Proof of Correctness and Equivalence of Combinational Circuits Using ROBDD and Logical Operations

Presented by: Rashed Hassan Siam
Class Roll: FH-2343
Registration No: 2022-518-433
Department of Computer Science and Engineering
University of Dhaka

Presented to: Professor Dr. Hafiz Md. Hasan Babu
Department of Computer Science and Engineering
University of Dhaka

Content Outline

- **Revisiting ROBDD**
- **Reduction Rules of ROBDD**
- **Canonicity of ROBDD**
  - Using Boolean Functions
  - Using Combinational Circuits
- **Logical Operations on ROBDD: Apply Algorithm**

Revisiting ROBDD

- **An OBDD is ROBDD when these rules are applied:**
  - Merge isomorphic sub-graphs
  - Eliminate node whose both children are isomorphic

- **Advantage of ROBDD is that it is canonical for a given function**
  - There is exactly one ROBDD for a given variable ordering
  - This property makes it useful in functional equivalence checking

Merging Isomorphic Sub-graphs

For any two nodes u and v,

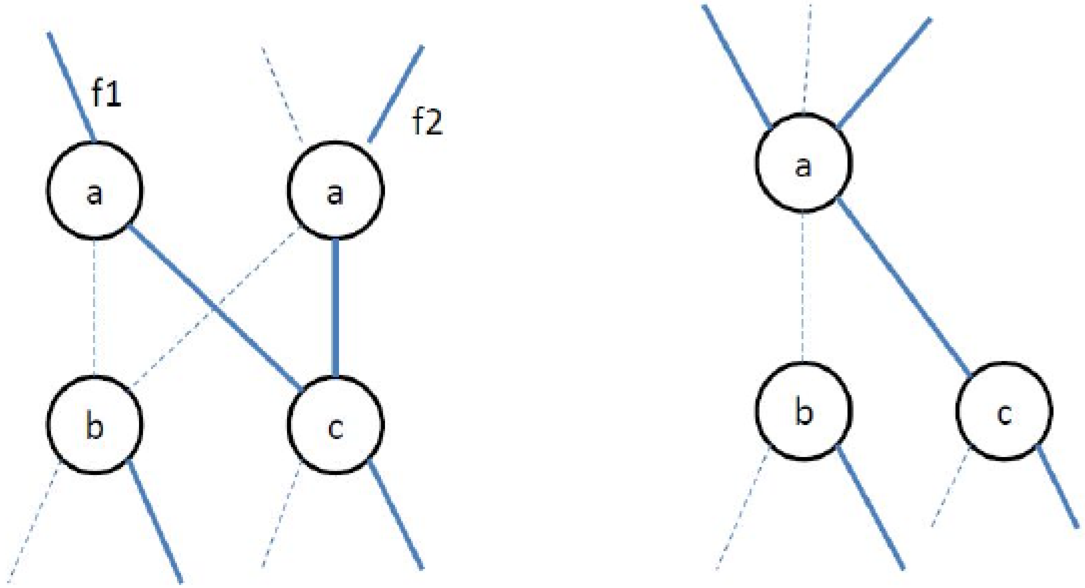
If var(u) = var(v),

low(u) = low(v)

and, high(u) = high(v)

That means, u = v

We can remove u and connect all parents of u to v.

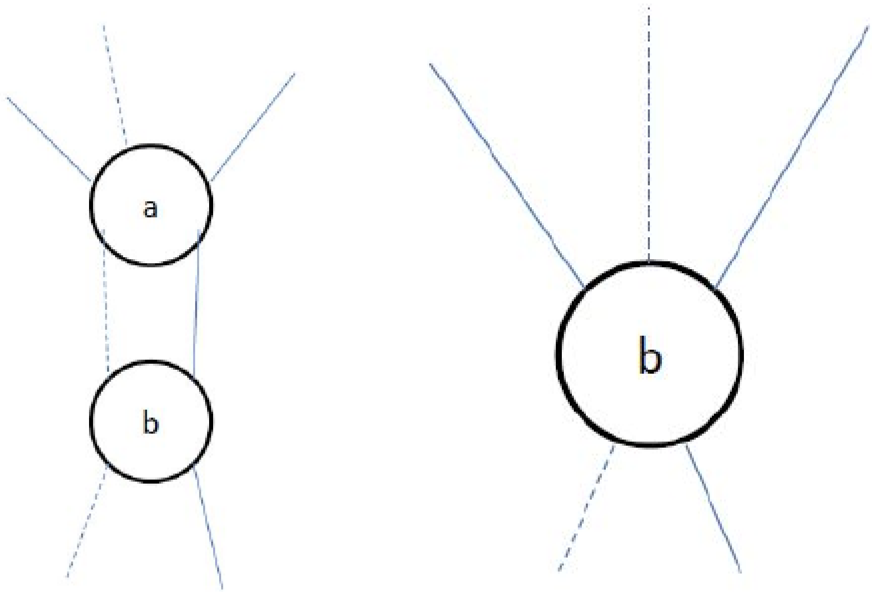


Isomorphic Children of Same Node

If there is a node u such that,

low(u) = high(u),

then remove u and connect all parent of u to low(u).



Properties of ROBDD

- **Uniqueness:**

  No two distinct nodes u and v are labeled with the same variable name and have the same low and high successor.

- **Non-redundant:**

  No variable node u has identical low and high successor.

Canonicity of ROBDD

Two Boolean Functions are equivalent iff their reduced OBDDs (ROBDD) are identical with respect to a specific variable ordering.

- For any boolean function *f*, there is a **unique** ROBDD with the respect to a **specific variable ordering**.
- We can transform it into a canonical form by repeatedly applying reduction rules.

Canonicity of ROBDD (Continued)

Due to the canonicity some possible applications can be :

- **Checking equivalence:** Verify isomorphism between ROBDDs.
- **Non satisfiability:** Verify if ROBDD has only one terminal node, labeled by 0.
- **Tautology:** Verify if ROBDD has only one terminal node, labeled by 1.

When Are Two Boolean Functions Logically Equivalent?

- **Two Boolean functions are logically equivalent –**
  - Iff their ROBDDs are isomorphic.
- **Two ROBDDs are isomorphic if there exists a bijection b between their graphs such that -**
  - Terminals are mapped to terminals.
  - Non-terminals are mapped to non-terminals.
  - For every terminal vertex v,
    - *value(v) = value(b(v))*
  - For every non-terminal vertex v,
    - *var(v) = var(b(v))*
    - *low(v) = low(b(v)), and*
    - *high(v) = high(b(v))*

Equivalence of ROBDD

P = (x XOR y)

x	y	P
0	0	1
0	1	0
1	0	0
1	1	1

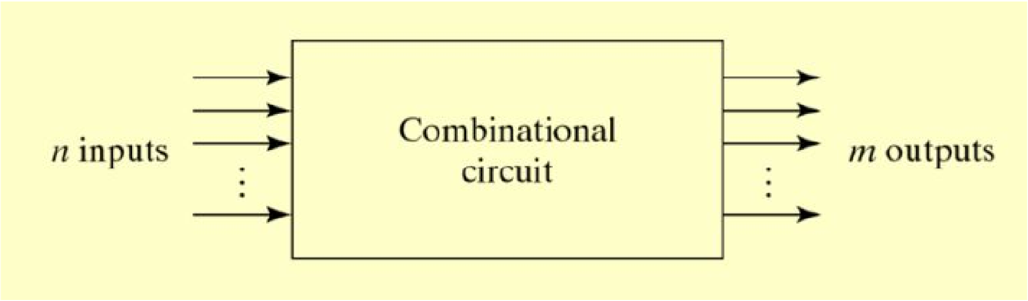
Logically Equivalent

Q = (x OR y) AND (x OR y)

x	y	x OR y	x OR y	Q
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

Combinational Circuits

- **Properties**
  - Current output depend on current inputs only
  - Consists of combinational gates



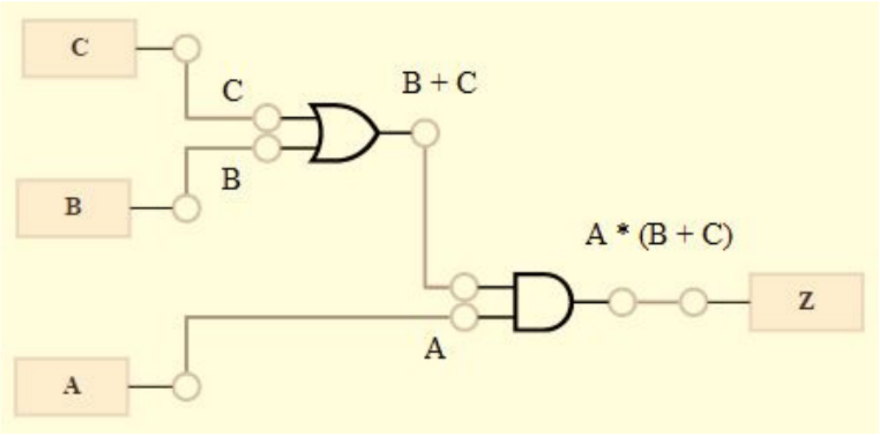
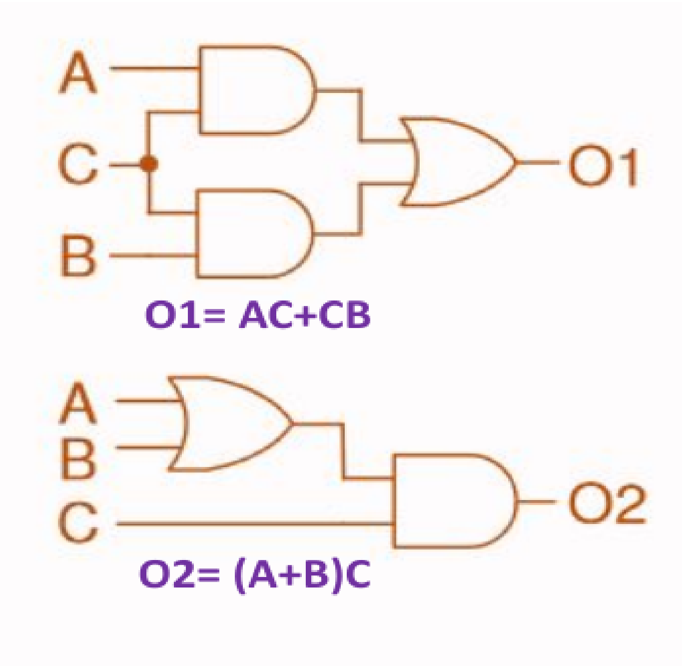
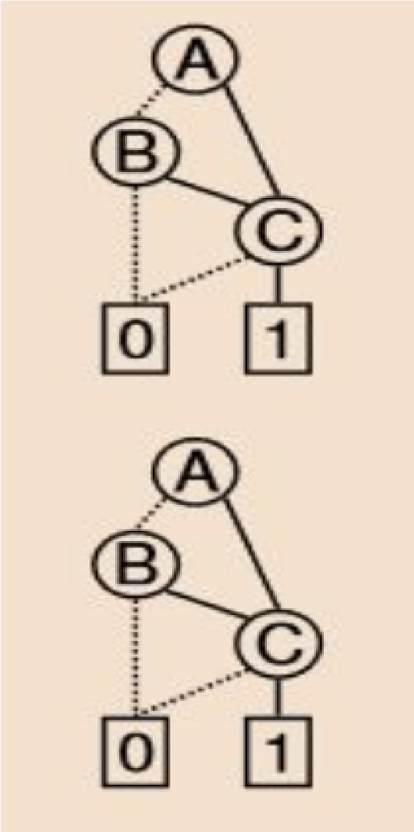


Figure: Combinational Circuits

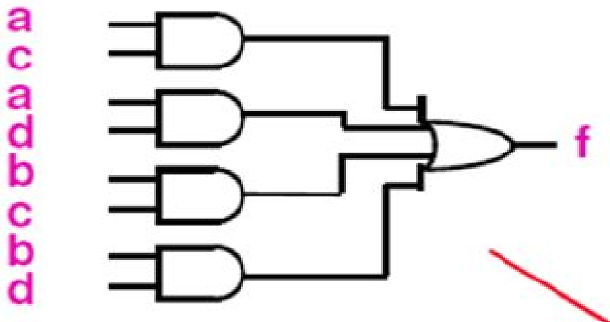
Equivalence of Combinational Circuits





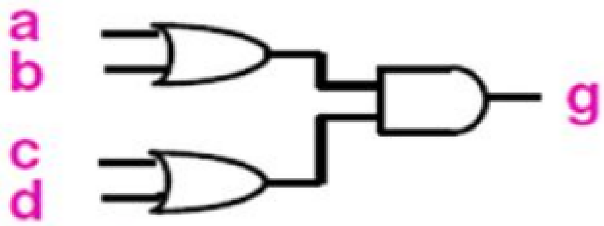
Isomorphism check

Canonicity of ROBDD From Combinational Circuits



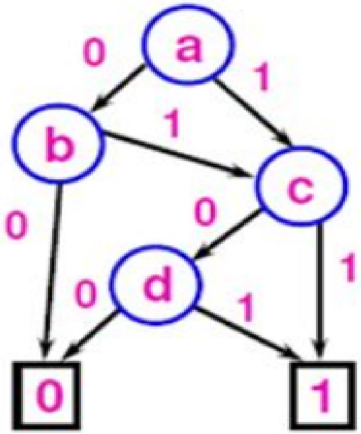
f = ac + ad + bc + bd

g = (c + d) (a + b)



ordering

a b c d



f = ac + ad + bc + bd

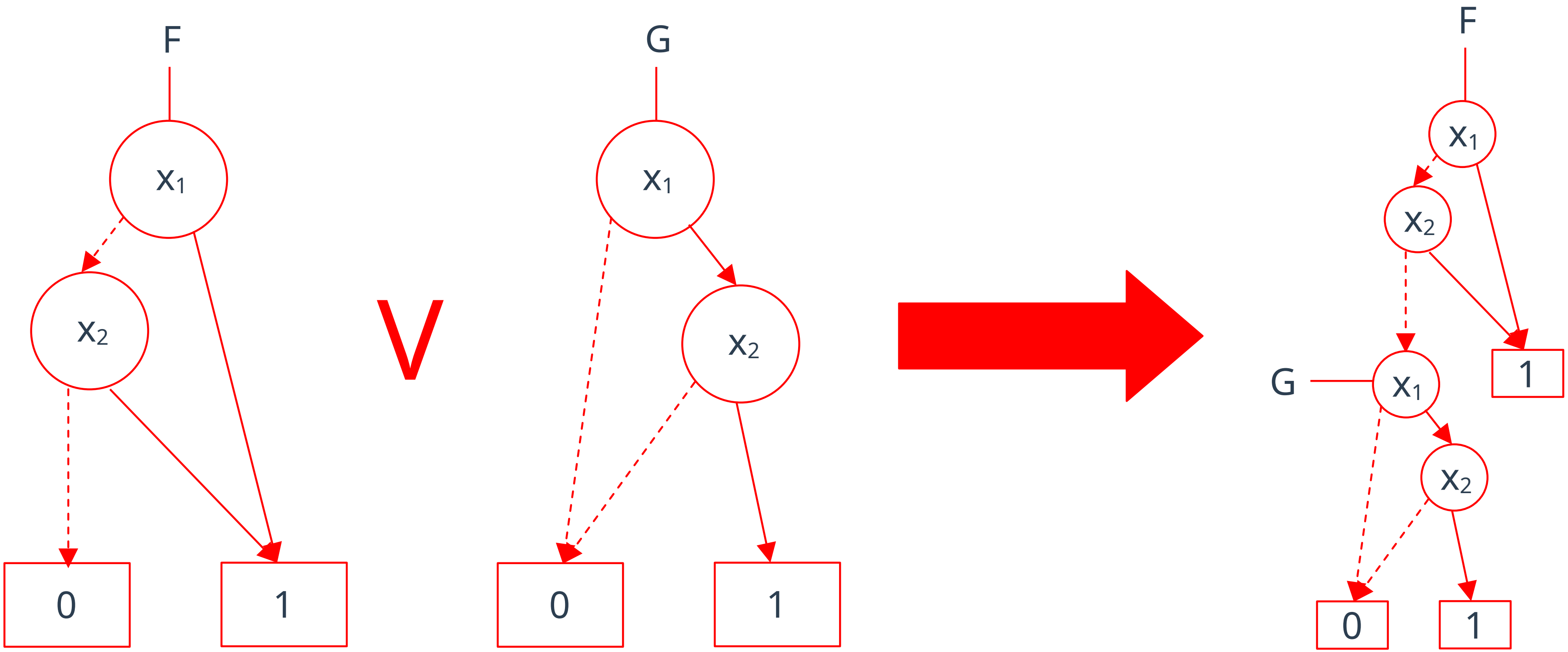
f = a (c + d) + b (c + d)

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

f = g

f and g are logically equivalent

Traditional Logical Operations



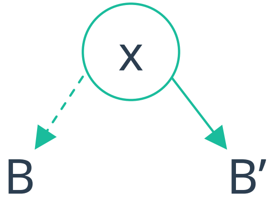
Apply Algorithm

- **Preconditions:**
  - Let's say we have two ROBDDs B_f and B_g
    - Must have compatible ordering (should follow same variable ordering)
    - Apply (op, B_f, B_g) ⇒ OBDD ⇒ ROBDD
  - Shannon's expansion is used for apply algorithm

$$F = \bar{x}.f[0/x] + x.f[1/x]$$
$$G = \bar{x}.g[0/x] + x.g[1/x]$$

  - So Shannon's expansion of f and g,
    - $\bar{x}(f[0/x]\,op\,g[0/x]) + \bar{x}(f[1/x]\,op\,g[1/x])$ *here op can be any operator*

Shannon's Expansion

If a BDD 

represents a Boolean function f, then:

1. B represents f[0/x] and B' represents f[1/x]; and
2. The BDD is effectively a compressed representation of f in Shannon normal form.

So, we implement apply recursively on the structure of the BDDs.

Apply Algortihm (Case 1)

Algorithm apply(op, B_f, B_g)

Let r_f be the root node of B_f and r_g be the root node of B_g.

> If both r_f and r_g are terminal nodes with labels l_f and l_g, respectively compute the value l_f op l_g and the resulting OBDD is B_0 if the value is 0 and B_1 otherwise.

Apply Algortihm (Case 2)

In the remaining cases, at least one of the root nodes is a non-terminal.

- If both nodes are x_i – nodes (i.e., non-terminal of some variables), create an x_i – node n (r_f, r_g) with a dashed line to apply(op, lo(r_f), lo(r_g)) and a solid line to apply(op, hi(r_f), hi(r_g)).

Apply Algortihm (Cases 3 and 4)

If r_f is an x_i-node, but r_g is a terminal node or an x_j-node with $j > i$, create an x_i-node n (called r_f, r_g) with a dashed line to apply(op, lo(r_f), r_g) and a solid line to apply(op, hi(r_f), r_g).

If r_g is an x_i-node, but r_f is a terminal node or an x_j-node with $j > i$, create an x_i-node n (called r_f, r_g) with a dashed line to apply(op, lo(r_g), r_f) and a solid line to apply(op, hi(r_g), r_f).

Apply Algorithm

apply: cases

- apply(□, x [B, B′], x [C, C′]) = x [apply(□, B, C), apply(□, B′, C′)]

  x

  B B′ C C′

  x

  apply(□, B, C) apply(□, B′, C′)

- apply(□, x [B, B′], C) = x [apply(□, B, C), apply(□, B′, C)]

  x

  B B′

  x

  apply(□, B, C) apply(□, B′, C)

  - When C is terminal node, or non-terminal with var(root(C)) > x
- apply(□, B, x [C, C′]) = x [apply(□, B, C), apply(□, B, C′)]

  x

  C C′

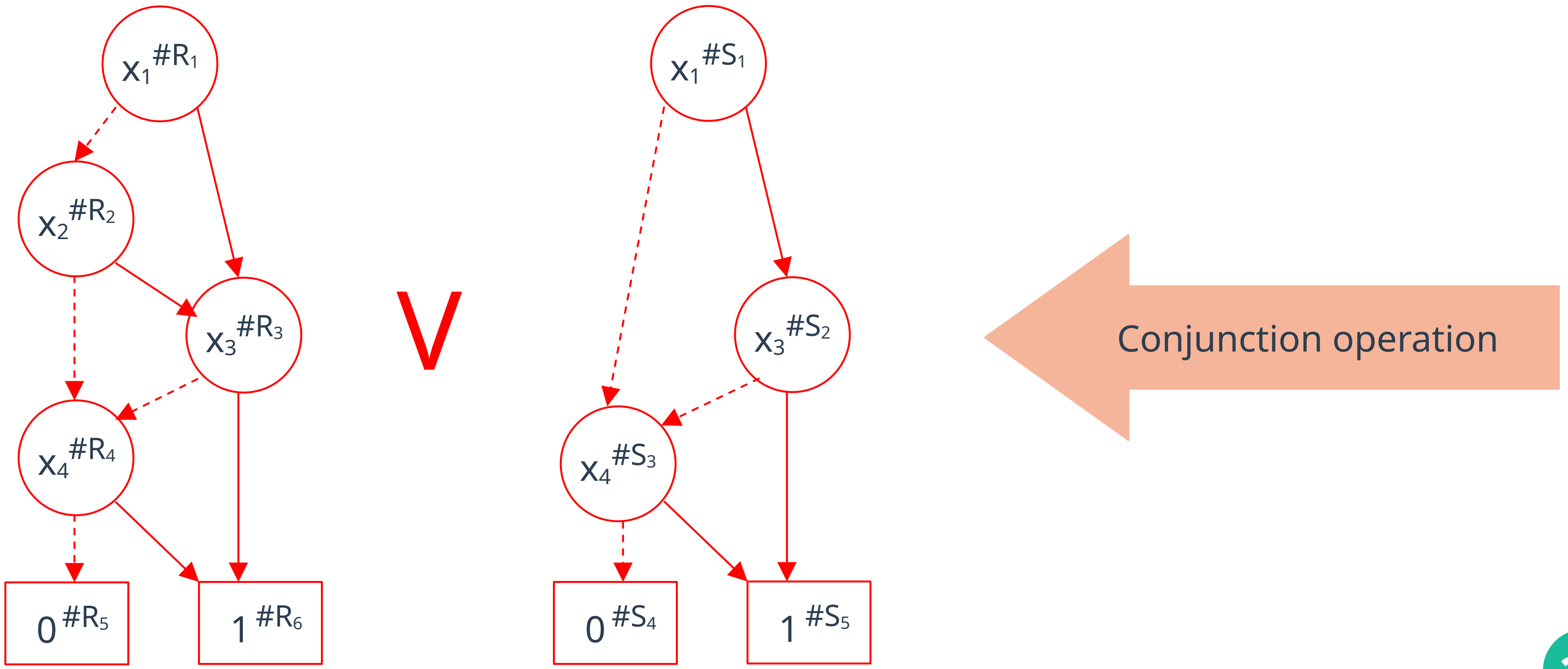
  x

  apply(□, B, C) apply(□, B, C′)

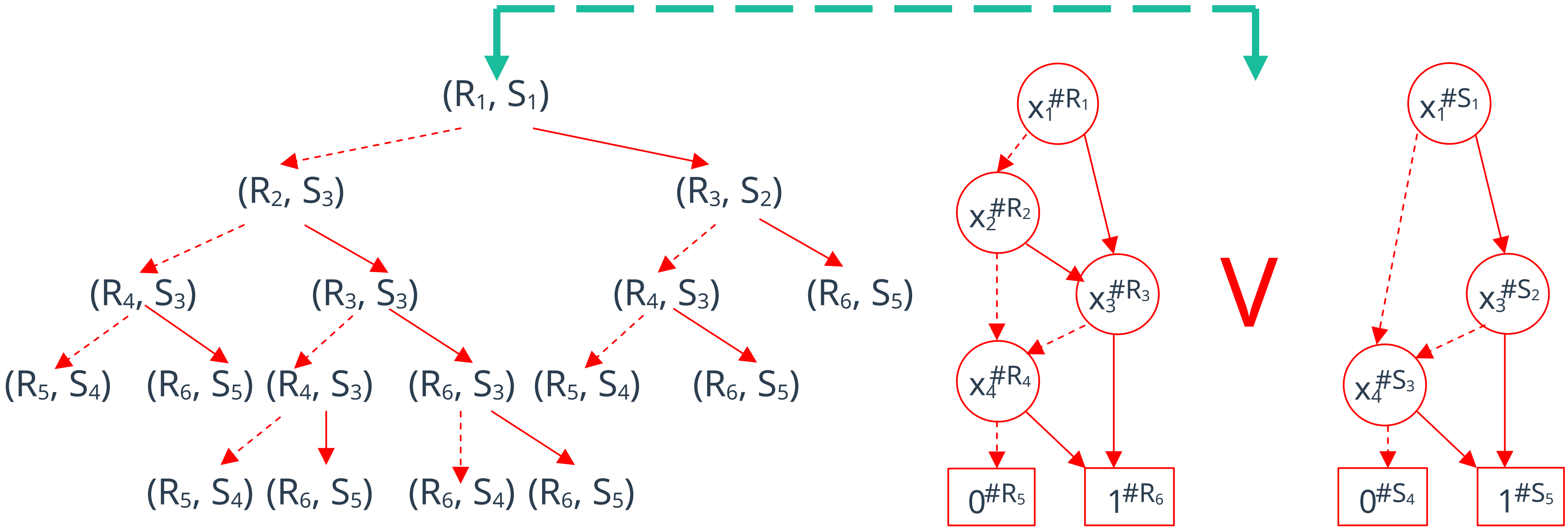
  - When B is terminal node, or non-terminal with var(root(B)) > x
- apply(□, $\boxed{u}$, $\boxed{v}$) = $\boxed{u \,\square\, v}$

Apply Example

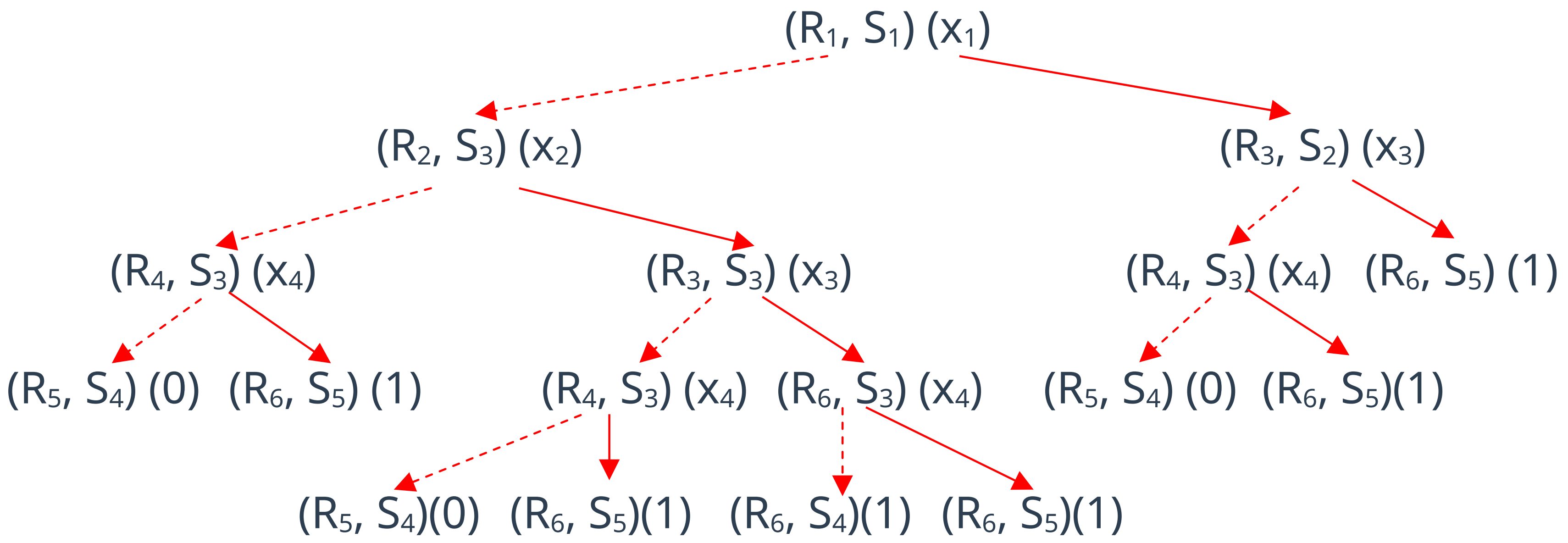
Two ROBDDs in the figure below:



Apply: Recursive Call

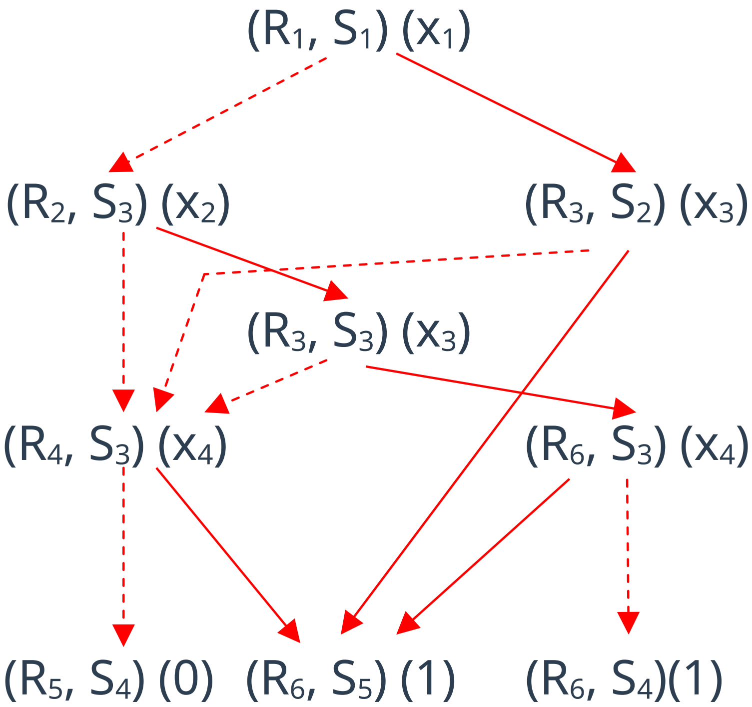


Apply Algorithm

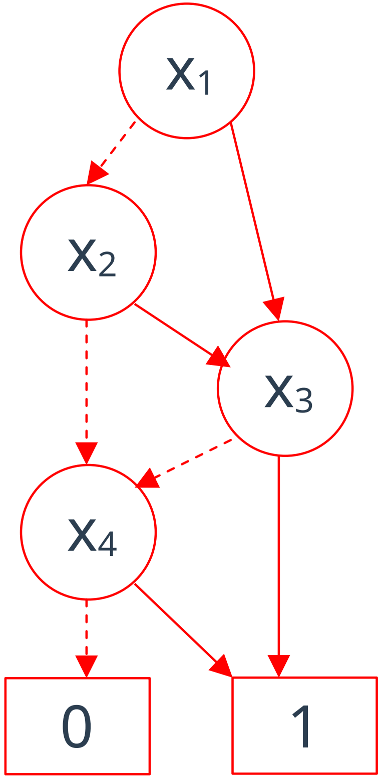


Apply Algorithm

Eliminate Redundancy



Reduced OBDD



References

- Formal Verification, Lecture 8: Operations on Binary Decision Diagrams (BDDs); Jacques Fleuriot, jdf@inf.ac.uk, Diagrams from Huth & Ryan, LiCS, 2nd Ed.
- Module 6 - Lecture 3: Operation on Ordered Binary Decision Diagram; IIT Guwahati Lectures, https://archive.nptel.ac.in/courses/106/103/106103116/

Thank you!
Any questions??