NPTEL Phase-II
Video course on

Design Verification and Test of Digital VLSI Designs

Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
IIT Guwahati

Module VI: Binary Decision Diagram

Lecture I: Binary Decision Diagram:
Introduction and construction

- Model checking algorithm
  - Polynomial in the size of state machine and the length of the formula
- Problem with model checking
  - State space explosion problem

Binary Decision Diagrams (BDD)

- Based on recursive Shannon expansion

$$f = x f_x + x' f_{x'}$$

- Compact data structure for Boolean logic

  can represents sets of objects (states) encoded as Boolean functions

- Canonical representation

  reduced ordered BDDs (ROBDD) are canonical

Shannon Expansion

$$f = x\, f_x + x'\, f_{x'}$$

$$f = ac + bc$$

$$f_{a'} = f(a{=}0) = bc$$

$$f_a = f(a{=}1) = c + bc$$

$$\begin{aligned} f &= a\, f_a + a'\, f_{a'} \\ &= a(c+bc) + a'(bc) \end{aligned}$$

Binary Decision Tree (BDT)

- Binary Decision Trees are trees whose non-terminal nodes are labeled with Boolean variables x, y, z, …. and whose terminal nodes are labeled with either 0 or 1.

Binary Decision Tree (BDT)

- Each non-terminal node has two edges, one dashed line and one solid line.
- Dashed line represents 0 and solid line represents 1.

Binary Decision Tree

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

Truth table

$f = ac + bc$

Truth Table $\rightarrow$ BDT

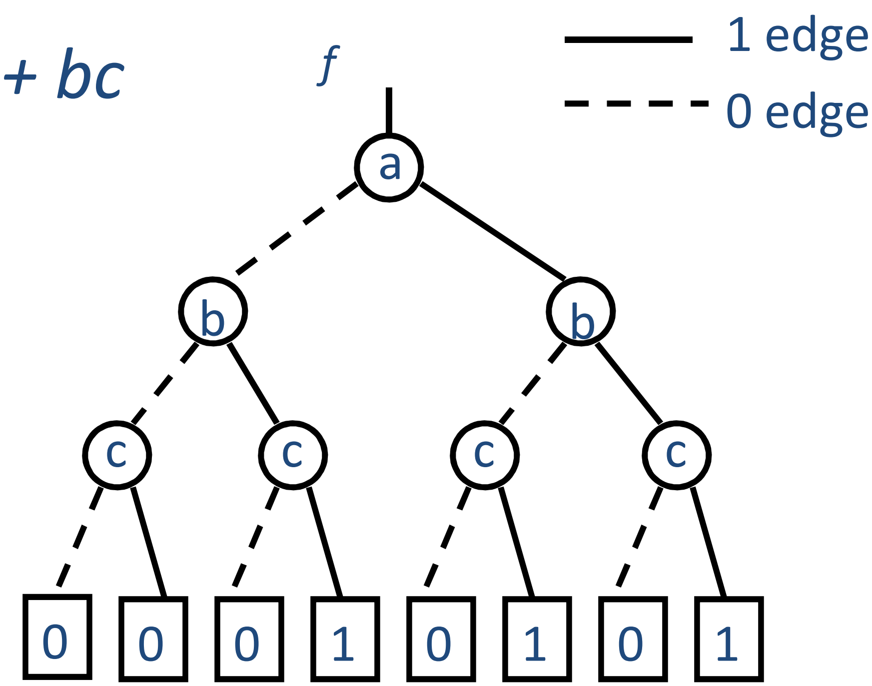
Binary Decision Tree

Truth Table $\rightarrow$ BDT

$$f = ac + bc$$

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

Truth table

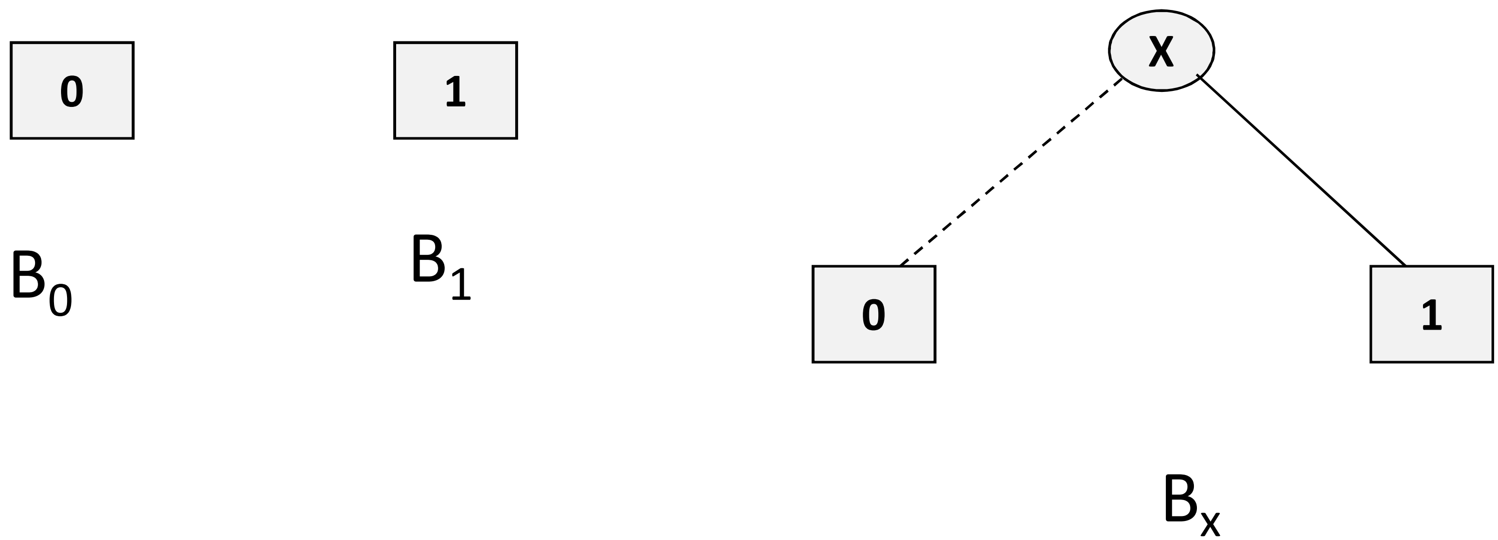


Decision tree

Binary Decision Diagram

- A Binary Decision Diagram (BDD) is a finite DAG with an unique initial node, where
  - all terminal nodes are labeled with 0 or 1
  - all non-terminal nodes are labeled with a Boolean Variable.
  - Each non-terminal node has exactly two edges from that node to others; one labeled 0 and one labeled 1; represent them as a dashed line and a solid line respectively

Binary Decision Diagram



B_0 representing the Boolean constant 0

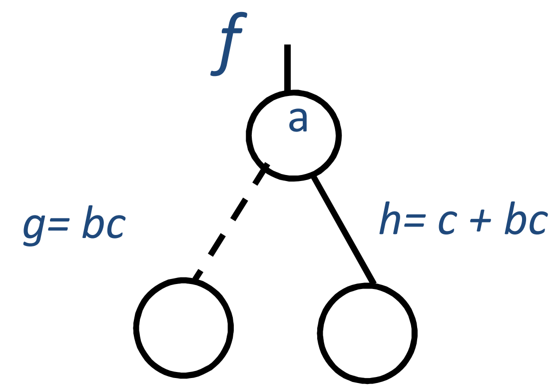
B_1 representing the Boolean constant 1

B_x representing the Boolean variable x

Shannon Expansion $\rightarrow$ BDD

$$f = ac + bc \qquad\qquad f = x f_x + x' f_{x'}$$

- $f_{a'} = f(a=0) = bc = g$
- $f_a = f(a=1) = c + bc = h$

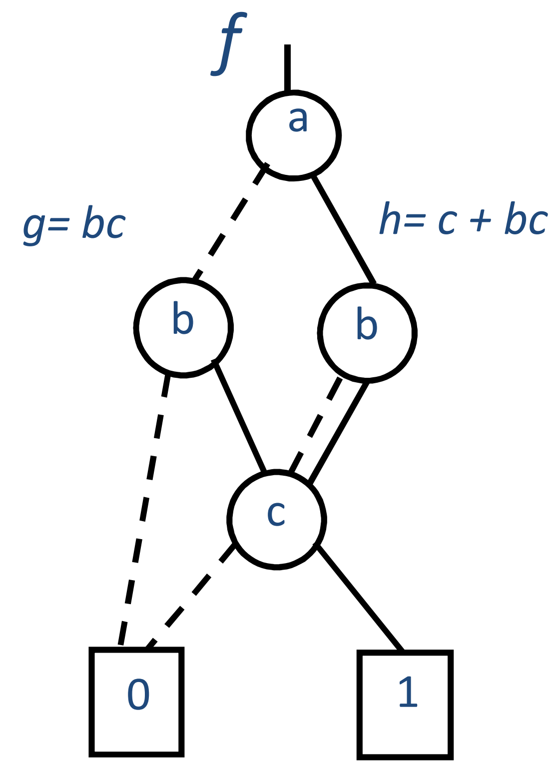


Shannon Expansion → BDD

$f = ac + bc$

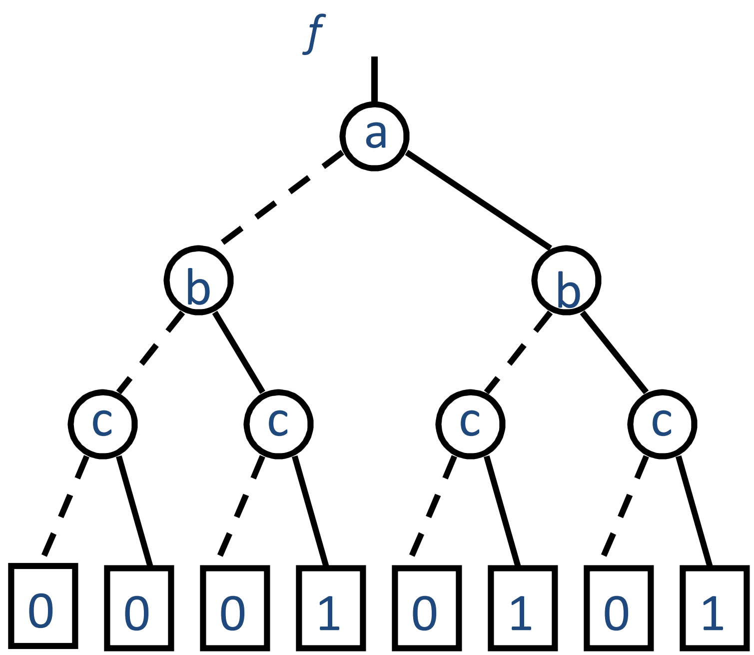
$f = x f_x + x' f_{x'}$

- $f_{a'} = f(a=0) = bc = g$
- $f_a = f(a=1) = c + bc = h$
- $g_{b'} = (bc)_{|b=0} = 0$
- $g_b = (bc)_{|b=1} = c$
- $h_{b'} = (c+bc)_{|b=0} = c$
- $h_b = (c+bc)_{|b=1} = c$

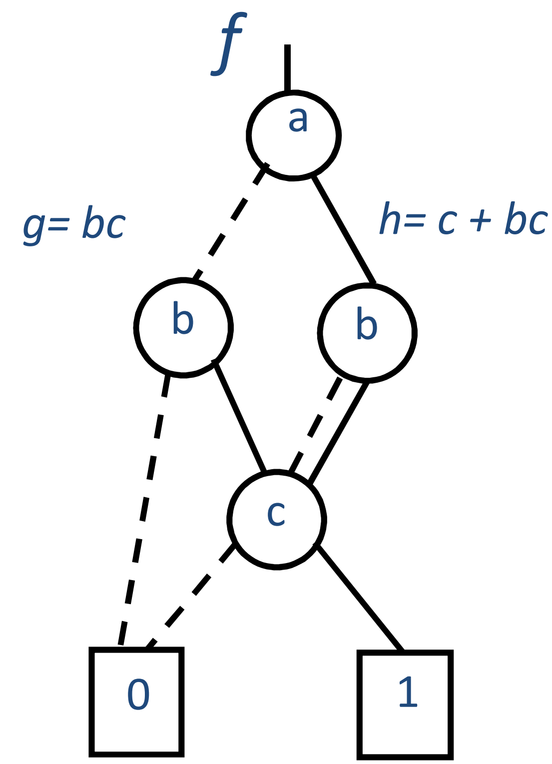


Binary Decision Tree and Diagram

$$f = ac + bc$$



From Truth Table



From Shannon Expression

BDD Reduction Rules -1

Eliminate *duplicate terminals*

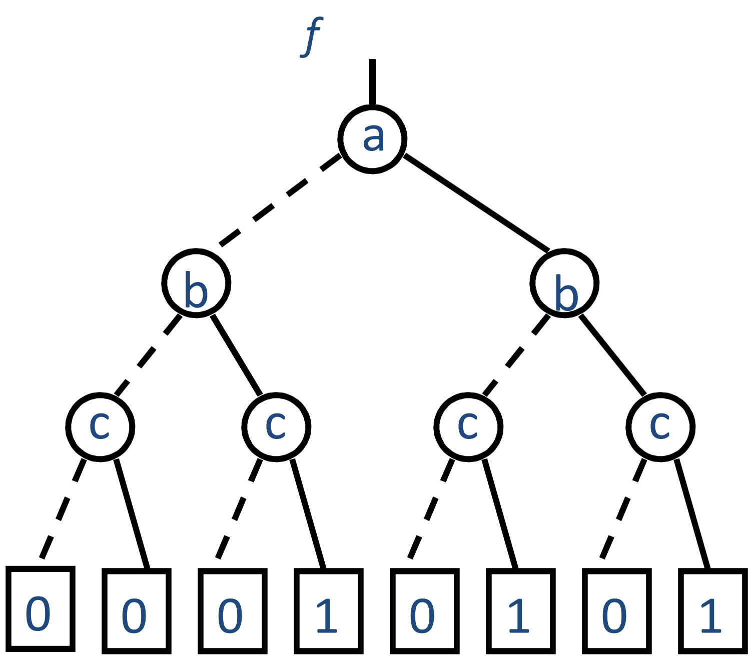
- If a BDD contains more than one terminal 0-node, then we redirect all edges which point to such a 0-node to just one of them.
- Similarly, we proceed for nodes labeled with 1.

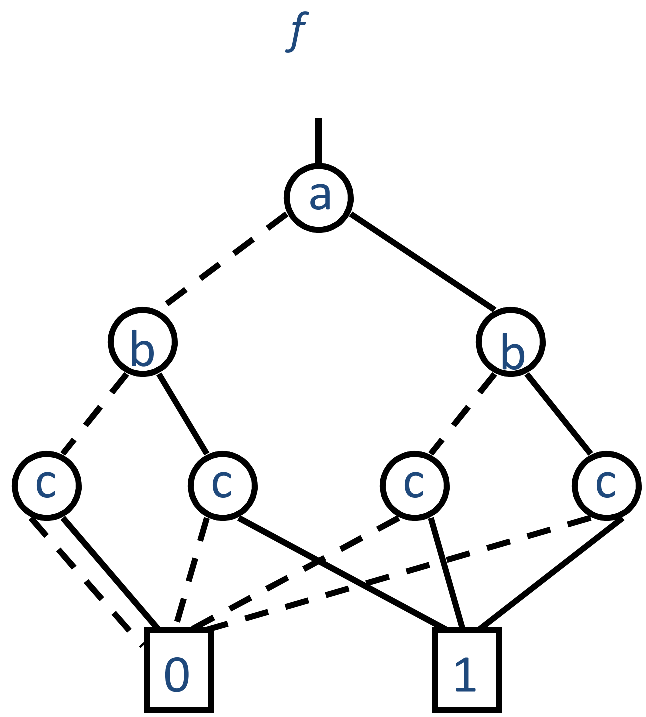
BDD Reduction Rules -1

Eliminate *duplicate terminals*

If a BDD contains more than one terminal 0-node, then we redirect all edges which point to such a 0-node to just one of them.

Similarly, we proceed for nodes labeled with 1.

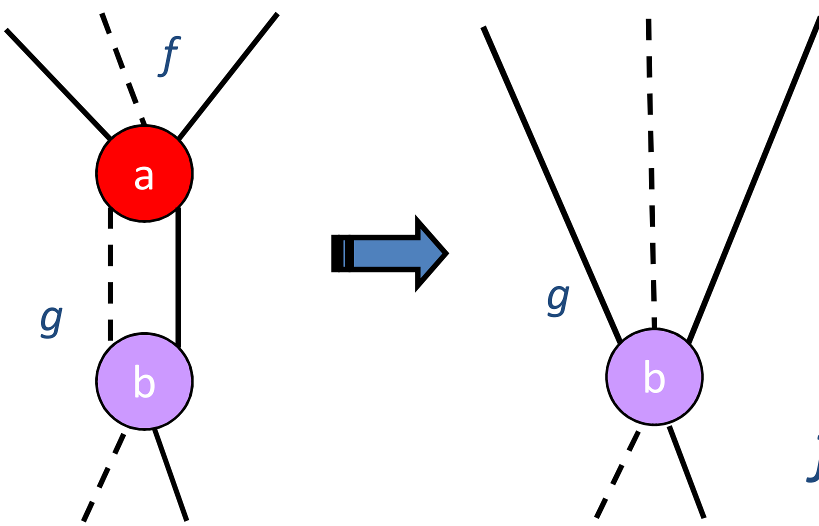




BDD Reduction Rules -2

Eliminate *redundant* nodes

(with both edges pointing to same node)



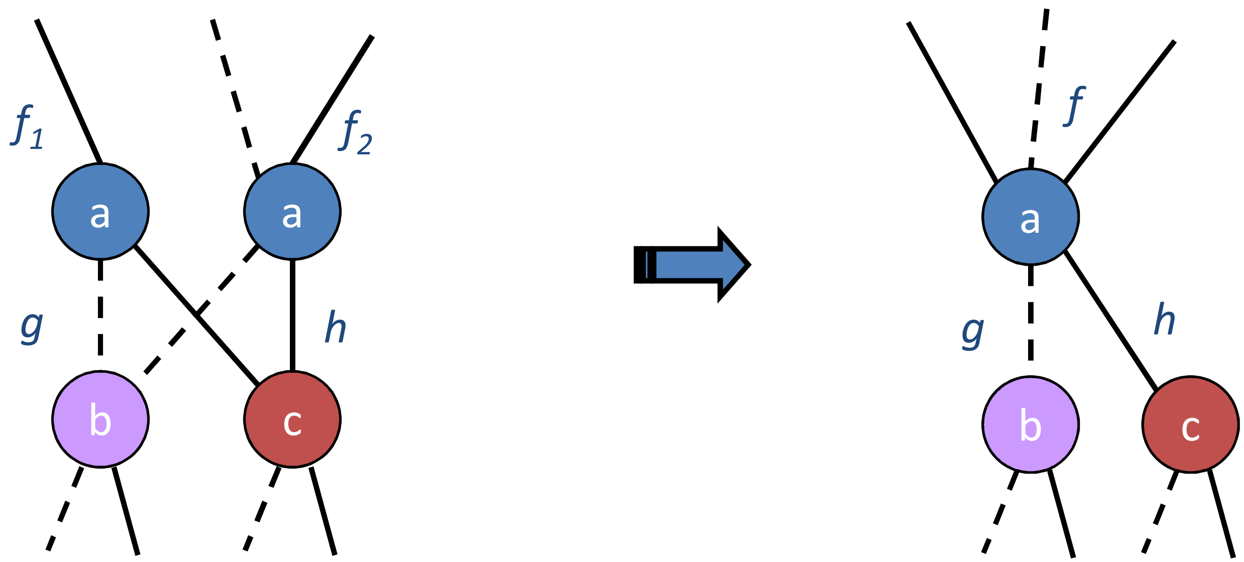
$f = a' \, g(b) + a \, g(b) = g(b)$

$(f_a + f_{a'} = 1)$

BDD Reduction Rules -3

Merge duplicate nodes

- Nodes must be unique



$f_1 = a' \, g(b) + a \, h(c) = f_2$

$f = f_1 = f_2$

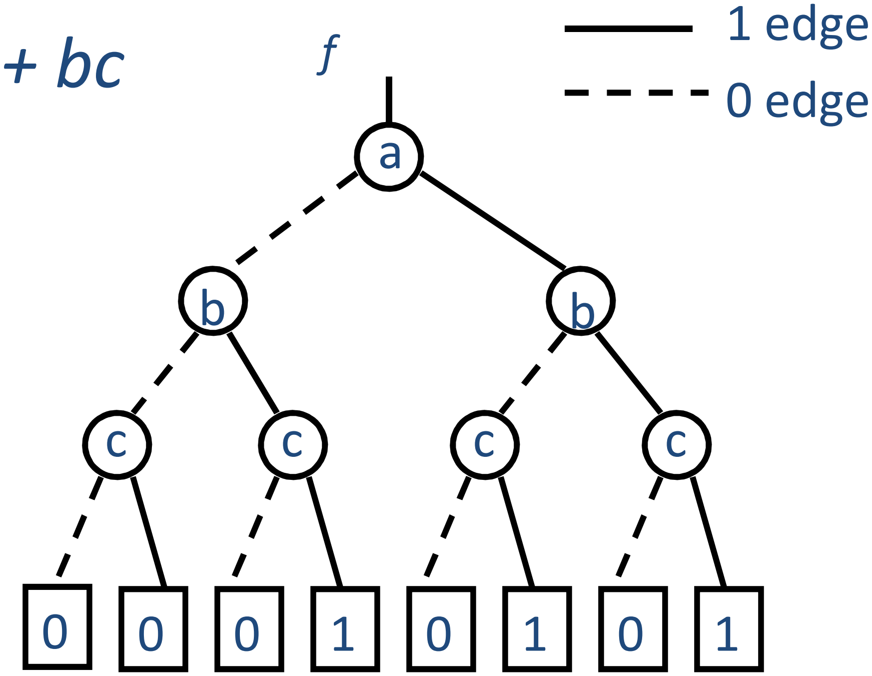
BDD Construction

- Reduced BDD

$$f = ac + bc$$

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

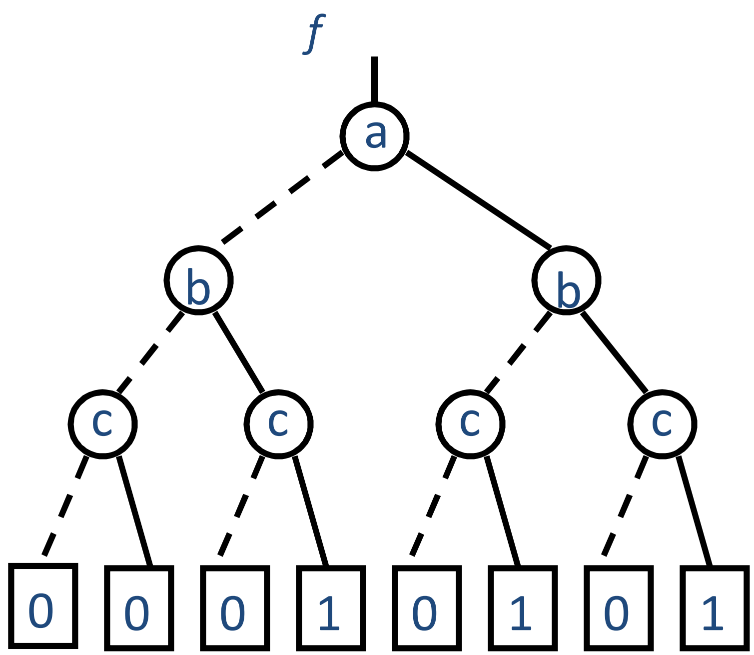
Truth table

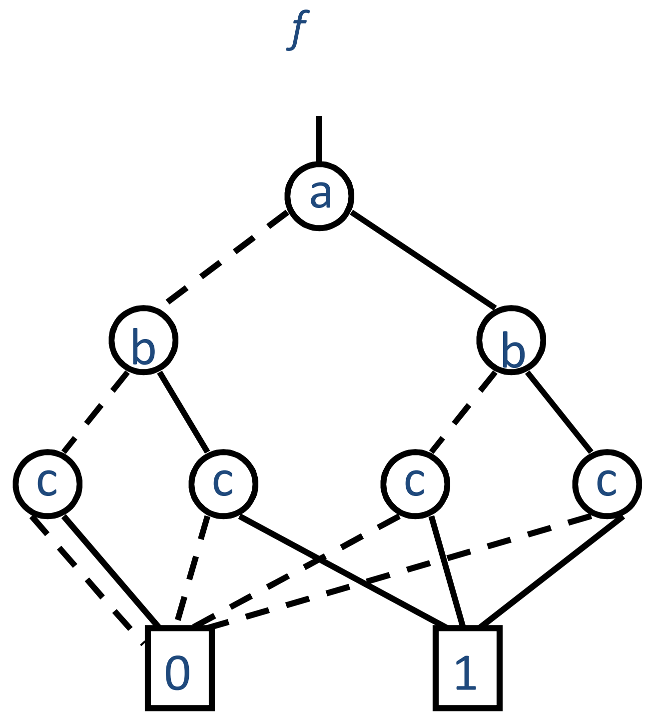


Decision tree

BDD Reduction

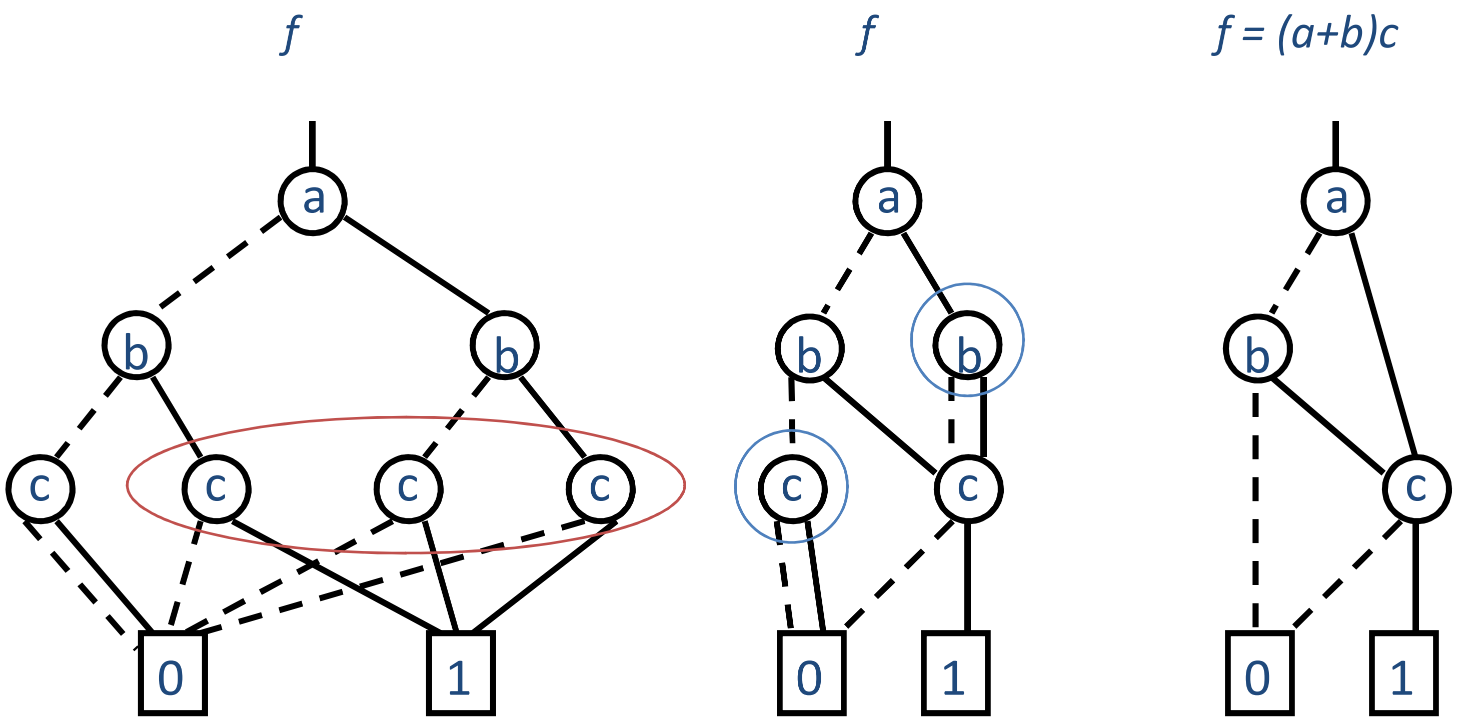
$f = ac + bc$





1. Merge terminal nodes

BDD Construction – cont'd



2. Merge duplicate nodes

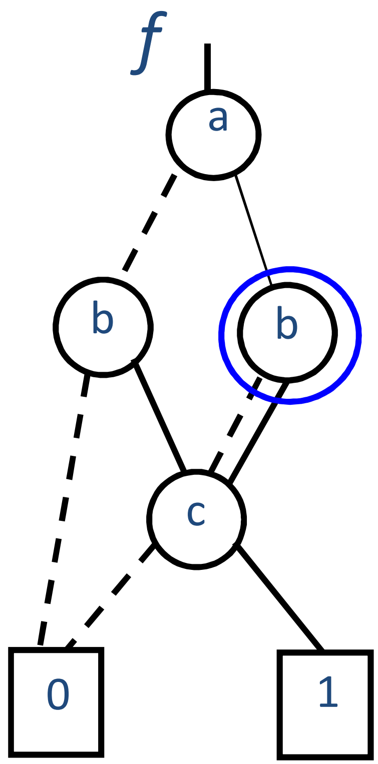
3. Remove redundant nodes

Reduced BDD

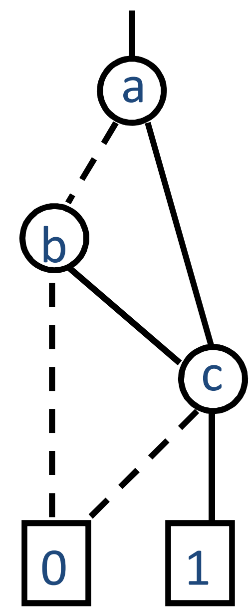
BDD Construction – cont'd

BDD constructed by Shannon Expression

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



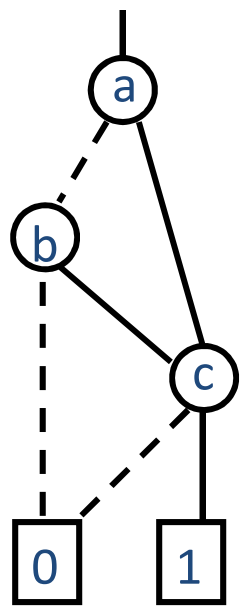
3. Remove redundant nodes

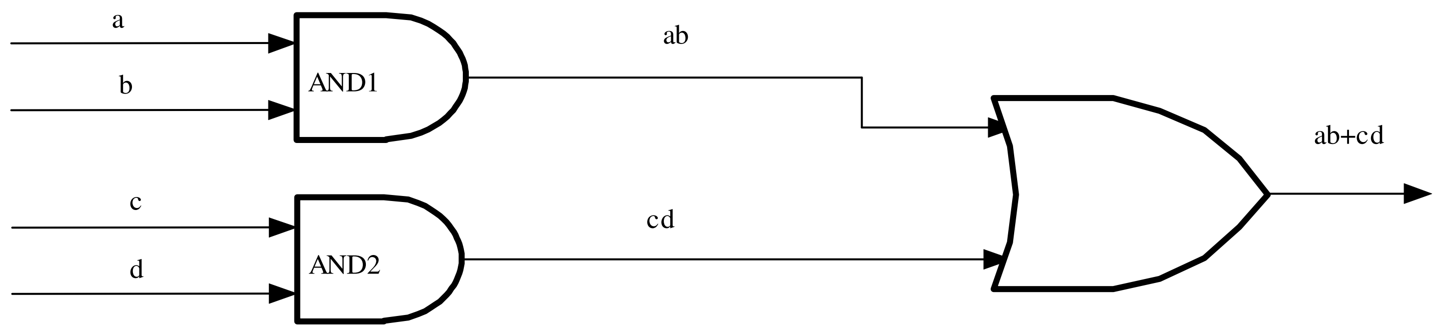


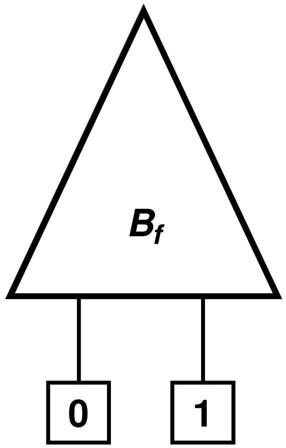
Reduced BDD

Reduced BDDs

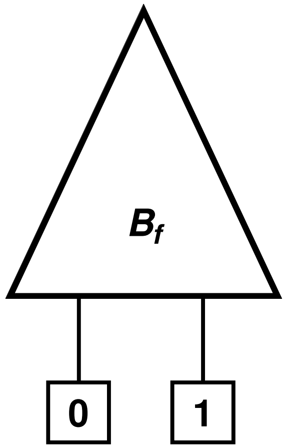
A BDD is said to be reduced if none of the reduction rules R1-R3 can be applied (i.e., no more reductions are possible)



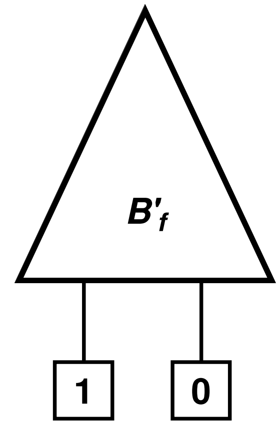

a
b
AND1
ab
c
d
AND2
cd
ab+cd



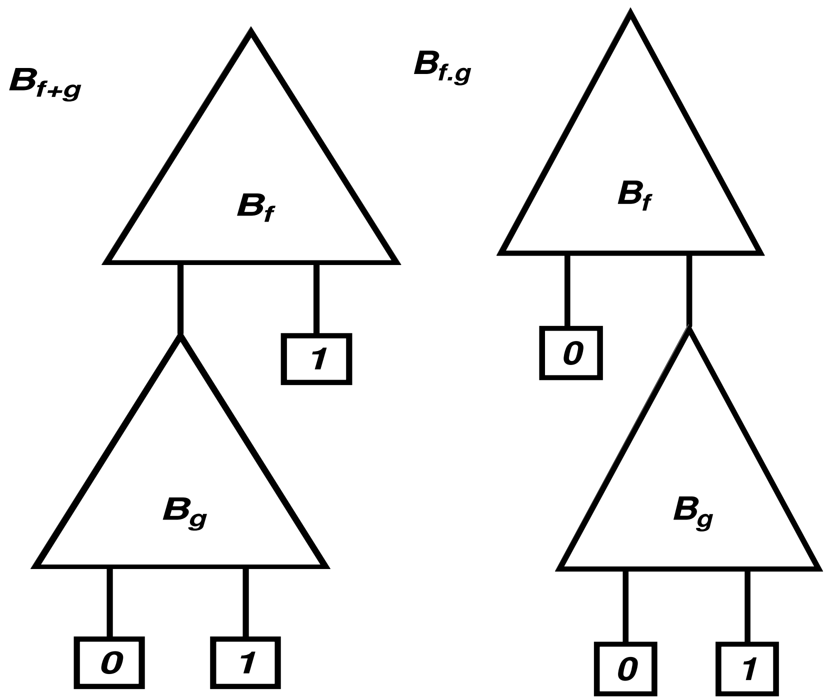
BDD B_f for Boolean function f



BDD B_f for Boolean function f



BDD B'_f for Boolean function f’



BDDs for f+g and f.g

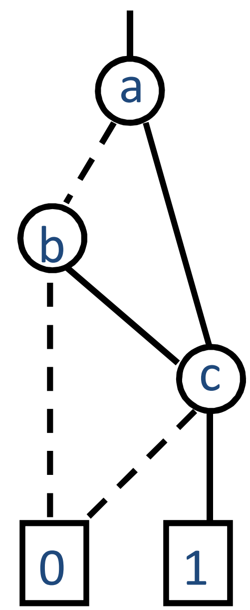
Questions

1. Do we get any advantage in using BDT.
2. While constructing the BDD, is it required to start from BDT.
3. The definition of BDD does not restrict the occurrence of a variable in any number of times in a path. Show that it may lead to inconsistency with an example.
4. Is reduced BDD of any function is unique.

Shannon Expansion → BDD

$f = ac + bc$ $\qquad$ $f = x f_x + x' f_{x'}$

- $f_{a'} = f(a{=}0) = bc = g$
- $f_a = f(a{=}1) = c + bc = h$
- $g_{b'} = (bc)_{|b=0} = 0$
- $g_b = (bc)_{|b=1} = c$
- $h_{b'} = (c+bc)_{|b=0} = c$
- $h_b = (c+bc)_{|b=1} = c$

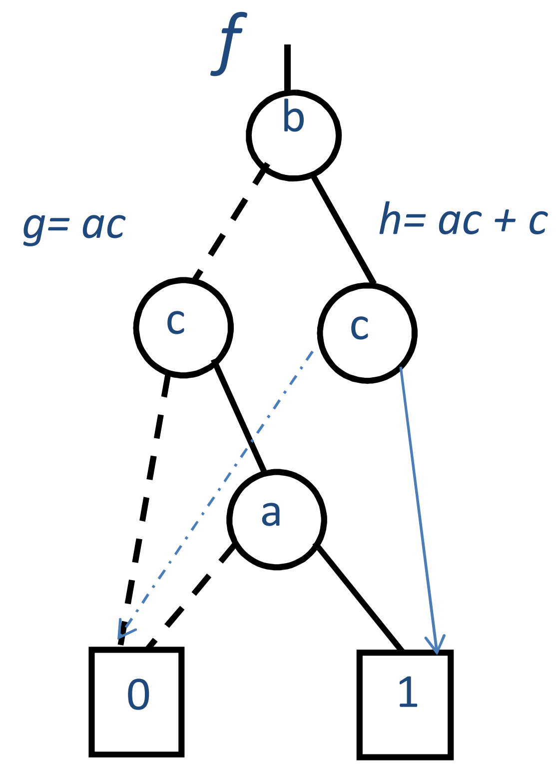


Shannon Expansion → BDD

$f = ac + bc$

$f = x f_x + x' f_{x'}$

- $f_{b'} = f(b{=}0) = ac = g$
- $f_b = f(b{=}1) = ac + c = h$
- $g_{c'} = (ac)_{|c=0} = 0$
- $g_c = (ac)_{|c=1} = a$
- $h_{c'} = (ac+c)_{|c=0} = 0$
- $h_c = (ac+c)_{|c=1} = 1$



NPTEL Phase-II
Video course on

Design Verification and Test of Digital VLSI Designs

Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
IIT Guwahati

Module VI: Binary Decision Diagram

Lecture II: : Ordered Binary Decision Diagram

Binary Decision Diagram

- Construction of BDD
- Reduced BDD

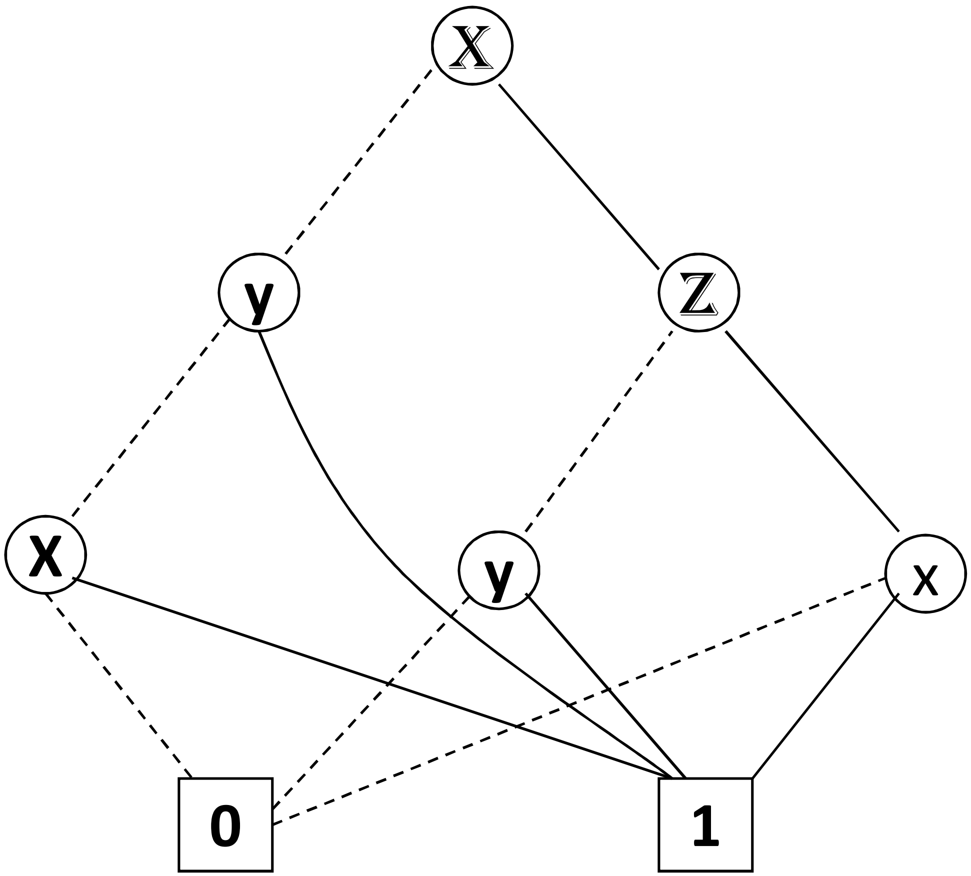
Binary Decision Diagram

- Occurrence of variables
- Ordering of variables

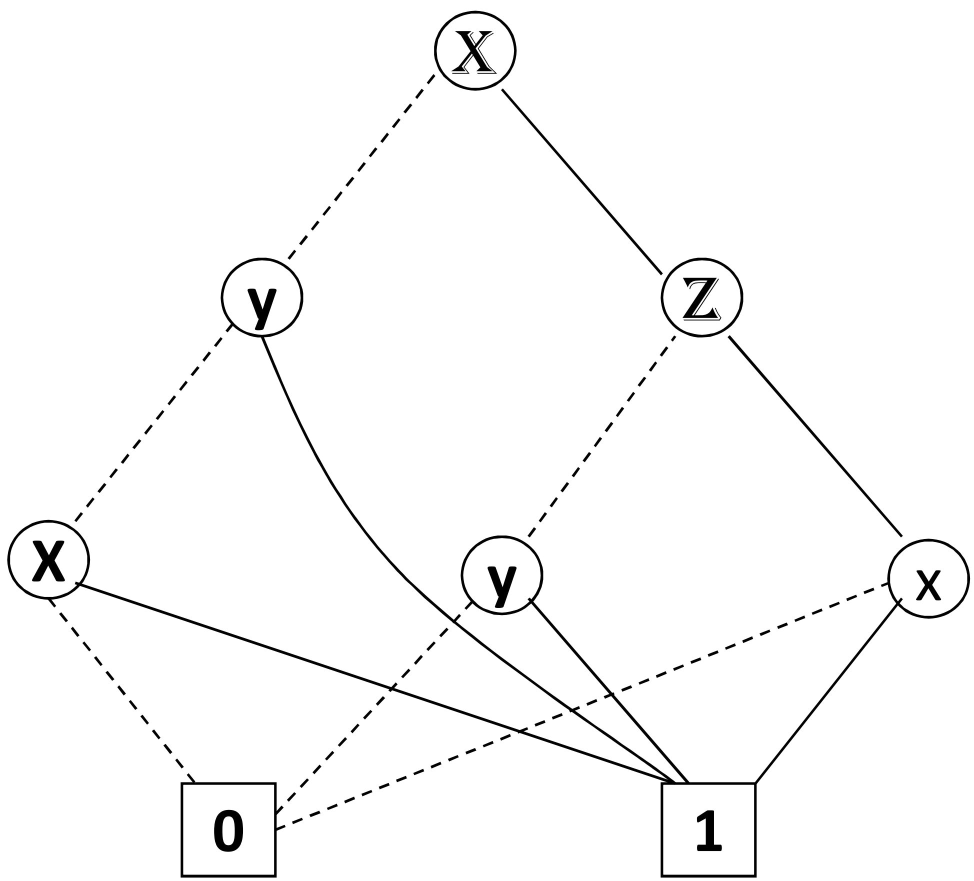
Binary Decision Diagram

- A Binary Decision Diagram (BDD) is a finite DAG with an unique initial node, where
  - all terminal nodes are labeled with 0 or 1
  - all non-terminal nodes are labeled with a Boolean Variable.
  - Each non-terminal node has exactly two edges from that node to others; one labeled 0 and one labeled 1; represent them as a dashed line and a solid line respectively

Ordering of Variables



Ordering of Variables

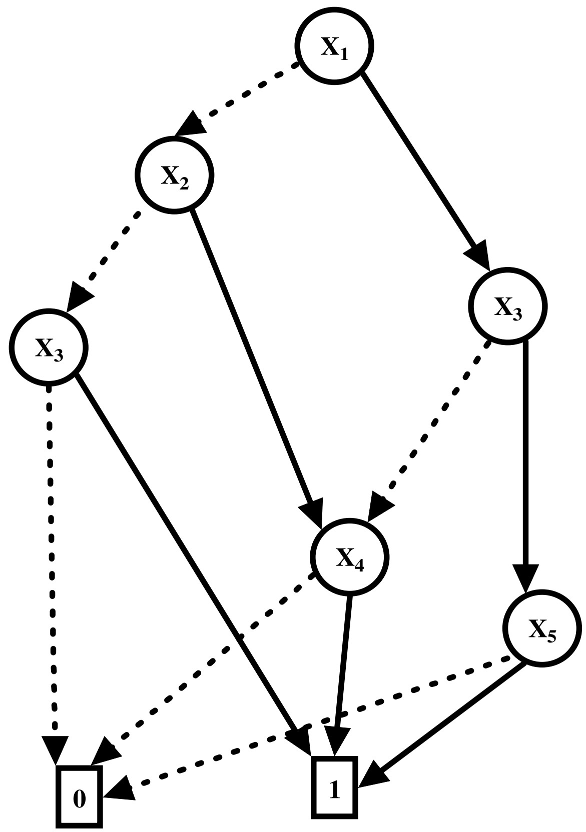


- Evaluation Path
  - Consistent
  - Inconsistent

Ordered BDDs (OBDDs)

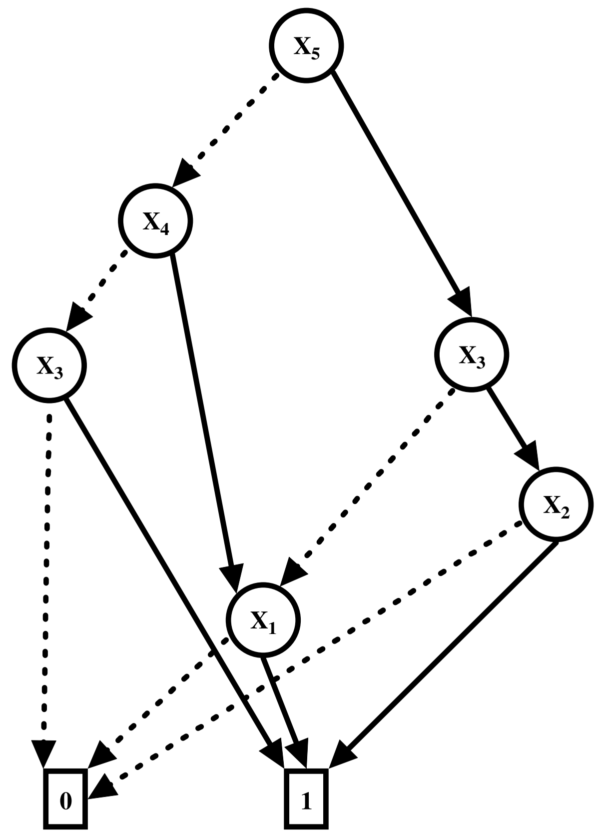
- Let $[x_1, x_2, \ldots., x_n]$ be an ordered list of variables without duplication and let B be a BDD all of whose variables occur somewhere in the list.
- We say that B has the ordering $[x_1, x_2, \ldots., x_n]$ if all variable labels of B occur in that list and, for every occurrence of x_i followed by x_j along any path in B, we have $i < j$.

OBDDs



BDD with variable ordering $[x_1, x_2, x_3, x_4, x_5]$

OBDDs

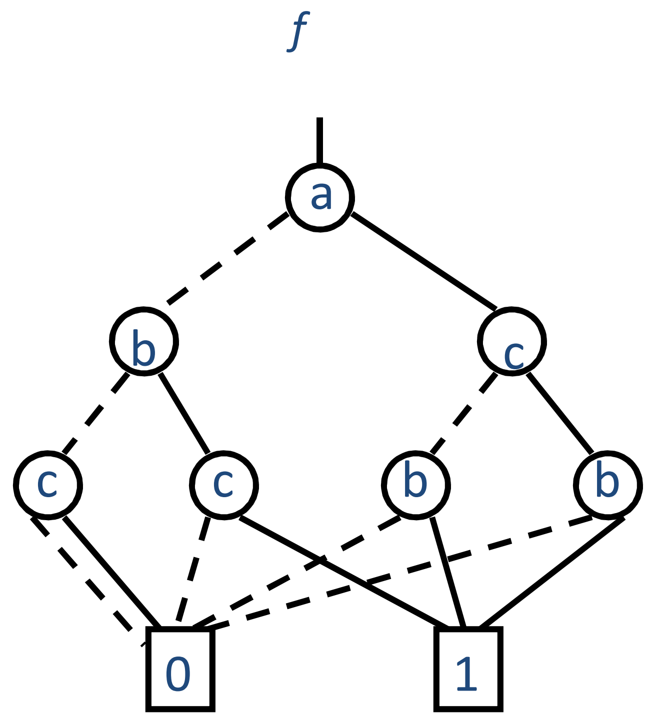


BDD with variable ordering $[x_5, x_4, x_3, x_2, x_1]$

Reduced Ordered BDDs (ROBDDs)

Not a Ordered BDD.

Not a Reduced BDD.



Impact of the chosen variable ordering

- In general the chosen variable ordering makes a significant difference to the size of the OBDD representing a given function.

Impact of the chosen variable ordering

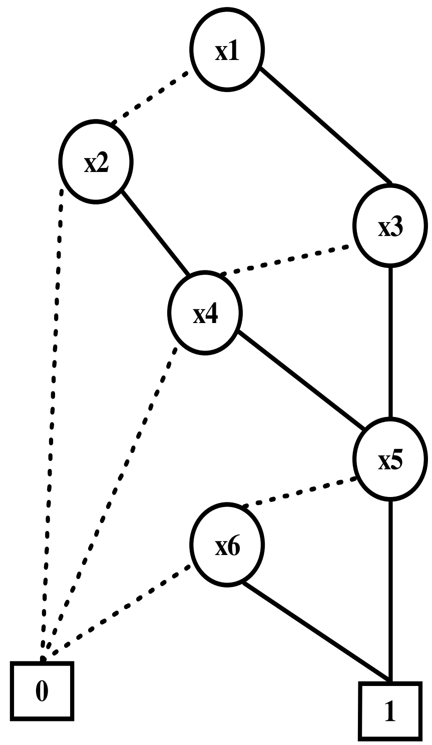
- Consider the Boolean function
  - $f = (x_1 + x_2).(x_3 + x_4).(x_5 + x_6)....(x_{2n-1} + x_{2n})$

Impact of the chosen variable ordering

- Consider the Boolean function
  - $f = (x_1 + x_2).(x_3 + x_4).(x_5 + x_6)....(x_{2n-1} + x_{2n})$

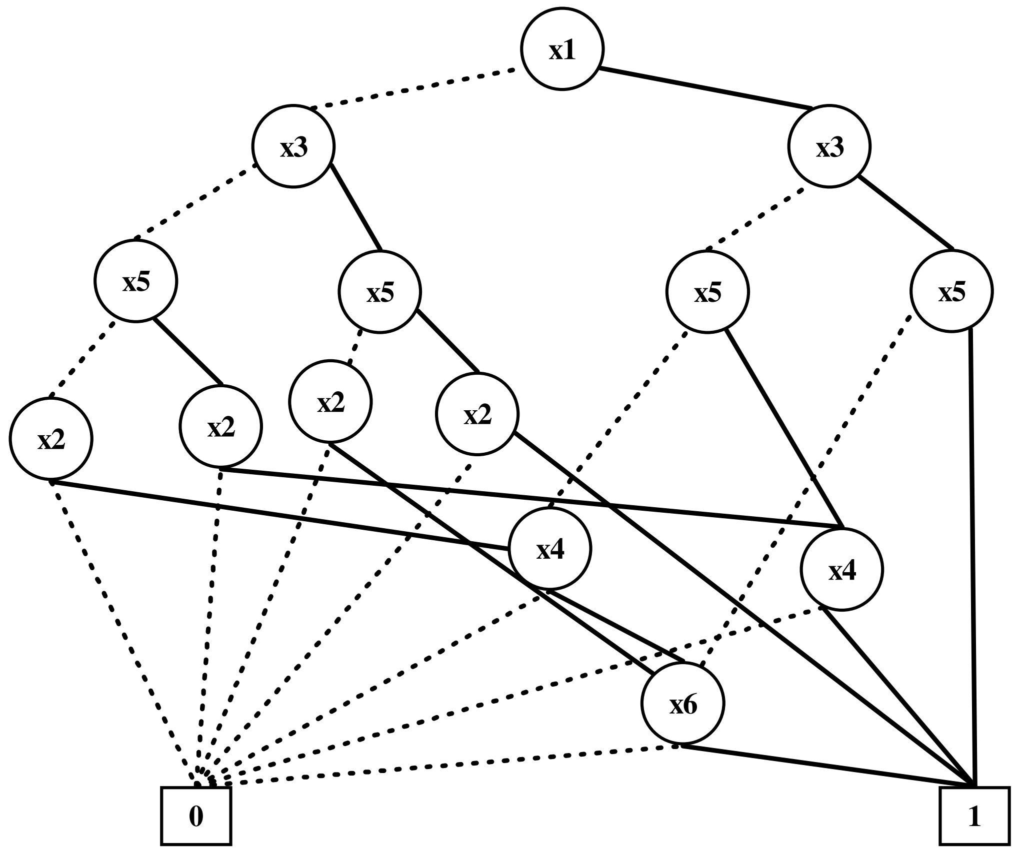
- If we chose the variable ordering $[x_1, x_2, x_3, x_4, ...]$, then we can represent this function as an OBDD with 2n+2 nodes.
- If we chose the variable ordering $[x_1, x_3, x_5, ..., x_{2n-1}, x_2, x_4, x_6, ..., x_{2n}]$, the resulting OBDD requires 2^{n+1} nodes.

OBDDs



$$f = (x_1 + x_2).(x_3 + x_4).(x_5 + x_6)$$

OBDDs



$$f = (x_1 + x_2).(x_3 + x_4).(x_5 + x_6)$$

Reduced ODBBs (ROBDDs)

A BDD is said to be reduced if none of the reduction rules R1-R3 can be applied (i.e., no more reductions are possible)

A OBDD is said to be reduced OBDD (ROBDD) if none of the reduction rules R1-R3 can be applied (i.e., no more reductions are possible)

Algorithm reduce

- The algorithm reduce provides the ROBDD of a given OBDD.
- If the ordering of B is $[x_1, x_2, \ldots, x_l]$, then B has at most $l+1$ layers.
- The algorithm reduce traverses B layer by layer in a bottom-up fashion.

Algorithm reduce

- We assign an integer label *id(n)* to each node of B.
- *Id(n)* equals to *id(m)* iff, the subOBDDs with root nodes *n* and *m* denote the same Boolean function.

Algorithm reduce

- Given a non-terminal node n in a BDD, we define *lo(n)* to be the node pointed to via the dashed line from *n*.
- Dually, *hi(n)* is the node pointed to via the solid line from *n*.

Algorithm reduce

- Labeling of terminal nodes:
  - Assign the first label (say #0) to the first 0-node it encounters.
  - All other terminal 0-nodes denote the same function as the first 0-node and therefore get the same label.
  - Similarly, the 1-nodes all get the next label (say #1)
- Reduction Rule (eliminate duplicate terminals)

Algorithm reduce

- Labeling of non-terminal nodes (Given an x_i node n and already assigned integer labels to all nodes of a layer > i):
  - If the label id(lo(n)) is same as id(hi(n)), then we set id(n) to be that label
  - (Reduction Rule:, Redundant nodes).

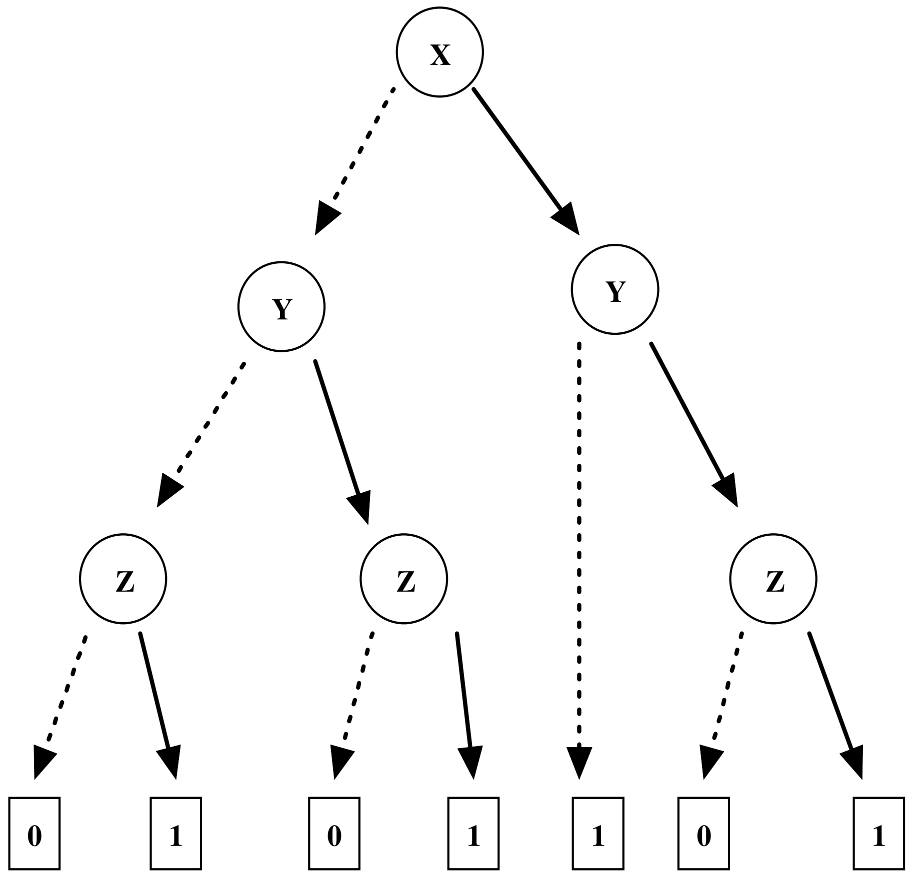
Algorithm reduce

- Labeling of non-terminal nodes (Given an x_i node n and already assigned integer labels to all nodes of a layer > i):
  - If there is another node m such that n and m have the same variables x_i, and **id(lo(n)) = id(lo(m))** and **id(hi(n)) = id(hi(m)),** then we set id(n) to be id(m).
  - (Reduction Rule, duplicate nodes)

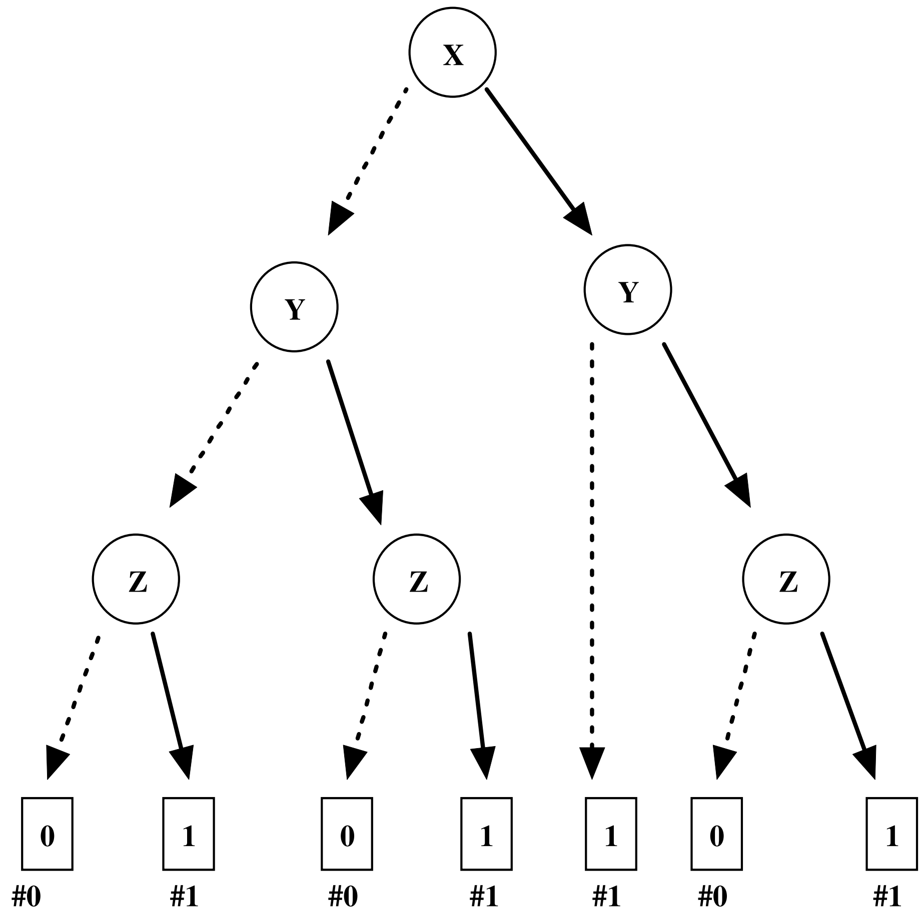
Algorithm reduce

- Labeling of non-terminal nodes (Given an x_i node n and already assigned integer labels to all nodes of a layer > i):
  - Otherwise, we set id(n) to the next unused integer label.

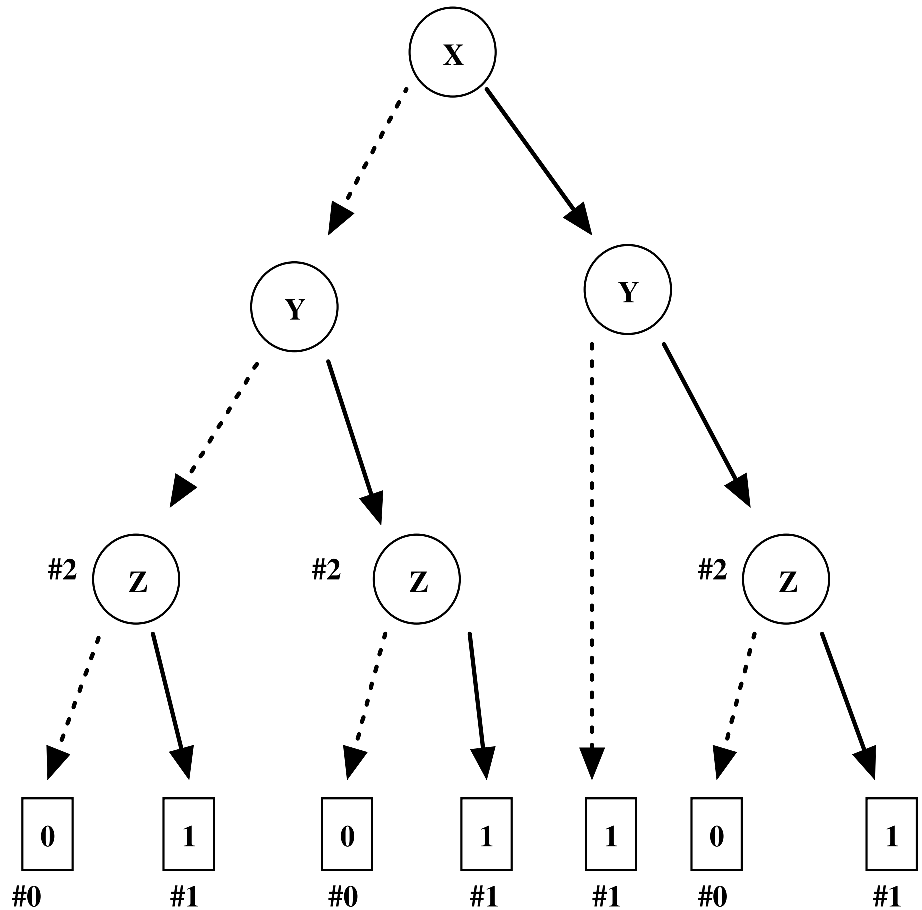
Algorithm reduce



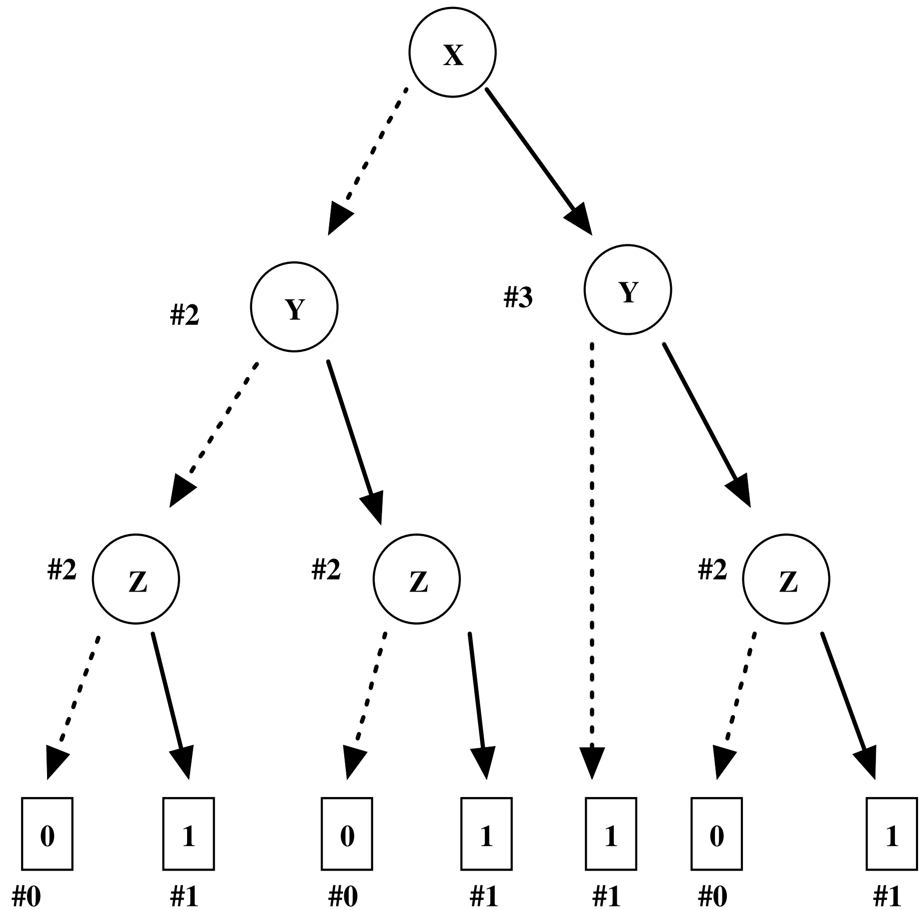
Algorithm reduce



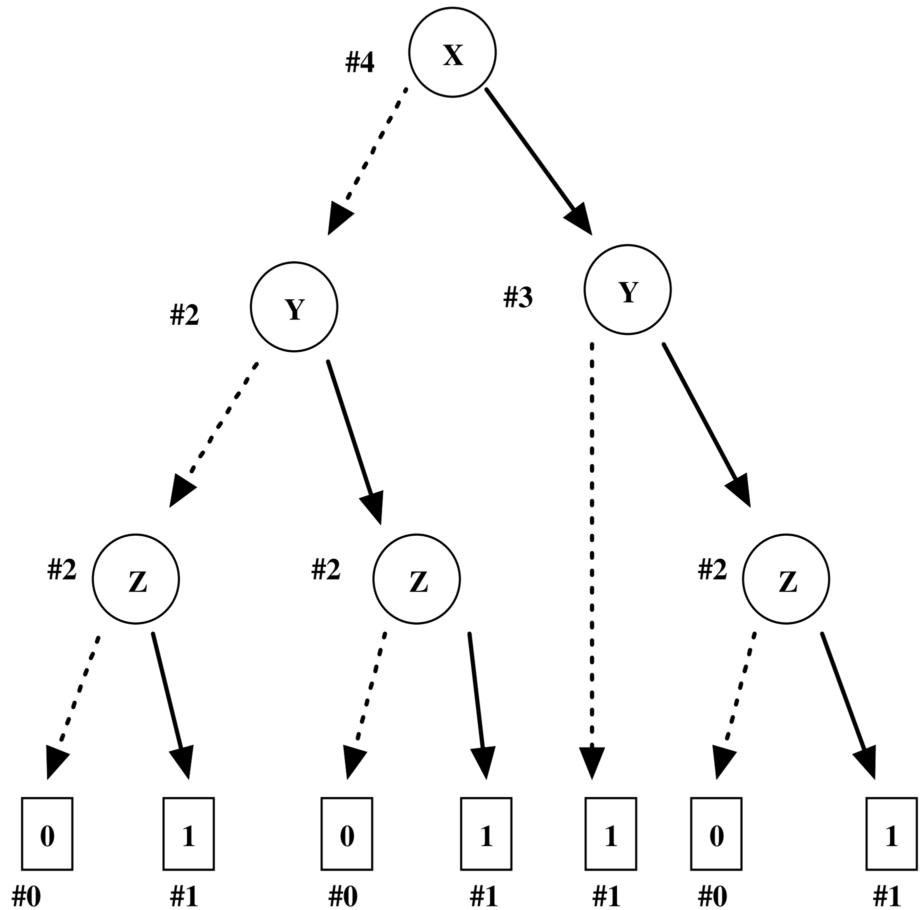
Algorithm reduce



Algorithm reduce



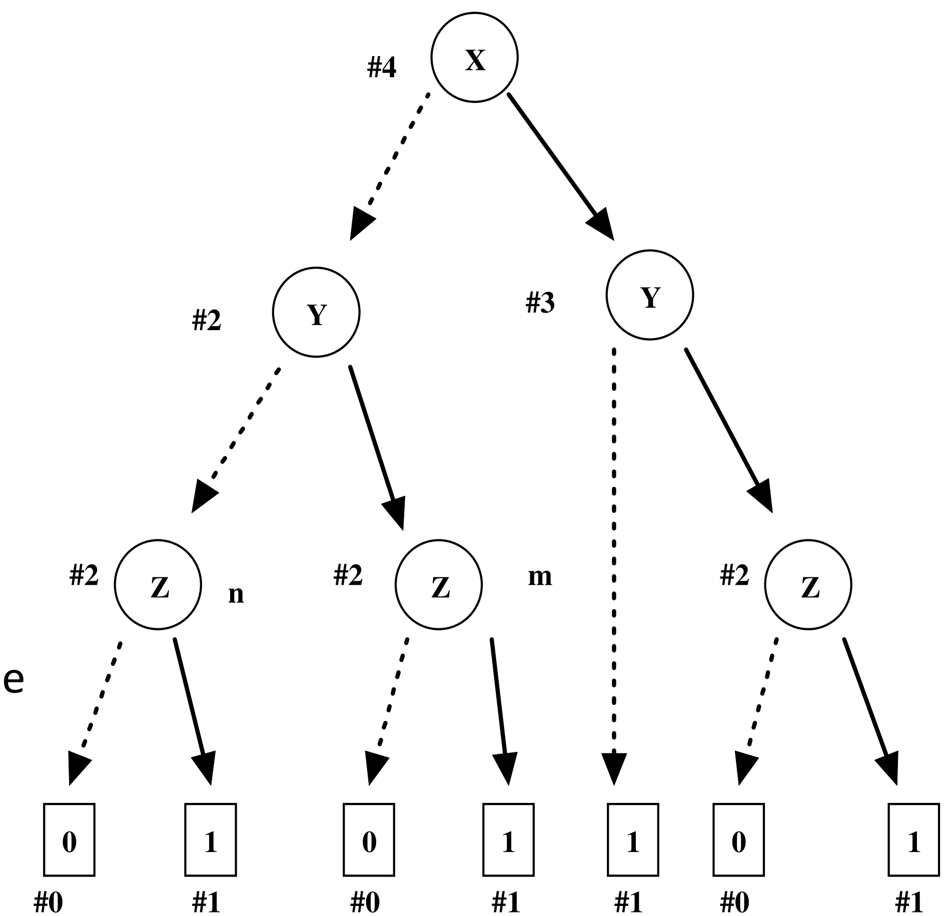
Algorithm reduce



Algorithm reduce

id(lo(n)) = id(lo(m))
Id(hi(n)) = id(hi(m))

Merge of duplicate node



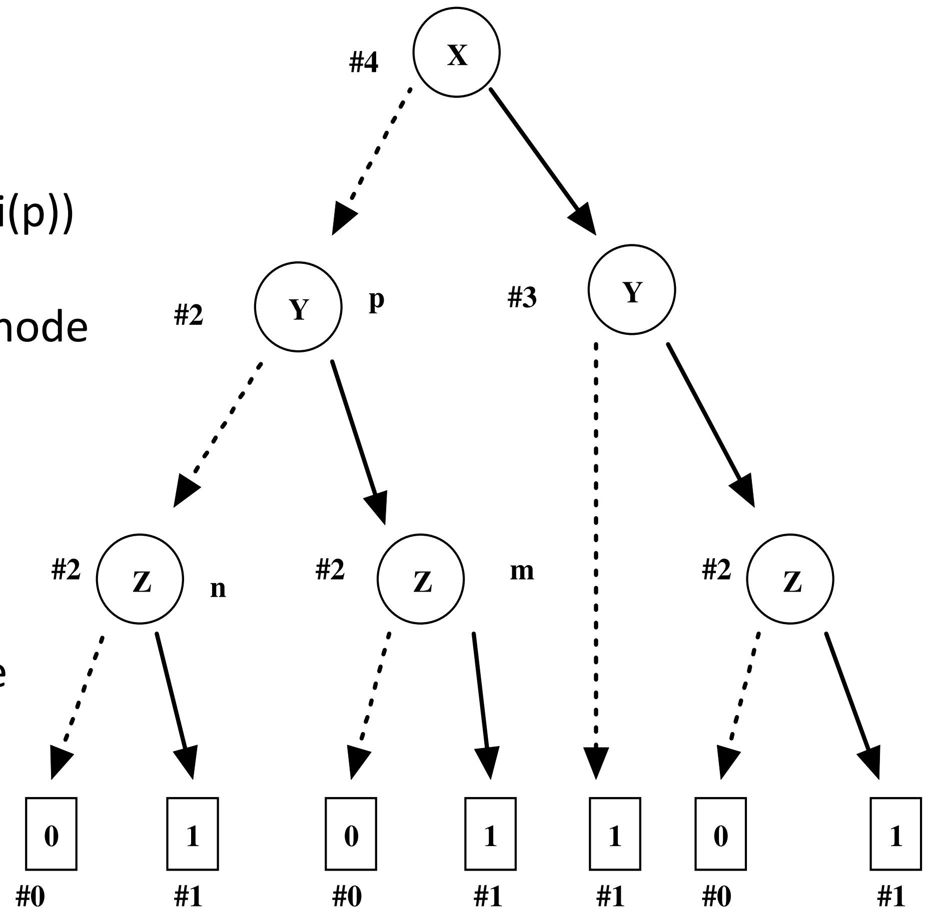
Algorithm reduce

id(lo(p)) is same as id(hi(p))

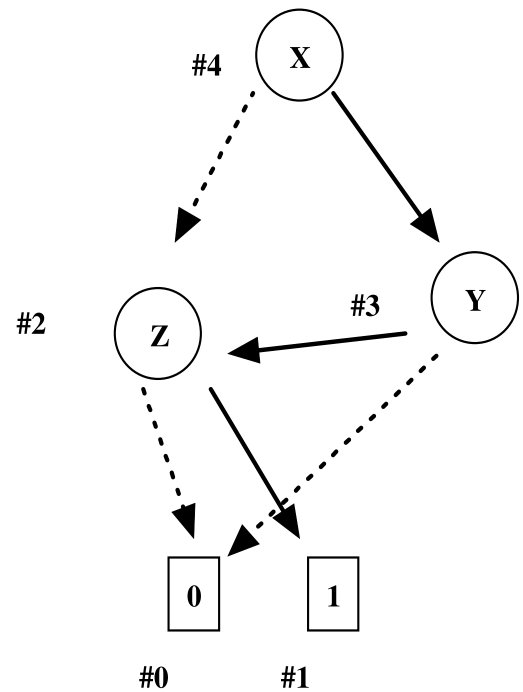
Removal of redundant node

id(lo(n)) = id(lo(m))
Id(hi(n)) = id(hi(m))

Merge of duplicate node



Algorithm reduce



Reduced Ordered BDD (ROBDD)

Reduced Ordered BDDs (ROBDDs)

- The reduced OBDD, representing a given function *f,* is *unique*.
- That is to say, let B_1 and B_2 be two reduced OBDDs with *compatible variable ordering*. If B_1 and B_2 represent the same Boolean function, then they have identical structure.
- The order in which we applied the reductions does not matter.
- OBDDs have a canonical form, their unique ROBDDs.

Reduced Ordered BDDs (ROBDDs)

Let B_1 and B_2 are the BDDs of Boolean function f_1 and f_2.

The orderings of B_1 and B_2 are said to be ***compatible*** if there are no variables x and y such that x comes before y in the ordering of B_1 and y comes before x in the ordering of B_2.

Application of BDDs

- **Test for Absence of redundant variables**
  - If the value of a Boolean function $f(x_1,x_2,\ldots x_n)$ does not depend on the value x_i, then any ROBDD which represents f does not contain any x_i-node.

Application of BDDs

- **Test for semantic equivalence**
  - B_f and B_g are the ROBDD representation of two functions f and g respectively with compatible variable ordering.
  - f and g denote the same Boolean function if, and only if, the ROBDDs have identical structure.

Application of BDDs

- **Test for Validity**
  - Consider the ROBDD of a Boolean function $f(x_1,x_2,\ldots x_n)$.
  - f is valid if, and only if, its ROBDD is B_1.

Application of BDDs

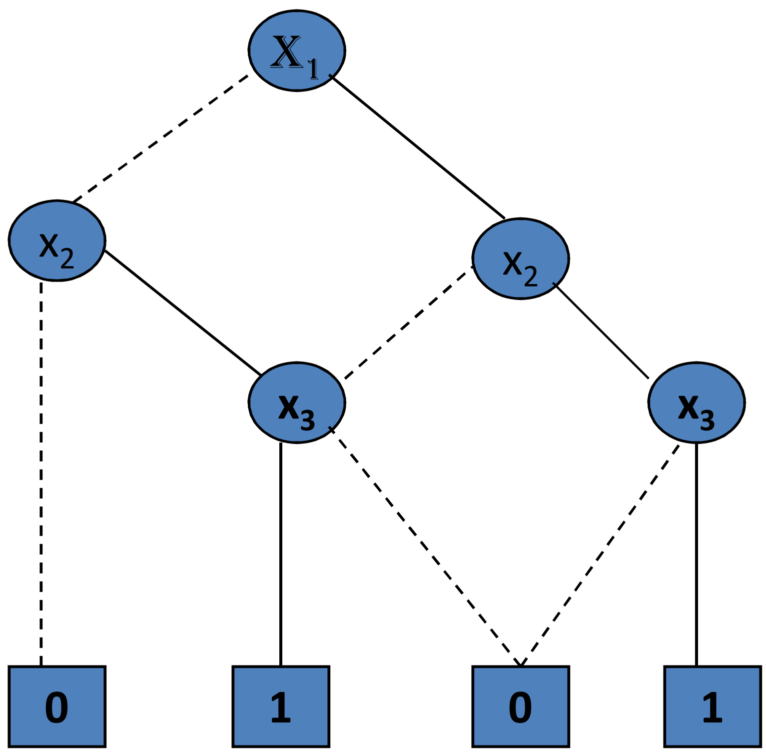
- **Test for Implication (f $\rightarrow$ g)**
  - We can test whether *f implies g* by computing the ROBDD of $(B_f \wedge \neg B_g)$
  - f implies g if, and only if, the resultant ROBDD of $(B_f \wedge \neg B_g)$ is B_0

Application of BDDs

- **Test for Satisfiability**
  - A Boolean function $f(x_1,x_2,\ldots x_n)$ is satisfiable if it computes 1 for at least one assignment of 0 and 1 values to its variables.
  - The function f is satisfiable if, and only if, its ROBDD is not B_0.

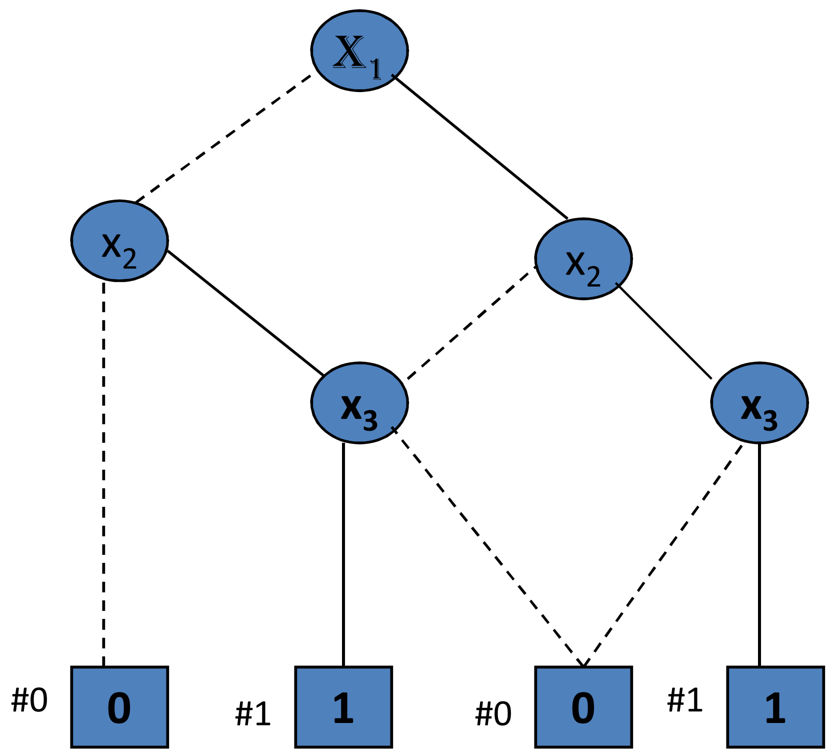
Question

- Apply the algorithm reduce



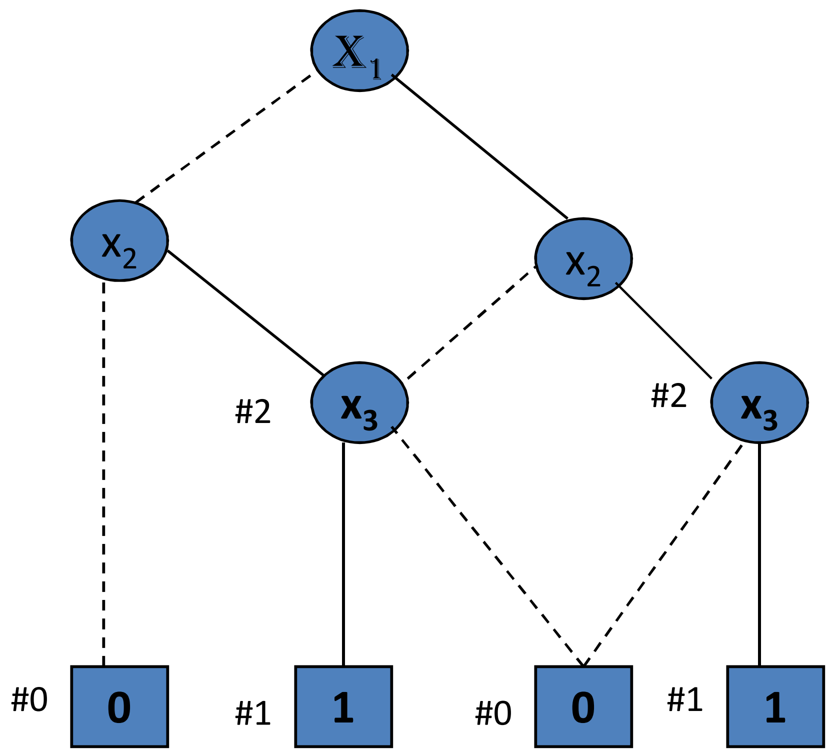
Question

- Apply the algorithm reduce



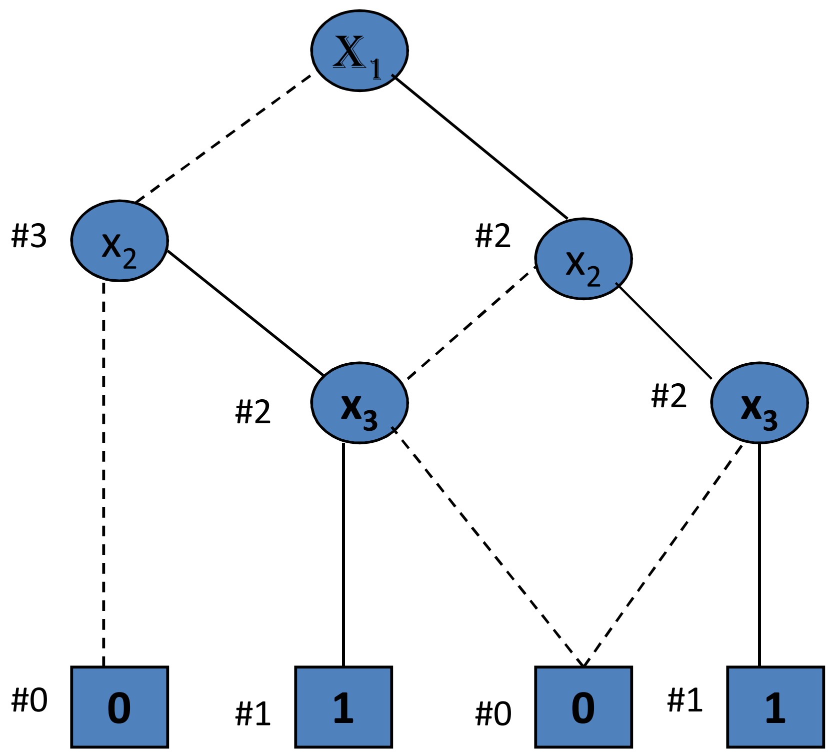
Question

- Apply the algorithm reduce



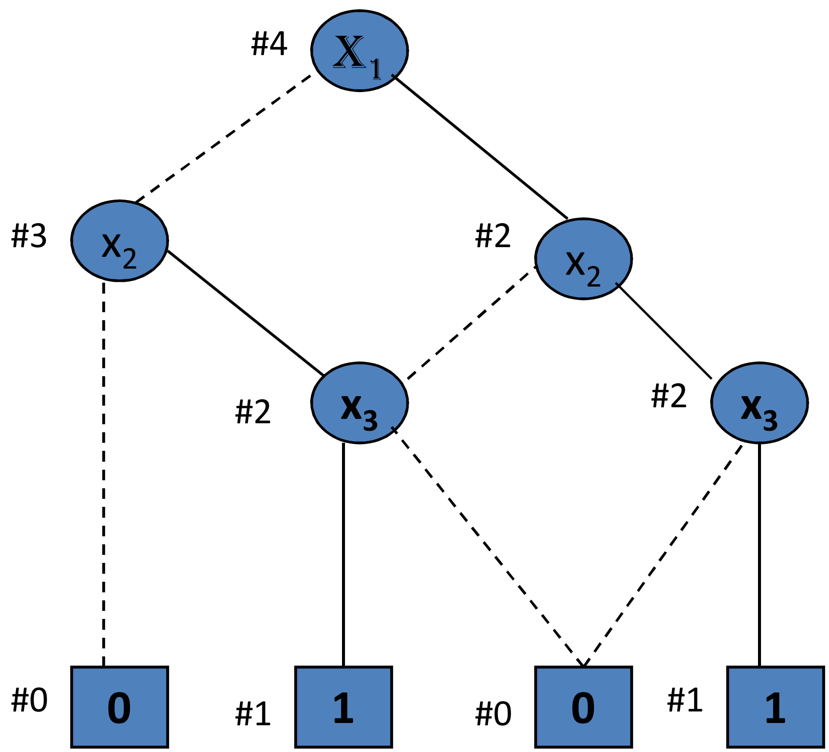
Question

- Apply the algorithm reduce



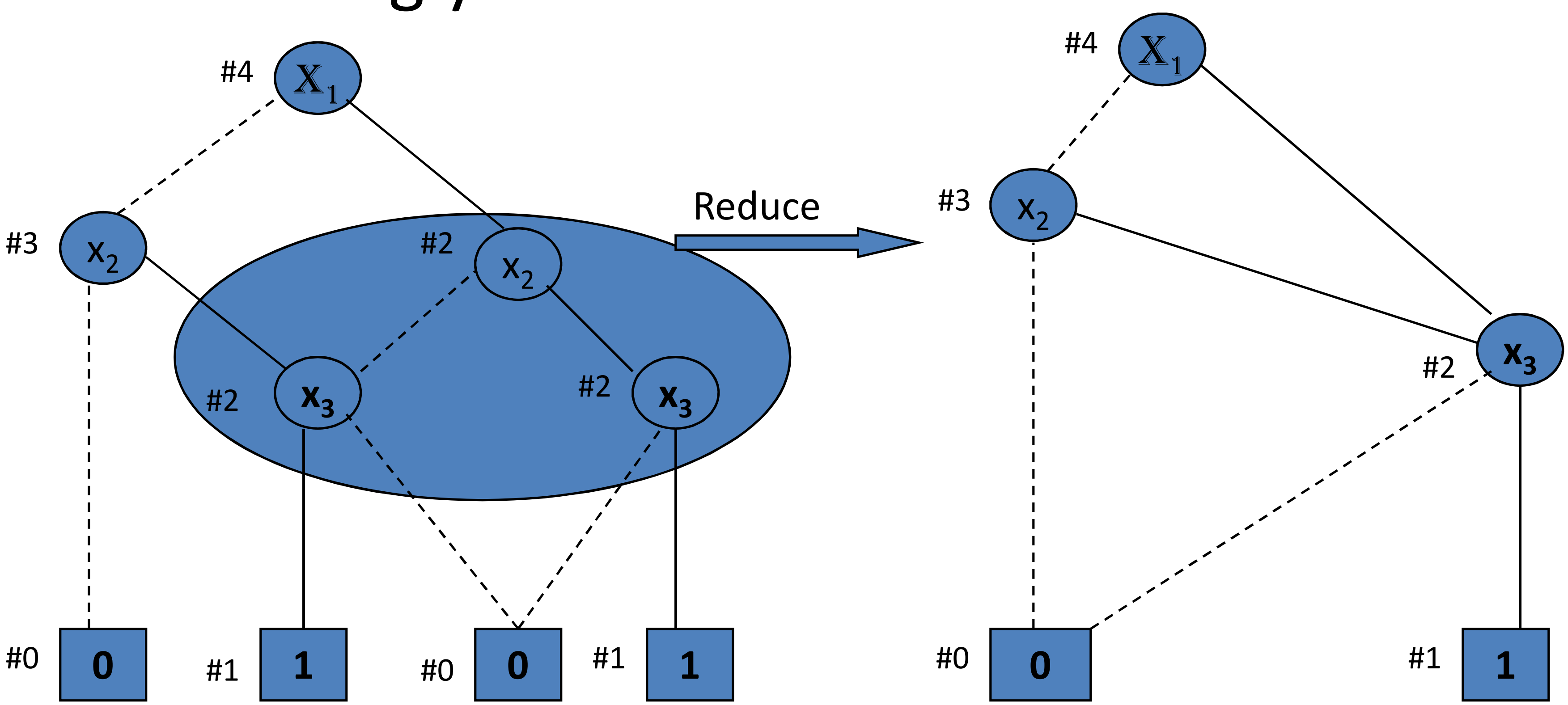
Question

- Apply the algorithm reduce



Algorithm *reduce* for BDDs

- Merge all nodes which have same label and redirect the incoming and outgoing edges accordingly.



Question

- Consider the following function
  - $f(x,y,z) = xz + xz' + x'y$

  Is it independent of any variables.

Question

- Consider the following function
  – $f(x,y,z) = xz + xz' + x'$

  Is it independent of any variables.

  Test for validity

Question

NPTEL Phase-II
Video course on

Design Verification and Test of Digital VLSI Designs

Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
IIT Guwahati

Module VI: Binary Decision Diagram

Lecture III: : Operation on Ordered Binary Decision Diagram

Operation on BDD

- f+g
- f.g

Operation on BDD

Operation on OBDD

Algorithm apply

To perform the binary operation on two ROBDD's B_f and B_g, corresponding to the functions f and g respectively, we use the algorithm ***apply*(op, B_f , B_g).** The two ROBDDs B_f and B_g have compatible variable ordering.

Operation on OBDD

Algorithm apply

Application of ***apply*(op, B_f , B_g)** will give a OBDD. The ordering of the resultant BDD is same as B_f or B_g but it may not be the reduced one. After constructing the resultant BDD, we may apply the reduce algorithm to get the ROBDD.

Operation on OBDD

Operation on OBDD

The function ***apply*** is based on the Shannon's expansion for f and g:

$$f = \overline{x}.f[0/x] + x \cdot f[1/x]$$

$$g = \overline{x}.g[0/x] + x \cdot g[1/x]$$

From the Shannon's expansion of f and g :

$$f \; op \; g = \overline{x}.(f[0/x] \; op \; g[0/x]) + x.(f[1/x] \; op \; g[1/x])$$

Operation on OBDD

This is used as a control structure of apply which proceeds from the roots of B_f and B_g downwards to construct nodes of the OBDD B_f op B_g.

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

Operation on OBDD

Algorithm apply(op, B_f, B_g)

1. 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.

Operation on OBDD

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 same variable), create an x_i-node n (called r_f,r_g) with a dashed line to ***apply*** (op, $lo(r_f)$, $lo(r_g)$) and a solid line to ***apply***(op, $hi(r_f)$, $hi(r_g)$).

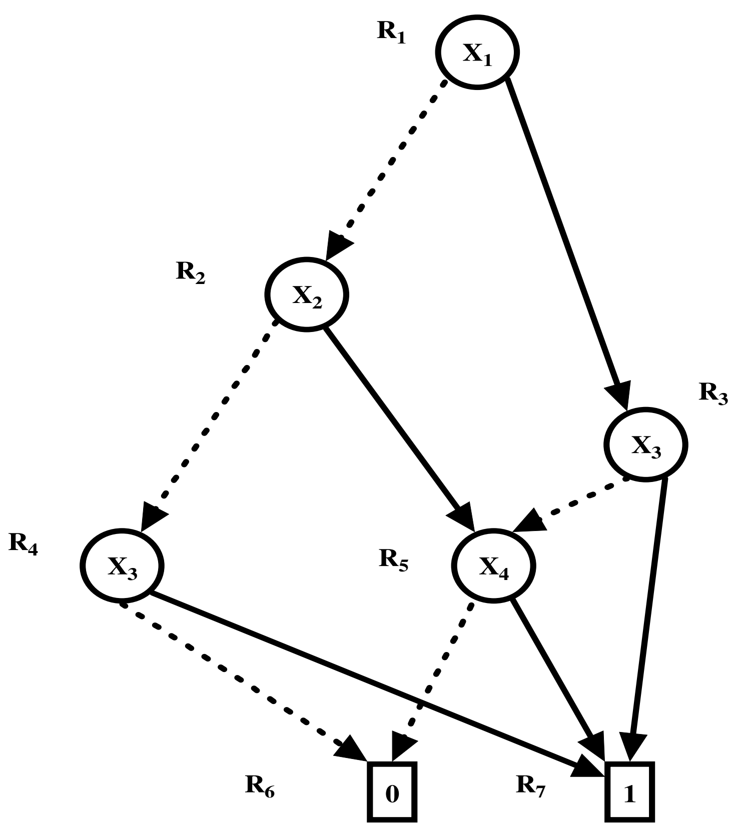
Operation on OBDD

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).

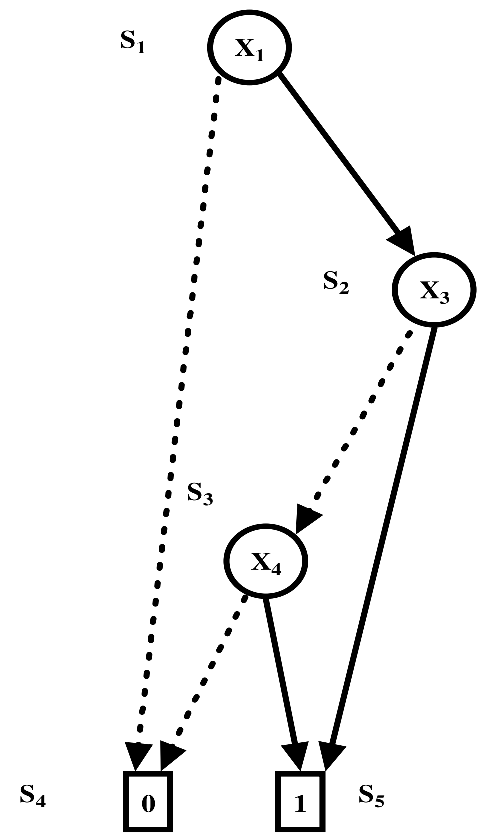
Operation on OBDD

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).

Operation on OBDD

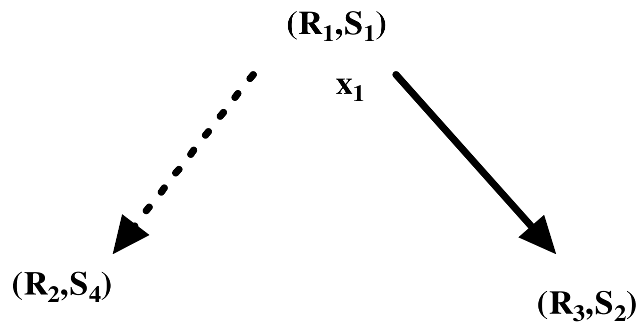


+

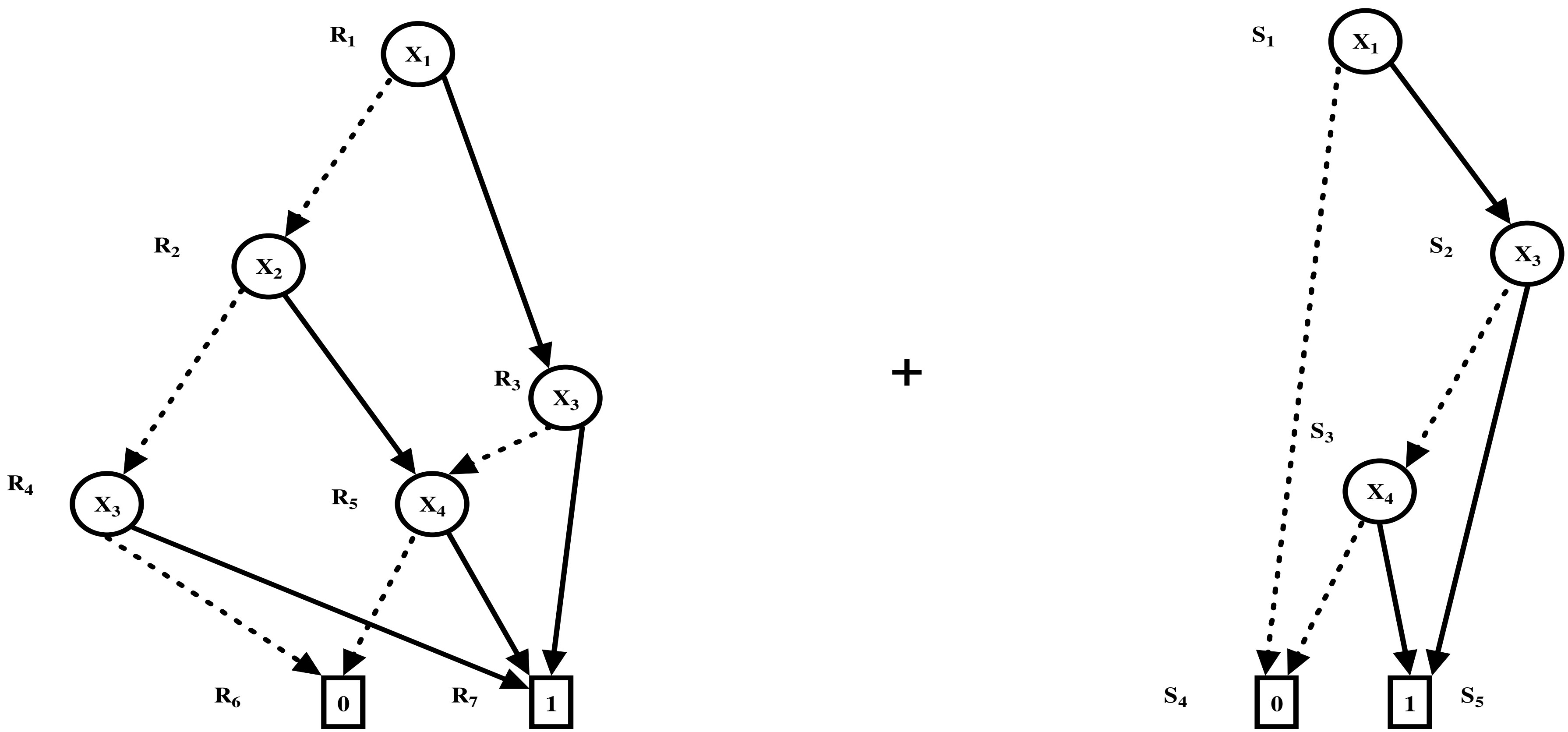


Variable ordering: [x_1, x_2, x_3, x_4]

Operation on OBDD



Operation on OBDD

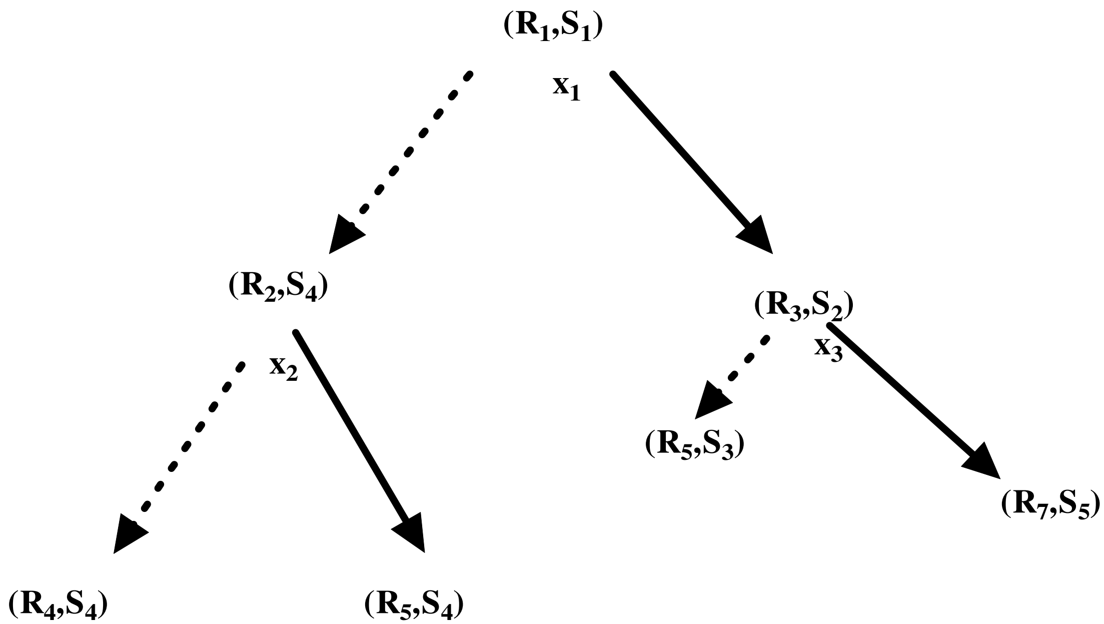


Variable ordering: $[x_1, x_2, x_3, x_4]$

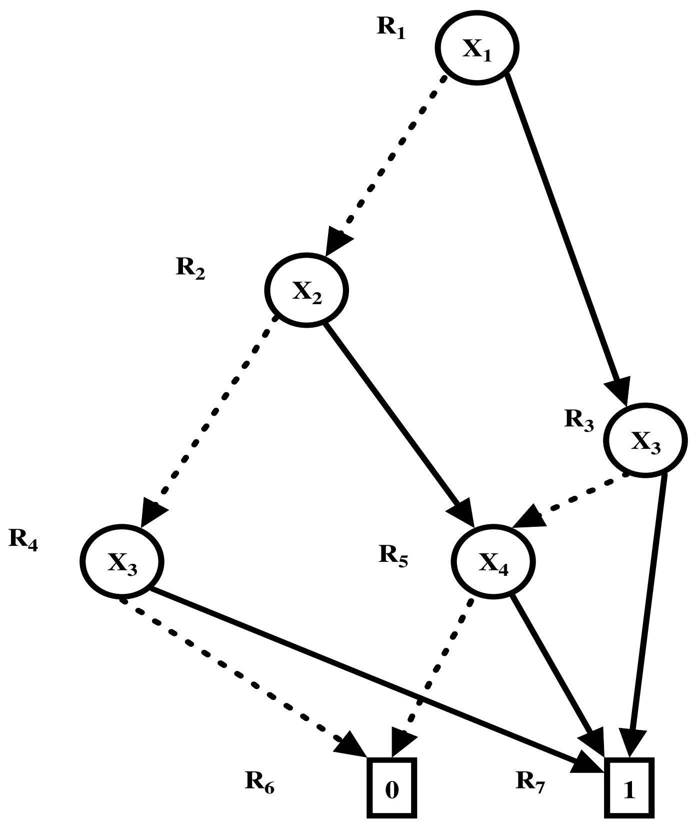
Operation on OBDD

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).

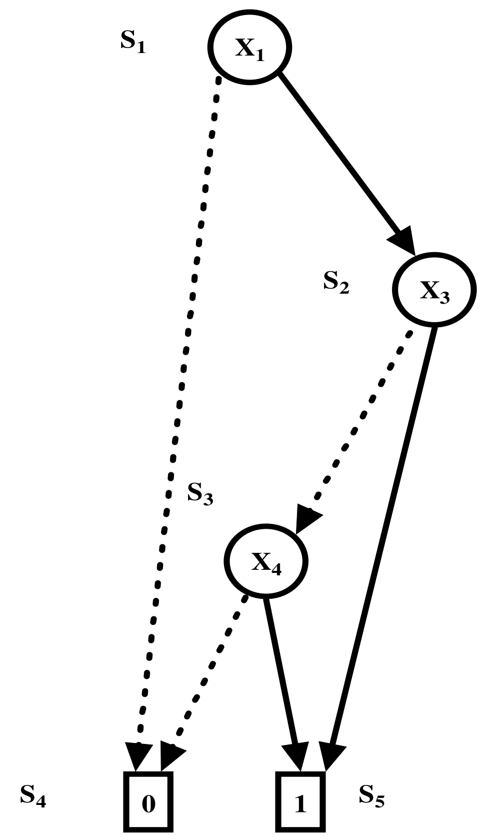
Operation on OBDD



Operation on OBDD

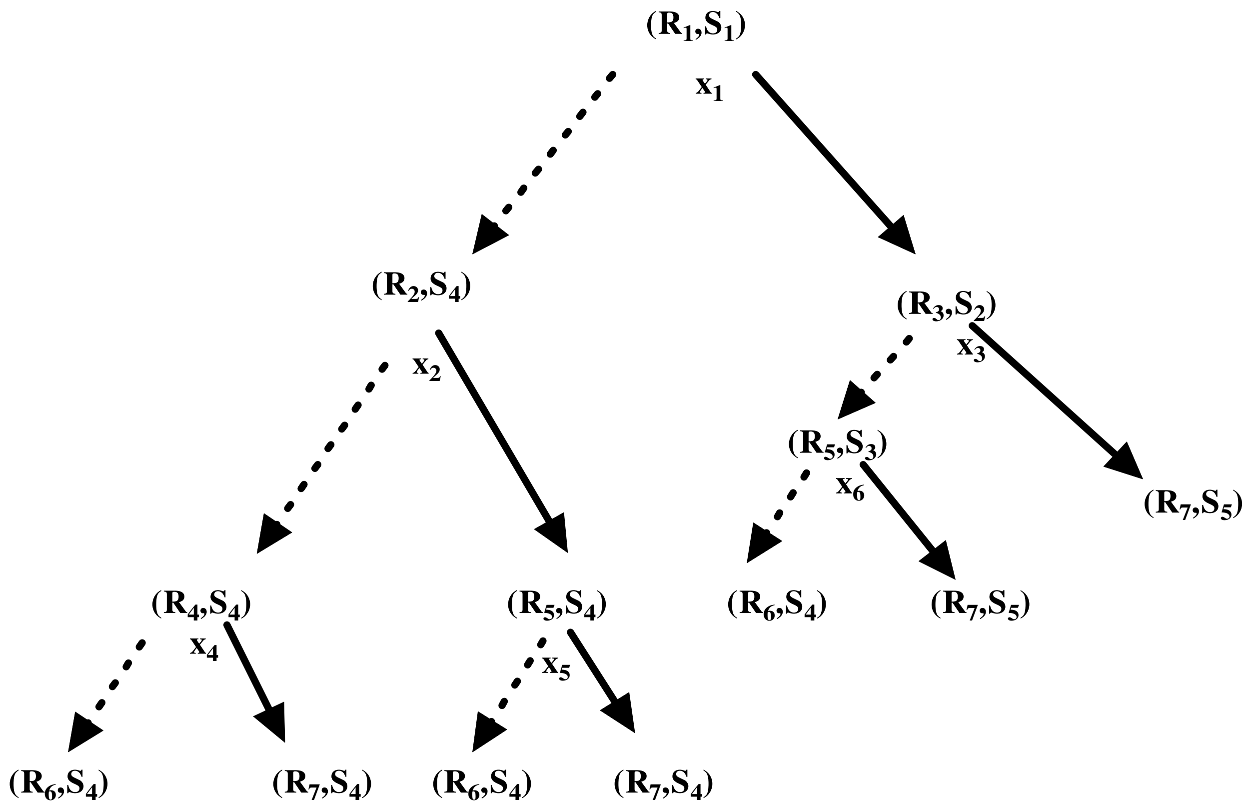


+

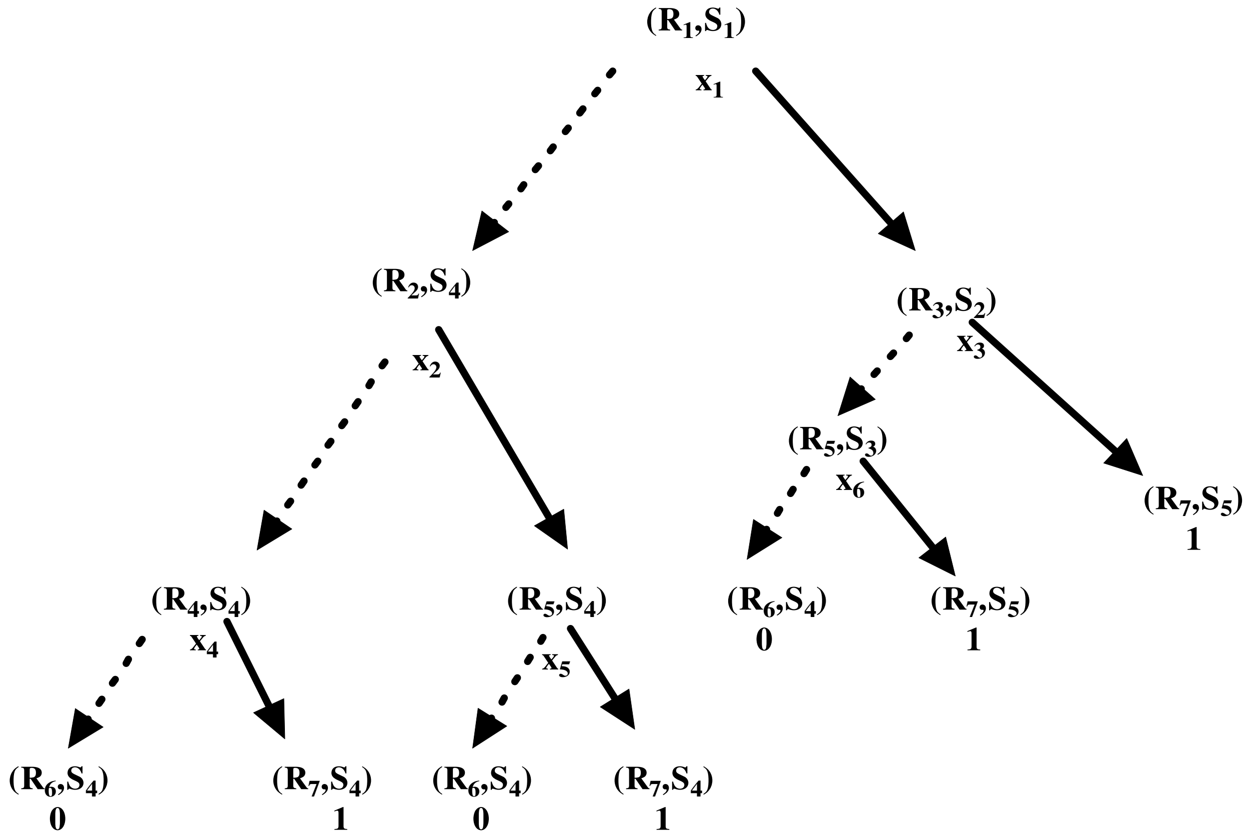


Variable ordering: $[x_1, x_2, x_3, x_4]$

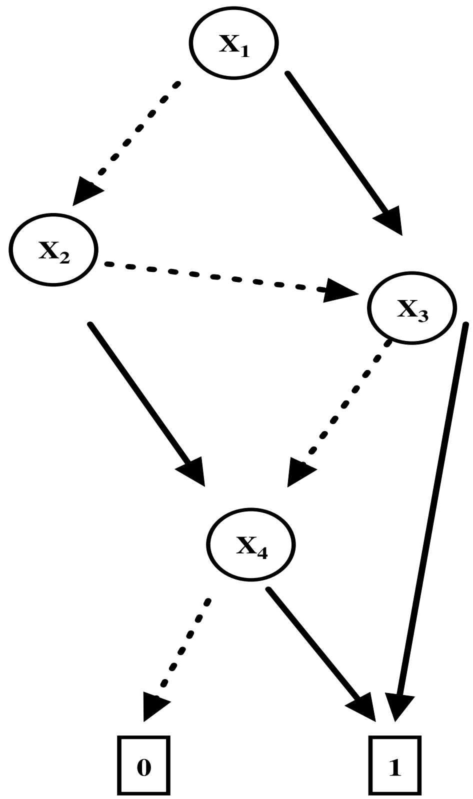
Operation on OBDD



Operation on OBDD



Operation on OBDD



Operation on OBDD

Algorithm restrict

The Boolean formula obtained by replacing all occurrences of x in f by 0 is denoted by $f[0/x]$.

The formula $f[1/x]$ is defined similarly.

The expressions $f[0/\mathrm{x}]$ and $f[1/x]$ are called restriction of f.

Operation on OBDD

restrict$(0, x, B_f)$

For each node n corresponding to x, remove n from OBDD and redirect incoming edges to $lo(n)$

restrict$(1, x, B_f)$

For each node n corresponding to x, remove n from OBDD and redirect incoming edges to $hi(n)$

Operation on OBDD

Sometimes we need to express relaxation of the constraint on a subset of variables.

If we relax the constraint on some variable x of a Boolean function f, then f could be made true by putting x to 0 or to 1.

Operation on OBDD

We write $(\exists x.f)$ for the Boolean function f with the constraint on x relaxed and it can be expressed as:

$$\exists x.f = f[0/x] + f[1/x]$$

i.e., there exists x on which the constraint is relaxed.

Operation on OBDD

Algorithm exists

The ***exists*** algorithm can be implemented in terms of the algorithms ***apply*** and ***restrict*** as

$$\exists x.f \;= apply(+,\; restrict(0, x, B_f),\; restrict(1, x, B_f))$$

Operation on OBDD

Algorithm exists

The exists operation can be easily generalized to a sequence of exists operations

$$\exists x1 . \exists x2 \ldots\ldots\ldots\ldots\ldots . \exists xn . f$$

.

Question

- Consider the following function

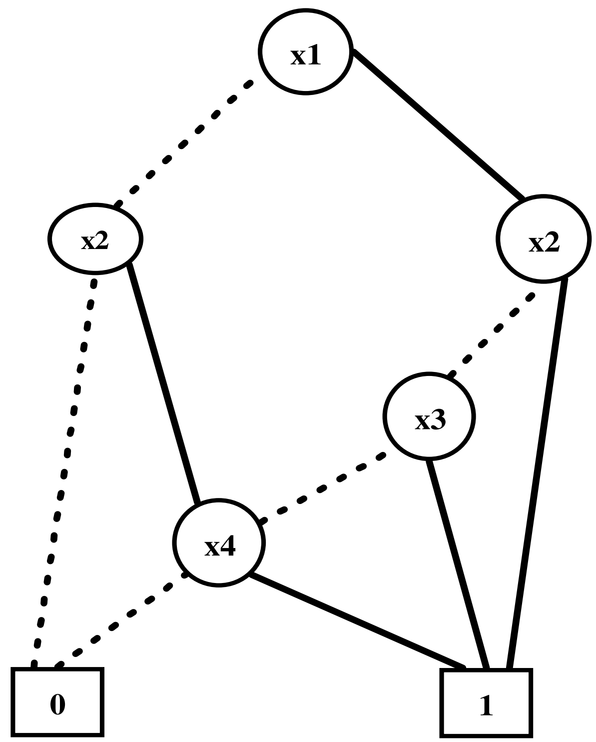
$f = x1'x2x4 + x1x2'x3 + x1x2'x3'x4 + x1x2$

Construct the ROBDD for f: B_f

restrict(0, x4, B_f) and restrict(1, x4, B_f)

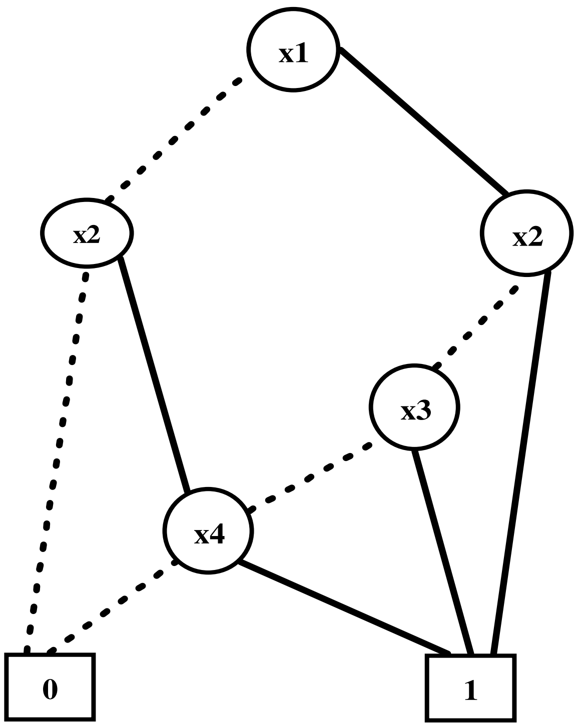
exists(x4, B_f)

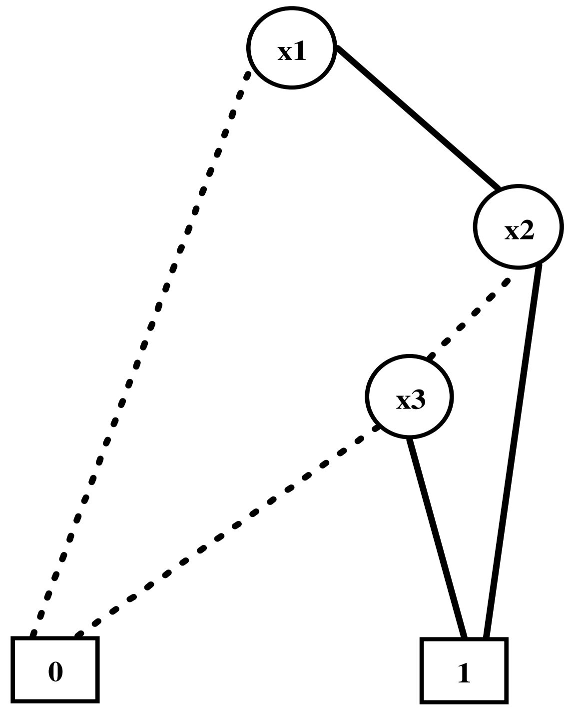
Question



f = x1’x2x4 + x1x2’x3 + x1x2’x3’x4 + x1x2

Question

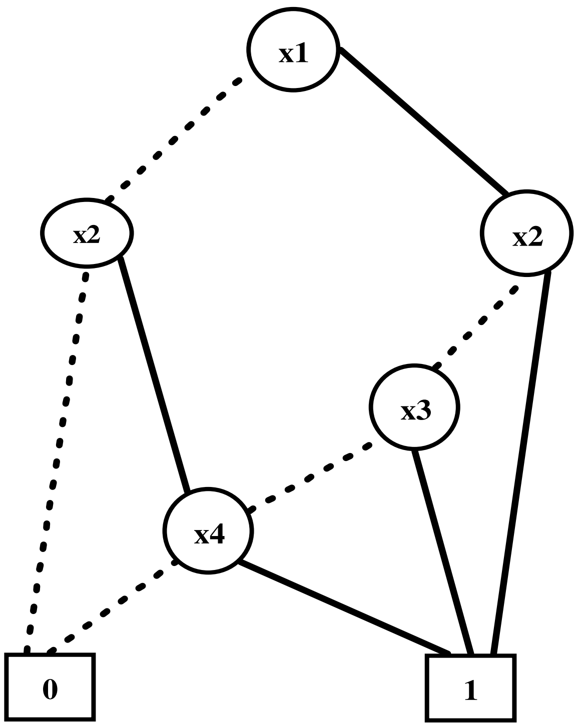


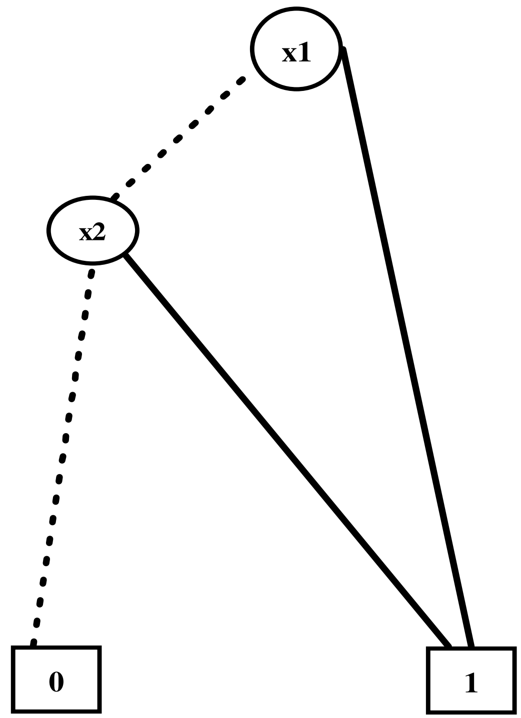


restrict(0,x4,B_f)

f = x1’x2x4 + x1x2’x3 + x1x2’x3’x4 + x1x2

Question

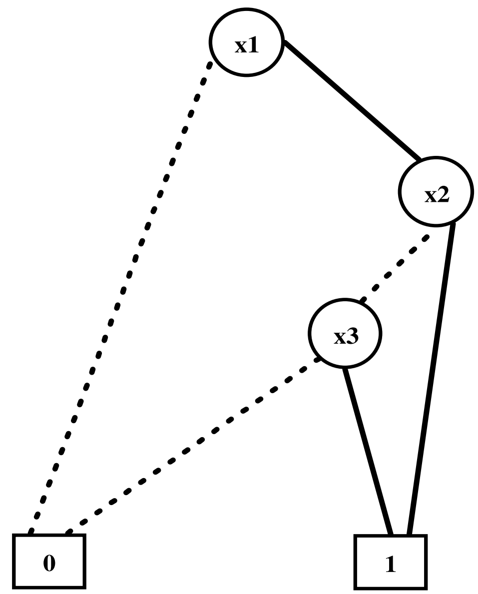




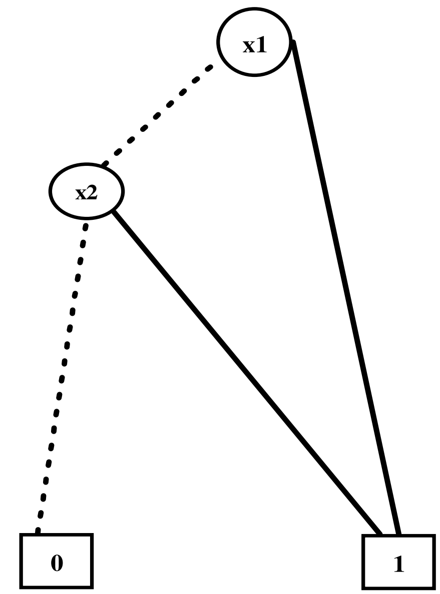
restrict(1,x4,B_f)

f = x1’x2x4 + x1x2’x3 + x1x2’x3’x4 + x1x2

Question



restrict(0,x4,B_f)



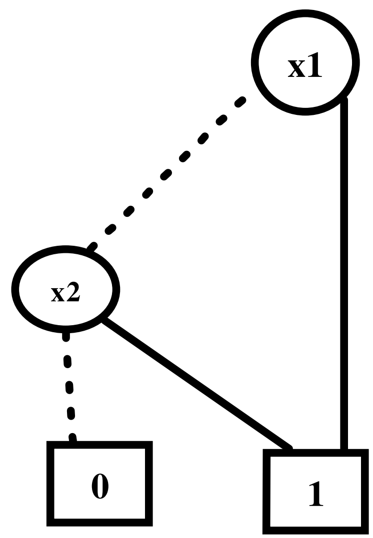
restrict(1,x4,B_f)

f = x1’x2x4 + x1x2’x3 + x1x2’x3’x4 + x1x2

Exists x4 f = apply(+, restrict(0, x4, Bf), restrict(1,x4,Bf))

Question

Question



Question

- Show that the formula $\exists x.f$ depends on all those variables that f depends upon, except x.
- If f computes to 1 with respect to a valuation v, then $\exists x.\ f$ computes 1 with respect to the same valuation.

NPTEL Phase-II
Video course on

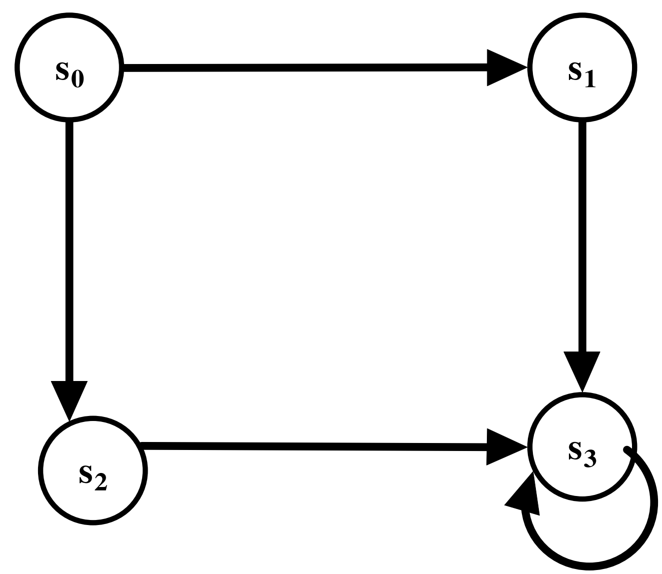
Design Verification and Test of Digital VLSI Designs

Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
IIT Guwahati

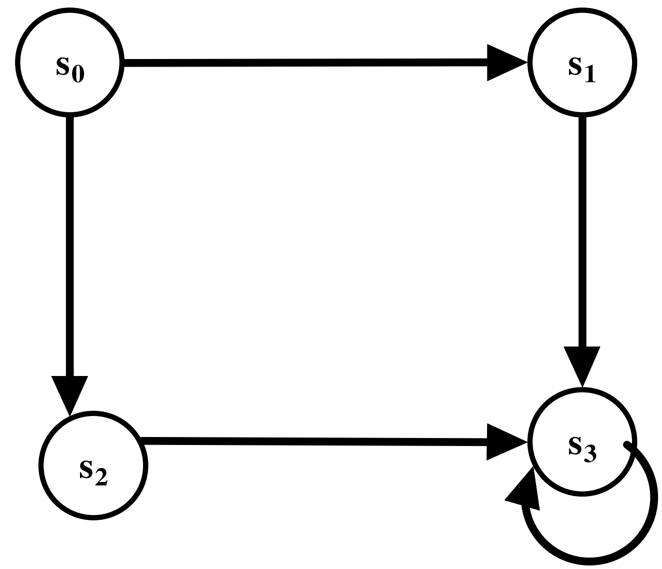
Module VI: Binary Decision Diagram

Lecture IV: Ordered Binary Decision Diagram for State Transition Systems

State Transition System



State Transition System



the states s_{0}, s_{1}, s_{2} and s_{3} can be distinguished using two state variables, say x_1 and x_2.

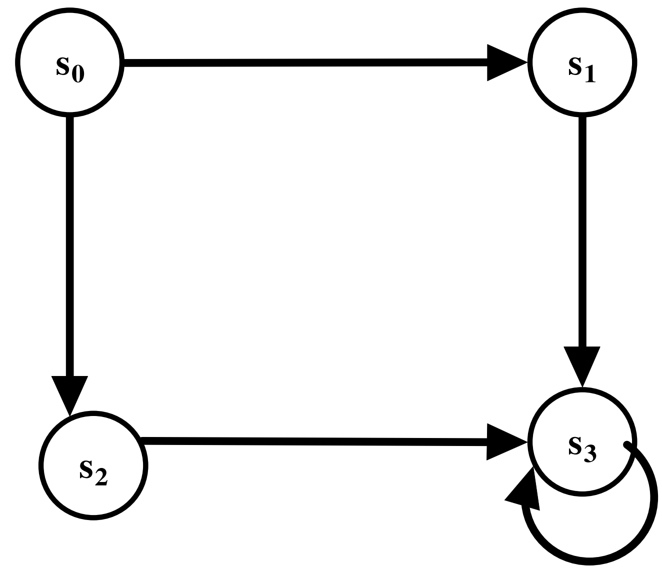
State Transition System

$$\{s_0\} = \overline{x_1}\,\overline{x_2}$$

$$\{s_1\} = \overline{x_1}x_2$$

$$\{s_2\} = x_1\overline{x_2}$$

$$\{s_3\} = x_1x_2$$



the states s_0, s_1, s_2 and s_3 can be distinguished using two state variables, say x_1 and x_2.

State Transition System: set of states

- Set of states

State Transition System: set of states

$$\{s_0, s_1\} = \overline{x_1}\,\overline{x_2} + \overline{x_1}x_2$$

$$\{s_0, s_2\} = \overline{x_1}\,\overline{x_2} + x_1\overline{x_2}$$

$$\{s_0, s_3\} = \overline{x_1}\,\overline{x_2} + x_1x_2$$

$$\{s_1, s_2\} = \overline{x_1}x_2 + x_1\overline{x_2}$$

$$\{s_1, s_3\} = \overline{x_1}x_2 + x_1x_3$$

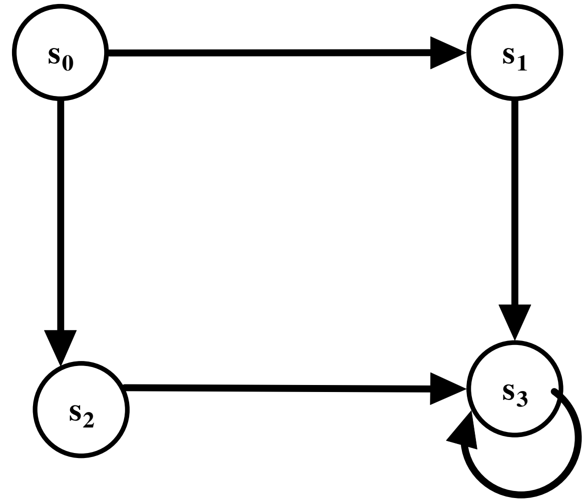
$$\{s_2, s_3\} = x_1\overline{x_2} + x_1x_3$$

$$\{s_0, s_1, s_2\} = \overline{x_1}\,\overline{x_2} + \overline{x_1}x_2 + x_1\overline{x_2}$$

$$\{s_0, s_1, s_3\} = \overline{x_1}\,\overline{x_2} + \overline{x_1}x_2 + x_1x_3$$

$$\{s_0, s_2, s_3\} = \overline{x_1}\,\overline{x_2} + x_1\overline{x_2} + x_1x_3$$

$$\{s_0, s_1, s_2, s_3\} = \overline{x_1}\,\overline{x_2} + \overline{x_1}x_2 + x_1\overline{x_2} + x_1x_3$$



$$\{s_0\} = \overline{x_1}\,\overline{x_2}$$

$$\{s_1\} = \overline{x_1}x_2$$

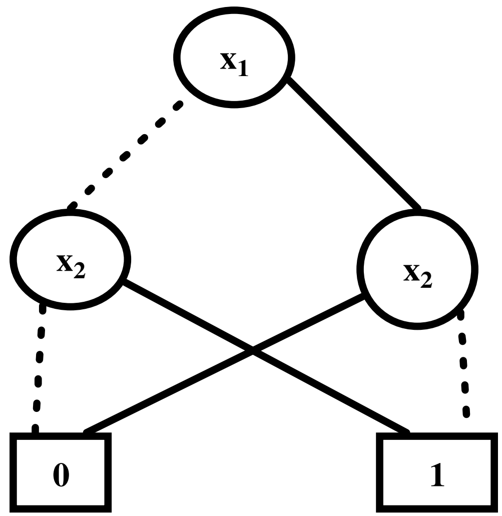
$$\{s_2\} = x_1\overline{x_2}$$

$$\{s_3\} = x_1x_2$$

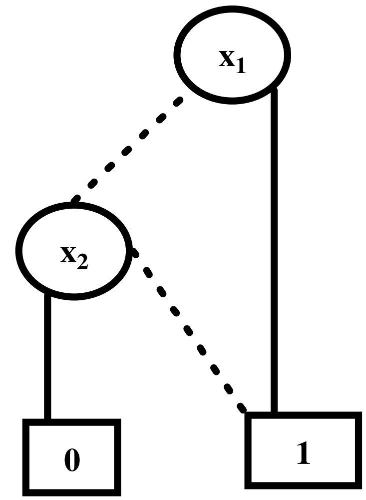
State Transition Diagram: set of states

- Set of states is represented by Boolean expression.
- OBDDs are used to represent Boolean expression.

State Transition Systems: set of states



ROBDD for {s1, s2}
x1’x2 + x1x2’



ROBDD for {s0, s2, s3}
x1’x2’ + x1x2’ + x1x2

State Transition Systems: Set of states

- Set operation:
  - Union, Intersection, etc
- S1 and S2 are two sets.

State Transition Systems: Set of states

- Set operation:
  - Union, Intersection, etc
- S1 and S2 are two sets.
- B_{S1} and B_{S2} are the OBDD representation of sets S1 and S2 respectively.
- Union of S1 and S2 is *apply*(+, B_{S1}, B_{S2})
- Intersection of S1 and S2 is *apply*(.,B_{S1}, B_{S2})

State Transition system: transition

- Transition of a system can be viewed as an ordered pair (s_p, s_n)
  - s_p: present state
  - s_n: next state

State Transition system: transition

- Transition of a system can be viewed as an ordered pair (s_p, s_n)
  - s_p: present state
  - s_n: next state
  - If n variables are used to represent the current state $x_1, x_2, \; x_3, x_4, \ldots\ldots, x_n$
  - We Need another n variables to represent the next state $x'_1, x'_2, \; x'_3, x'_4, \ldots\ldots, x'_n$

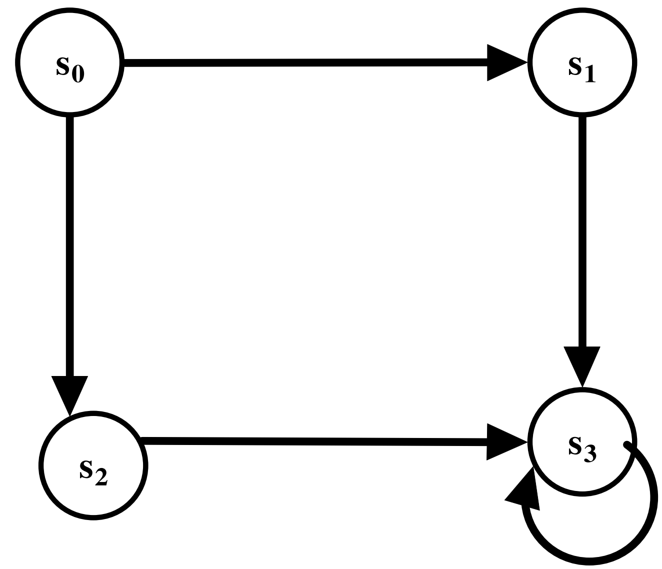
State Transition System: Transitions

$$\{s_0\} = \overline{x_1}\,\overline{x_2}$$

$$\{s_1\} = \overline{x_1}x_2$$

$$\{s_2\} = x_1\overline{x_2}$$

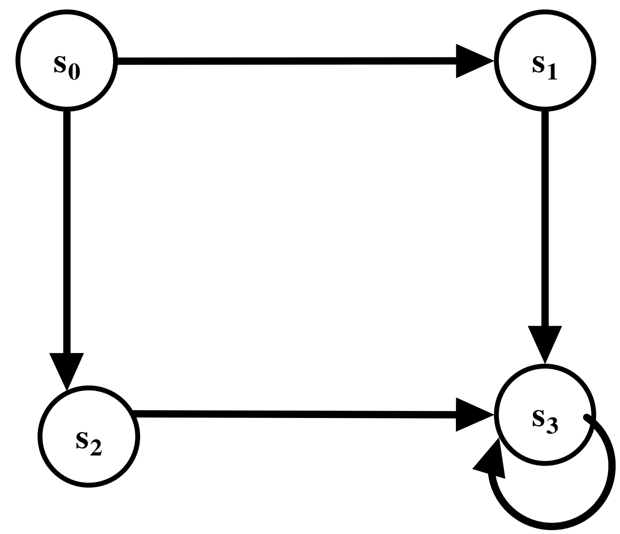
$$\{s_3\} = x_1 x_2$$



the states s_0, s_1, s_2 and s_3 can be distinguished using two state variables, say x_1 and x_2.

State Transition System: Transitions

Next state variables: x1’ and x2’



$$\{s_0\} = \overline{x_1}\,\overline{x_2}$$

$$\{s_1\} = \overline{x_1}x_2$$

$$\{s_2\} = x_1\overline{x_2}$$

$$\{s_3\} = x_1x_2$$

State Transition system: transition

State Transition system

- State transition system can be represented by Boolean expression.
- OBDD is used to represent Boolean expression.

Verification: Model Checking

- Model of the system: Kripke structure
  - Set of states
  - Transitions
  - Labeling function
- Specification/Property: CTL
- Verification Method: Model Checking method

Model Checking

- Graph traversal algorithm
- State space explosion problem
- OBDD can be used to represent kripke structure
  - State transition system
  - Labeling function

Model Checking

- Symbolic Model Checking

CTL Model Checking

Temporal Operator:

AF p

- If any state s is labeled with p, label it with AF p
- Repeat: label any state with AF p if all successor states are labeled with AF p until there is no change.

Symbolic Model Checking

- Requirements:
  - Find the predecessor state(s) of a state or a set of states

Symbolic Model Checking

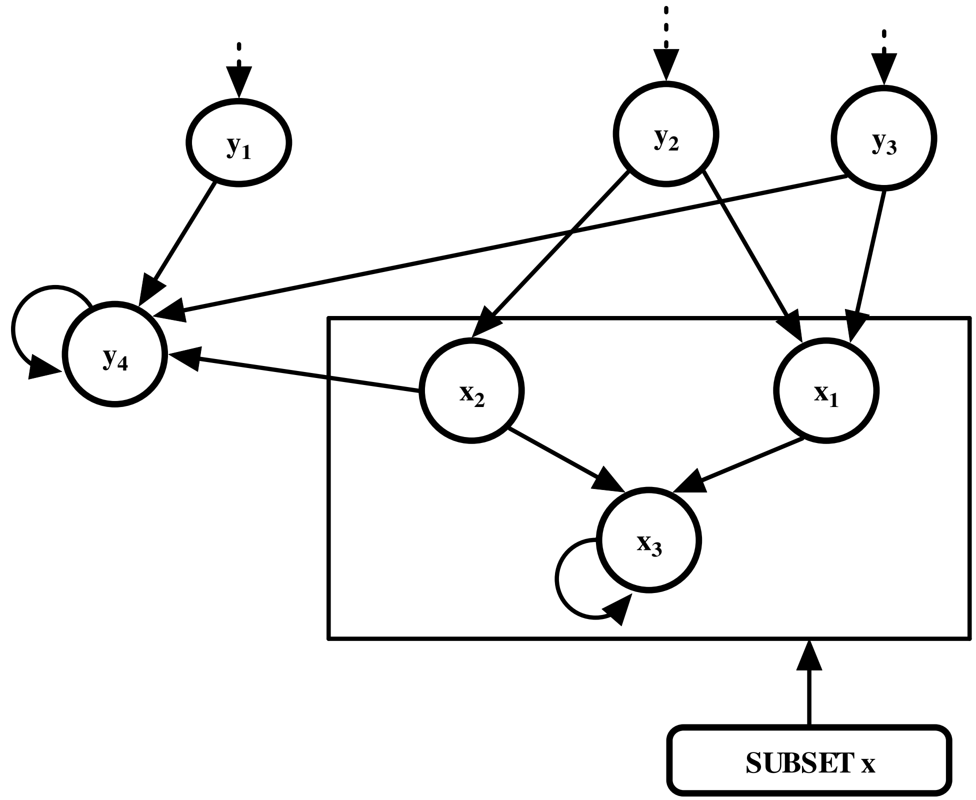
- To find the predecessor states, we define two functions:
  - $Pre_{\exists}(X)$: takes a subset X of states S and return the set of states which can make a transition into X.
  - $Pre_{\forall}(X)$: takes a subset X of states S and return the set of states which can make a transition **only** into X.

Symbolic Model Checking

$$\mathrm{Pr}\,e_{\exists}(X) = \{ s \in S \mid \exists s', (s \rightarrow s' \ and \ s' \in X) \}$$

$$\mathrm{Pr}\,e_{\forall}(X) = \{ s \in S \mid \forall s', (s \rightarrow s' \text{ and } s' \in X) \}$$

Symbolic Model Checking



Symbolic Model Checking

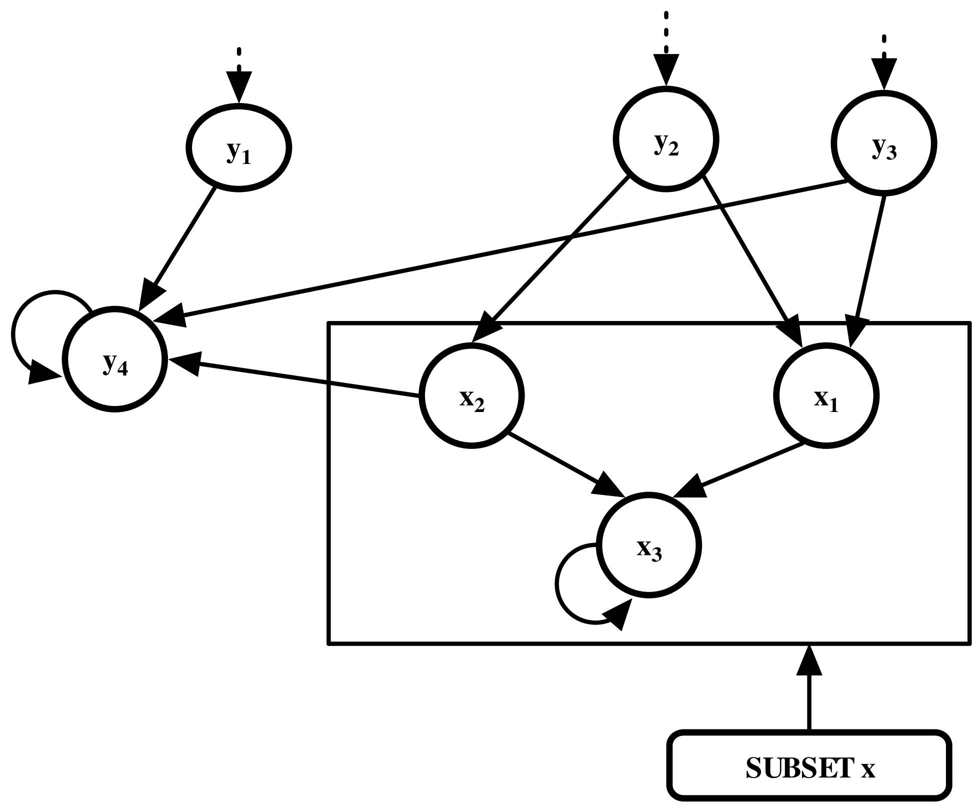
- Important relationship between $Pre_{\exists}(X)$ and $Pre_{\forall}(X)$:

$$Pre_{\forall}(X) = S - Pre_{\exists}(S - X)$$

S: Set of all states

X: Subset of S

Symbolic Model Checking



Transition System: Represented by ROBDD
Subset X: Represented by ROBDD

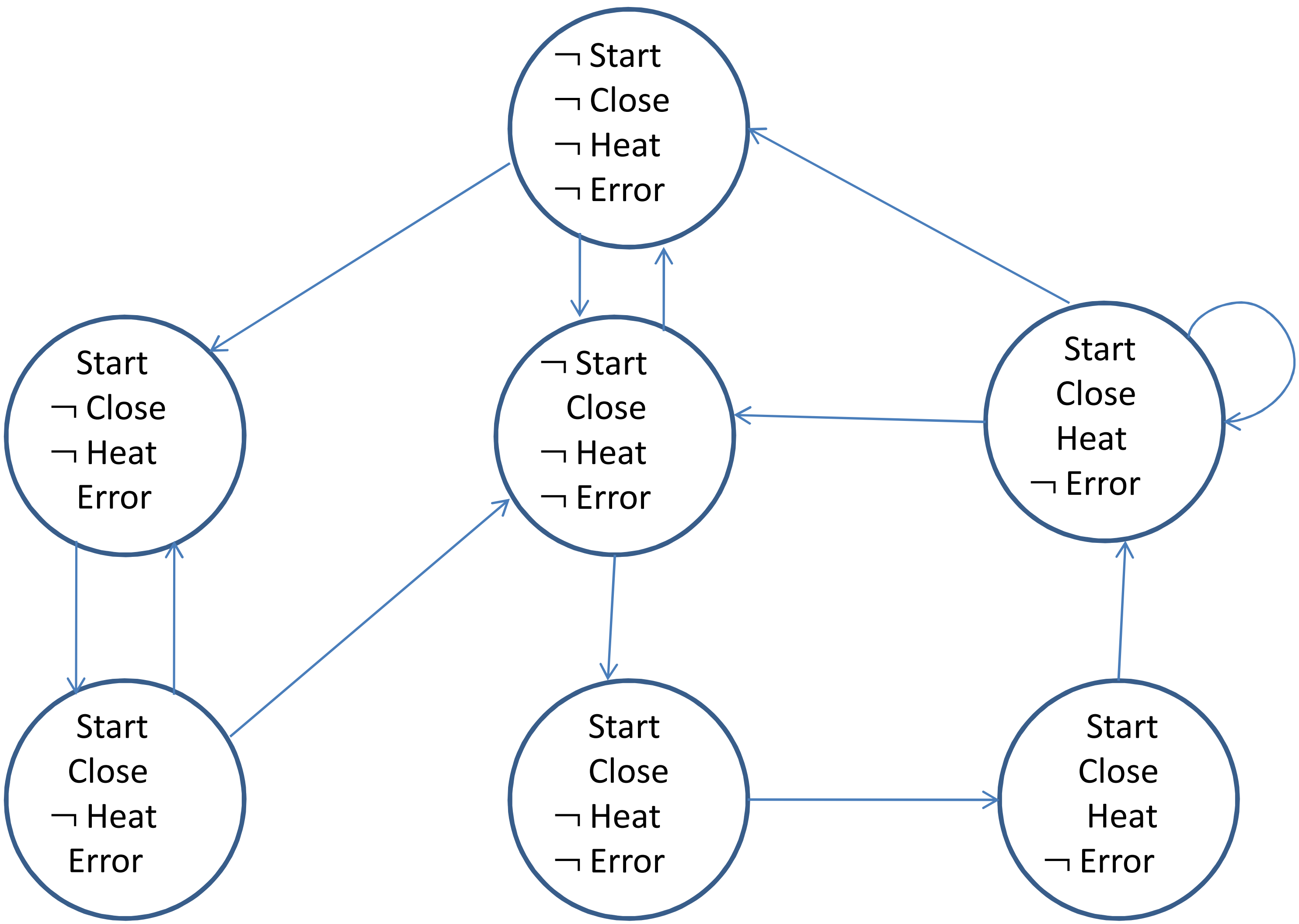
Question

- Draw the state transition diagram of MOD-6 counter.
  - Give a binary encoding to the states
  - Give the Boolean expression for the transition system
  - Indicate the labeling function

Question

- Consider the microwave oven controller and give the state encoding. What is the Boolean expression for the state transition diagram.

Question


¬ Start
¬ Close
¬ Heat
¬ Error
Start
¬ Close
¬ Heat
Error
¬ Start
Close
¬ Heat
¬ Error
Start
Close
Heat
¬ Error
Start
Close
¬ Heat
Error
Start
Close
¬ Heat
¬ Error
Start
Close
Heat
¬ Error

NPTEL Phase-II
Video course on

Design Verification and Test of Digital VLSI Designs

Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
IIT Guwahati

Module VI: Binary Decision Diagram

Lecture V: Symbolic Model Checking

Symbolic Model Checking

- Represent the transition systems with ROBDD
- Set of states can be represented by ROBDD

Symbolic Model Checking

- Basis of Model Checking
  - Graph Traversal algorithms
  - Need to find the predecessor states of a given state or a set of states

Symbolic Model Checking

Symbolic Model Checking

- Important relationship between $Pre_{\exists}(X)$ and $Pre_{\forall}(X)$:

$$Pre_{\forall}(X) = S - Pre_{\exists}(S - X)$$

S: Set of all states

X: Subset of S

Symbolic Model Checking

Procedure for $\text{Pre}_{\exists}(X)$

Given,

- B_X: OBDD for set of states X.
- $B_{\rightarrow}$: OBDD for transition relations.

Procedure,

- Rename the variables in B_X to their primed versions; call the resulting OBDD B_X'.
- Compute the OBDD for *exists*(x' , *apply*($\bullet$, $B_{\rightarrow}$, B_X')) using the **apply** and **exists** algorithms.

Symbolic Model Checking

– Rename the variables in B_X to their primed versions; call the resulting OBDD B_X'.

Symbolic Model Checking

– Compute the OBDD for exists(x' , apply($\bullet$, $B_{\rightarrow}$, B_X')) using the **apply** and **exists** algorithms.

CTL Model Checking

Function $SAT_{EX}(p)$

/* determines the set of states satisfying EXp */

local var X,Y

begin

  X := SAT(p)

  Y := $\{s_0 \in S \mid s_0 \rightarrow s_1 \text{ for some } s_1 \in X\}$

  return Y

end

Symbolic Model Checking

EX(B_ϕ):

B_ϕ:OBDD for set of states where ϕ is true.

// *Analogous to $X := SAT(\phi)$;*

$B_\rightarrow$:OBDD for transition relation.

Return $Pre_\exists(B_\phi)$. // *Analogous to $Y := \{s \in S \mid \text{exists } s', (s \rightarrow s' \text{ and } s' \in X)\}$;*

Evaluation of $Pre_\exists(X)$

Symbolic Model Checking

CTL Model Checking

```
Function SAT_AF(p)
/* determines the set of states satisfying AFp */
local var X, Y
begin
    X :=S,  Y := SAT(p),
    repeat until X = Y
    begin
            X:=Y
            Y := Y ∪ {s | for all s' with s → s' we have s' ∈ Y}
    end
    return Y
end
```

CTL Model Checking

Symbolic Model Checking

AF(B_ϕ):

B_ϕ:OBDD for set of states where ϕ is true.// *Analogous to "$Y := SAT(\phi)$";*

$B_\rightarrow$:OBDD for transition relation.

B_X : OBDD for all states of the system. // *Analogous to "$X:=S$";*

repeat until B_X=B_φ // *Analogous to "Repeat until $X=Y$"*

B_X:=B_φ // *Analogous to "$X := Y;$"*

B_φ:=apply(+, B_φ, $Pre_\forall$ (B_φ)) // *Analogous to "$Y := Y \cup \{ s \in S \mid$ for all s', ($s \rightarrow s'$ implies $s' \in Y)\}$"*

end

return B_φ

$Pre_\forall(X) = S - Pre_\exists(S - X)$

CTL Model Checking

Function $SAT_{EU}(p,q)$

```
/* determines the set of states satisfying E(p U q) */
local var W,X,Y
begin
    W := SAT(p), X := S, Y := SAT(q)
    repeat until X = Y
    begin
        X := Y
        Y := Y ∪ (W ∩ {s | exists s' such that s → s' and s' ∈ Y}
    end
    return Y
end
```

CTL Model Checking

Symbolic Model Checking

EU($B_{\psi 1}$, $B_{\psi 2}$):

B_X: OBDD for all states of the system. // *Analogous to "X := S*

$B_{\psi 1}$:OBDD for set of states where ψ_1 is true. // *Analogous to "W := SAT (ψ_1);"*

$B_{\psi 2}$:OBDD for set of states where ψ_2 is true. // *Analogous to"Y:=SAT(ψ_2);"*

$B_{\rightarrow}$:OBDD for transition relation.

repeat until $B_x = B_{\psi 2}$

$B_x := B_{\psi 2}$ // *Analogous to "X :=Y;"*

$B_{\psi 2}$:= apply(+, $B_{\psi 2}$, apply(•, $B_{\psi 1}$, $Pre_{\exists}(B_{\psi 2})$))) // *Analogous to "Y := Y$\cup$ (W $\cap$ { s $\in$ S | exists s', (s $\rightarrow$ s' and s' $\in$ Y)});"*

end

return $B_{\psi 2}$

Tools

- CUDD
  - CU Decision Diagram Package
  - University of Colorado at Boulder
- nuSMV
  - Extension of SMV, the first model checker based on BDD
- SPIN
  - LTL model checker developed at BELL labs

System Design Verification

- Model of the system
  - Kripke structure (kind of FSM)
- Specification
  - Specification language (like CTL)
- Verification Method
  - Model Checking

System Design Verification

- Model Checking Algorithms
  - Polynomial algorithm
  - Method can be easily automated
  - It provides counter example
- Problem with model checking
  - State space explosion problem
- Symbolic Model Checking
  - Use of OBDDs to content the state space explosion problem

Question

- We have discussed system model for
  - Elevator controller
  - Microwave oven controller
- Specification and verification of
  - Traffic light controller
  - Controller for ATM
- Use of tools
  - nuSMV and SPIN

Design Cycle: Digital Systems

- Specification
- **Design**
- **Verification**
- Implementation
- **Testing**
- Installation/marketing
- Maintenance

The Course: Digital VLSI Design

- This course is about Digital VLSI Design
- This course consists of three parts
  - Design
  - Verification
  - Test