CAD for VLSI

Boolean Function

Outline

- Binary system representations
- Definitions of BDDs, OBDDs and ROBDDs
- Logic operations on BDDs
- The ITE operator
- Variable ordering (static and dynamic)

Basic Definitions

- Let B = {0,1} Y = {0,1,2}
  - A logic function f in n inputs $x_1, x_2, \ldots x_n$ and m outputs $y_1, y_2, \ldots y_m$ is a function

$$f : B^n \longrightarrow Y^m$$

$$X = [x_1, x_2, \ldots, x_n] \in B^n \quad \text{is the input}$$

$$Y = [y_1, y_2, \ldots, y_m] \in Y^m \quad \text{is the output}$$

  - m=1 → a single output function
  - m>1 → a multiple output function

Basic Definitions

- For each component f_i, i = 1,2, ...,m, define
  - ON_SET: set of input values x such that $f_i(x) =1$
  - OFF_SET: set of input values x such that $f_i(x) = 0$
  - DC_SET: set of input values x such that $f_i(x) = 2$

- Completely specified function: DC_SET = ϕ, $\forall$ f_i
- Incompletely specified function: DC_SET $\neq$ ϕ , for some f_i

Boolean Representations

- **Truth table representation**

Full adder

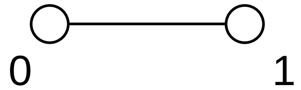
X	Y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- A multiple output function
- Sum
  - on-set = {(0 0 1), (0 1 0), (1 0 0), (1 1 1)}
  - off-set = {(0 0 0), (0 1 1), (1 0 1), (1 1 0)}
- A completely specified function

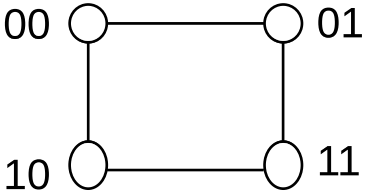
Boolean Representations

- Geometrical representation

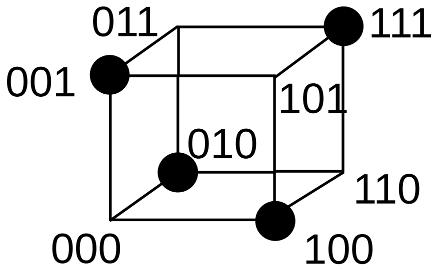
1 variable



2 variables



3 variables



sum : on-set ={(0 0 1),(0 1 0),(1 0 0),(1 1 1)}

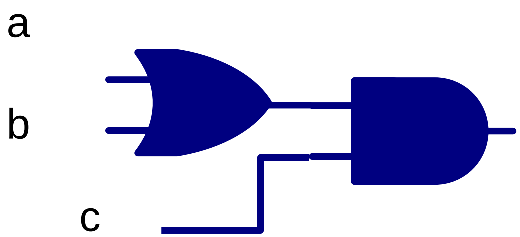
off-set ={(0 1 1),(1 0 1),(1 1 0),(0 0 0)}

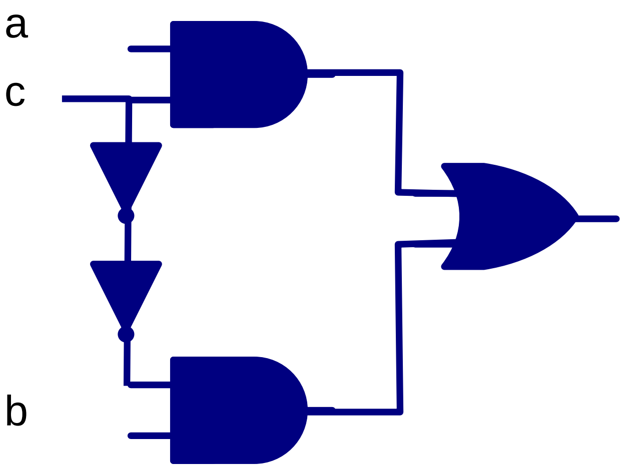
Boolean Representations

- Algebraic representations
  - Canonical sum of minterms
    - $C_{out} = x'yC_{in} + xy'C_{in} + xyC_{in}' + xyC_{in}$
  - Reduced sum of products
    - $C_{out} = yC_{in} + xC_{in} + xy$
    - $C_{out} = yC_{in} + xC_{in} + xyC_{in}'$
  - Multi-level representation
    - $C_{out} = C_{in}(x + y) + xy$

Boolean Representations

- Logic gate representations





Boolean Representations

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

  - Often abbreviated as DAG

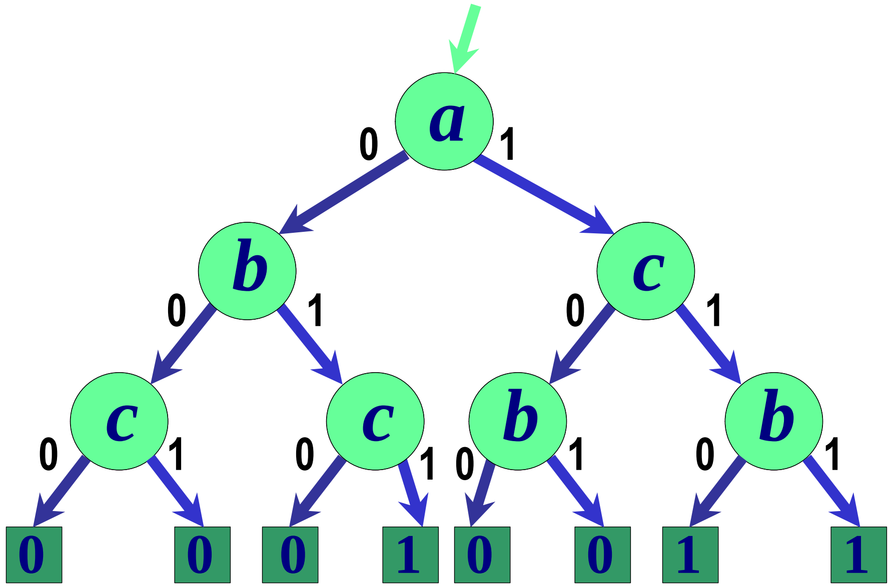
Binary Decision Diagram (BDD)

- A BDD graph which has a vertex v as root corresponds to the function F_v:
  - If v is a terminal node:
    - if value (v) is 1, then $F_v = 1$
    - if value (v) is 0, then $F_v = 0$
  - If F is a non-terminal node (with index(v) = i)
    - $F_v (x_i, \ldots x_n) = x_i' F_{low(v)} (x_{i+1}, \ldots x_n) + x_i F_{high(v)} (x_{i+1}, \ldots x_n)$

BDD Example

- F = (a + b) c

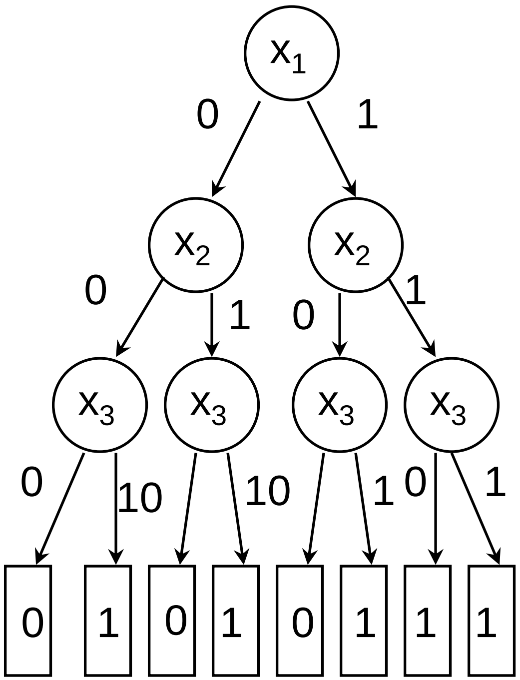
a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



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 Example

- Binary Decision Diagram (BDD)
- $f = x_1x_2 + x_3$

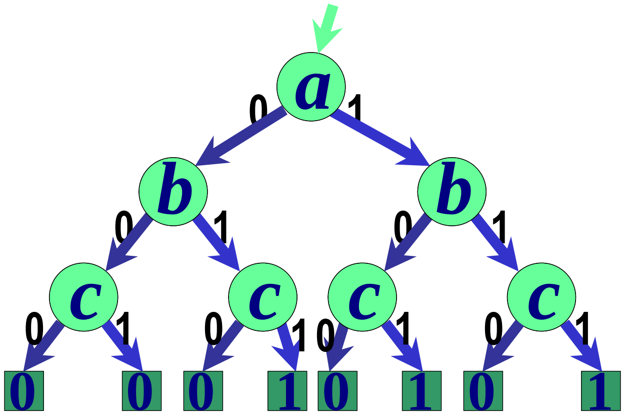


- Terminal node:
  - Attribute
    - value (v) = 0
    - value (v) = 1
- Non-terminal node:
  - index (v) = i
    - Two children nodes
    - low (v)
    - high (v)
- Evaluate an input vector

BDD – Observations

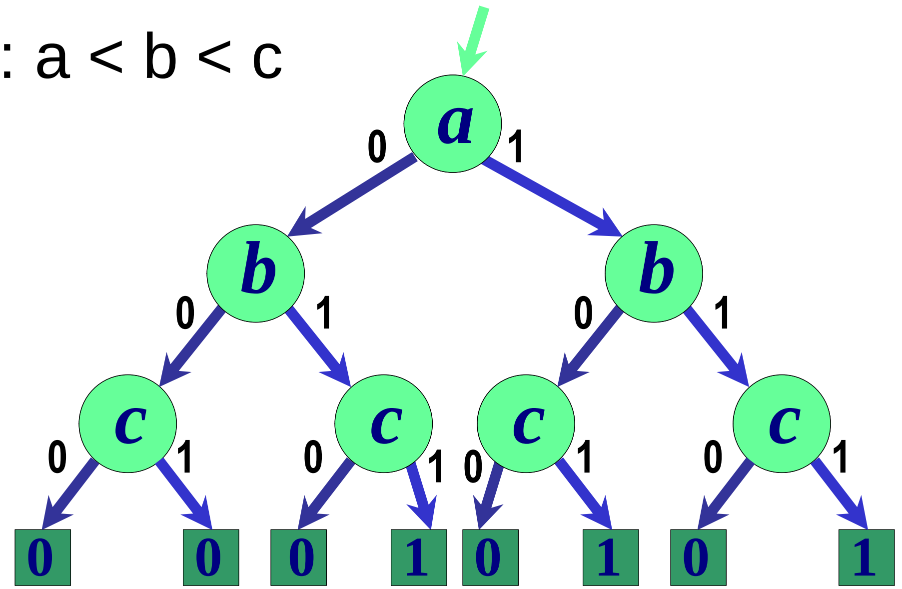
- The size of a BDD is as big as a truth table:
  - – 1 leaf per row

- Each path from root to leaf evaluates variables in some order
  - - but the order is not fixed:



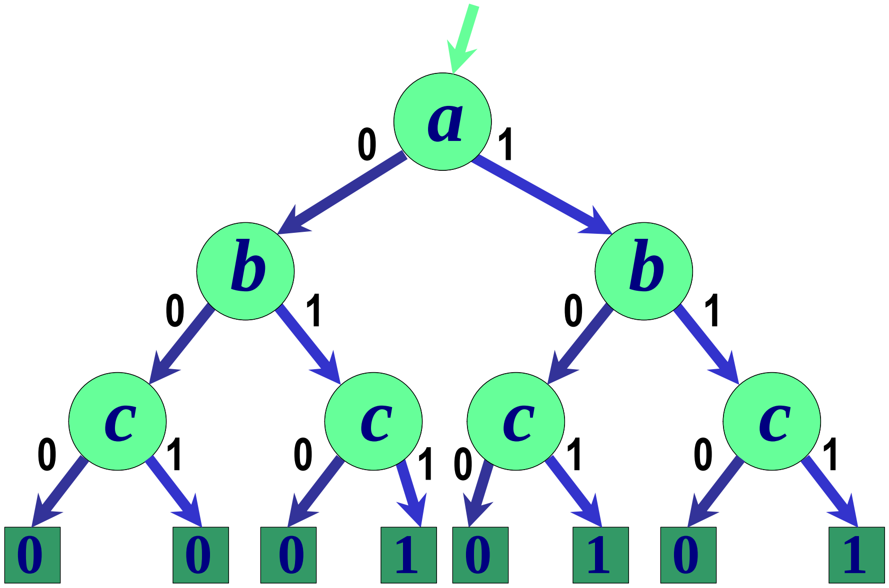
1st idea: Ordered BDD (OBDD)

- 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



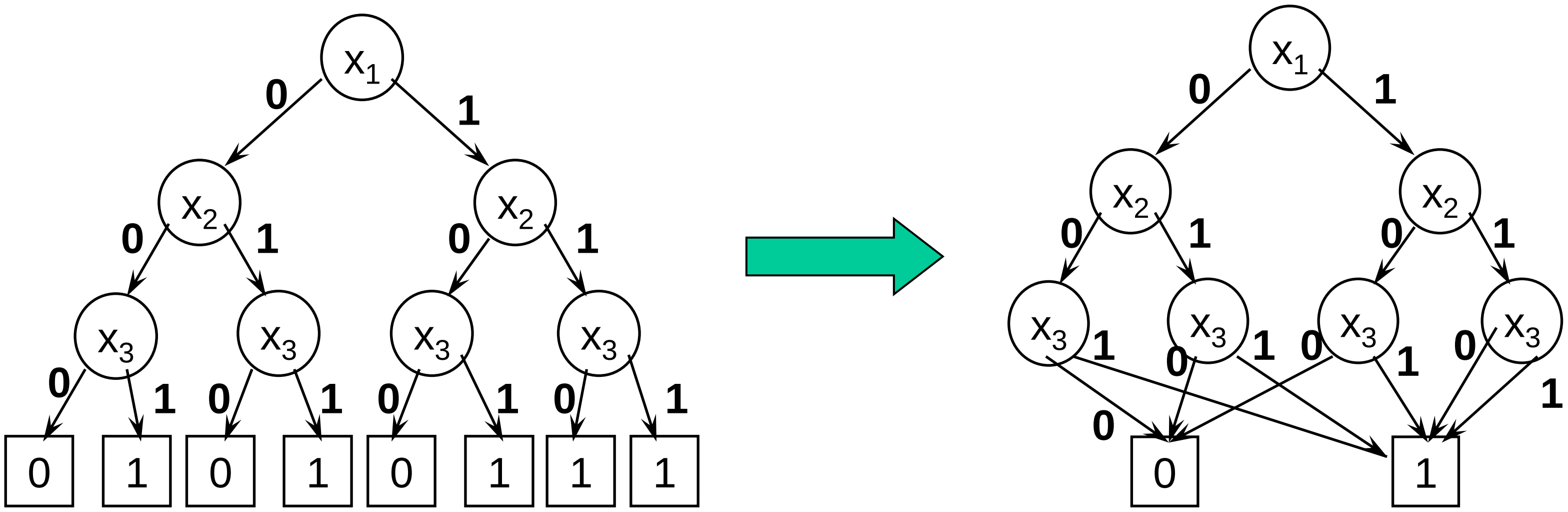
2nd idea: Reduced OBDD (ROBDD)

- Reduced OBDD:
  - No distinct vertices v and v' such that subgraphs rooted by v and v' are isomorphism
  - No vertex v with low (v) = high (v)



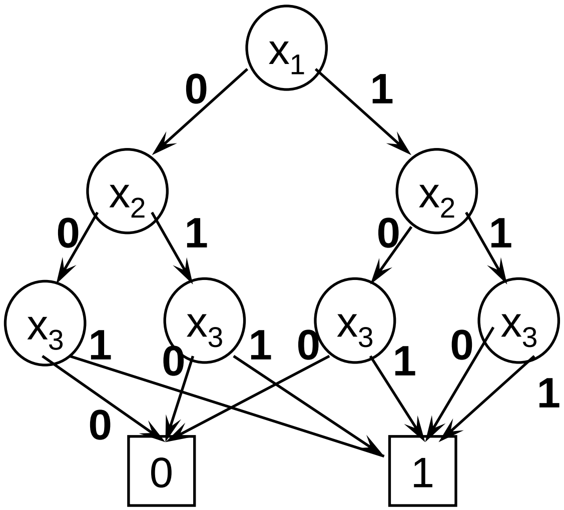
ROBDD Example

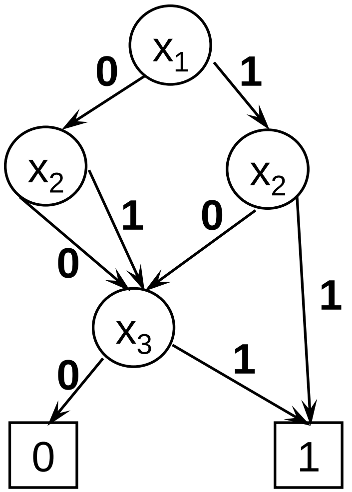
- $F = x_1x_2+x_3$



ROBDD Example

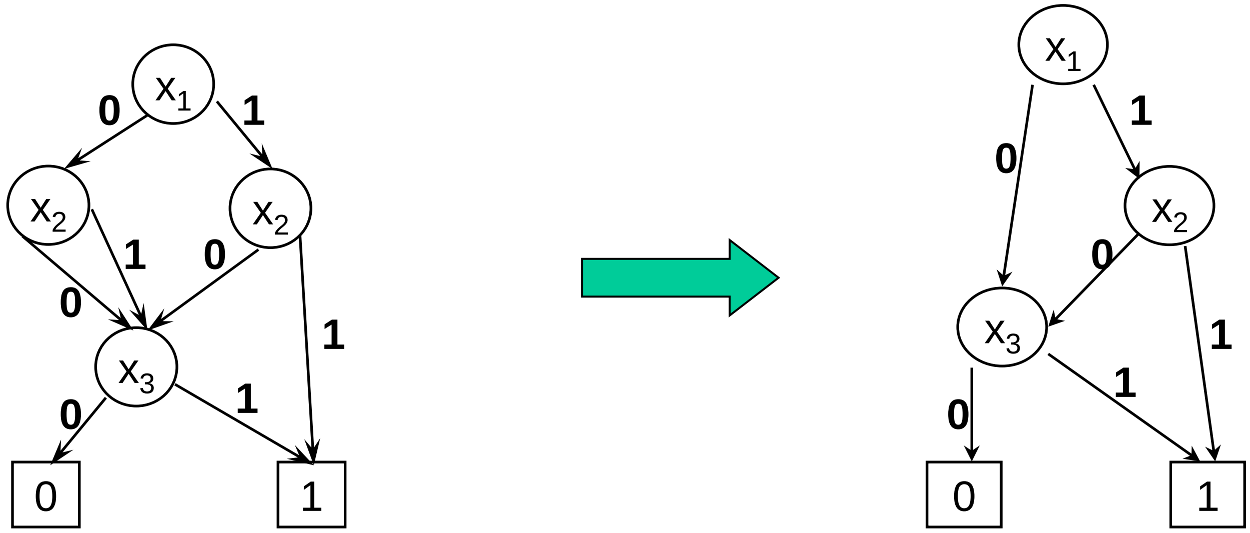
- $F = x_1x_2+x_3$





ROBDD Example

- $F = x_1x_2+x_3$

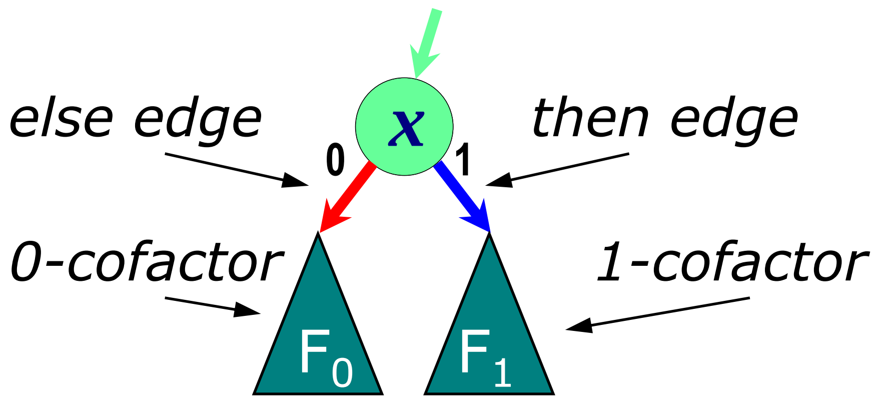


BDD Semantics

Constant nodes

0 **1**

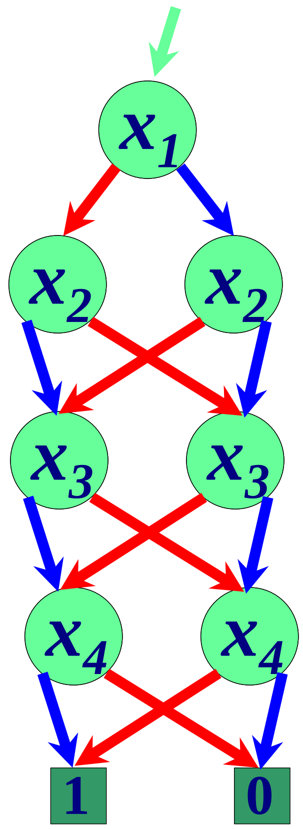
ITE(x,F1,F0)



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

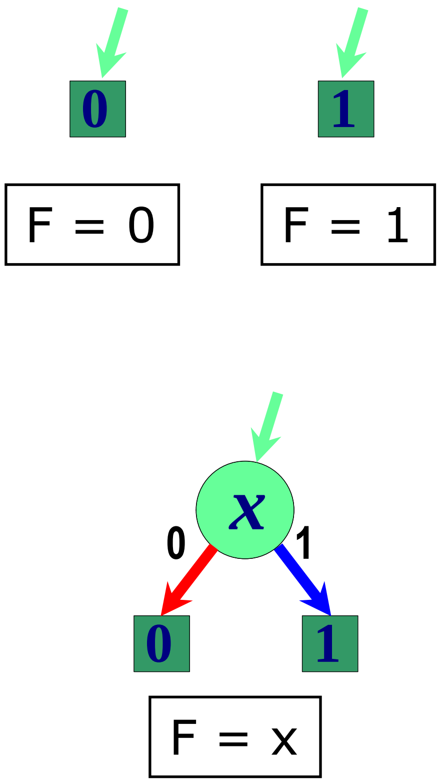
ROBDD Property

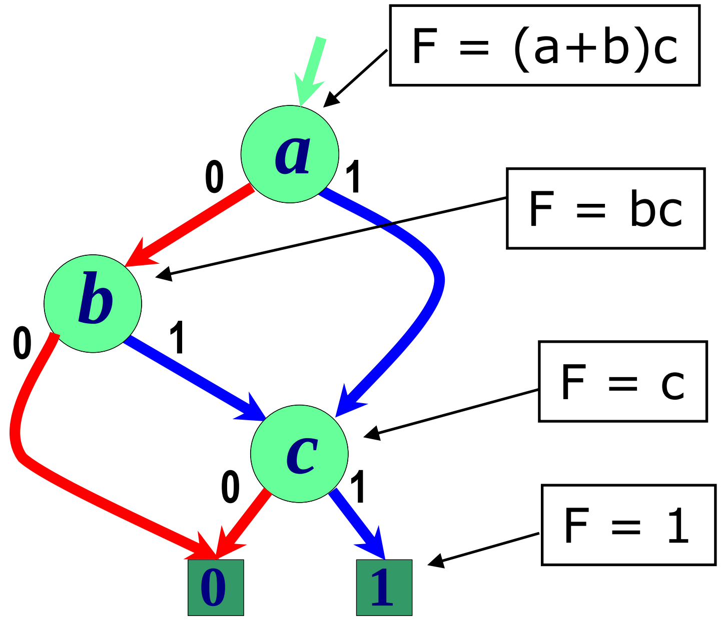
- ROBDDs are canonical
  - For a given variable order
- ROBDDs are more compact than other canonical forms
- ROBDDs size depend on the variable order
  - many useful functions have linear-space representations



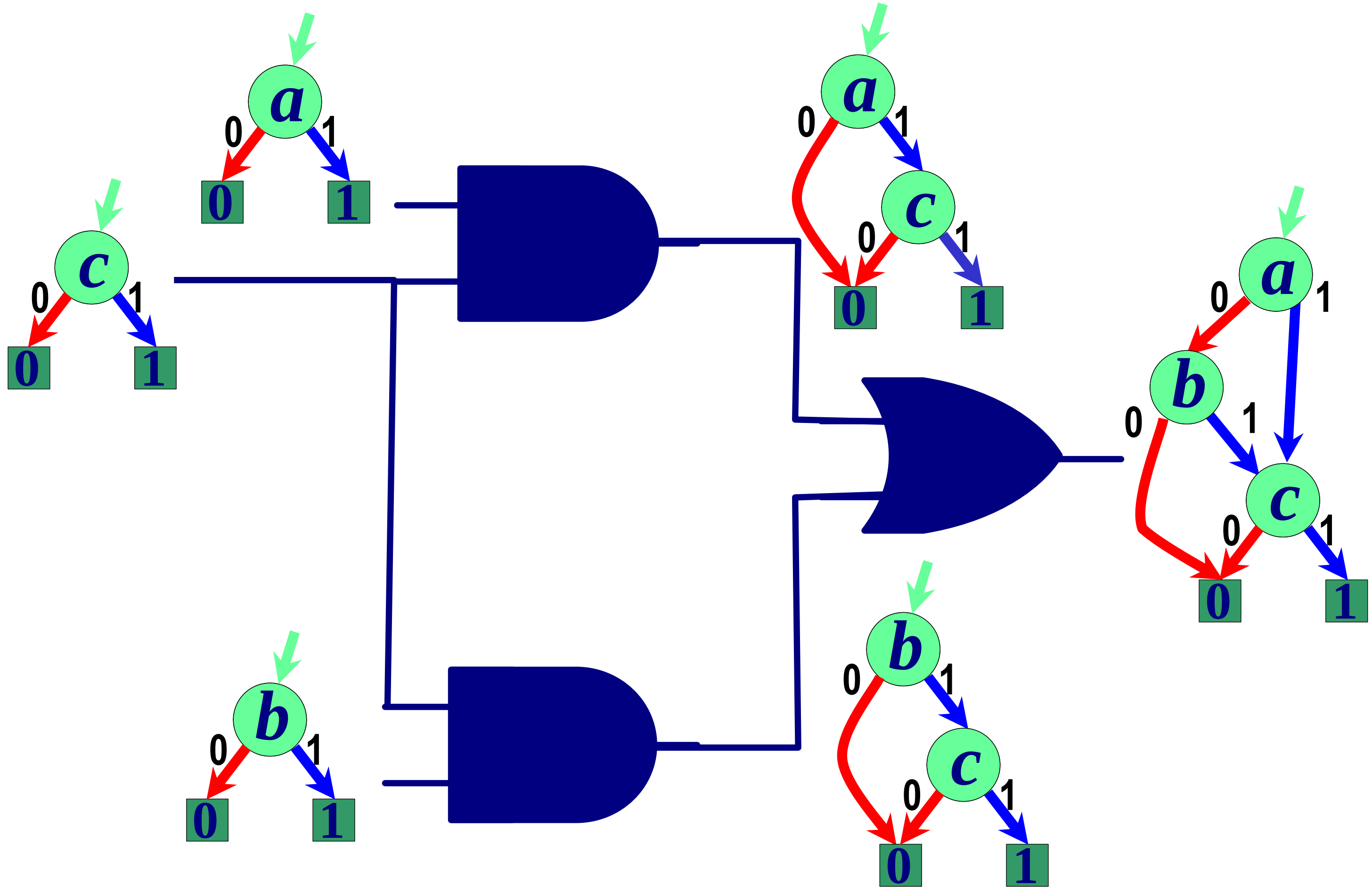
$$F = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$

A Few Simple Functions





A Network Example



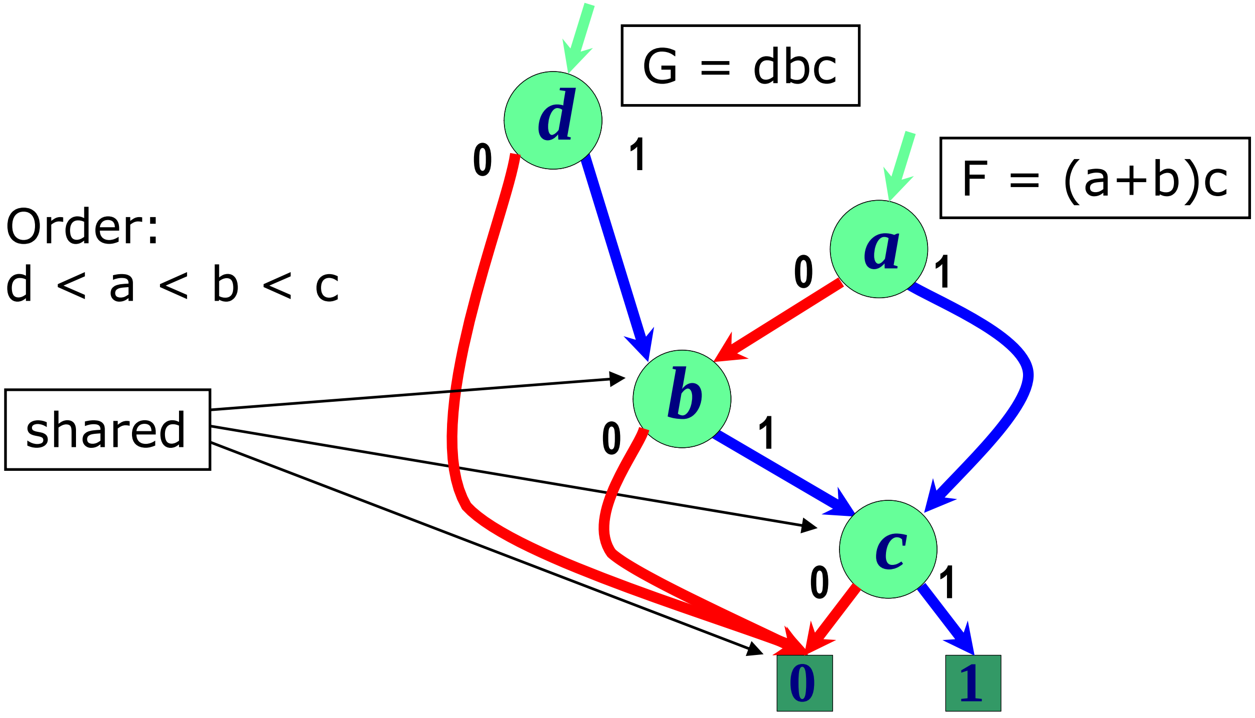
ROBDDs – Why Do We Care?

- Easy to solve some important problems:
  - Tautology checking
    - just check if BDD is identical to function $\boxed{1}$
  - Identity checking
  - 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
  - – We can share also among multiple ROBDDs!

Order:
d < a < b < c



Logic Operations with ROBDD

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

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

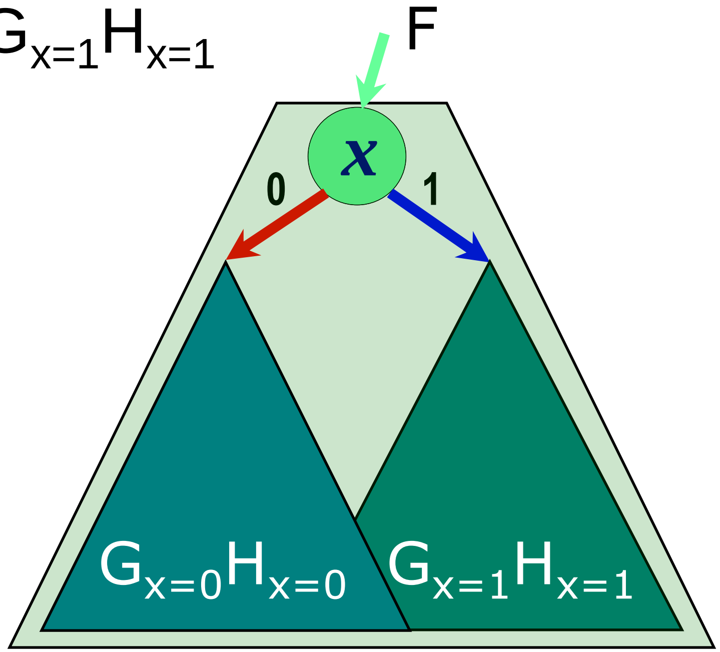
Example

- 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 is 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

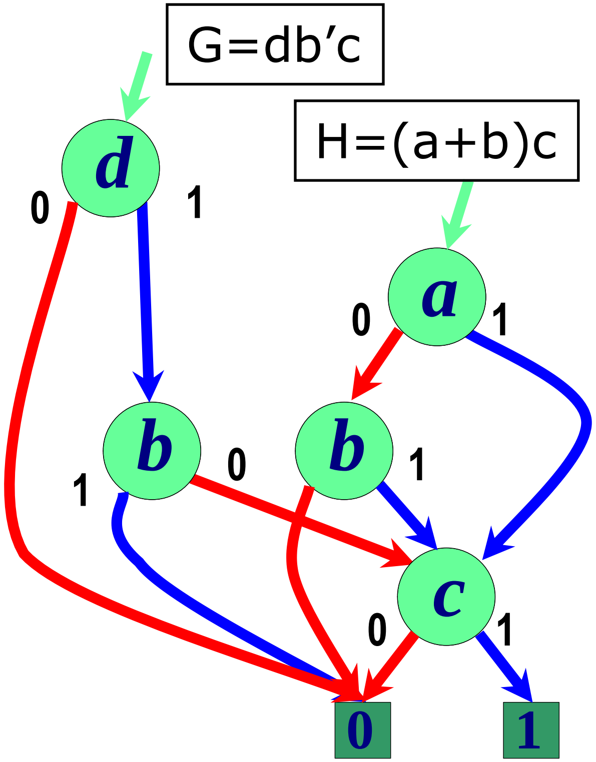
Putting All Together

```
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 – Example

Order:d < a < b < c



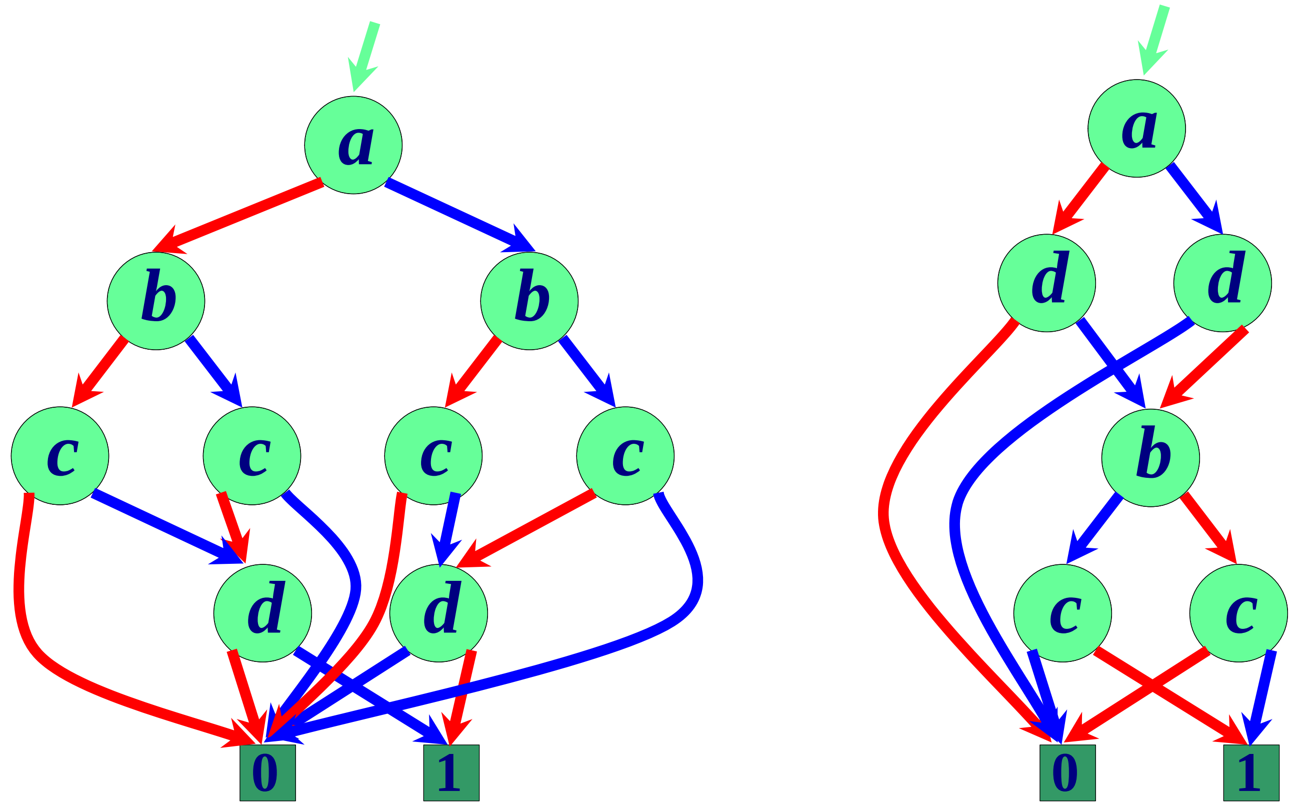
Split Variable	G cof	H cof	AND(G,H)
d=0	0	(a+b)c	
d=1	b'c	(a+b)c	
d=1,a=0			

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

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

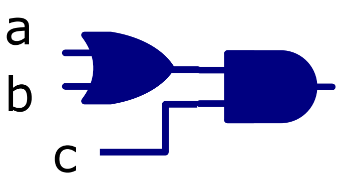
- In practice:
  - Many common functions have reasonable size
  - Can build ROBDDs with millions of nodes
  - Algorithms to find good variables ordering

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

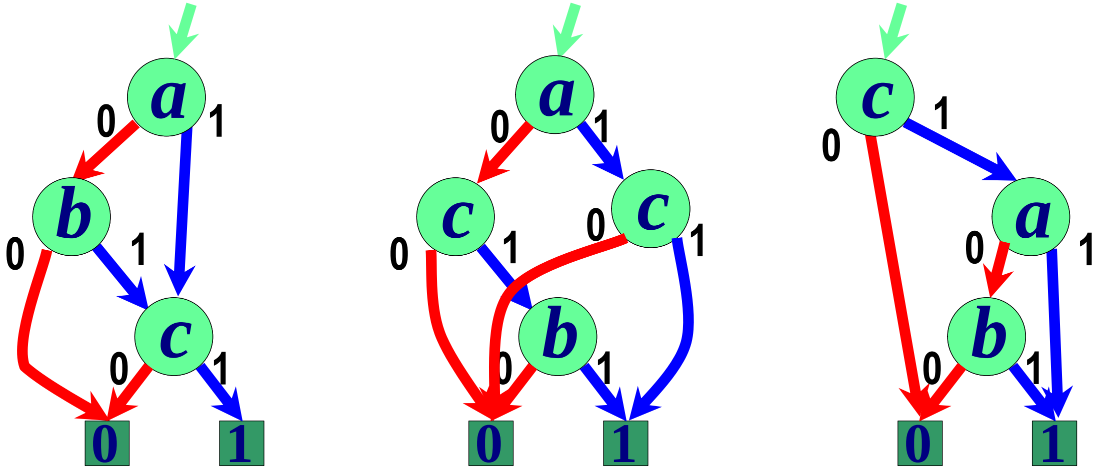
  - Disclaimer: it's a heuristic, results are not guaranteed

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



initial order:
a < b < c

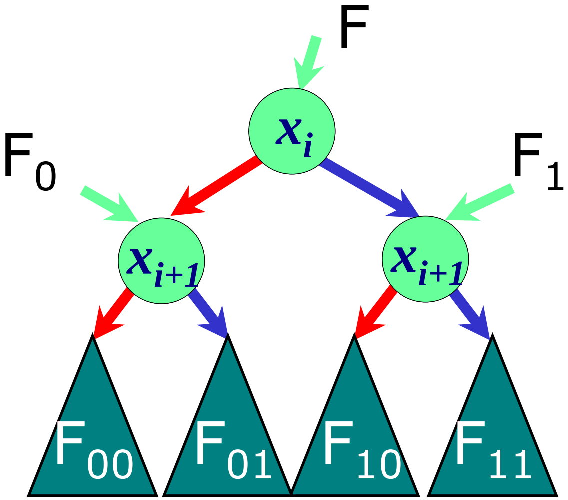
Swap (b, c):
a < c < b

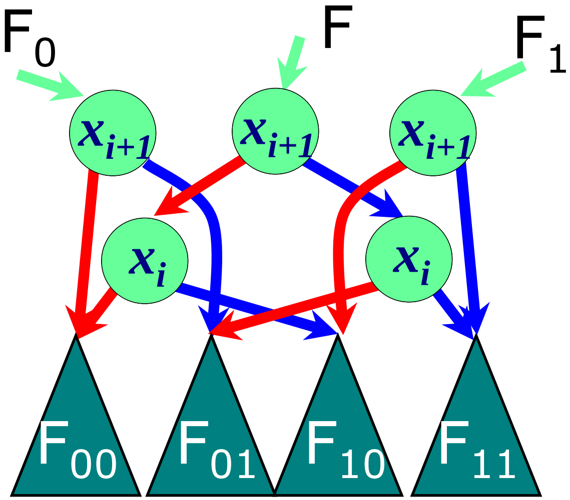
Swap (a, c):
c < a < b

Final order:
c<a<b

Variable Swapping

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



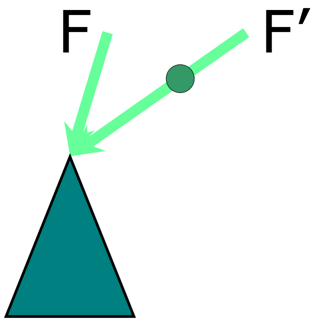


Dynamic Variable Ordering

- 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:
    - The technique is still non optimal
    - At convergence, you most probably have found only a local minimum

Improvements of BDDs

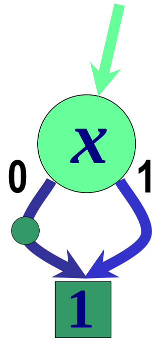
- Complement edges (1990)
  - Creates more opportunities for sharing
    → fewer nodes

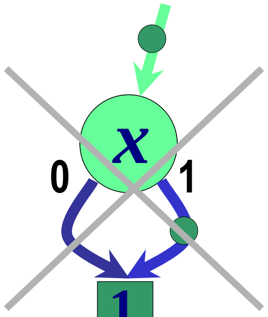


  - For every pair (F,F’), we
    - only construct the ROBDD for F
    - F’ is given by using a complement edge to F
  - Which do you pick?
    - THEN edge can never be complemented
    - Only constant value 1

Complement Edges

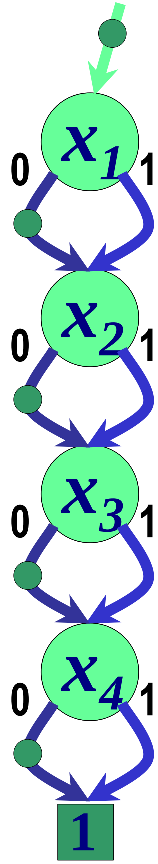
- F = x





- Still canonical

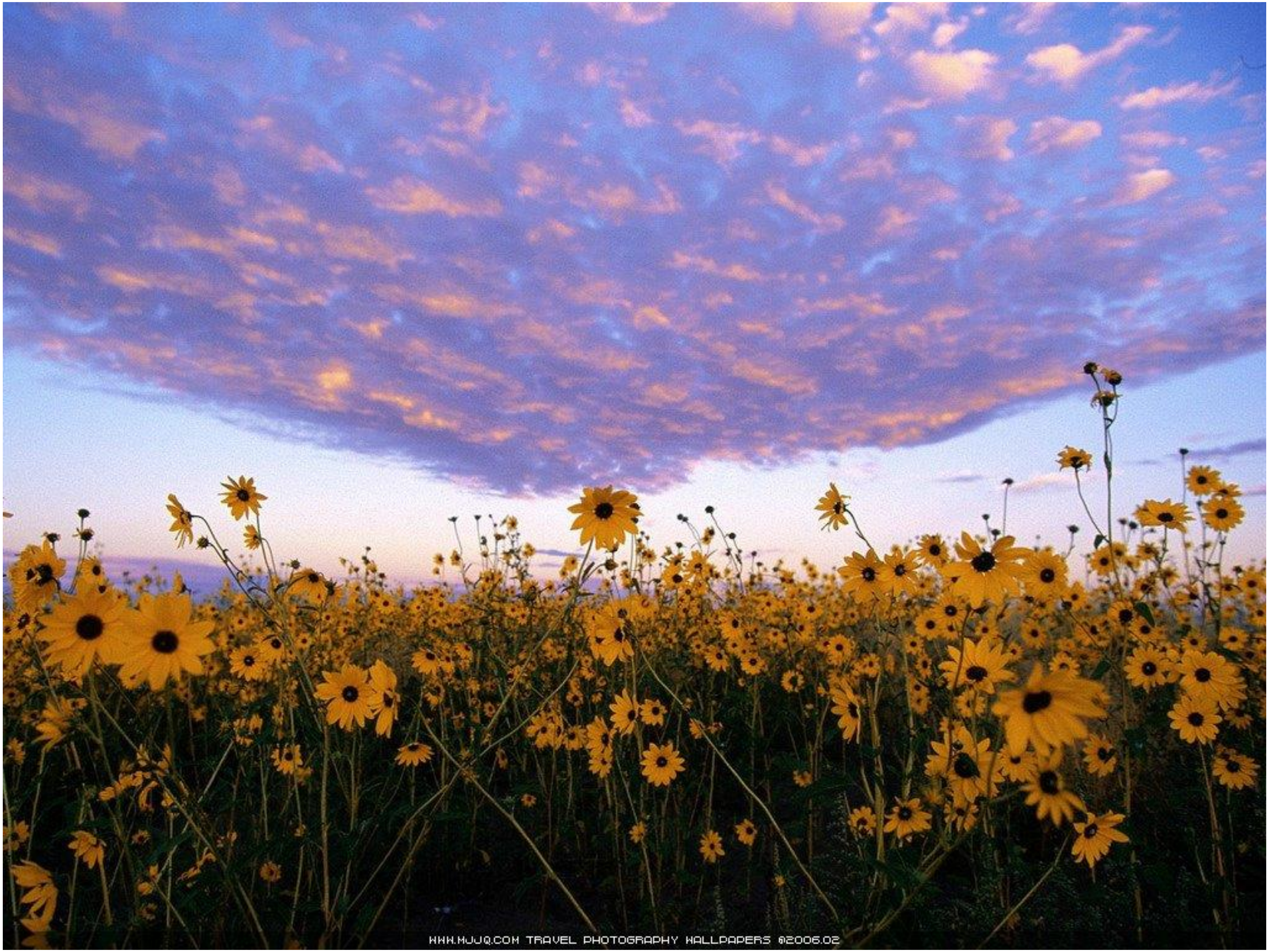
$$F = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$



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



WWW.MJJQ.COM TRAVEL PHOTOGRAPHY WALLPAPERS @2006.02