COL:750

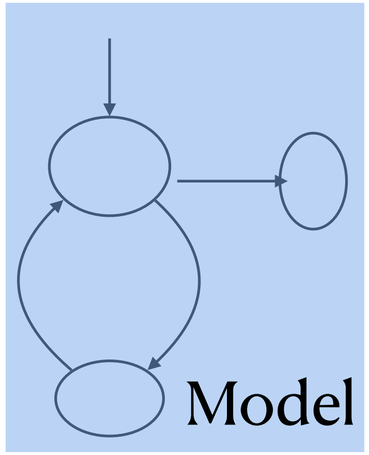
Foundations of Automatic Verification

Instructor: Priyanka Golia

Course Webpage 

https://priyanka-golia.github.io/teaching/COL-750/index.html

Does Code Satisfy Requirements ?

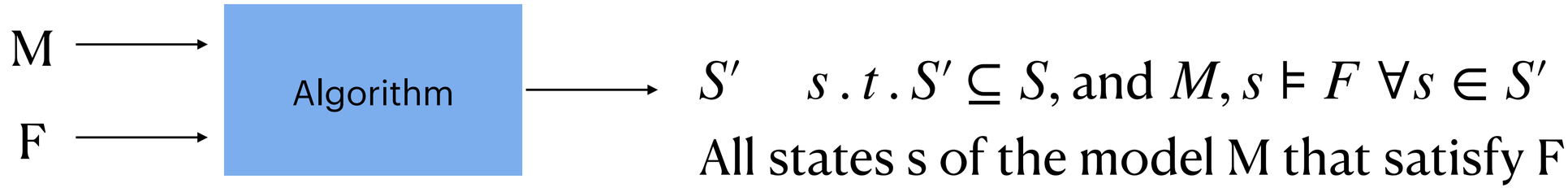


Does Model Satisfy Logical formulation: LTL/CTL Formula ?

Model Checking

Model Checking Algorithm

$$M, s \models F?$$



Note that not necessarily $I \subseteq S'$

Labelling Algorithm —

1. Does not scale well to large systems due to state explosion.
2. Memory-intensive as it maintains explicit labels for each state.

We need better data structure.

CTL Model Checking Algorithm — BDD basedAlgorithm

1. Input — a Model M, and a CTL formula F.
2. Output — S’ (the set of states of M that satisfy the formula F.)

BDD — Binary Decision Diagrams.

BDD — Binary Decision Diagrams

A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

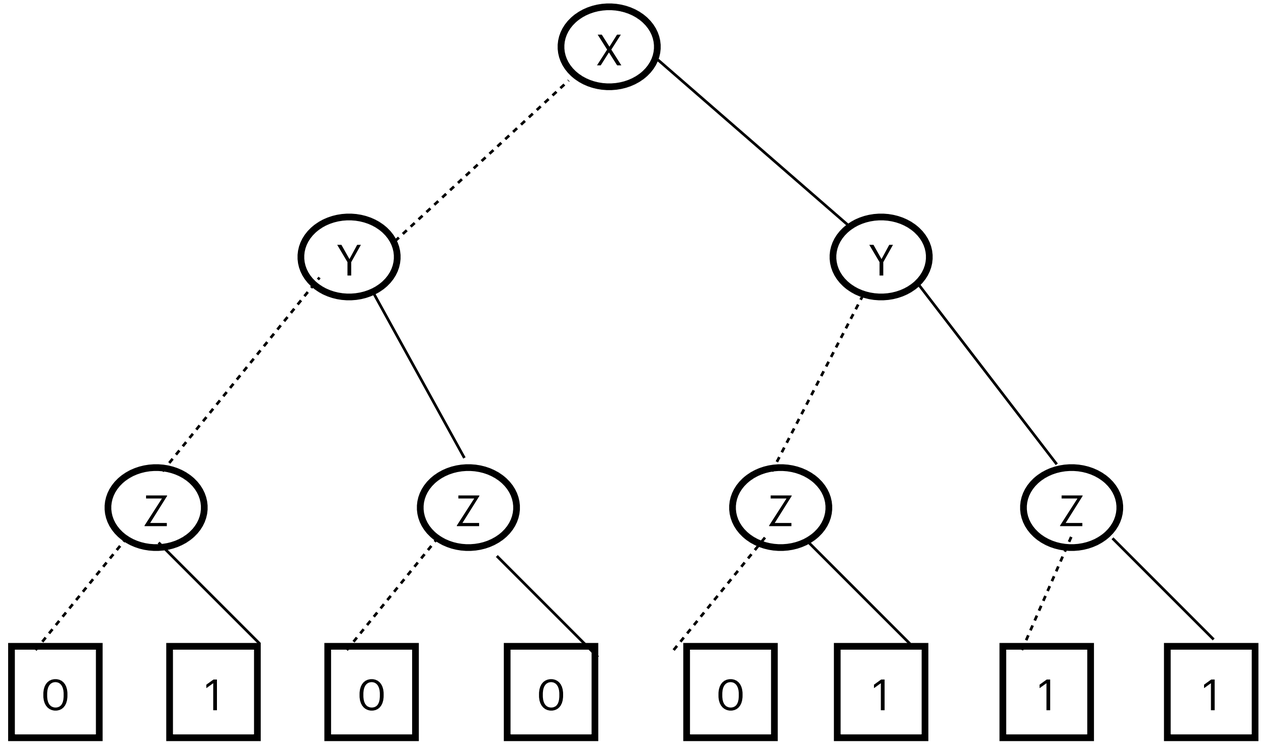
BDD — Binary Decision Diagrams

A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Binary Decision Diagram

BDD — Binary Decision Diagrams

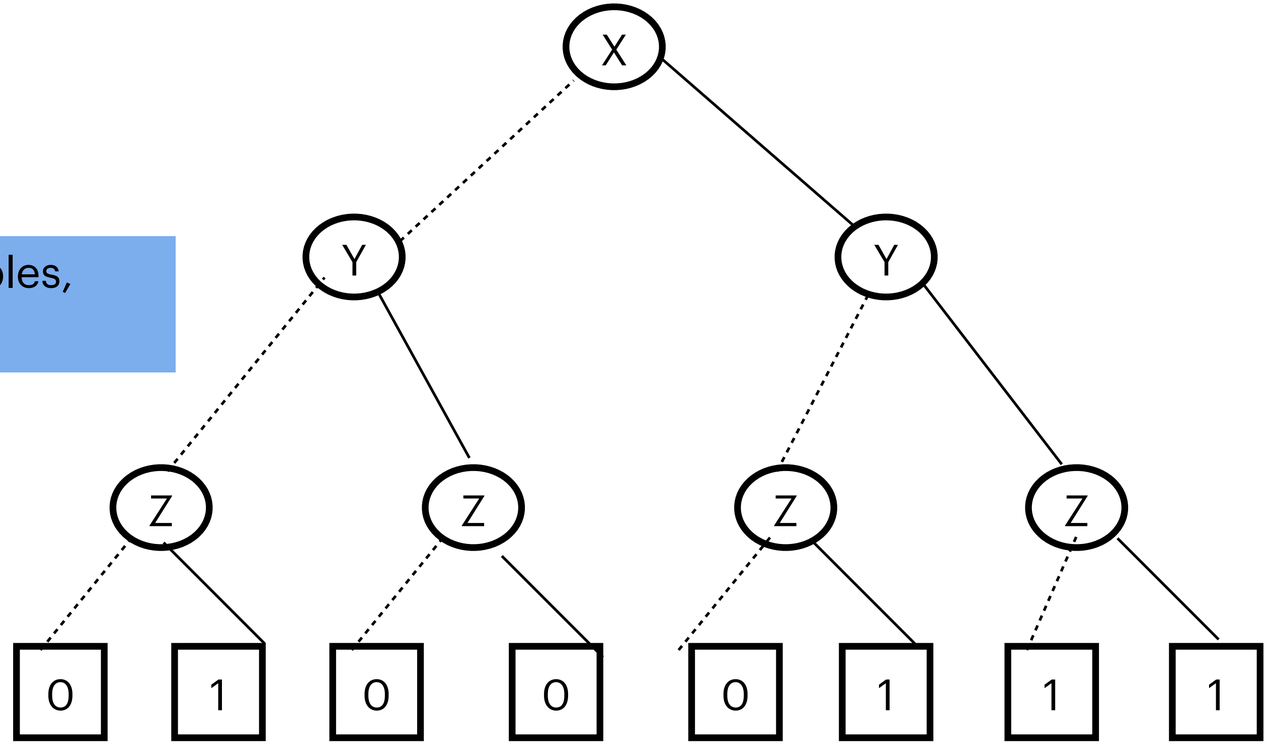
A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Space: for n variables,
$O(2^{n+1} - 1)$



Binary Decision Diagram

BDD — Binary Decision Diagrams

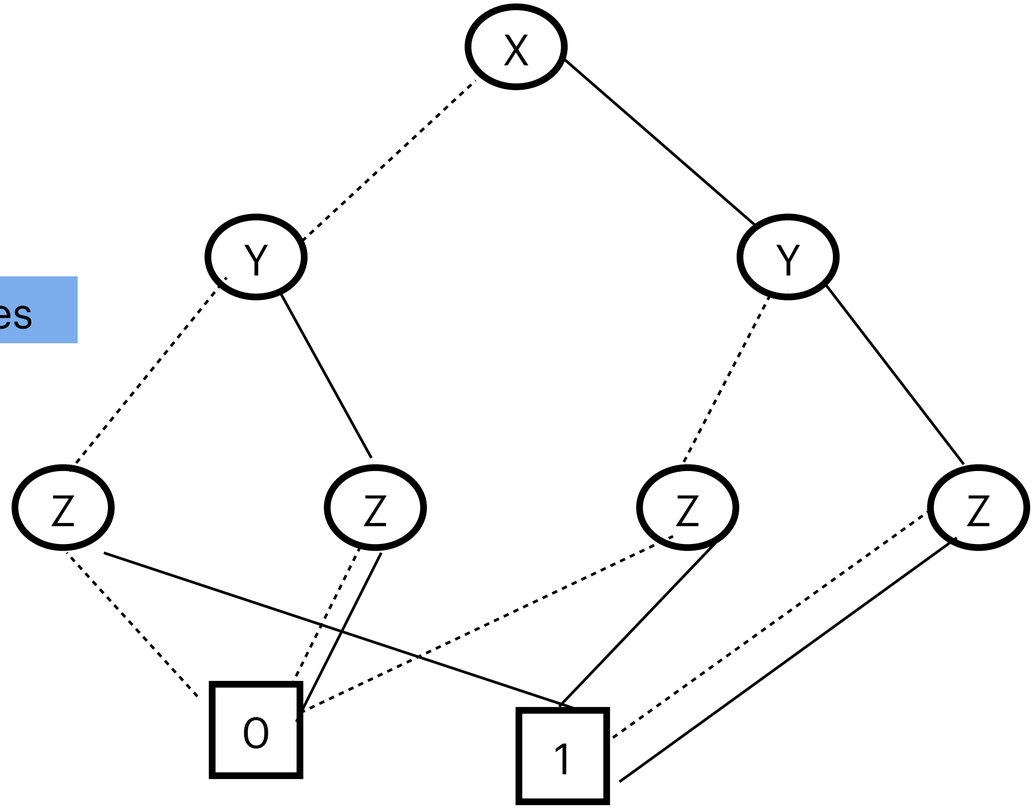
A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Removal of duplicate leaves



Binary Decision Diagram

BDD — Binary Decision Diagrams

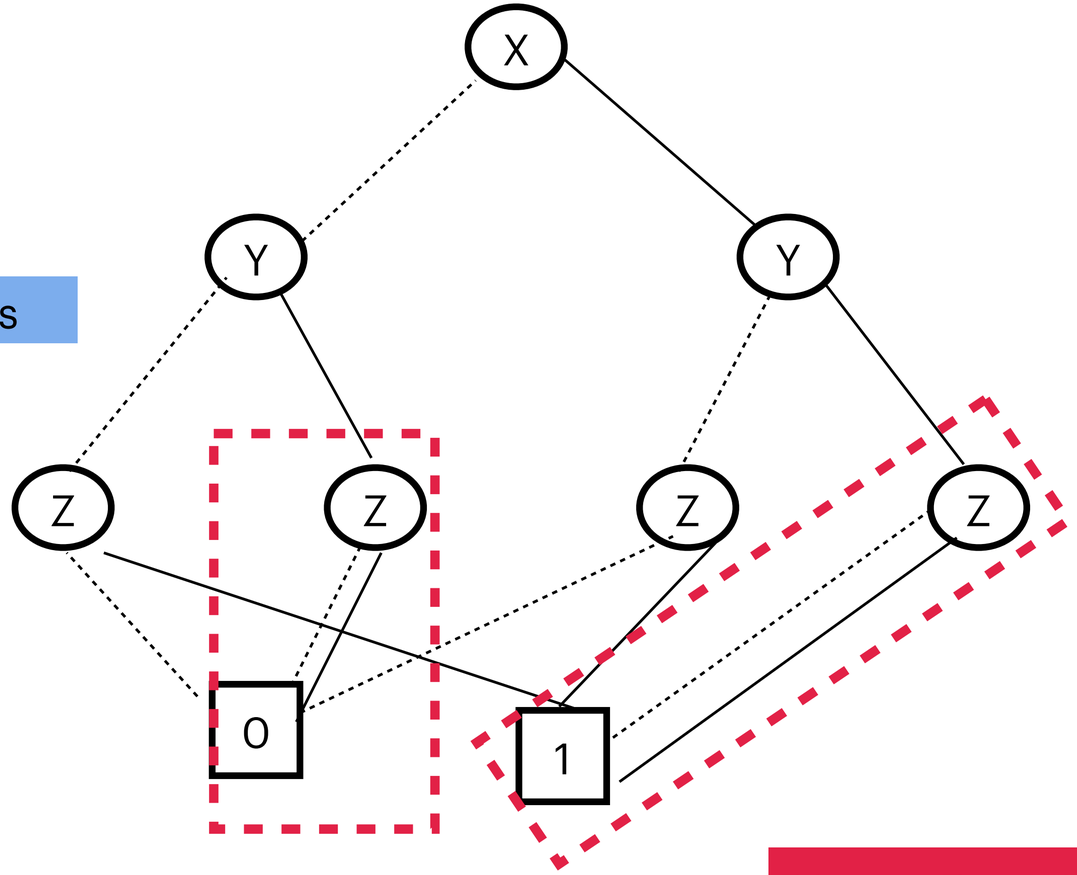
A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Removal of duplicate tests



Binary Decision Diagram

BDD — Binary Decision Diagrams

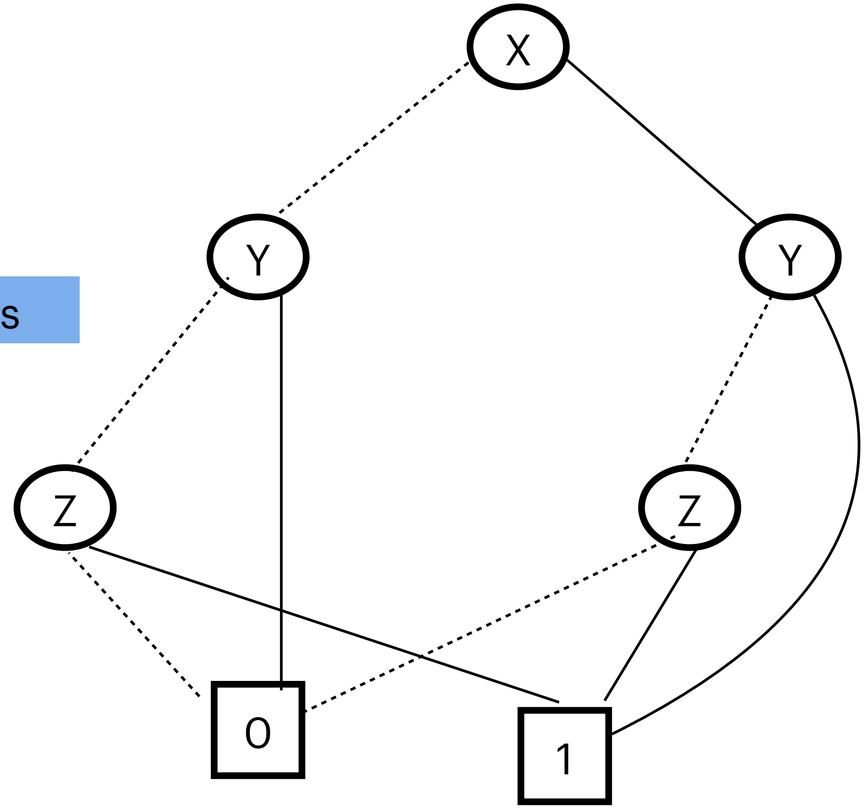
A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Removal of duplicate tests



Binary Decision Diagram

BDD — Binary Decision Diagrams

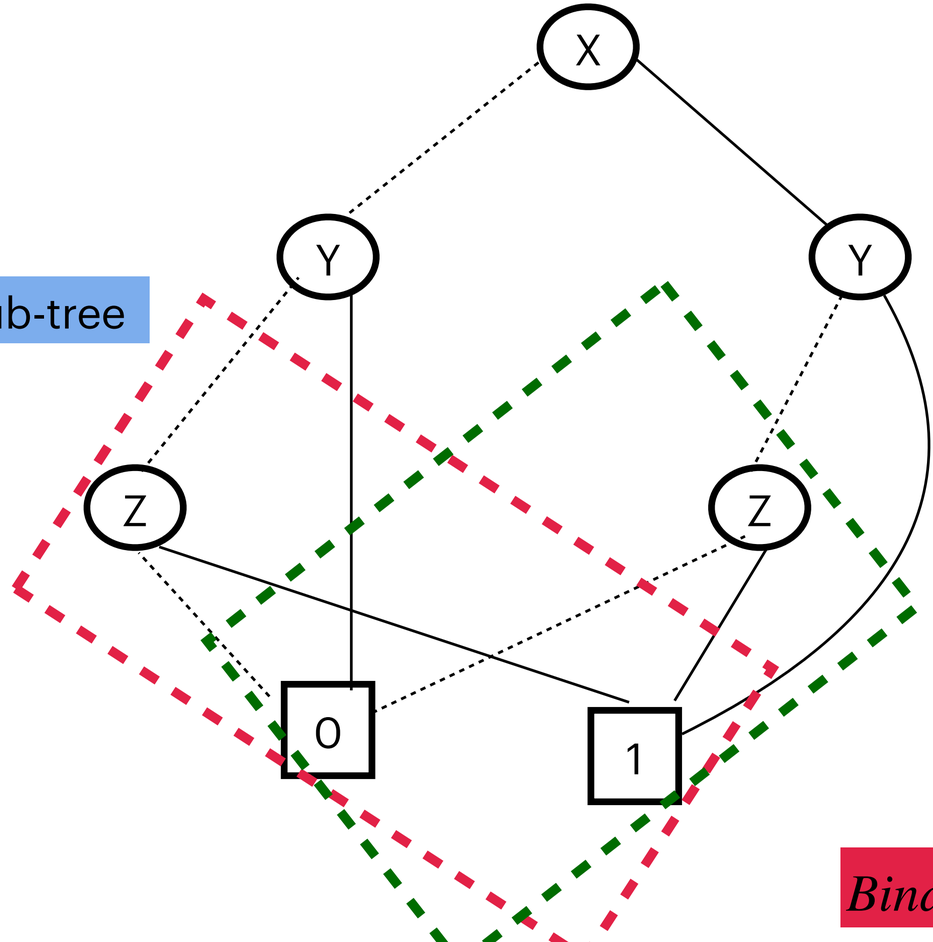
A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Removal of duplicate sub-tree



Binary Decision Diagram

BDD — Binary Decision Diagrams

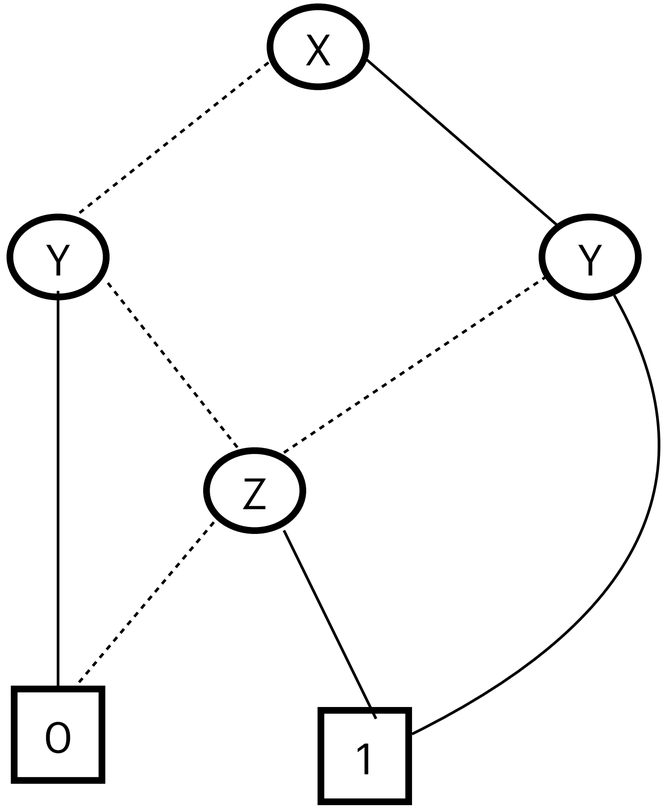
A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. $\{x, y, z, \ldots, \} \rightarrow \{0,1\}$

BDDs can be considered as a compressed representation of sets or relations.

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Removal of duplicate sub-tree

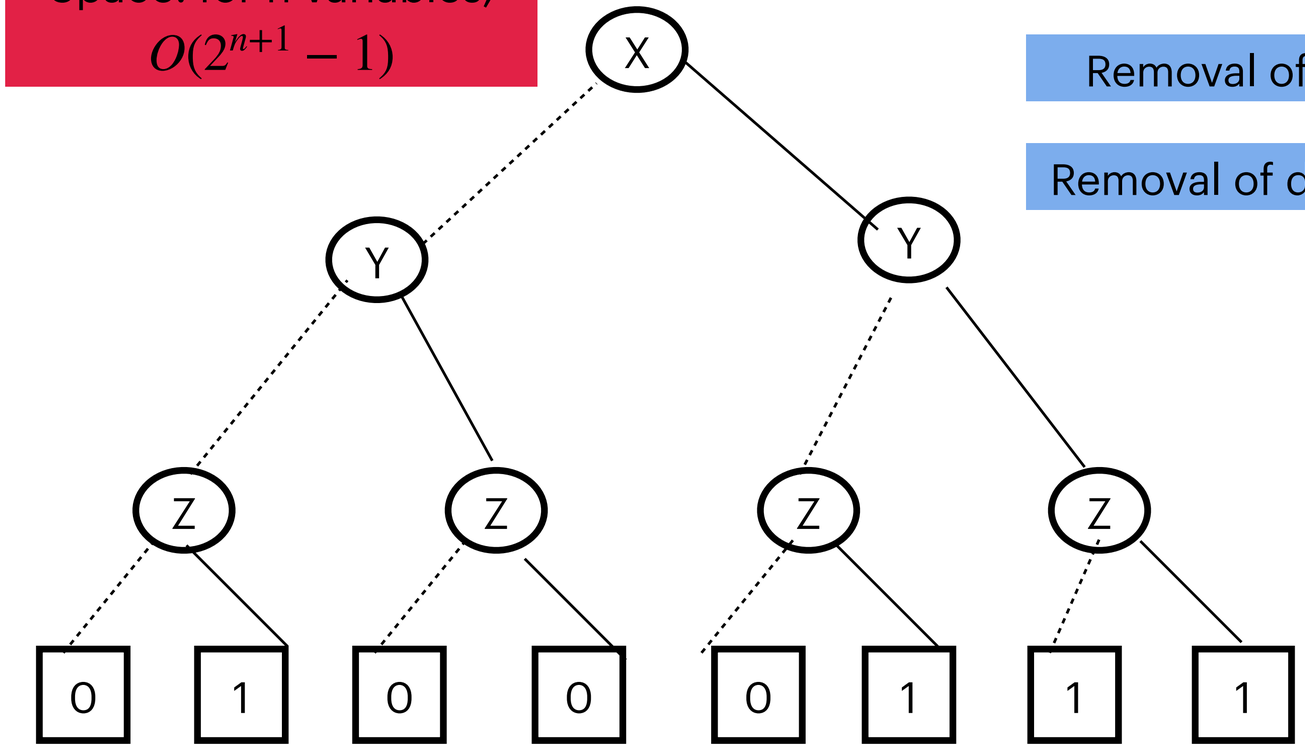


Binary Decision Diagram

RBDD — Reduced Binary Decision Diagrams

$$F = (x \wedge y) \vee (\neg y \wedge z)$$

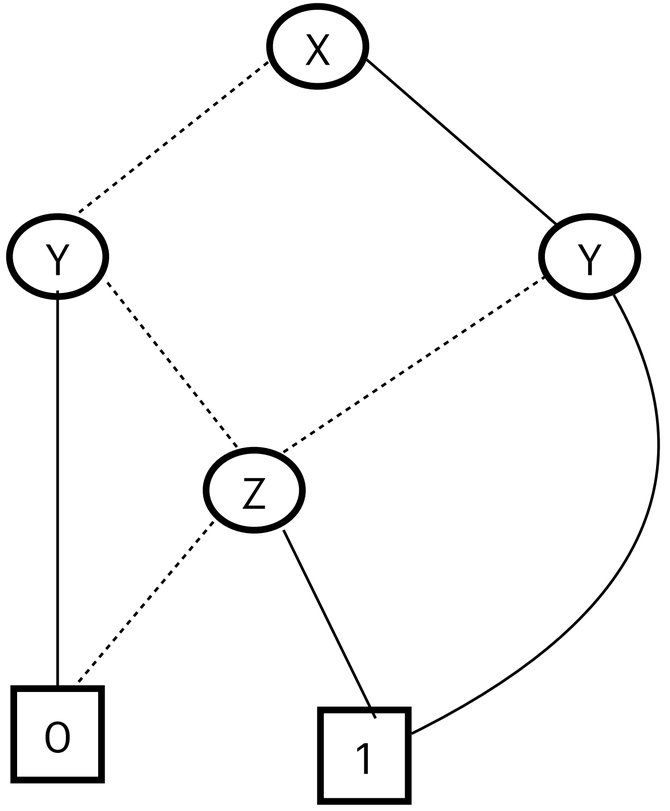
Space: for n variables, $O(2^{n+1} - 1)$



Removal of duplicate leaves

Removal of duplicate tests

Removal of duplicate sub-tree



BDD — Binary Decision Diagrams

$$F(x_1, x_2, x_3, x_4) = \begin{cases} 1 - \text{if even number of variables are } 1 \\ \\ 0 - \text{otherwise} \end{cases}$$

Create a ROBDD.

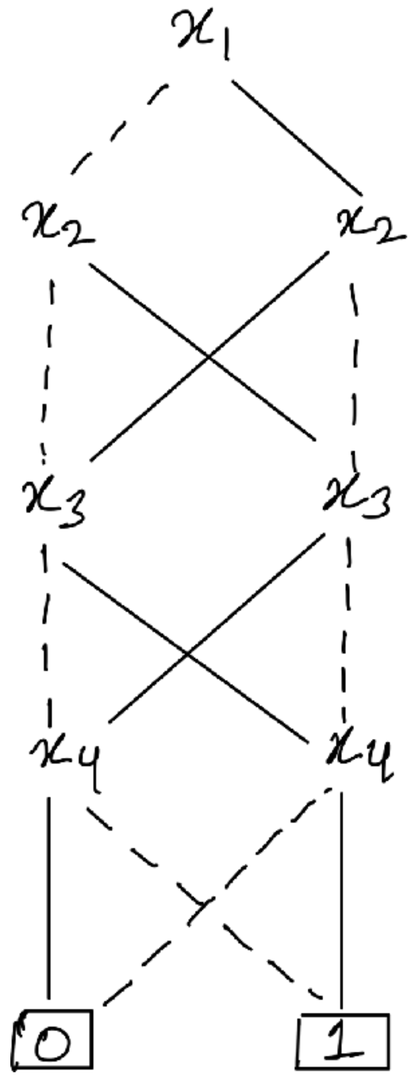
Assuming order to be x_1, x_2, x_3, x_4

ROBDD — Reduced Ordered Binary Decision Diagrams

$$F(x_1, x_2, x_3, x_4) = \begin{cases} 1 \text{ — if even number of variables are } 1 \\ 0 \text{ — otherwise} \end{cases}$$

Create a ROBDD.

Assuming order to be x_1, x_2, x_3, x_4

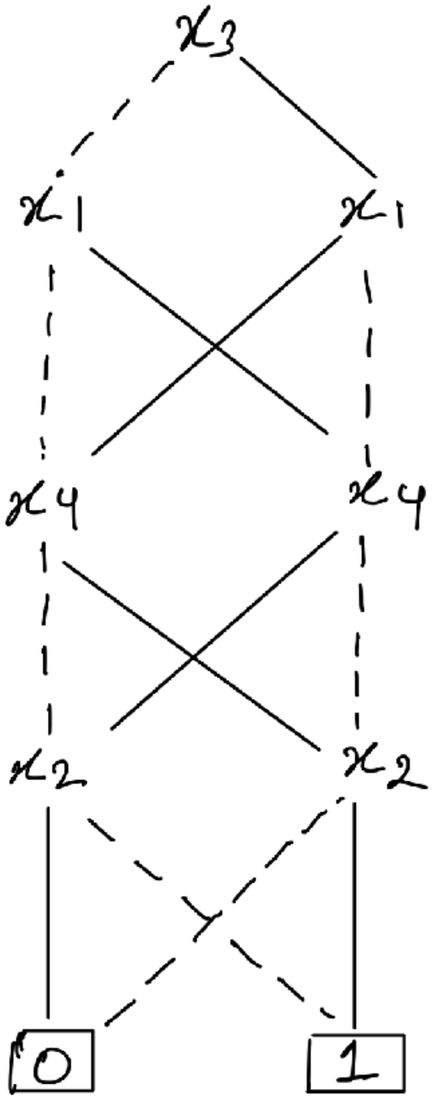


BDD — Binary Decision Diagrams

$$F(x_1, x_2, x_3, x_4) = \begin{cases} 1 - \text{if even number of variables are } 1 \\ 0 - \text{otherwise} \end{cases}$$

Create a ROBDD.

Assuming order to be x_3, x_1, x_4, x_2

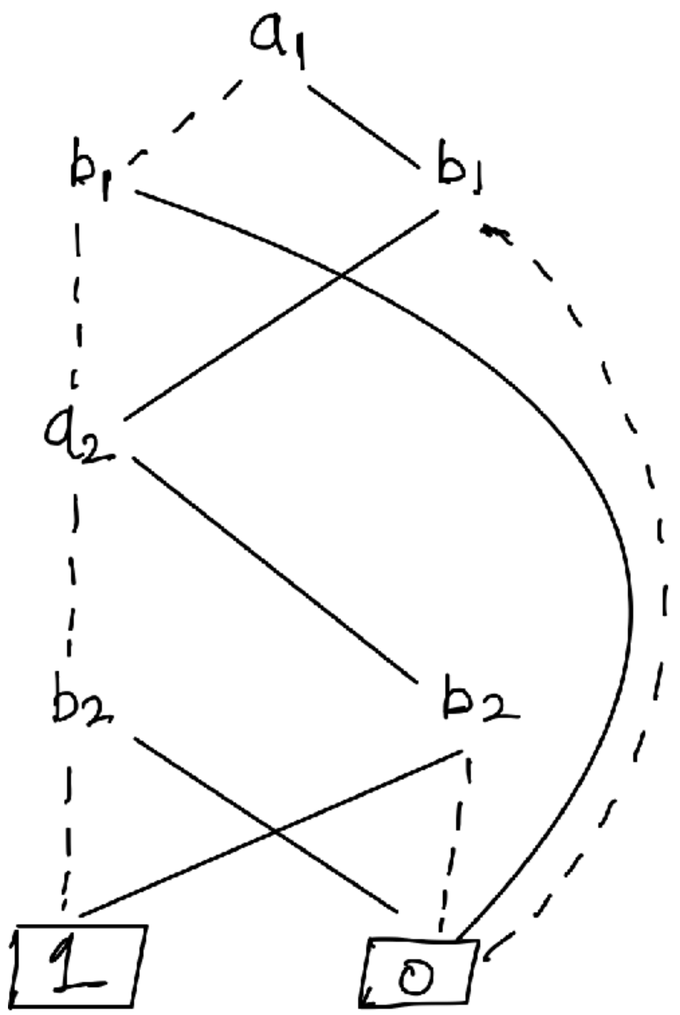


ROBDD — Reduced Ordered Binary Decision Diagrams

$$F = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$

Create a ROBDD.

Assuming order to be a_1, b_1, a_2, b_2



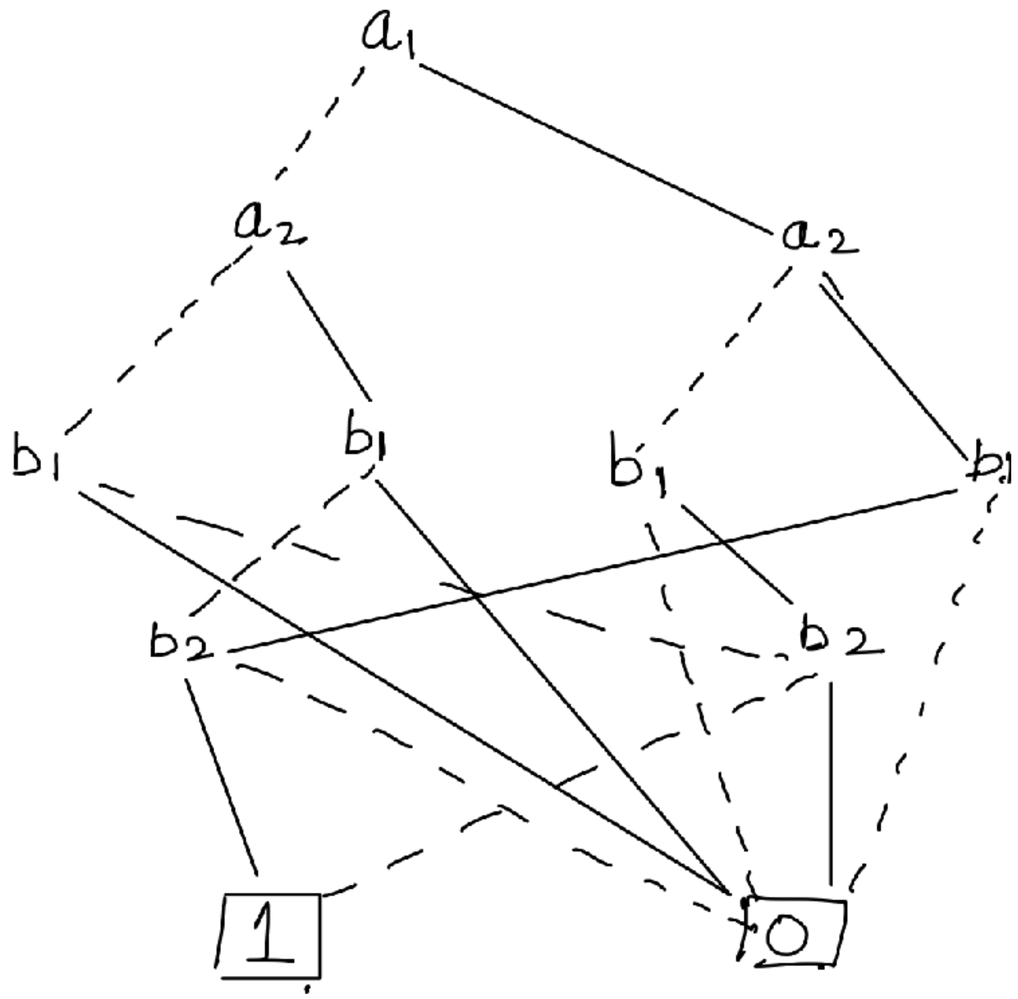
Number of nodes $2n + n + 2$

ROBDD — Reduced Ordered Binary Decision Diagrams

$$F = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$

Create a ROBDD.

Assuming order to be a_1, a_2, b_1, b_2



Number of nodes $3 \times 2^n - 1$

ROBDD — Reduced Ordered Binary Decision Diagrams

For an n-bit comparator:

if we use the ordering $< a_1, b_1, a_2, b_2, \ldots, a_n, b_n >$, the number of vertices will be $3n + 2$.

if we use the ordering $< a_1, a_2, \ldots, a_n, b_1, b_2 \ldots, b_n >$, the number of vertices is $3 \times 2^n - 1$.

Moreover, there are boolean functions that have exponential size OBDDs for any variable ordering.

An example is the middle output (nth output) of a combinational circuit to multiply two n bit integers

Given an order, ROBDD is always unique

ROBDD Operations

Assuming two ROBDDs over same variable ordering.

Given argument functions f and g, and a binary operator ,

- **APPLY** returns the function F <op> G.
- Works by traversing the argument graphs depth first.

Expanding for any variable x

$$F < op > G = \neg x(F|_{x=0} < op > G|_{x=0}) + x(F|_{x=1} < op > G|_{x=1})$$

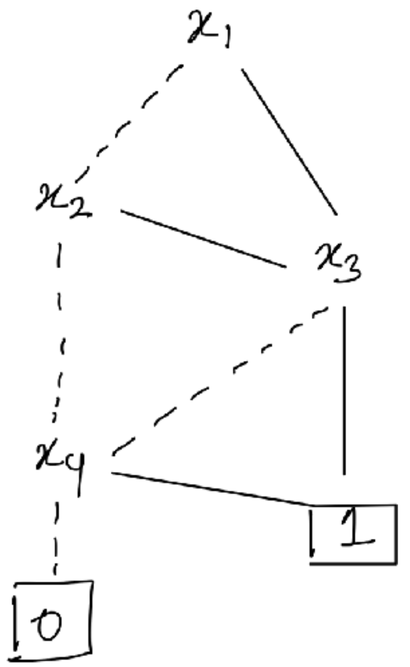
ROBDD Operations

Assuming two ROBDDs over same variable ordering.

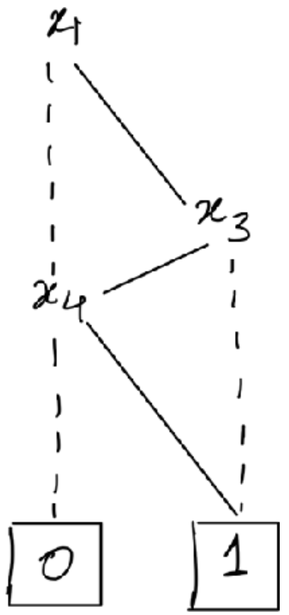
Given argument functions f and g, and a binary operator ,

- **APPLY** returns the function F <op> G.
- Works by traversing the argument graphs depth first.

Expanding for any variable x



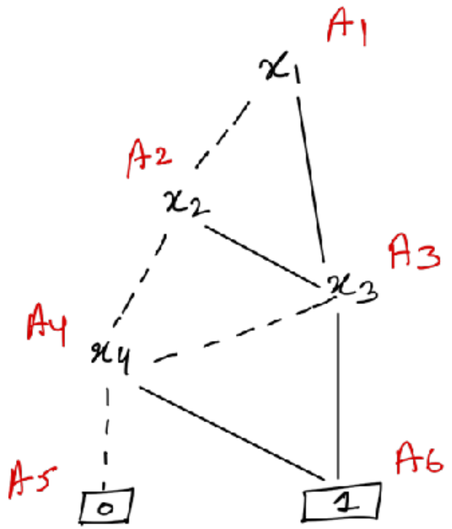
$F(x_1, x_2, x_3, x_4)$

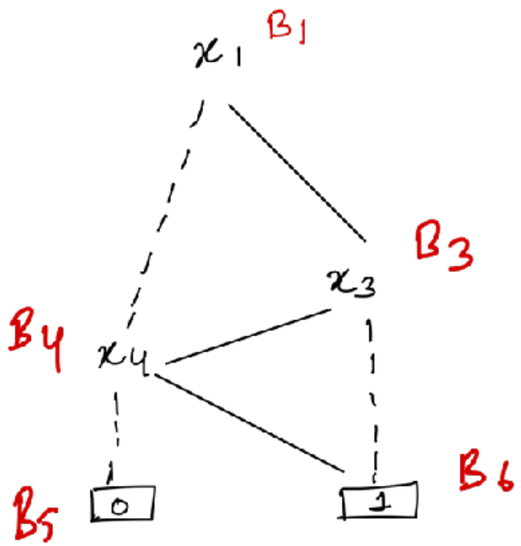


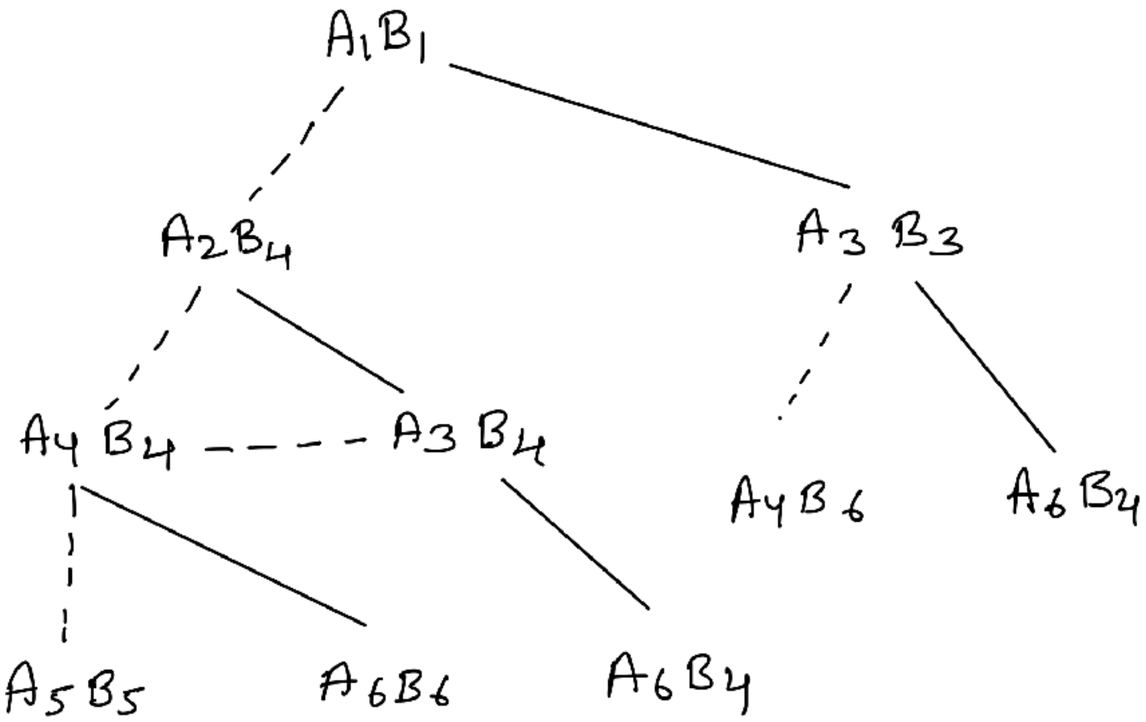
$G(x_1, x_2, x_3, x_4)$

How about $F \vee G$?

ROBDD Operations (Apply)





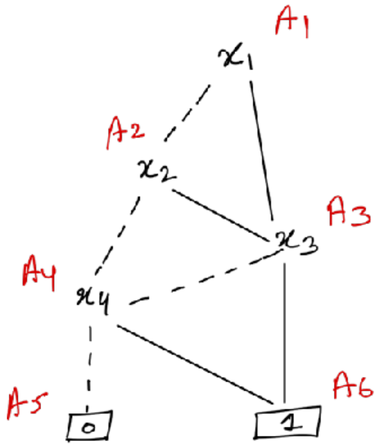


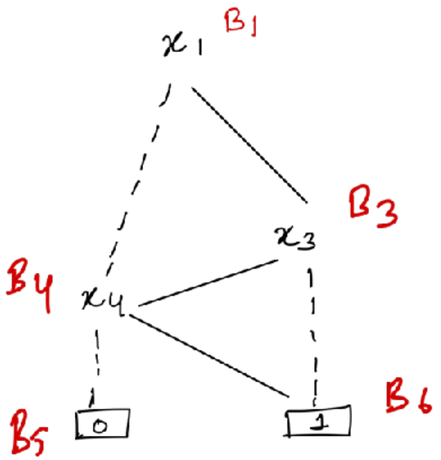
1. Depth first search — respect the ordering.

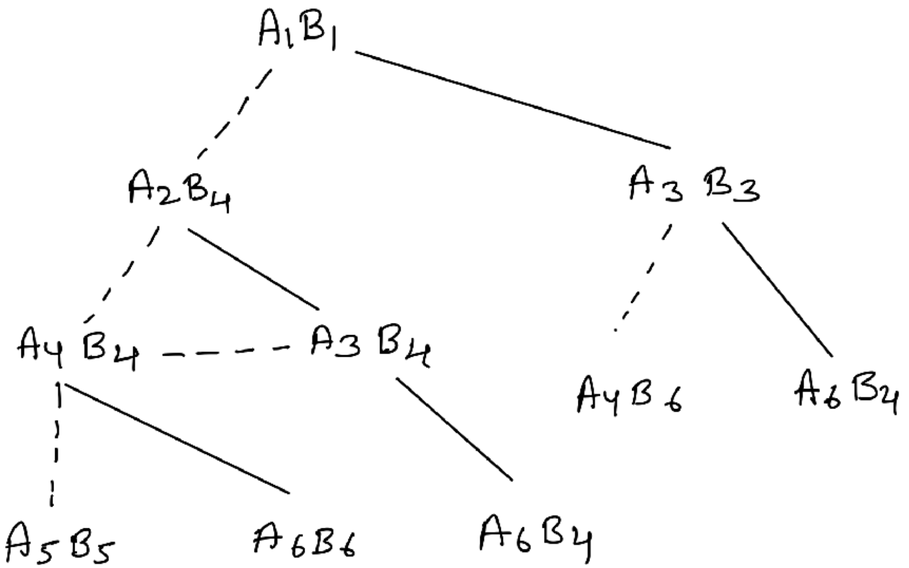
2. Reaching a terminal with a dominant value (e.g 1 for OR, 0 for AND) terminates recursion and returns an appropriately labeled terminal

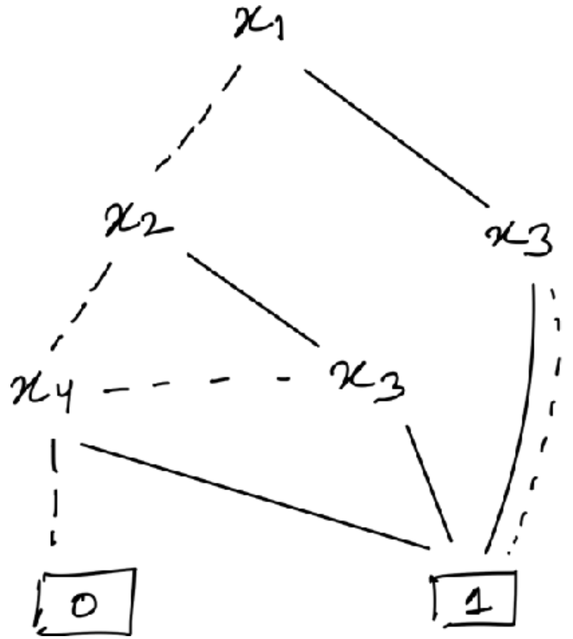
3. Avoid multiple recursive calls on the same pair of arguments by a hash table

ROBDD Operations (Apply)

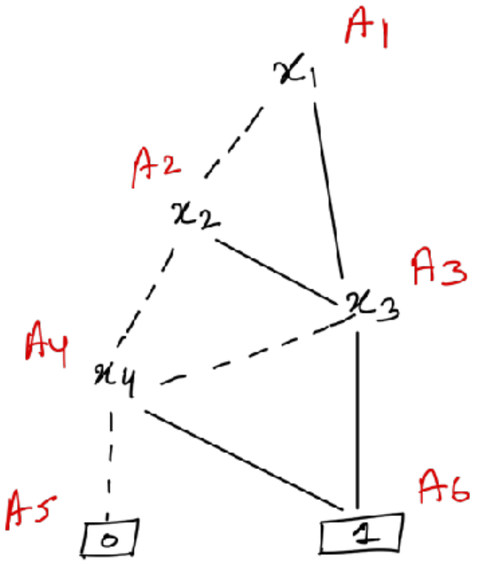




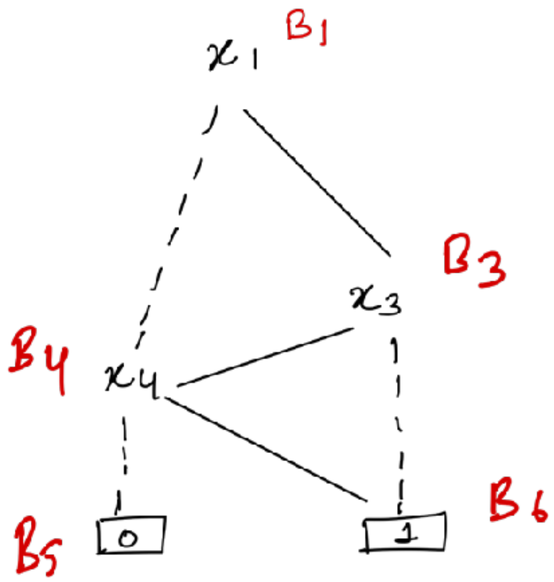




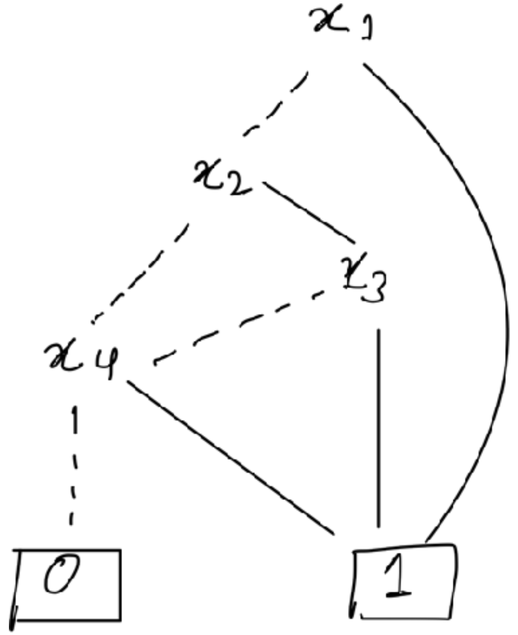
ROBDD Operations (Apply)



F



G



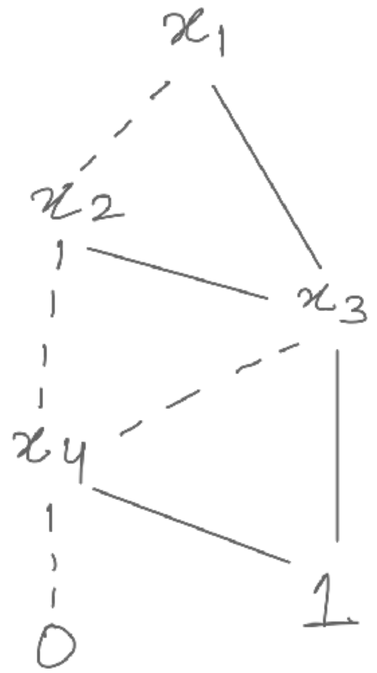
$F \vee G$

ROBDD Operations (Restrict)

Effect to setting a function argument x_i to a constant 0/1

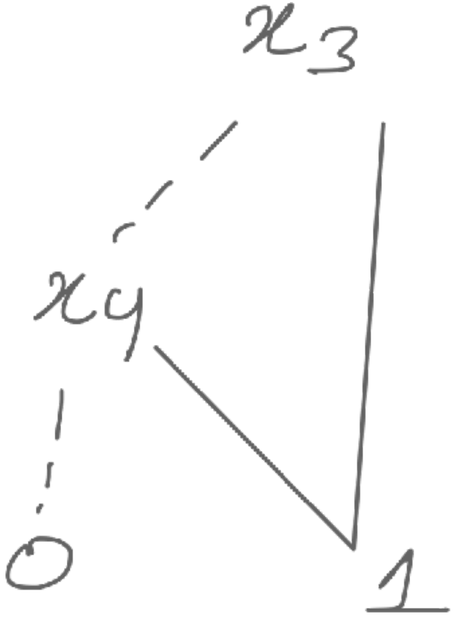
Depth-first traversal.

Redirecting arcs according to constant.



F

$F[x_2 = 1]$



$F[x_2 = 1]$

ROBDD Operations (Exists (x,F))

Compute ROBDD for $\exists xF$

1. Uses the identity:

$$\exists xF \equiv F[x = 0] \vee F[x = 1]$$

2. Realized using the restrict and apply functions

$$\text{Apply}(\vee, Restrict(x,0,F), Restrict(x,1,F))$$

$$\exists x_1, x_2 F \equiv ?$$

$$\exists x_1, x_2 F \equiv F(x_1,1,x_3, \ldots, x_n) \vee F(x_1,0,x_3, \ldots, x_n)$$

$$\exists x_1, x_2 F \equiv F(1,1,x_3, \ldots, x_n) \vee F(0,1,x_3, \ldots, x_n) \vee F(1,0,x_3, \ldots, x_n) \vee F(0,0,x_3, \ldots, x_n)$$

ROBDD Operations

$\text{Apply(op, F, G)} : O(|F|) \times O(|G|)$

$\text{Restrict(x, F, b} = \{0,1\}) : O(|F|)$

Exists(x, F) :?