Algebraic decision diagrams**

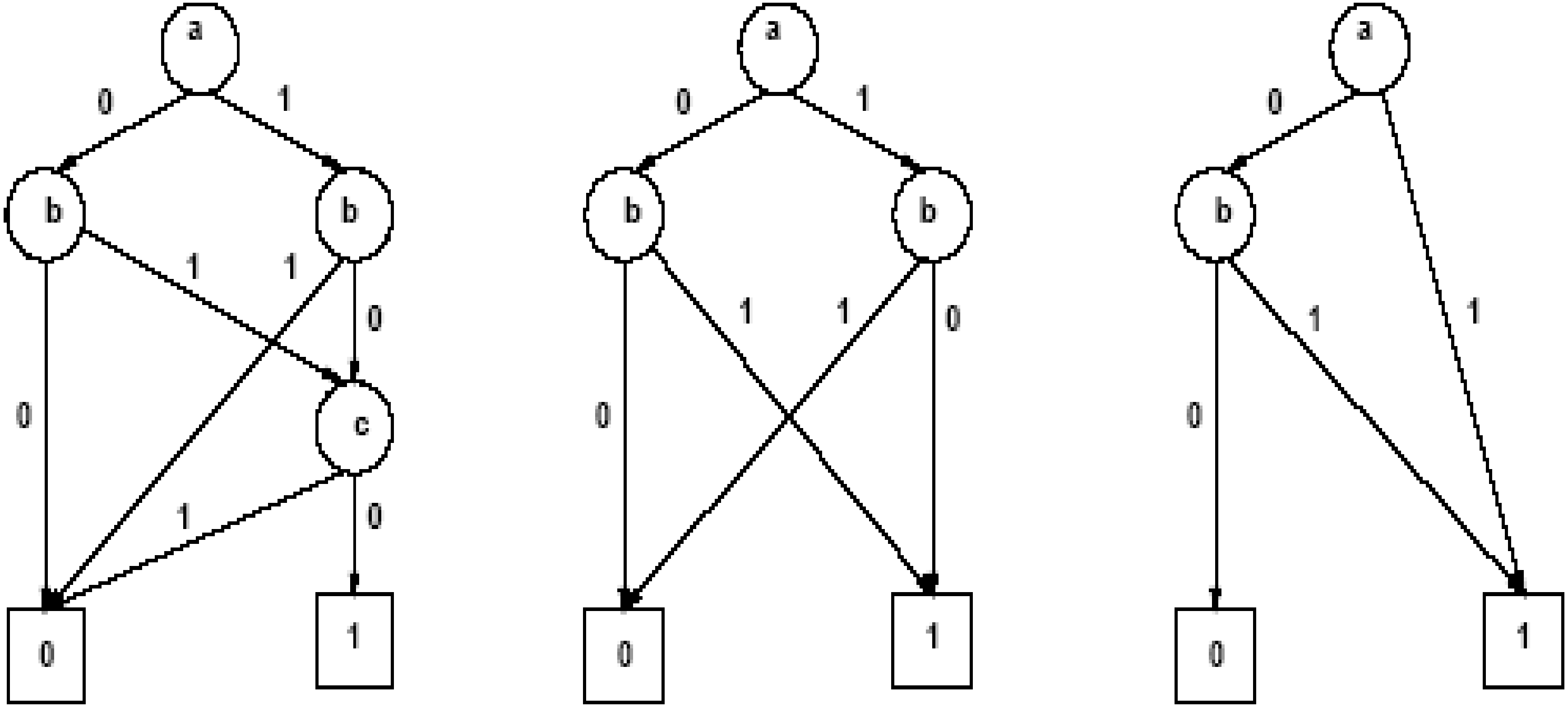


ZDDs -- Zero-suppressed BDDs

- 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
- Applicability
  - Good for representing ensembles of subsets
  - Most ensembles are very sparse: i.e., subsets have few elements

Example

f = ab'c' + a'bc' 100 + 010



BDD MODIFIED BDD ZDD

Summary

- BDDs
  - + Very efficient data structure
  - + Efficient manipulation routines
  - – A few important functions don't come out well
  - – Variable order can have a high impact on size
- Application in many areas of CAD
  - Hardware verification
  - Logic synthesis