Verifying Multilevel Circuits

Given two multilevel circuits to be checked for equivalence flatten them to two levels.

$\Rightarrow$ use two-level equivalence algorithm

$$f = (a_1 + b_1)(a_2 + b_2)(a_3 + b_3)$$

$$\downarrow$$

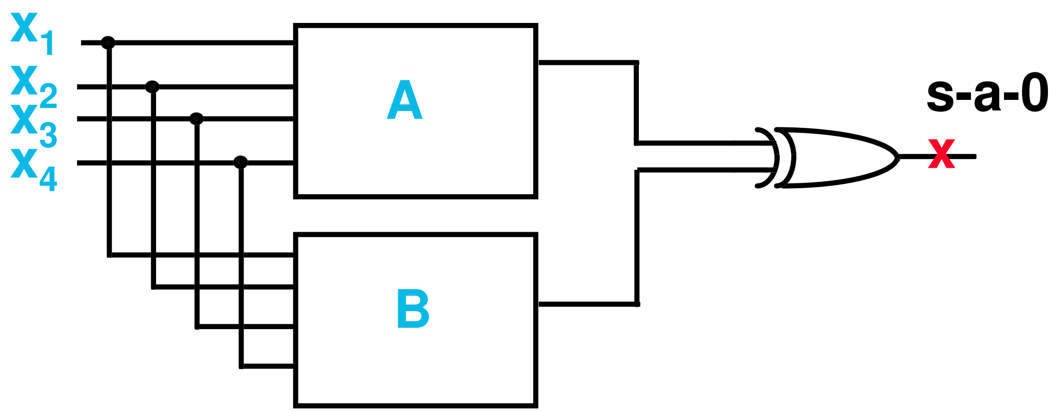
$$f = a_1 a_2 a_3 + a_1 a_2 b_3 + a_1 b_2 a_3 + a_1 b_2 b_3 + b_1 a_2 a_3 + b_1 a_2 b_3 + b_1 b_2 a_3 + b_1 b_2 b_3$$

$$f = a_1 \oplus a_2 \ldots \oplus a_n \longrightarrow 2^{n-1} \text{ product terms}$$

Verification and Testing

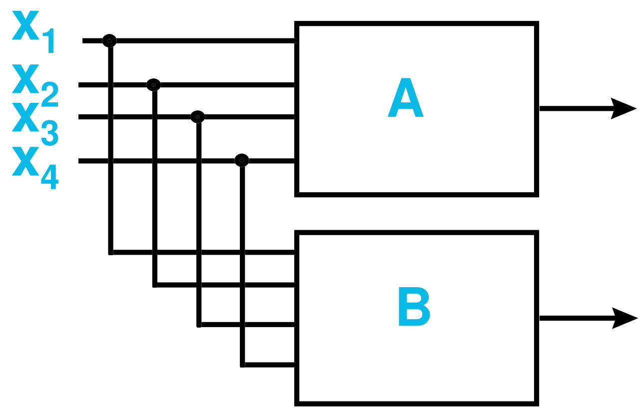
Given two single-output circuits A and B

Are A and B equivalent can be posed as: Is there a test for F s-a-0?



If F s-a-0 is redundant, A $\equiv$ B else test vector produces different outputs for A and B.

Enumeration-Simulation Methods

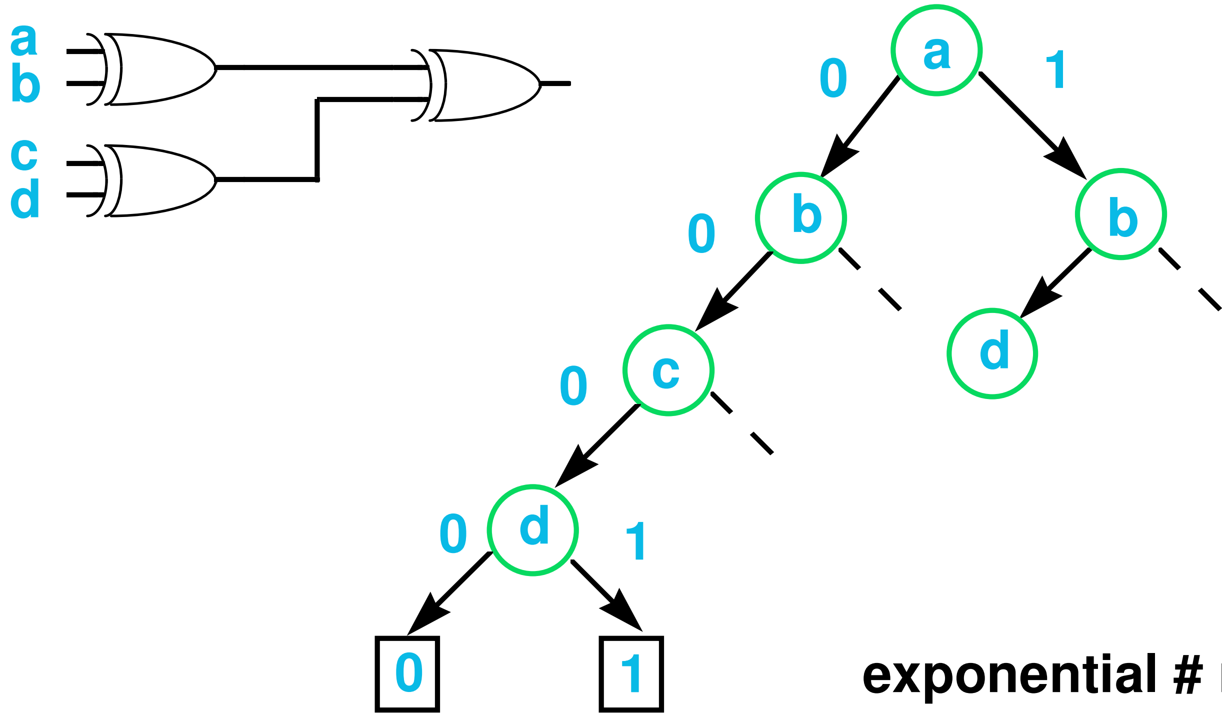


Enumerate the ON-set cubes, i.e., all tests for s-a-0 on A. Cube-simulate these tests on B to check if B produces a 1. If not $A \not\equiv B$.

Do so for OFF-set of A also.

This is similar to containment check of a cube in a cover, except we do not explicitly store the two-level covers.

Decision Tree

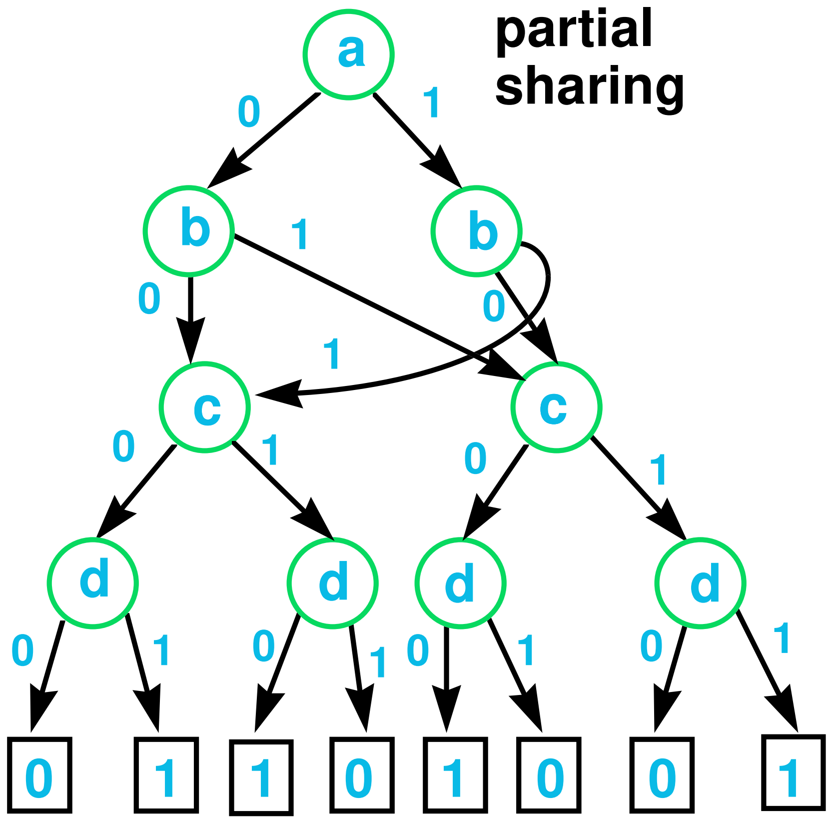


exponential # nodes.

Do not have to store entire set of nodes, but have to enumerate them (slight improvement over two-level tautology).

Decision Graph

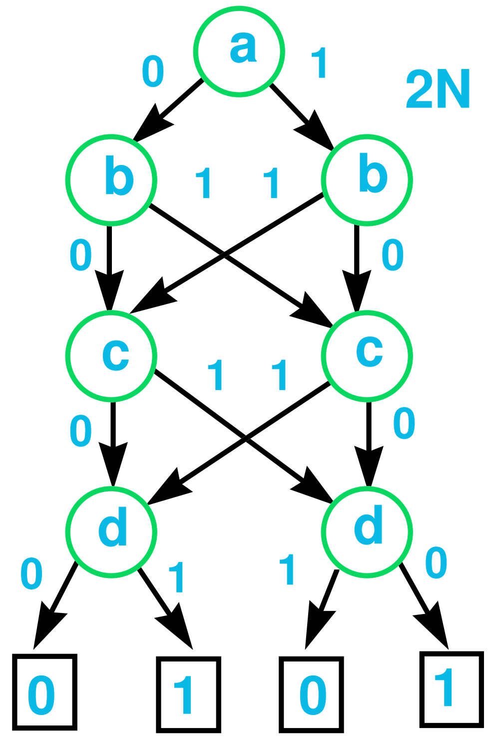
Share nodes in tree ⇒ graph.



partial sharing

FULLY SHARED

2N nodes



Verification Using Boolean Graphs

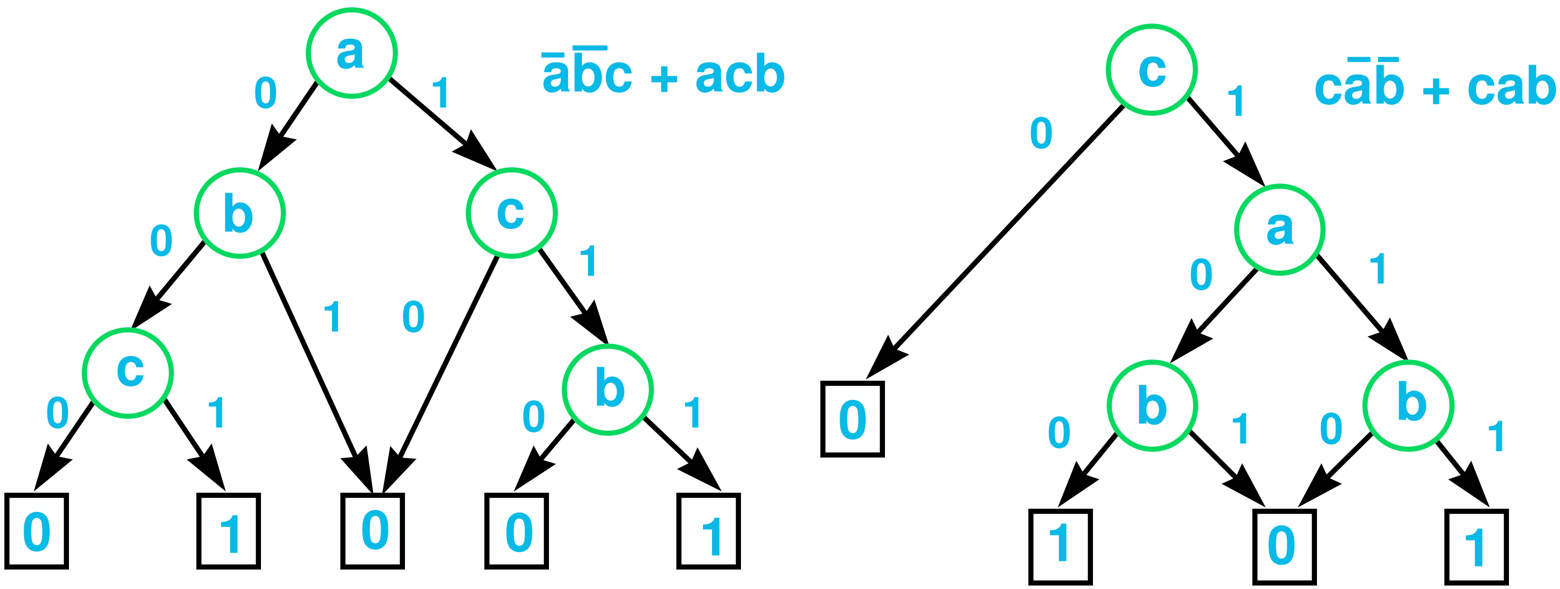
Construct Boolean graphs for the two circuits

Verify the equivalence of the Boolean graphs

- Can't really "simulate" one graph on the other, without unfolding it.

- Have to store and manipulate both graphs.

Free Boolean Graphs

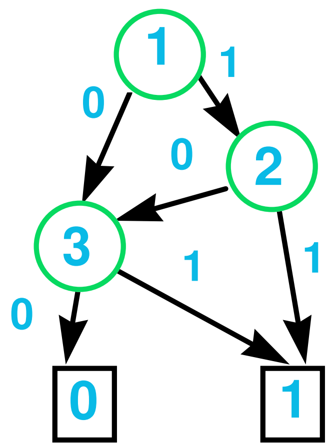


Verifying the equivalence of free Boolean graphs is co-NP-complete

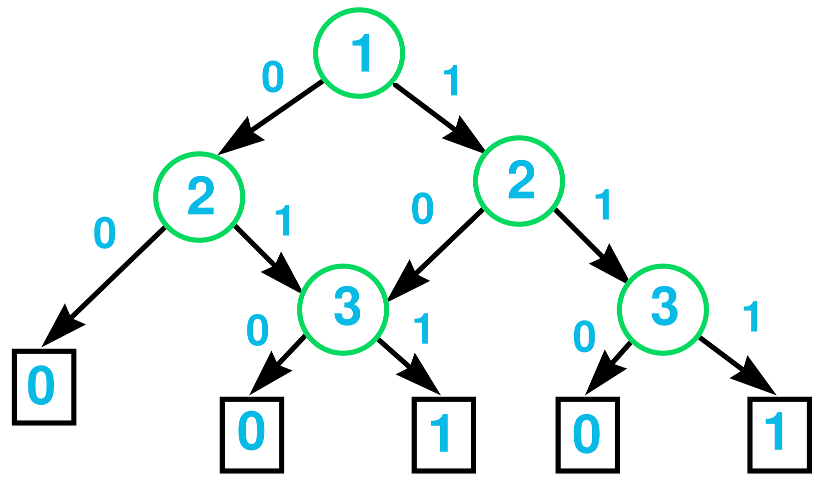
- Satisfiability in linear time
- Probabilistic verification possible in polynomial time

Ordered Boolean Graphs

$f = x_1\, x_2 + x_3$



$f = x_1\bar{x}_2x_3 + x_1x_2x_3 + \bar{x}_1x_2x_3$



Inputs satisfy ordering restriction. Each node/vertex v in the graph has index(v). Two children are low(v) and high(v). $\boxed{0}$ and $\boxed{1}$ are terminal vertices, others are non-terminal.

$$index(v) < index(low(v)) \qquad \text{for all } v$$

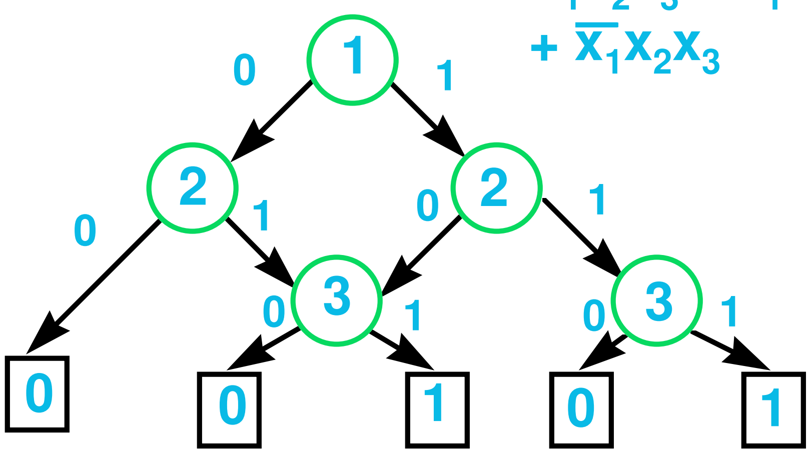
$$index(v) < index(high(v))$$

Reduced, Ordered BDDs

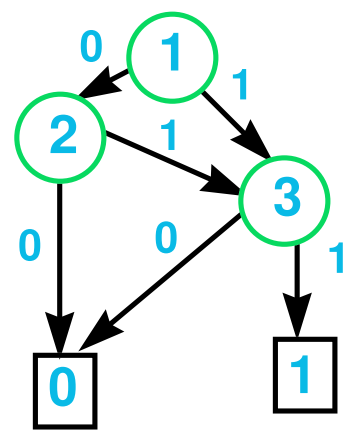
An ordered Binary Decision Diagram (OBDD) may have redundant vertices.

Definition: An OBDD is reduced, if it contains no vertex v with $low(v) = high(v)$, nor does it contain distinct vertices v and v^1 such that the subgraphs rooted by v and v^1 are isomorphic.

$f = x_1x_2x_3 + x_1\overline{x_2}x_3 + \overline{x_1}x_2x_3$



$f = x_1x_3 + \overline{x_1}x_2x_3$



Can reduce an OBDD in $O(|G| \log |G|)$ time.

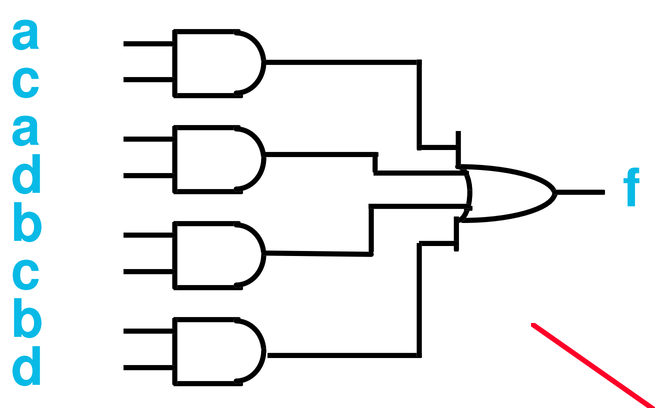
ROBDDs are Canonical

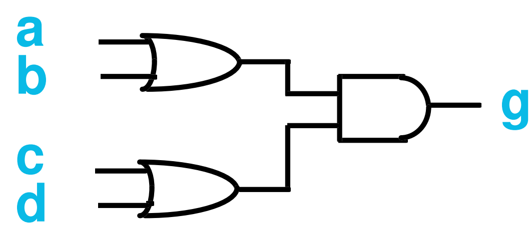
Given an ordering, a logic function has a unique ROBDD.

Given two logic functions, checking their equivalence reduces to a graph isomorphism check between their respective ROBDDs

- can be done in linear time in $|G_1|$ (= $|G_2|$).
- constructing ROBDD for a given function and ordering could take exponential time.

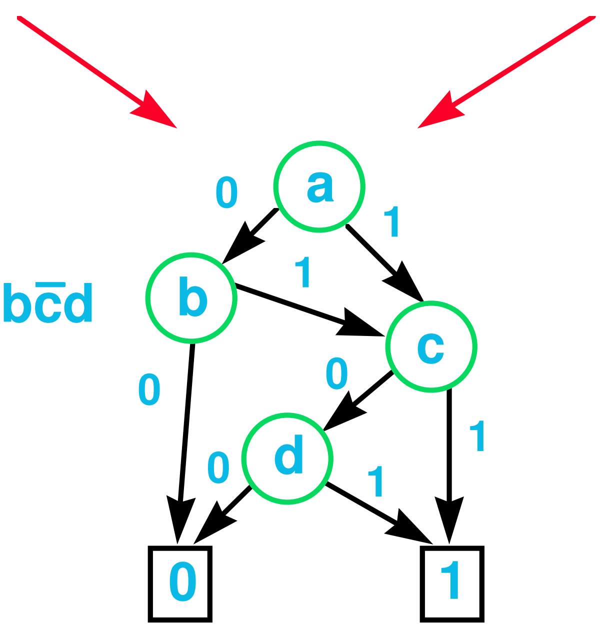
Example





$f = ac + \bar{a}bc + a\bar{c}d + \bar{a}b\bar{c}d$

disjoint cover



ordering

a b c d

Why are ROBDDs Canonical?

Induction on # inputs.

Base Case: Boolean functions with zero inputs.
f can be $\boxed{0}$ or $\boxed{1}$ ROBDDs.

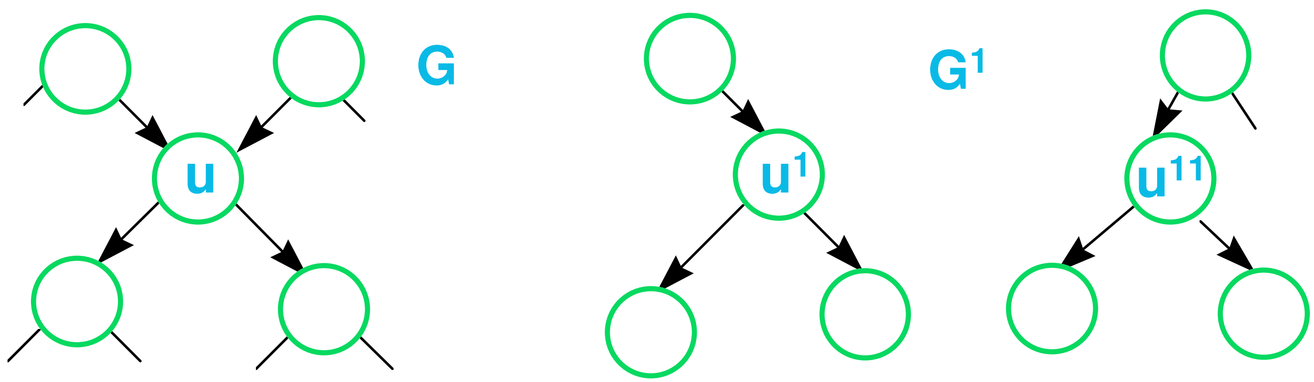
Induction Hypothesis: Any function with # inputs < k has a unique ROBDD.

Choose a function f with k inputs. Let G, G^1 be two ROBDDs for f under the same ordering.

Let x_i be the input with lowest index in the ROBDDs G, G^1

Why are ROBDDs Canonical? - 2

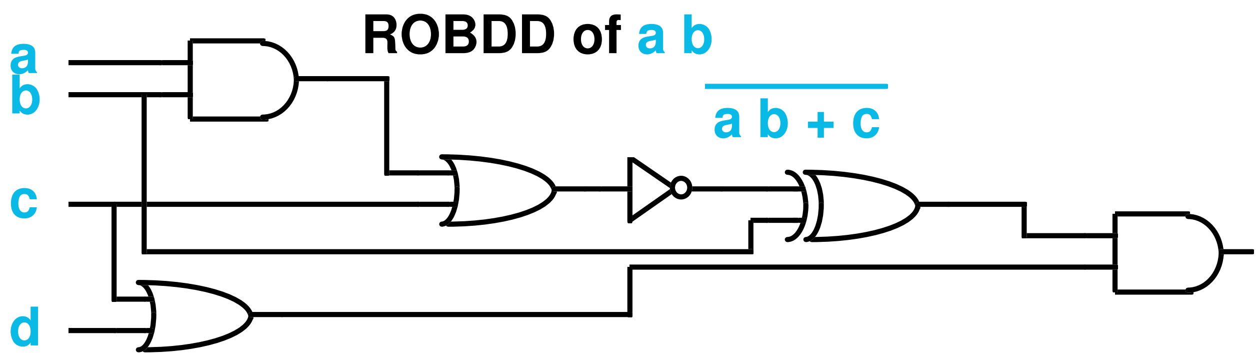
By hypothesis, $f0 \equiv f0^1$ $f1 \equiv f1^1$. If no sharing of vertices between f0, f1 and $f0^1$, $f1^1$, then $G \equiv G^1$. Say a vertex u is shared across f0, f1.

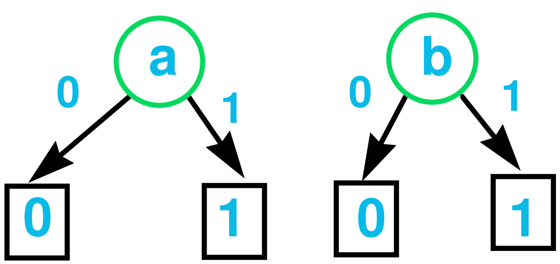


There will be a unique u^1 shared in $f0^1$, $f1^1$. If u $\nearrow u^1$ $\searrow u^{11}$ G^1 is not reduced.

ROBDD Construction

Given ordering and multilevel network.





Begin with ROBDDS for primary inputs

Proceed through network, constructing the ROBDD for each gate output, by applying the gate operator to the ROBDDs of the gate inputs

Procedure Apply

Compute f_1 <op> f_2

<op> can be AND, OR, XOR, XNOR, etc.

To apply the operator to the ROBDDs represented by f_1 and f_2

1) If v_1 and v_2 are terminal vertices, simply generate a terminal vertex u with $value(u) = value(v_1) <op> value(v_2)$

2) Else if $index(v_1) = index(v_2) = i$

   Apply algorithm recursively on $low(v_1)$ and $low(v_2)$ to generate $low(u)$, $high(v_1)$ and $high(v_2)$ to generate $high(u)$, after creating vertex u, $index(u) = i$

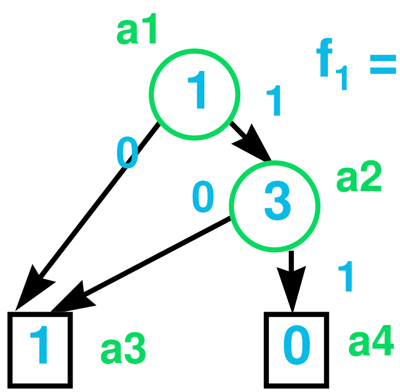
Procedure Apply - 2

3) If $index(v_1) = i$, but $index(v_2) > i$, then create a vertex u having index i, and apply algorithm recursively on $low(v_1)$ and v_2 to generate $low(u)$, and on $high(v_1)$ and v_2 to generate $high(u)$.

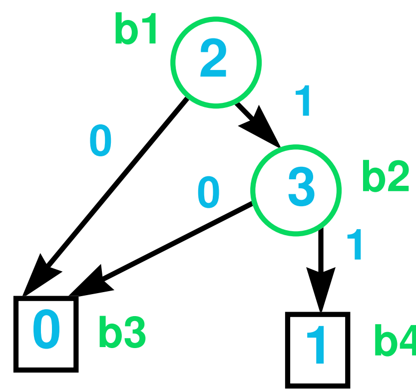
4) Reverse of (3)

$O(|G_1| \cdot |G_2|)$ complexity (though recursive).
"Multiplying" the two graphs.

Example OR'ing of ROBDDs

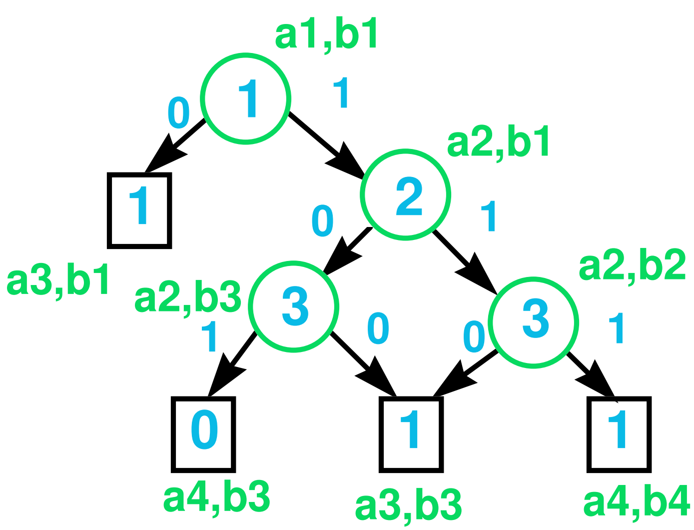


$f_1 = \overline{x_1 x_3} = \overline{x}_1 + \overline{x}_3$

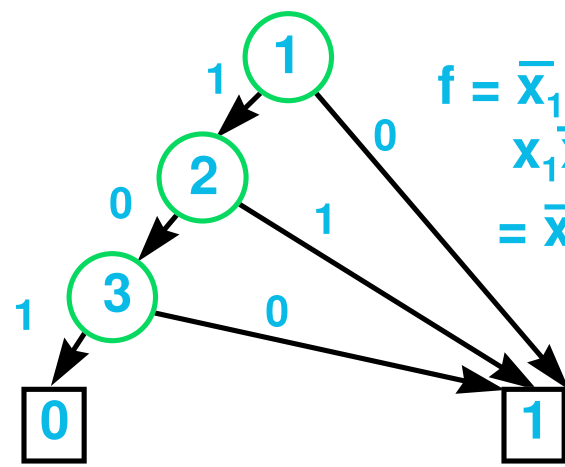


$f_2 = x_2\, x_3$

Evaluation graph



After reduction



$f = \overline{x}_1 + x_1 x_2 + x_1 \overline{x}_2 \overline{x}_3$

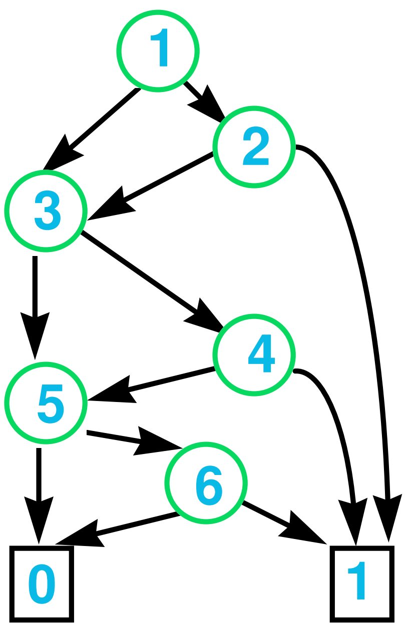
$= \overline{x}_1 + \overline{x}_3 + x_2$

Sensitivity to Ordering

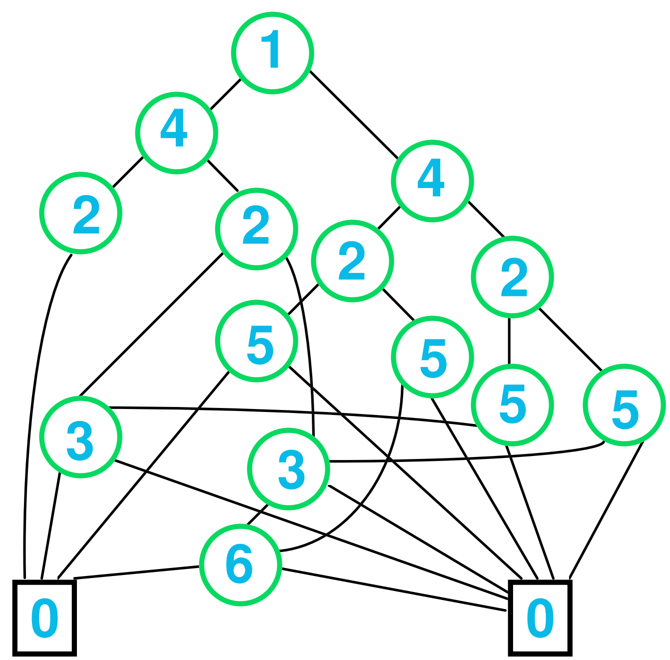
Given a function with n inputs, one input ordering may require exponential # vertices in ROBDD, while other may be linear in size.

$$f = x_1 x_2 + x_3 x_4 + x_5 x_6$$

$x_1\ x_2\ x_3\ x_4\ x_5\ x_6$



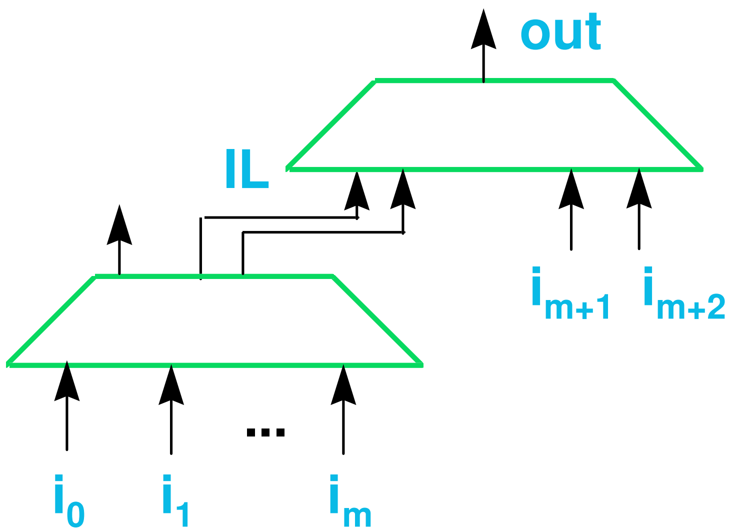
$x_1\ x_4\ x_2\ x_5\ x_3\ x_6$



Heuristic Input Ordering

BDD can be viewed as a multiplexor-based multilevel circuit.

Look at an (optimized) multilevel network and decide ordering for the BDD.



order i_{m+1}, i_{m+2} after $i_0, i_1, \ldots, i_m$ since IL appear to be a good "encoding" for $i_0, i_1, \ldots, i_m$

Generalize to multiple levels.

Resolve "conflicts" heuristically.