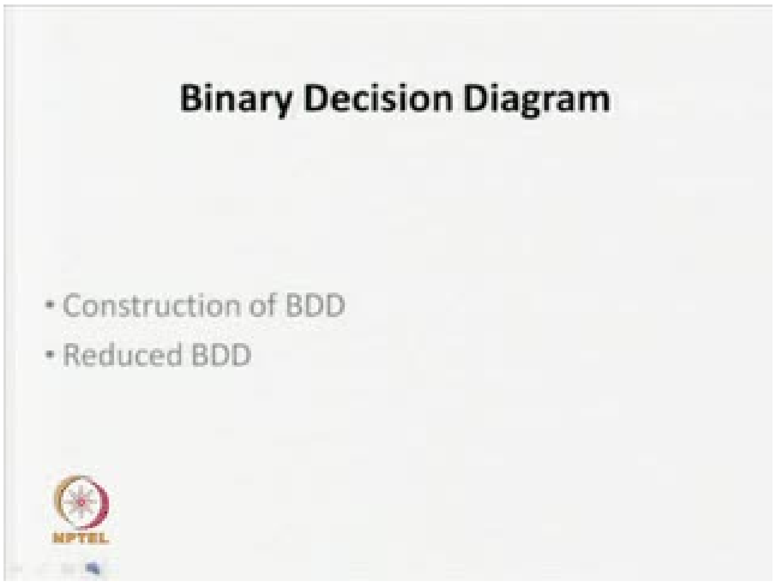
Design Verification and Test of Digital VLSI Designs

Prof. Dr. Santosh Biswas
Prof. Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

Module - 6
Binary Decision Diagram
Lecture - 2
Ordered Binary Decision Diagram

In last class, we have introduced the data structures called binary decision diagram and with the help of this binary decision diagram, we can represent any Boolean function or expression. And we have seen, how to construct this particular binary decision diagram.

(Refer Slide Time: 00:38)

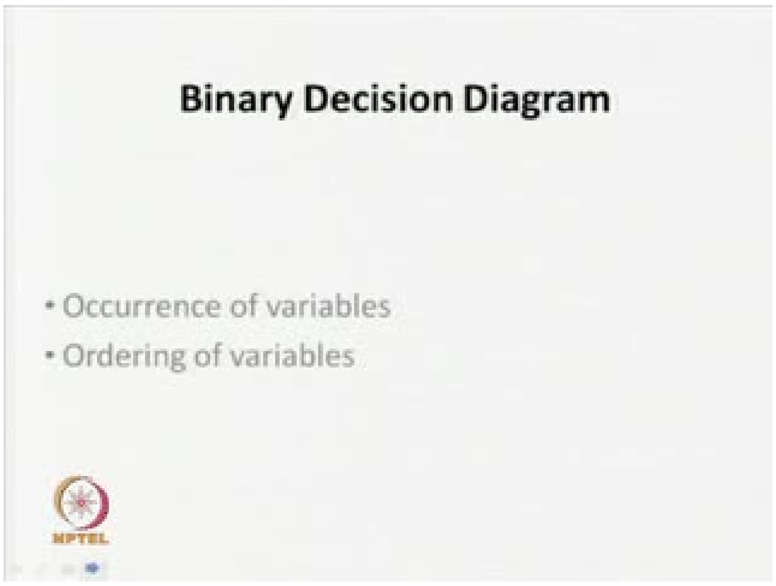


Binary Decision Diagram

- Construction of BDD
- Reduced BDD

So construction can be done from your two table, where we are going to get a binary decision tree or we can use the Shannon expansion of your Boolean expression to construct the B D D, binary decision diagram. And after that we have seen some reduction rule basically, we have 3 reduction rule; 1 elimination of duplicate terminals removable of redundant nodes and margining of duplicate non-terminals. With the help of these 3 reduction rule, we can reduce the BDD and we get reduced binary decision diagram. And in most of the cases, we have seen that we are going to get a compact representation of Boolean function; if we use reduce B D D, reduce binary decision diagram.

(Refer Slide Time: 01:34)

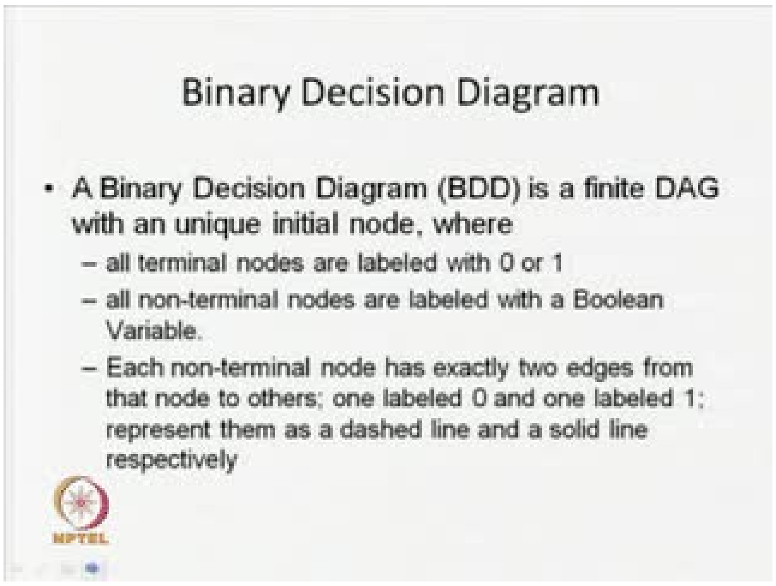


Binary Decision Diagram

- Occurrence of variables
- Ordering of variables

But, when you look into the construction of our B D D, we will see that we are not putting any restriction on the ordering of variable, the way it may appear in our BDD and again we are not giving any restriction of the occurrence of variables in a particular path.

(Refer Slide Time: 01:53)



Binary Decision Diagram

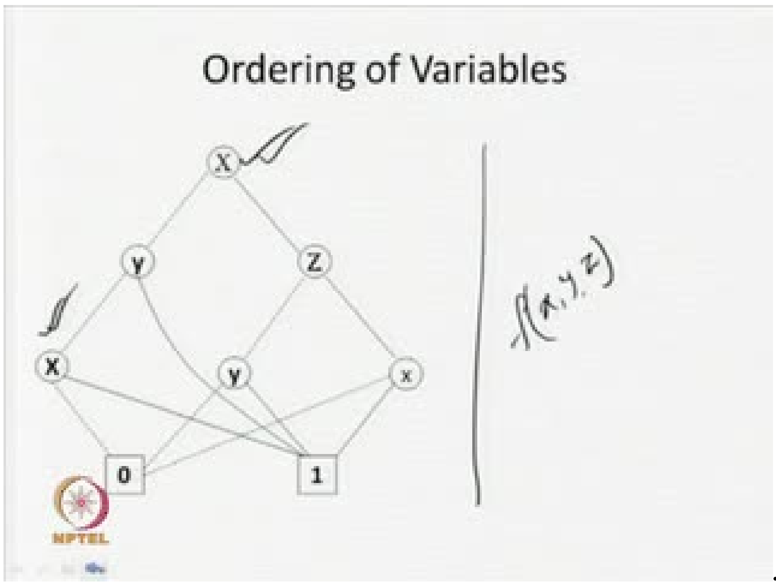
- A Binary Decision Diagram (BDD) is a finite DAG with an unique initial node, where
  - all terminal nodes are labeled with 0 or 1
  - all non-terminal nodes are labeled with a Boolean Variable.
  - Each non-terminal node has exactly two edges from that node to others; one labeled 0 and one labeled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

It may appear in many places, because if we look into the definition we are saying that a binary decision diagram is a finite DAG with an unique initial node. It is a finite DAG directed as I click up with an initial unique node, where all terminals are labeled with 0 or 1; all non-terminal nodes are labeled with Boolean variable. So, if we are having a

Boolean expression, we are having those particular variable in the Boolean expression and the term non-terminal nodes will be label by this particular Boolean variable and its non-terminal nodes are having two outgoing edges; one represent by dash line which specifically indicates, it is the valuation of that particular variable is 0 and another one is given by solid line which says that, the valuation of this particular variable is 1.

So this is the definition of binary decision diagram and we can construct binary decision diagram of any Boolean expression and while we are talking about this or when you look into the definition of this particular B D D, it is not talking about any, not talking anything about your occurrence of that variables, how many times it should occurred? Or it is not keeping any particular ordering.

(Refer Slide Time: 03:10)

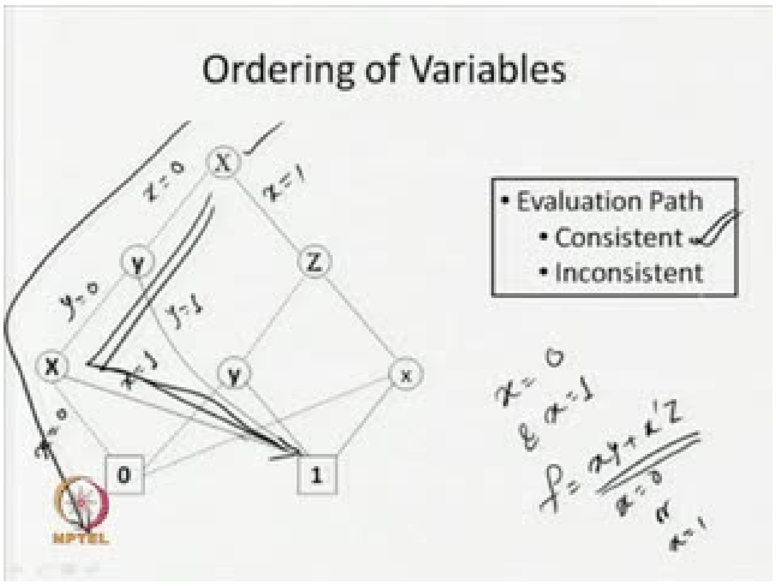


Ordering of Variables

.

So, if you look into this particular binary decision definition of binary decision diagram, then will say that, this is a binary decision diagram. This is simple DAG; we are having three variables say, it is a function of three variable f x y z. So x I am having over here, then this is dash line indicate the valuation of this x is 0 and valuation of this particular x is 1. Like that y z's are another variable and these are the two terminals known one is 0 and second one is 1. So if you look into it you see that, here in this particular case this particular x is appearing twice in this particular path because as per definition we do not have any restriction. Similarly, we can say that y is also appeared in two different position but if they are in a different parts.

(Refer Slide Time: 04:05)



Ordering of Variables

- Evaluation Path
  - Consistent
  - Inconsistent

Now in this particular case if you look into it, then by looking into this particular variable B D D, we are going to see some problems. We are talking in that particular case, we are going to have the notion of consistent evaluation path and inconsistent evaluation path. So when we draw this particular B D D, some of the path may be in consistent; I will say why it is inconsistent. So die valuation of this Boolean function will be done through consistent part only. Now you just see that, I am saying that this is x variable here I am talking x equal to 0 and in this particular part I am talking x equal to 1. When value of x equal to 0, I am going to take decision of y, this is in this particular part y equal to 0 and in this particular point y equal to 1.

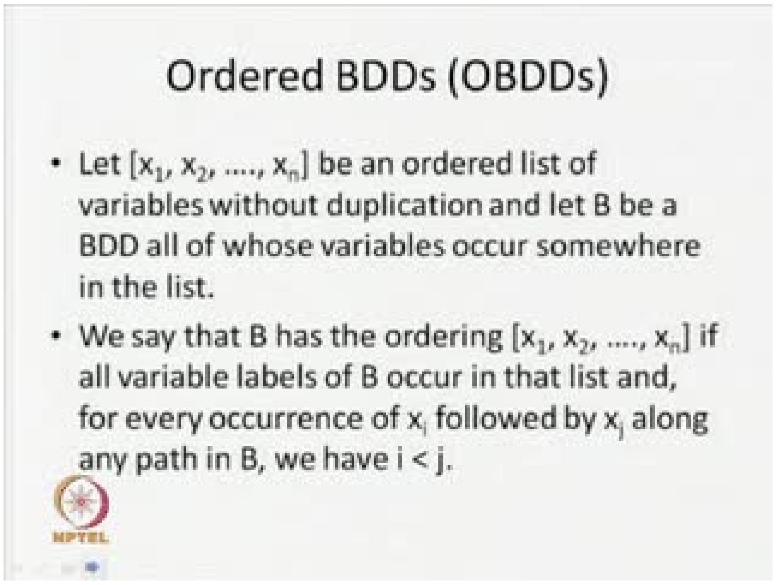
After that when I come to this particular node, then again I am going to take decision on x; then I am going to say that in if a follow this particular path, then x equal to 0 and if I follow this particular part then x equal to 1. Now this is the way that we can see an ultimately, we are going to get this functional value. Now if I follow this particular path in this evaluation path, it is same for that value of x equal to 0 and value of y equal to 0 and we are going to say that, the functional value is 0 if x equal to 0 and y equal to 0; it is independent object in this particular evaluation path.

Now if I look into this particular evaluation path, I am showing by double line. Now what happens? I am going to say that, x equal to 0 and valuation of y equal to 0 and x equal to 1. Now in this particular path you just see that, value of x is taken as equal to 0 and it is also taken as x equal to 1. Now, when we are going to look for a valuation of any particular function at any instant, we can have only one valuation for one variable; so just say that if I am randomly writing on function say f equal to x y plus x bar z.

Now in this particular case, when I am going to look for the valuation or evaluation of this particular Boolean function, either I am going to say that, evaluate it either x equal to 0 or I am going to say that either x equal to 1. So for these two different combination, I am going to get two different valuations, but in this particular path, what will happen? The value is treated as x equal to 0 as well as x equal to 1 which is not possible and which is not permitted.

So in this case we are going to s ay that, this is an inconsistent path. So that means, we are having the notion of inconsistent path in B D D, if we look into the basic definition of BDD that means; the valuation of the function or evaluation of the function has to be done through consistent path only that means; we should not look for the inconsistent.

(Refer Slide Time: 07:07)



Ordered BDDs (OBDDs)

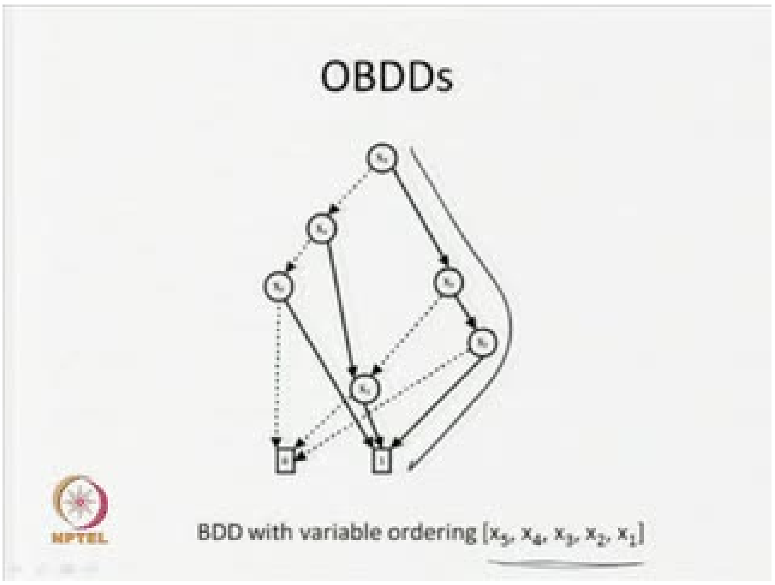
- Let $[x_1, x_2, \ldots, x_n]$ be an ordered list of variables without duplication and let B be a BDD all of whose variables occur somewhere in the list.
- We say that B has the ordering $[x_1, x_2, \ldots, x_n]$ if all variable labels of B occur in that list and, for every occurrence of x_i followed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

Now due to this particular problem, we are trying to eliminate this particular problem and in this particular case, what we are going to do; we are going to put some restrictions on the occurrence of the variables and in that particular case, we are going to put a particular ordering of that variables and after putting this particular ordering of the variable, we are going to get ordered B D D; ordered binary decision diagram. So in case of ordered binary decision diagram that means; variable will come in a particular order.

Let consider that, x 1, x 2, x n be an order list of variables without duplication and let b be a B D D, all of whose variables occurs somewhere in the list. Now we are considering a BDD where, all the variables of this particular list is occurring somewhere in this particular BDD and we are saying that, this is an order list where duplication is not allowed. We say that b has an ordering x 1, x 2, x n, if all variable of b occur in that list and for every occurrence of x i followed by x a along any path in b, where i is less than j that means; if we are going to fallow this particular ordering say x 1 to x n. So when we draw this particular B D D, then x 1 must always come before x 2 or may be x 1 always comes earlier point of x n; it should follow this particular ordering.

So when we are following or practicing this particular ordering of B D D, then we will find that, one particular path only 1, the variable will appear only once that means; since it is appends only once. So that multiplicity of valuation like, x equal to 0 and x equal to 1 that when the way we have seen in the earlier example we will not occur. So that means; in case of ordered B D D, we are going to follow a particular ordering of the variable.

(Refer Slide Time: 08:38)



OBDDs

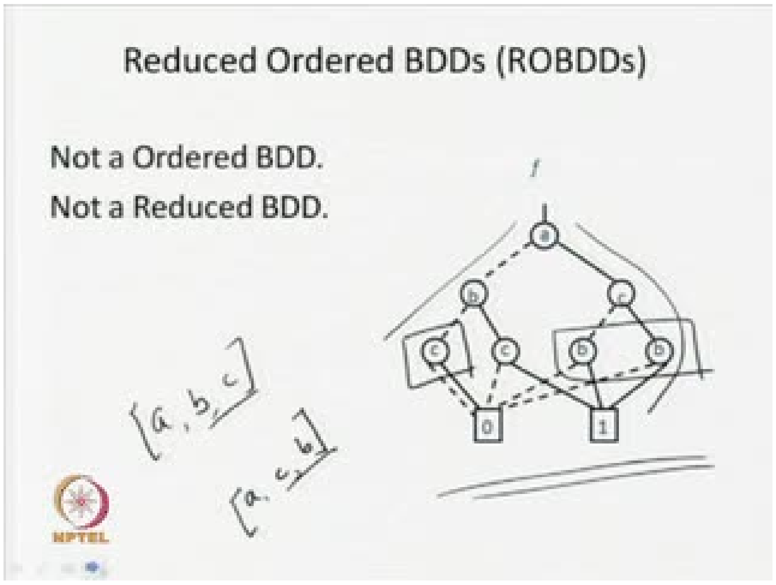
BDD with variable ordering $[x_5, x_4, x_3, x_2, x_1]$

Just a simple example you see that, I am having an BDD over here and we are saying that, we are having five variables x 1 to x 5 and we are considering this particular ordering that means; x 1, x 2, x 3. x 4 and x 5, this is the ordering of variable. When we mention this particular ordering, then what we have to see that x 1 always must occur before x 2, x 3, x 4 and x 5. Similarly, x 3 must always occur before x 5 in any path in this particular B D D. So if you look in any path, you will find that this is x 1, x 2, x 4 then we are having evaluation values.

So now where it is saying that, it is evaluate everything this particular ordering that means; x 3 is not coming before x 2 in any of path. Similarly, x 5 is not coming before x 1 in any of the evaluation value. So in this particular case, we are going to said this is an ordered binary decision diagram and that means the variables are following a particular ordering. Since they are following a particular ordering, duplication occurrence of a multiple times of a variable in a particular path is not allowed. So this the way that we can say this is ordered binary decision diagram.

Similarly, this is another binary decision diagram, ordered binary decision diagram and in this particular case, we have following these particular ordering say; x 5, x 4, x 3, x 2, x 1. That means when you look a name evaluation path in this particular path the variable must occur in this particular ordering that means; x 5 will come first then x 3, then x 2, then it is coming to the terminal node 1 so it will follow this particular 1.

(Refer Slide Time: 10:52)



Reduced Ordered BDDs (ROBDDs)

Not a Ordered BDD.

Not a Reduced BDD.

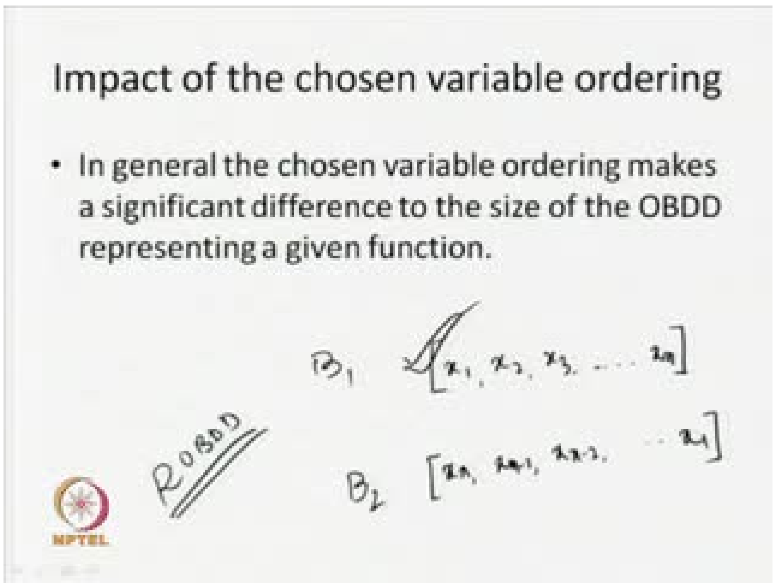
Now when we are using this particular ordered binary decision diagram, then variable will appear in a particular order in any execution, any evaluation path of this particular B D D. So appearing a particular variable multiple times in a particular path is avoided that means; the notion of in consistent path is avoided over here that means; all evaluation paths are now consistent. So this is the notion about ordered binary decision diagram, so we are going to pull a particular ordering of the variable.

Now if you consider this particular B D D, just say I am drawing a random BDD over here, I think in last class also I have drawn this particular B D D; now slickly sensed a level over here. So in this particular case, I am saying that it is not an ordered B D D; why it is not a ordered B D D? It is not fallowing any particular order in an angle.

Say, if I fallow this particular path, then the ordering is your a, b, c. This is the ordering of variable in this particular model but if I fallow this particular order path, evaluation path, then I am going to get the ordering as your a, c, b. You now see that, a is appearing before b is All Right but in this particular path b is appearing before c and in this particular path c is appearing before b that means; it is note following any particular ordering of the variable, so that is why it is not an ordered B D D.

Secondly, we are saying that, this is not a reduced BDD why? We can see some instances look into this particular node c, we are having a retardant test over here. So c equal to 0 or c equal to 1 it is equal to 0 that means; this is retardant test; so this retardant test can be removed. Secondly if you look into this particular two nodes, b equal to 0 it is evaluated 0, here also b equal to 0 evaluated 0; in this case b equal to 1 evaluated to 1 b equal to 1 evaluated to 1 that means; these are read you can say that duplicate non-terminals.

(Refer Slide Time: 12:37)



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.

$B_1 \; [x_1, x_2, x_3, \ldots x_n]$

$B_2 \; [x_n, x_{n-1}, x_{n-2}, \ldots x_1]$

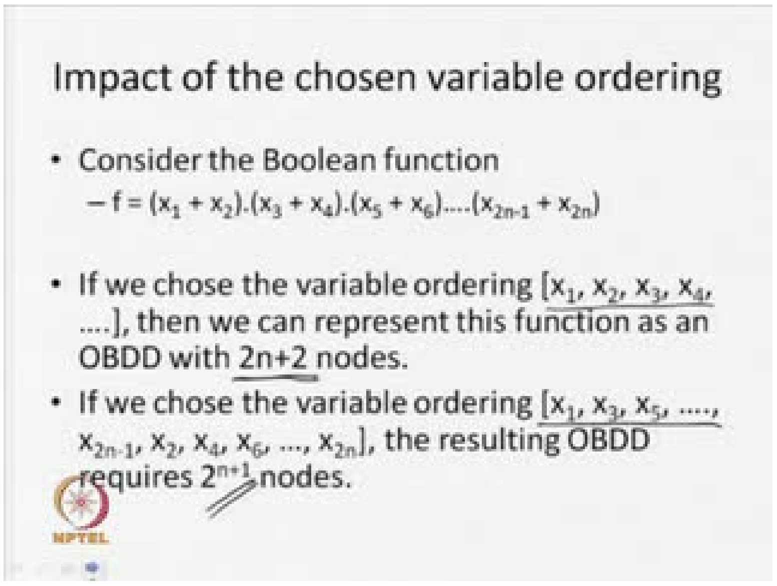
ROBDD

So these non-terminals can be merge to one non-terminal; so since we can apply the reduction rule to this particular B D D. So that is why it is not an reduced B D D, this is not ordered because we are not having the similar order in out of it and it is not reduced; so this is simple example that I am giving.

Now we are having some impact of the chosen variable. Now we can say that I am going to say that we are going to put a variable ordering, say some x I am having some variable x 1, x 2, x 3 like that say x n; this may be one variable ordering. Secondly I can say that; x n, x n minus 1, x n minus 2 like that up to x 1 this may be another variable ordering. So with a help of this particular variable ordering, I can get an BDD I will say that, this is your BDD B 1 and with a help of this variable ordering, I can again have the BDD representation of the given function and I say that this is your b 2. Now we are getting these two B D Ds but these two B D Ds are representing the same Boolean function.

Now after that will use the reduction of these things, so we are going to get reduce order binary decision diagram. Now in this particular case, we have to see what will be the size of b 1 and b 2, it may happen that the size may vary in both the cases; one may be bigger one and one may be smaller one. So that means the ordering of variable is going to material doped for the size of our BDD; I will explain this things with the help of one simple example.

(Refer Slide Time: 14:05)



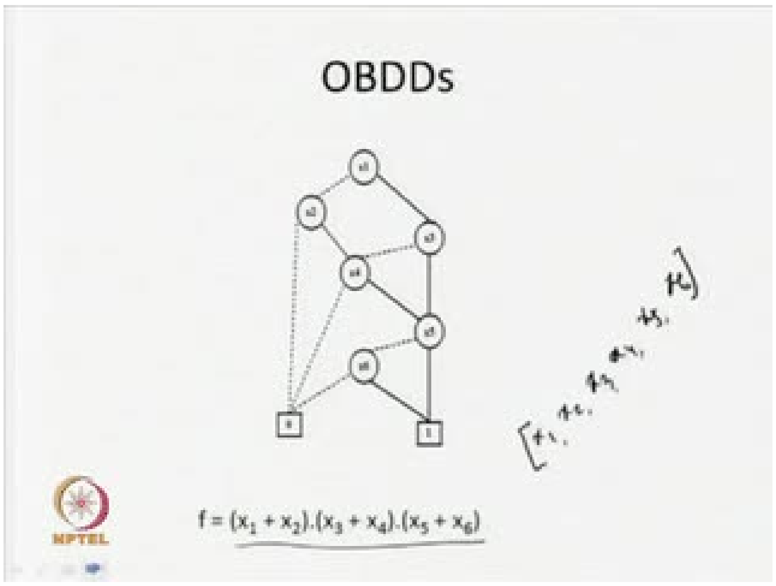
Impact of the chosen variable ordering

- 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.

So just see that, consider this particular Boolean function f, we are saying that x 1 plus x 2 x 3 plus x 4 x 5 plus x 6 like that up to x twice n minus 1 plus x twice n that means; we are having twice n variables. So this is a function of twice n variable and this is simple function. Now in this particular case, if we choose the variable ordering x 1 to x 2 up to x n x twice n say this is one particular variable ordering. Then in that particular case what happens, we can draw the BDD for this particular function and the ordered BDD will have twice n plus two nodes, we will see. So we can construct the BDD for this particular Boolean function and that particular Boolean function will have 2 to the power twice n plus two nodes and we will see that, no more deduction can be possible and we can said that, this is the reduced order binary decision diagram.

On the other hand, if we choose the variable ordering something like that x 1, x 3, x 5 then up to x twice n minus 1 then x 2, x 4, x 6 up to twice n that means; all the odd subscription I am going to put first and then even subscription I am going to put that all. So this is also possible variable ordering, so if we use this particular variable ordering, then will see that total number of nodes that will be having is your 2 to the power n plus 1. So, if we having n twice n variable the total number of your this things, that nodes that will appear in this BDD is your 2 to the power n plus 1 and the earlier case it is twice n plus 1. So you can see that they have considered difference of our size of the BDD and it basically depends on the variable ordering.

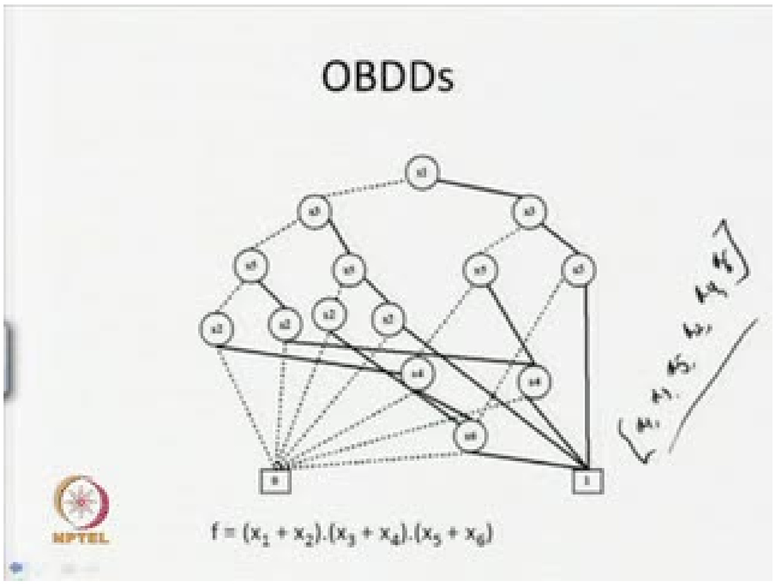
(Refer Slide Time: 15:59)


OBDDs

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

You just see, this is the BDD I have drawn this is basically the same function x 1, x 2, x 3, x 4, x 5, x 6 and the variable ordering that we are having is your x 1, x 2, x 3, x 4, x 5, x 6. So in this particular case, I am getting this particular BDD I am having six non-terminal nodes and two terminal nodes.

(Refer Slide Time: 16:25)



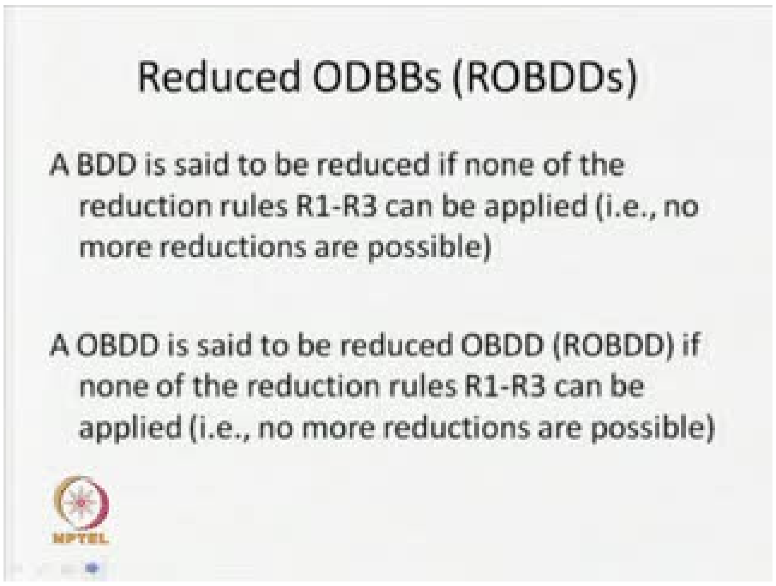
OBDDs

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

Now if I sends the variable ordering, then we are going to get this particular representation this BDD so here, the variable ordering basically I am taking x 1, x 3, x 5, x 2, x 4, x 6; this is the variable ordering x 1, x 3, x 5 then x 4, x 6, here x 3 separate like that. So in all the evaluation path, you will find that it is following this particular variable ordering. Now when I am having this particular variable ordering, then you see that I am getting a b or B D D.

So this is the same function I am representing with one ordered B D D, so we are ordering is this from x 1, x 2, to x 6 and the same Boolean function I am representing with another order BDD where the order is different and we have see that the size is more. So the size basically, size of the ordered BDD depends on your the variable ordering and if you look into this particular things no more reduction rule is possible for this particular B D Ds and on the other hand, for this BDD is also no more reduction rule is possible. So these are basically, reduce ordered binary decision diagram.

(Refer Slide Time: 17:33)



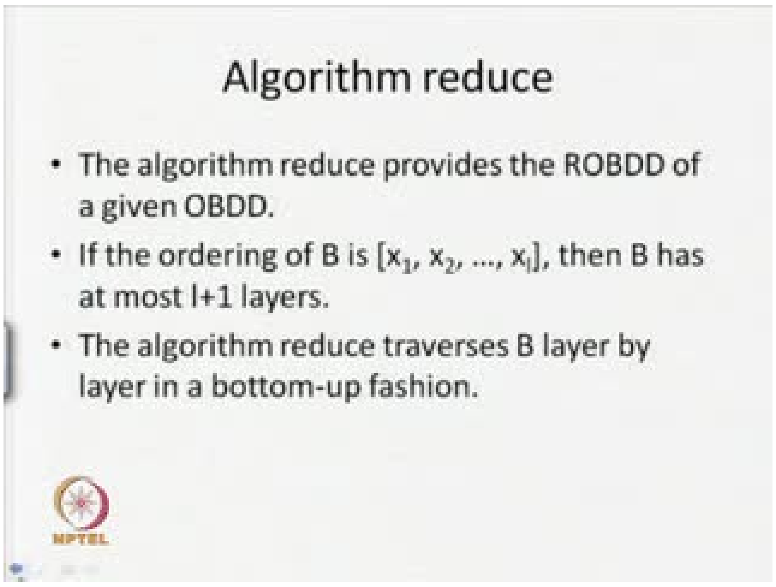
Reduced ODBBs (ROBDDs)

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)

So that is why you are saying that a BDD is said to be reduced, if none of the reduction rule can be apply and similarly, for ordered BDD ordered binary decision diagram we are going to said that it is a reduce ordered binary decision diagram ROBDD if none of the reduction rules can be apply in of order. So eventually you are coming to reduce order binary decision diagram.

(Refer Slide Time: 18:24)



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.

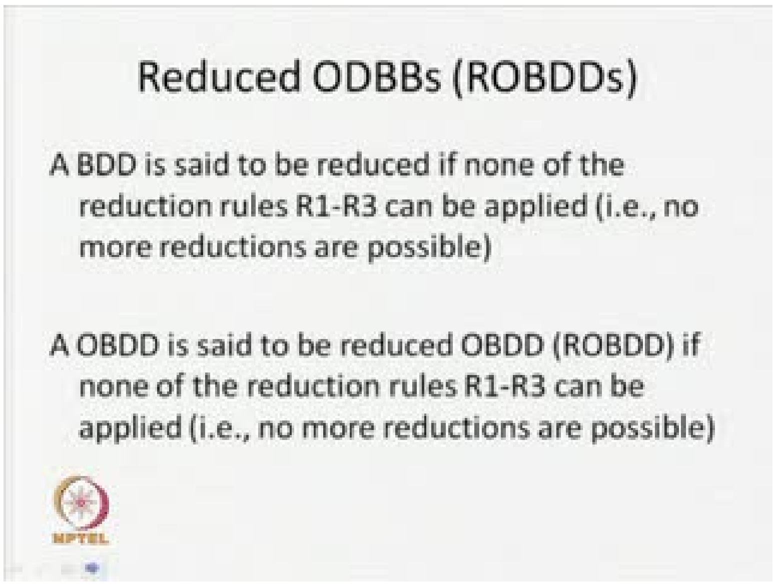
So by it is a binary decision diagram so variable are having a particular ordering so we are going say this is the ordered binary decision diagram. Entirely, it is a reduced one no more reduction rule is possible so we are going to say this is reduced order binary decision diagram and eventually we are going to worked with ROBDD, which is having all the required properties from us.

Now in this particular case you just see that, already I have mention about this particular ordering of variables depending on the ordering of our variables, the size of our ordered BDD will vary. In some cases, we are going to get a compact representation and in some cases, we are going to get slightly bigger representation or bigger ordered BDD or bigger reduce ordered B D D.

Now, how to get the proper variable ordering? So, that we can get a compact representation of this particular Boolean function, so that means we have to look for the proper variable ordering but to get the proper variable ordering to get a compact representation of the function in BDD is a hard problem. We do not have any method or we do not have any algorithm to say that, this particular variable ordering is the best variable ordering and it will give us the compact representation of a given Boolean function.

So this is a hard problem, it is difficult to find out that particular based variable ordering. So this is a open problem plus till working on it whether it can be solve or not. Since it is an open problem, it is a hard problem. So currently, we are using some heuristics and with the help of this heuristics method, we will try to find out some variable ordering which are going to give us some compact representations. It may not be the minimal one the BDD that, we are going to get may not be minimal one but it is going to give us some compact representation. So we apply some heuristic and use by using this particular heuristic thing, we try to find out some variable ordering and after that, we apply this particular variable ordering to construct the BDD and in most of a cases, we have found that we are getting a each and every good size of BDD representation of any function.

(Refer Slide Time: 20:19)



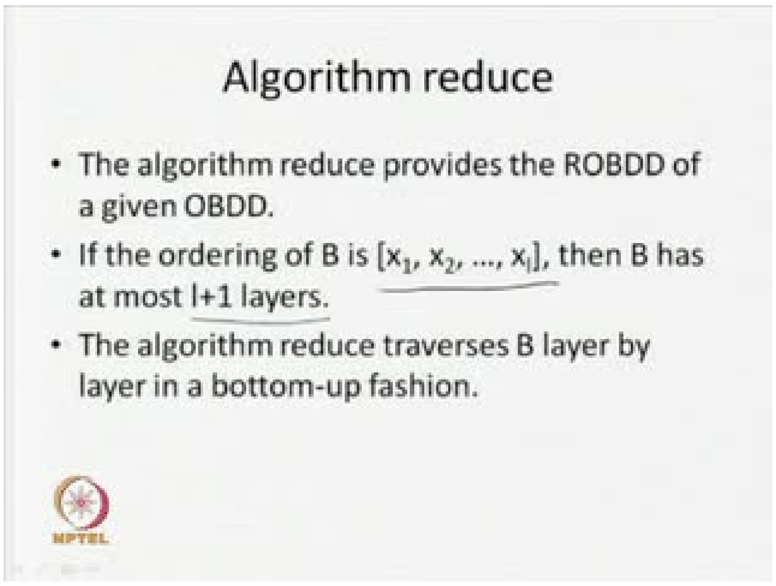
Reduced ODBBs (ROBDDs)

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)

Now, we have introduce a data structure call BDD binary decision diagram, which is use to represent any Boolean function and later on we have seen that; ordering of the variable and reduction of that particular ordered binary decision diagram and eventually we are coming to ordered binary decision diagram and we use this particular ROBDD to represent any Boolean function.

(Refer Slide Time: 20:47)



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 $\underline{l+1 \text{ layers}}$.
- The algorithm reduce traverses B layer by layer in a bottom-up fashion.

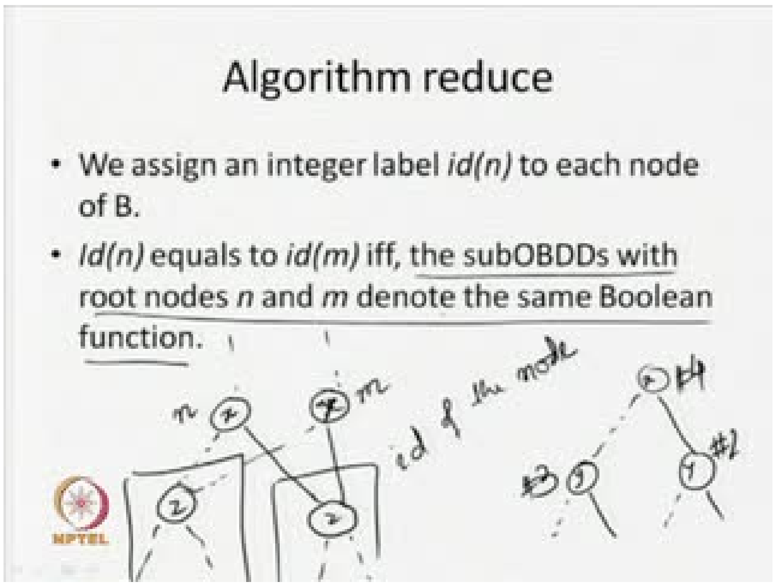
Now after that we now, since we are having this particular BDD representation of Boolean function, we may need some algorithm or we need some method to work with those particular BDDs. Now once you just see that I am giving a Boolean function and you have come up with a variable ordering or you have chosen a particular variable ordering, then we are going to construct the BDD, or ordered BDD for that particular function and you can very well constructed, with the help of Shannon’s expansion of the Boolean function. But eventually whatever you are getting it whether it will be a reduced one or not, you have to check it or if it is not reduced one, then you have to construct the reduce BDD reduced ordered binary decision diagram and already we have mention that, we are having three rules to go for a reduce BDD and we can apply this particular three rules to the initial B D D.

Now we will going to see an algorithm, we are going to put this particular rule in an algorithm form and we will say or we will get an algorithm call the algorithm reduce and we will apply this particular algorithm reduce to reduce any BDD. So basically, what will happen in this particular algorithm reduce, we are going to give a BDD or basically we are going to give a ordered BDD as an input and the output, that we are going to get a reduce order BDD and the variable ordering of the output B D Ds will be same as with the input BDD. So we will look into this particular algorithm.

So, if we are going look an ordering of some variable say x 1 to x l that means; we are having l variables, then we can have at most l plus 1 layer in our BDD. So we are going to have l plus 1 layer at most, but in some cases in some part it may be less also because that particular function may be independent of that variable.

Now algorithm traverse this particular, that algorithm reducible traverse that B, that final decision diagram B layer by layer in a bottom-up fashion. So we are going to have a representation of our Boolean function BDD representation and this algorithm reduce is going to traverses it from the bottom layer to the top layer. So that means we will start form the terminal nodes and after that, we will traverse it back to the root nodes and eventually, we are trying going to apply some rules; so that we can get the reduce BDD.

(Refer Slide Time: 23:29)



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.

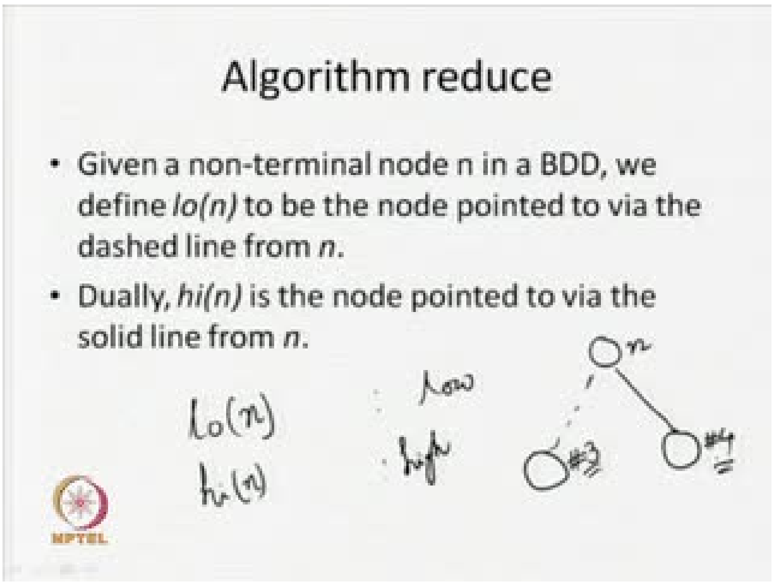
n
x
x
m
2
2
id of the node
#4
#3
#2

Now here in this particular case we are going to an assign an integer label to each node, so first one is the labeling after node. So say if I am having some BDD something like that, say this is your x, this is variable y we are going to have some ordering. So in this particular case, we are going to assign some integer label to each node, I can say that I am not assigning that integer label 3 to 8 and I am assigning that integer label 2 to 8 and I am going to said that, integer label 4 assign to this particular reduce. So this is an integer label of these particular nodes and we are going to say that, this is basically the id of this particular node, id of the node.

So basically we are going to give an id and id is nothing but the integer label of its node and we can say that id of a particular b d node and will be equal to id of m if the sub ROBDDs with root nodes n and m denote the same Boolean function. So basically if sub BDD denotes the same Boolean function, then we are going to give them the same your this thing, so recall integer label just I will say that this is one node say labeled with your x and say this is y. So with evaluation 0, we are going to same node and with evaluation 1, we are coming to same node say this is z and z.

Now in this particular case, what will happen? You just see that and after that we may have this particular BDD. Now for is particular node, say this is n I am talking about and say this is m. So once we take the value of x equal to 0, it is going to look for this particular sub BDD. Similarly, for this particular node also say y is equal to 0, then sorry if they are having the same. So in this particular case, it is also x is equal to 0, they are going to have the some sub BDD. Similarly, for 1 both are having the same sub BDD that means; both this particular nodes are evaluating the same sub formula for value x equal to 0 and x equal to 1. So in that particular case, we are to give the same label to this node n and m so we are going to give some label or id and this label will be same for this particular this nodes. So if they are going to have the same sub B D Ds, then their label will be same; this is in the rule through assign the label to each and every node.

(Refer Slide Time: 26:09)



Algorithm reduce

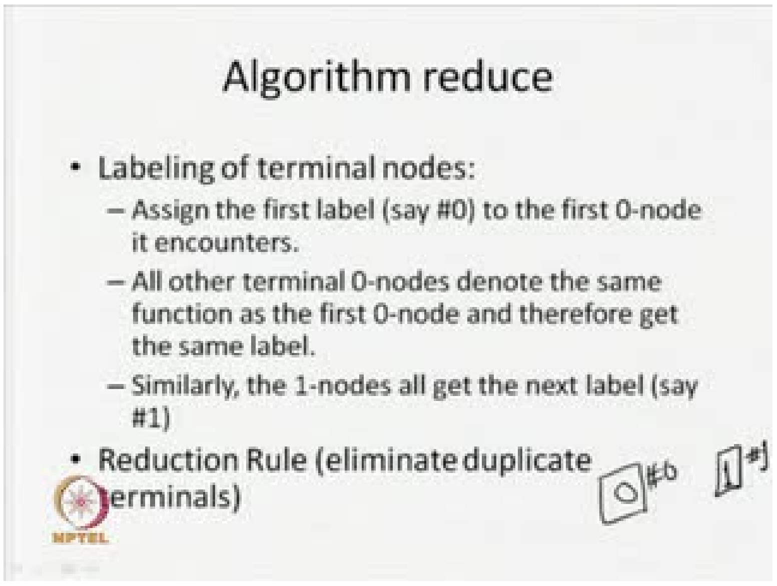
- Given a non-terminal node n in a BDD, we define *lo(n)* to be the node pointed to via the dashed line from *n*.
- Dually, *hi(n)* is the node pointed to via the solid line from *n*.

Lo(n)
hi(n)
low
high
n
#3
#4

And another one we are going to define which is basically two function; one is your low and one is your high. So for every non-terminal nodes we define a functional call lo n, so this is basically low to be the node pointing to via the dashed line from n and similarly, we can have hi of n which is basically high. So if I am having any node say on over here, so I can have the dash line and solid line. So in this particular case, are the label of this particular node is a 3 and label of this particular node is say 4.

So in this particular low of n, it is going to written this particular label 3 because it is low to be the node pointed to via the dash line; so it is pointing this particular case. Similarly, high will written the value 4, because it is pointing to this particular node whose label is 4. So basically low allow, low and high it say are two function, it is going to give me the nodes pointed by dash line and node pointed by solid line respectively. So we are defining these two functions, say when with the help of this thing, we are going to construct this particular reduce algorithm; so first task is to label each and every node.

(Refer Slide Time: 27:36)



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)

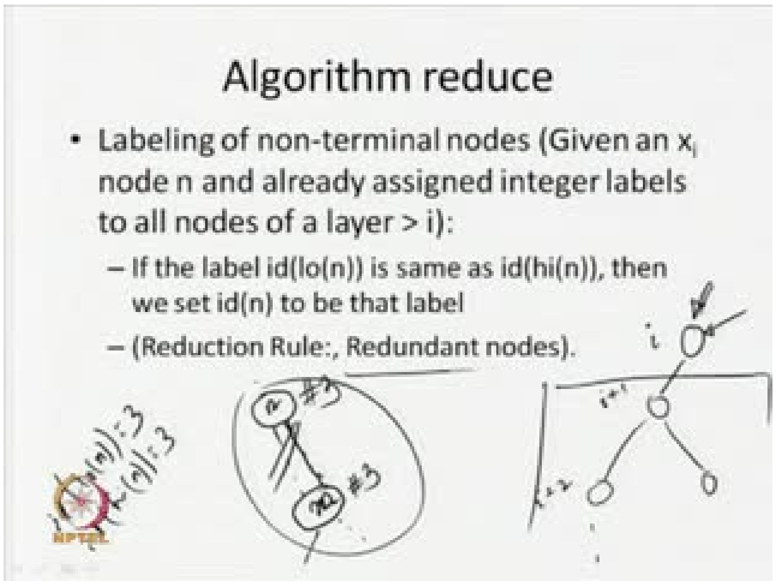
$\boxed{0}$ #0
$\boxed{1}$ #1

Now next task is to label the nodes, so how we are going to label the nodes? So label the first label say 0 to the first 0-node it encounters. So for basically say, we are going to follow from the terminal nodes and following in the bottom of expose approach. So first we are going to get some node 0-node, we may have one 0-node and more 0-node label by 0. Then we are going to assign a label 0 to those particular 0-nodes and similarly, we are having some nodes which are label terminal nodes, which are labeled with 1 and we are going to assign label 1 to those particular node.

So if we are having more 0-nodes, we are going to give the same label to those particular 0-nodes and we are going to give a 0, label 0 and we may have more 1-nodes to all the 1-nodes, we are going to get label s 1. So basically this is the starting of our labeling algorithm and we are starting from the terminal nodes and we are going to follow the bottom of approach.

Now what we can say, now in this particular case, we are giving that label 0 and 1 now. Later on what we are going to do, the nodes which are having the same label will be merge together basically, this is a second phase of our reduce algorithm first we are going to label each and every node and next phase, we are going to merge the nodes which are having the similar label. So in this particular case, all 0-nodes will be label merge to 1-node then 1-node will be merge to 1-node label by 1. So this is nothing but the reduction rule 1, that we have elimination of duplicate terminals so that means; we are going to take care of this duplicate terminals one, say removal of duplicate terminals so all zero node will be merge to one 0-node and all one nodes will be merge to 1 1.

(Refer Slide Time: 29:41)



Algorithm reduce

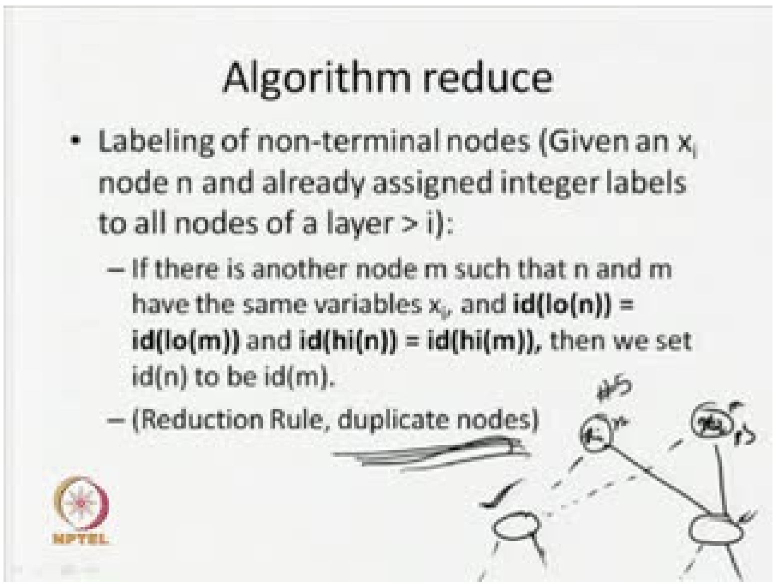
- Labeling of non-terminal nodes (Given an x_i node n and already assigned integer labels to all nodes of a layer > i):
  - If the label id(lo(n)) is same as id(hi(n)), then we set id(n) to be that label
  - (Reduction Rule:, Redundant nodes).

Now secondly if we are having now, once we label those particular terminal nodes, next we are going to follow the bottom of approach. Then we are going to get some non-terminal nodes, labeling of non-terminal nodes given an x i node n and already assign integer label to all nodes of layer i is, layer is greater than i that means; when we follow this particular labeling. So this is I am coming label i, then i plus 1, then i plus 2 like that up to terminal labeling. When we come to label this particular node, then what it will happen? All the nodes below this particular label particular label must be marked, we should have all the integer labeling of all those particular node below this particular then only we can label.

Since we are following the bottom up approach so when we reach this particular node, then all the nodes below this particular node will be all ready label. If the label l id lo n is same as id hi n, then we said id n to be that node, of that label that means; you can say that, if I am having a particular node n over here and say low and high both are coming to one particular node say m. So in that particular case say, id of low n say this is say I can have say some label 3, so id of low of n is equal to 3 similarly, id of i of n is equal to 3.

So if both are same then what will happen? We are going to give the same label to this particular node, so I can say that this is also labeled with 3. So what basically it means, that now in the next page we are going to merge them together, so these will be merged together because they are having the same label and you said that this is nothing but the reduction rule of redundant nodes specially removal of redundant node because here I am having some redundant test for both 0 and 1, it is going to give the same sub BDD. So if the id of low n and id of high n is same, then n will get the label of this particular n only. So this is basically we are labeling putting an integer label and basically, it is going to help us to use this particular reduction rule removal of redundant nodes.

(Refer Slide Time: 32:07)



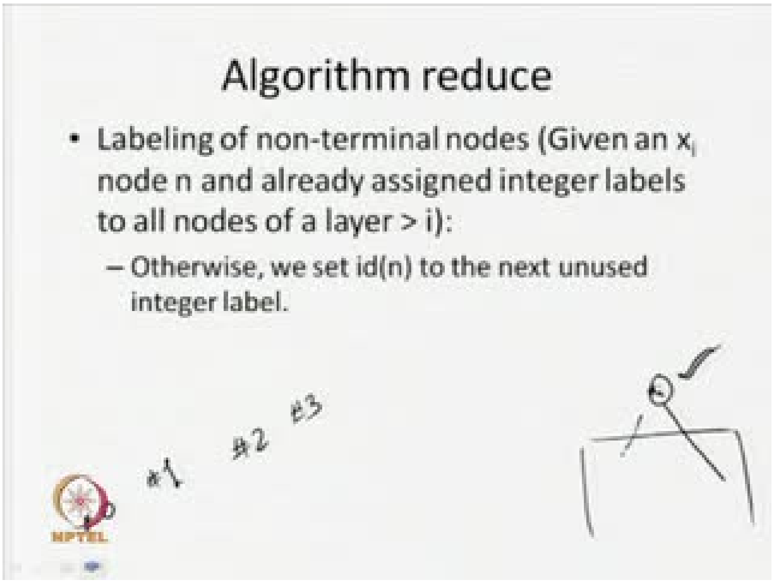
Algorithm reduce

- 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)

Next we are going to say that, already I have say that if I am coming to x i node, then all the nodes below that label is already marked. Now if there is another node m such that m and n have the same variable x i that means; I am having a node n and I am having a node m and both both are labeled with same variable say x i. Now what will happen in this particular case, if id lo n is equal to id lo m that means; this is low and this is low some label we having sorry this is low.

Similarly, i and i coming to the same nodes or it may cover to the define node but both are having the same label. So in this particular case, say low and low n and low m is coming to the same sub BDD and similarly, high n and high m are coming to the same sub BDD then id of n will be equal to id of m. So if I am having a already say, id of this particular nodes say m say is equal to 5, then I am going to assign the same label to this particular node also and it is 5. So if you see this things later on we are going to merged them together that means; we are this is basically corresponds to the rule of removal of your duplicate nodes. Basically these two are duplicates because; they are having the same sub BDD below that they are going to evaluate the same sub function so we can merge them together. So this is basically nothing but the removal of these particular duplicate nodes.

(Refer Slide Time: 33:59)



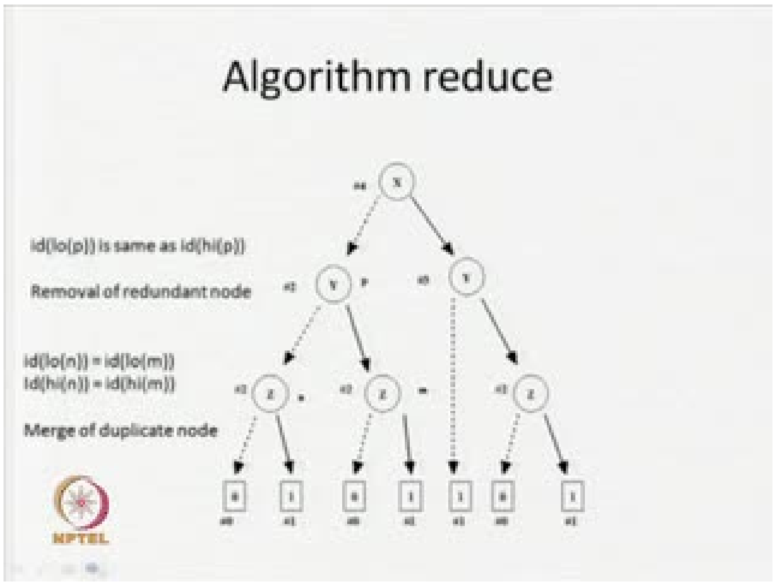
Algorithm reduce

- 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.

So in this particular case, if it is not getting this to two nodes say I am coming to your particular node x i and all the node below this particular node is label and if it is not following these particular nodes say, this is the first one and this is the second one; it is not following this particular two cases, then what will happen? Otherwise we set id of n to be the next unused integer label. So here starting with integer 0 for the 0-node, then we are starting with integer 1, then if they are not equal then I will go to next unused number 2; then go to next unused number 3 like that. So if it is not following these two particular rules that we have already mentioned, then set id of n to be the next unused number.

So next unused number will be given me that means; it is an new node we are getting and it will be appeared or it will be present in r it will present in our reduced BDD. So once we are having these things, then what we have going to do then already I have mentioned that. The next phase what we have going to do? We are going to merge the nodes which are having similar label that means; that can be removed.

(Refer Slide Time: 35:01)



Algorithm reduce

So just we are going to look an example say, this is an BDD we are constructing x y z and this is basically BDD also. So if you look into it how we are going to start it? First we are going to start the terminal nodes. So in this particular case, we are going to give 0 to all 0-node and assign 1 label 1 to all 1-nodes; this is the first two labeling the terminal nodes 0 and 1. Then we are going to look for this particular thing say, in this particular case z 0 is going to 0-node and 1 is going to 1-node. So since these two are define so it is going to get a new label.

Now when we are coming to this node, say these two are having define so it should get a a new level but secondly already we are having a z-nodes so we will compared these two nodes and the low of this particular z is same with low this particular one and high of this particular z is high of this one. So this node will get the same label over here because there they are having the same some BDD for 0 and 1.

And similarly, when I come to this particular node z label by z, I can we find that they are having two deferent nodes 0 and 1; so it should be at a defined label but if I consider about these nodes, then what will happen the behavior is same so it is going to get this particular label 2. So now, first we are labeling the terminal nodes, then we are coming one label up of after label this particular node; then will go to the next label. So since I am having three variables, so total we are having four labels you are distinct what we called non-terminal nodes for three variables and one label is the terminal nodes.

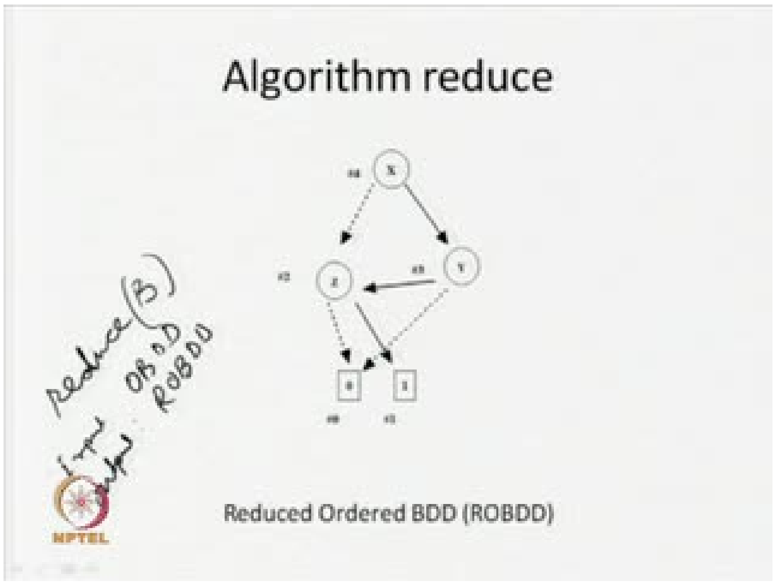
Now similarly, now when we come to this particular node, you just see that low of your this particular node is pointing to that label two nodes and high of these particular node is label 2 this particular label 2-nodes. So that means in low of your this nodes is same with the high of this nodes that means; this is removal of your redundant node. So since they are same this particular node is going to get this particular label so it will be labeled with 2.

Now, when we come to this particular node, then we will find that low is coming to labeled 1 and high is coming to label 2 that means; low and high is not giving the same representation, we are having the deferent representation, deferent sub B D Ds, different function so it is bond to get a new label. Similarly, again behavior of these two nodes are not same, second third condition will also clear so it is going to get a new label and we are all going to labeled it by 3 so it is assigning a label 3.

Now when I am coming to these particular things, then low of this particular node is 2 and high of this particular node is your 3. So id of low is 2 and id of high is 3 that means; they are having different sub function so it is going to get a new label 4. So we are starting from this particular terminal nodes and we are following the bottom of approach and coming to this particular root nodes and we have label all this particular nodes. Now in this particular case, now next one is your merging of our duplicates nodes. So, if it is id of low of n is equal to id of low m and id of high n is equal to id of high m so which is the senior, then we have going to merged. So merging of duplicate nodes in this particular case, these three nodes will be merged together.

And secondly, we are having another one removal of redundant nodes so if id of low of p is same as id of high of p. So this is say you have just saying that, this is labeled with p and this is a label n and m so that is why I am saying that n and m so that can be merged. Hence similarly, it can be merged and now since for this particular node p low and high as same so you can remove these things. So basically these are the rules that we are going to apply so in this particular case, now we can merge these particular nodes which are having the same label and eventually, we are going to get this particular reduce BDD.

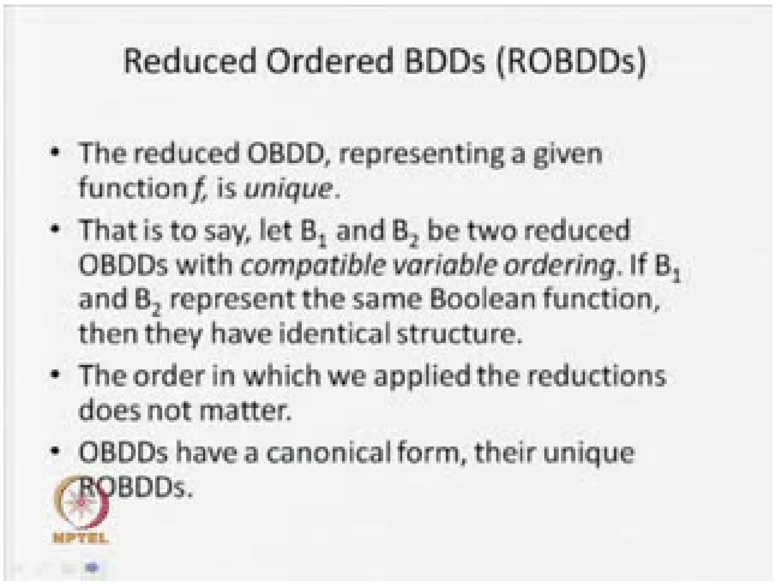
(Refer Slide Time: 39:26)



So this is the reduced ordered binary decision diagram of the BDD that we have studied. So now with the help of this reduce algorithm, we can use we can reduce the B D Ds and eventually we are going to get a reduced ordered binary decision diagram that means; we are having an algorithm called reduce where we are going to give an BDD B. So that means; input is your ordered BDD and what output we are going to get the output of this algorithm is your ROBDD, reduced ordered binary decision diagram and the variable ordering of the output BDD same with the variable ordering of that input BDD. So now you just see that, we can construct a BDD for given any Boolean function and after that after construction, we can use this particular reduced algorithm to get the reduce order binary decision diagram.

Now this is some properties you can see, now reduced ordered binary diagram, binary decision diagram so the reduced ordered binary decision diagram representing a given function f is unique. So if you are going to look a particular function f and we are going to construct the reduced ordered binary decision diagram of that particular function. Then this particular binary reduced ordered binary decision diagram will be unique with respect to that particular variable ordering because if you sense the variable ordering then size will very because already we have seen that means you we are going to get that deferent representation deferent BDD. So with a particular variable ordering we are going to get an unique representation.

(Refer Slide Time: 40:35)



Reduced Ordered BDDs (ROBDDs)

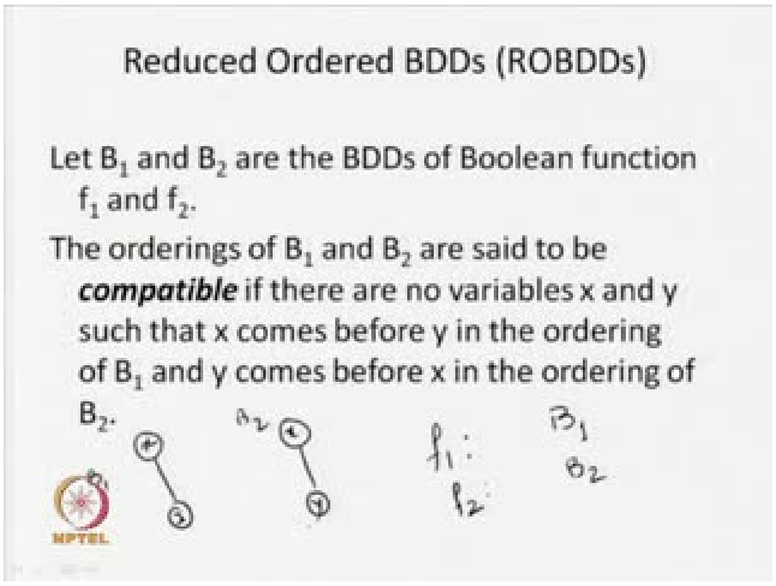
- The reduced OBDD, representing a given function *f*, is *unique*.
- That is to say, let B_1 and B_2 be two reduced OBDDs with *compatible variable ordering*. If B_1 and B_2 represent the same Boolean function, then they have identical structure.
- The order in which we applied the reductions does not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

Secondly, we are having a notion of, notion of compatible variable ordering. So if you conceder two BDDs B 1 and B 2, if they are having a same variable ordering then we say that, they are having the compatible variable ordering. And and this two BDDs say B 1 and B 2, if they are having the compatible variable ordering and if they are representing the same Boolean function, then their structure is same. You just see that we are going to get a unique representation with a particular variable ordering so if two BDDs are having compatible variable ordering and secondly, if they are representing in a same Boolean function then these two BDDs will be identical.

So again that is why your are saying that for a particular variable ordering we are going to get an unique BDD, ROBDD representation. So that is why we are saying that ROBDD representation of any function is going to give us the canonical representation of that particular function and secondly, already you have seen that how to apply this particular reduce algorithm, we are going to get the reduced BDD.

On the other hand, you can apply this particular reduction rule also and the order in which you are applying this particular reduction rule is immaterial, we are going to, always going to get the same reduced ordered binary decision diagram. So this is the unique property, that we are having, that we are going to get an unique representation of a Boolean function with respect to a particular variable ordering; that is why we say that ROBDD gives as the canonical representation of Boolean function so this is the main picture of ROBDD and you can use this particular BDD.

(Refer Slide Time: 43:13)



Reduced Ordered BDDs (ROBDDs)

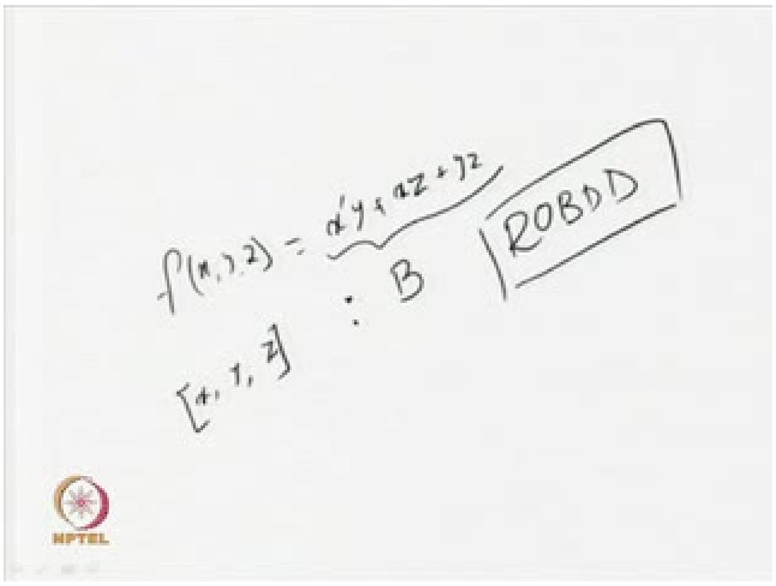
Let B_1 and B_2 are the BDDs of Boolean function f_1 and f_2.

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.

Now similarly, reduced ordered binary decision diagram we are saying that, if we are having two B D Ds B 1 and B 2 and this B 1and B 2 are representing say two Boolean function f 1 and f 2, say we are having one Boolean function f 1 and we are saying that this is the BDD representation B 1 and we are having another Boolean function f 2 and BDD representation B 2.

Now ordering of B 1 and B 2 are said to be compatible, if there are low variable x and y such that; x comes before y in the ordering of B 1and y comes before x in the ordering of B 2. So we do not have such type of scenario, then we can say that this is the compatible variable ordering that means; in B 1 if say x is coming before y, then in B 2 also x must come before y. So if y comes before x in B 2, then we are not going to say that these two are having the compatible variable ordering. So now you can look for any variable ordering and we can look for two BDDs, we will say that these two BDDs will have compatible variable ordering, if they are having the same variable ordering.

(Refer Slide Time: 44:38)

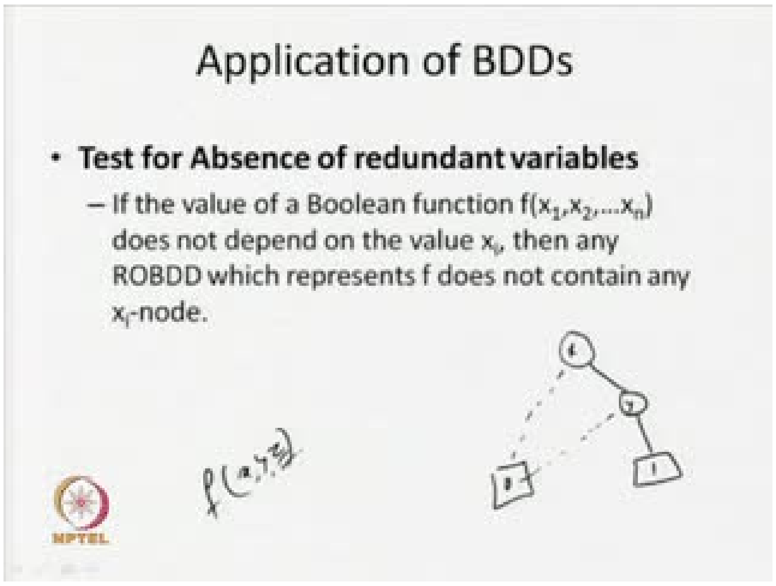


Now, we can have any Boolean function say, if I am having a function x, y, z say some random function I am going to say that, x bar y plus x z plus y z. Now if we are having such type of function, that I can always keep some chose some variable ordering say I am going to chose a variable ordering x, y, z and I can represent this particular function with the help of BDD B and if we apply the reduction rule or we apply the reduced algorithm to this particular B, we are going to get a ROBDD.

So once we are getting this ROBDD, then what happens? It is basically nothing but we are representing this particular Boolean function, we representing this particular Boolean function with the help of ROBDDs. Now once we have this particular ROBDD representation of this Boolean function, now what we can do? Already we have mention that, most of the cases we are going to get the compact representation of our Boolean function and after that we can do some operation, we can do some manipulation on this particular BDDs also so for any function we are going to have the BDD representation. Now this BDD representation is going to help us to take some decision very quickly so what are those particular decisions?

So first one is I am talking about test for absence of redundant variables, say if say I am giving one particular function, can you say that whether this function is redundant of a this is independent of some variables. Say I am giving this particular function x bar y plus x z plus y z can you see tell me what are independent of some variables? This is small function, we can see it and you can say that it may be independent it may not be independent but if we are having a BDD representation then by looking into a structure of BDD, easily we can say that whether it is independent of some variable or not.

(Refer Slide Time: 45:37)



Application of BDDs

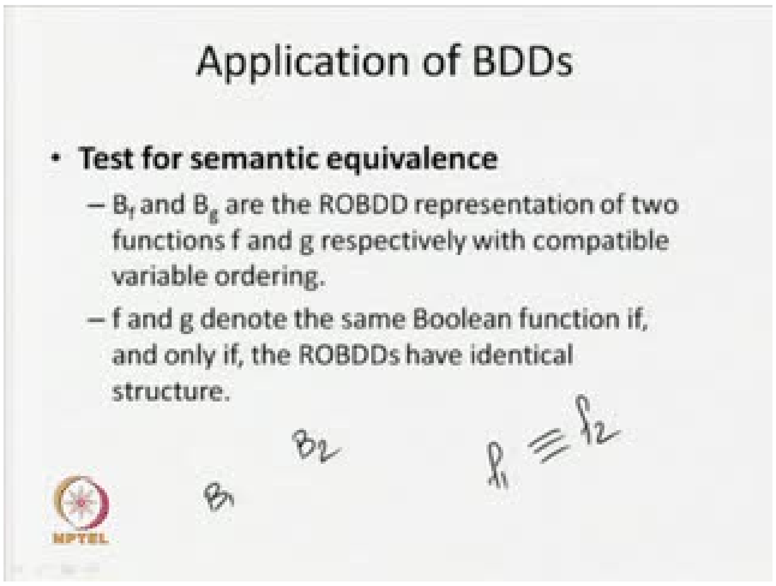
- **Test for Absence of redundant variables**
  - If the value of a Boolean function $f(x_1,x_2,...x_n)$ does not depend on the value x_i, then any ROBDD which represents f does not contain any x_i-node.

f(x,y,z)
x
y
0
1
NPTEL

So how we can say that? If the value of a Boolean function say f x 1 to x n does not depends on a particular variable say x i, then any ROBDD which represent f does not contain any x i node. So if I am having a BDD representation of a function say x is coming something like that, say this is y, say this is 0, 0 and say this is 1 so this is a function f x, y, z. If I look into the BDD, in this particular BDD we have found that this particular B D Ds not having this particular variable z. So we can very well say that this function is independent of this particular variable z.

So test for absence of redundant variable is easily check by looking in the structure of this particular ROBDD because ROBDD is going to give us a unique representation with respect to particular variable ordering, so you are not going to get any other representation for that particular variable ordering. So if one particular variable is not appearing in this particular structure; very well you can say that, it is independent this particular variable.

(Refer Slide Time: 47:53)



Application of BDDs

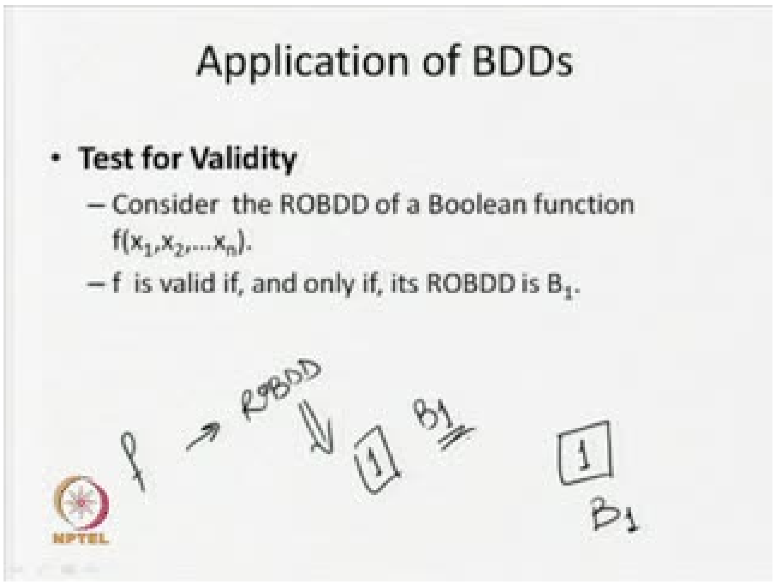
- **Test for semantic equivalence**
  - B_f and B_g are the ROBDD representation of two functions f and g respectively with compatible variable ordering.
  - f and g denote the same Boolean function if, and only if, the ROBDDs have identical structure.

B_2 B_1 $f_1 \equiv f_2$

Similarly, we can say that test for semantic equivalence because with respect to particular variable ordering; it is going to give me a unique BDD representation. Now if I am giving two function, say f 1 and f 2 and I am asking whether f 1 is equivalent to f 2 or not, whether they are representing the same Boolean function or not. Then what will happen? We are going to do some Boolean manipulation on this particular f 1 and we try to say whether we can get the f 2 or not. Now, if we construct now BDD B 1 for f 1 and BDD B 2 for f 2, with a particular variable ordering that means; they should have a compatible variable ordering.

Now once we construct this particular B D Ds for function f 1 and f 2; and eventually if we find that, both are having the identically structure. Then we can say that both f 1 and f 2 are representing the same Boolean function that means; f 1 and f 2 equivalent. So the semantic equivalence can be check very easily once we construct or once have the BDD for these two functions, if they are having the identical BDD representation with respect to a compatible variable ordering, then we can say that these two functions are equivalent.

(Refer Slide Time: 49:04)



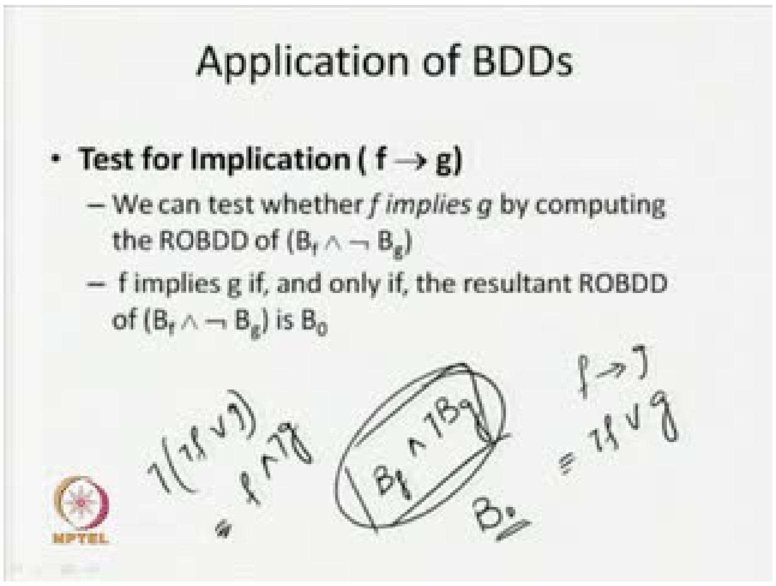
Application of BDDs

- **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.

f
ROBDD
B1
1
1
B1

Similarly, we can say that, test for validity we know when we say that, a particular variable Boolean function is valid or not. For all valuation of this particular variable it is evaluate to 1, then we can say that this is a valid function. So in this particular case, now if we construct a BDD and if your BDD is your B 1, then we can always say that it is a valid function. So, basically we can say that BDD B 1 if it is on the terminal node, then we are going to say this is your BDD B 1. So if I am giving any function f, now construct the ROBDD and with respect to particular variable ordering and if this ROBDDs happen to be your the BDD B 1, then we can say that this is a valid function. So you just see that, if I am giving a long expression to check whether it is a valid or not it is a the (( )) but if I am having a BDD representation and if BDD representation is B 1, then we can very well say that this is a valid function.

(Refer Slide Time: 50:12)



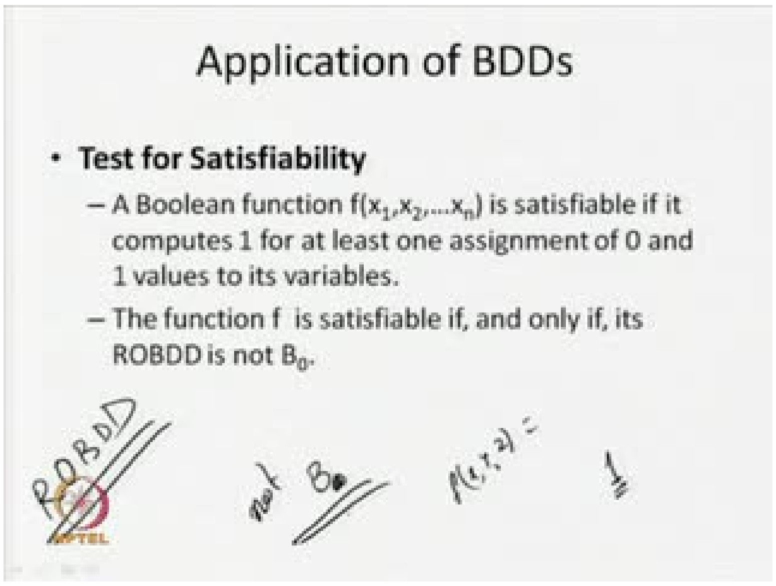
Application of BDDs

- **Test for Implication (f → 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

$\neg(\neg f \vee g)$
$= f \wedge \neg g$
$\boxed{B_f \wedge \neg B_g}$
B_0
$f \to g$
$\equiv \neg f \vee g$

Similarly, test for implication, whether f implies g or not so f implies g so this is something like that; f implies g is equivalent or knot of f or g. So in this particular case what I am going to do? I am going to have a take the negation of this thing so negation of f implies g. So if I take the negation that means what happens? I am going to get the knot of f or g applying De Morgan's law, what I am going to get? This is your f and g that means; if I am going to construct the BDD b f and knot of sorry, this is knot of this knot of B g and if this particular BDD is your B 0 because I am taking the negation you just see. So that that means the negation is an contradiction that means; your negation is a contradiction that possibility gives a satisfaction. So in that particular case, if this is equal to your B 0, that you can say that, f implies g. So these is also but how to do this operation we are going to check may be discuss this things in our next class.

(Refer Slide Time: 51:17)



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.

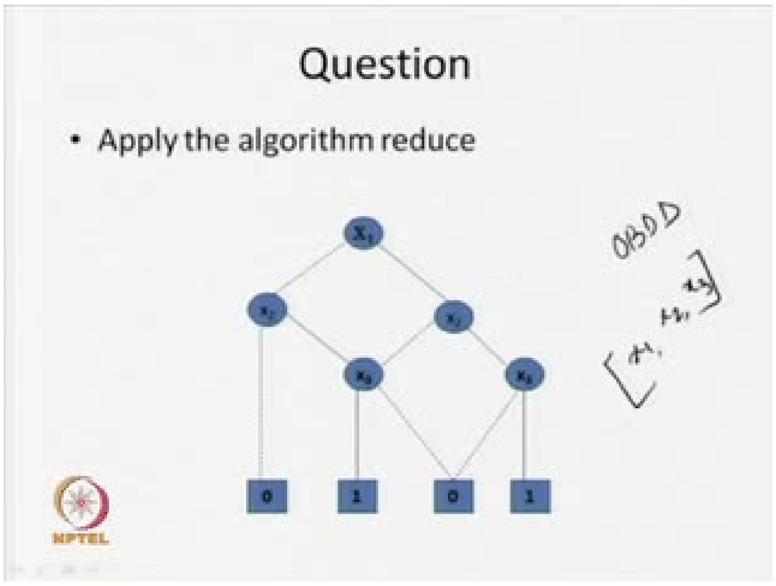
ROBDD
not B0
f(x,y,z) =
1

Similarly, we can say that test for satisfiability, when we say that a function is satisfiable if for at least one valuation or even one evaluation it gives the value as 1, then we can say that it is satisfiable that means; if I am having a function say f x, y, z then at least for one valuation it should be made of functional value as 1. Then in the particular case, we are going to say that this function is satisfiable. Now how to check it? Now we have to look for all will be valuation or if we having a truth able representation and we are having ending one entry as 1, then we can say that it is satisfiable.

So if I am having a BDD representation of this particular function and if this BDD representation is not equal to your B 0, if it is not B 0, then we can say that this is a satisfiable function that means; if it is B 0 for all valuation it will go to B 0 that means, for all valuation we are going to get 0. So if the resultant BDD is not equal to B 0, I can say that it is your satisfiable because for some valuation it is going to get the terminal node 1.

So by looking into the structure, looking into the resultant ROBDD because ROBDD is your a unique representation of the function and which is the canonical representation of the function. If the ROBDD is not equal to your B 0, then you can say that the function is satisfiable you just see that, now if we are having a Boolean function and if we can represent this Boolean function with the help of ROBDD with respect to a particular variable again, then some decision can be taken very quickly like; whether it is valid or not? Whether it is your satisfiable or not? Whether this function is independent of some variables or not? Whether two functions are equivalent or not? So such type of decision can be taken very quickly. So, these are the advantage of using ROBDD.

(Refer Slide Time: 53:12)



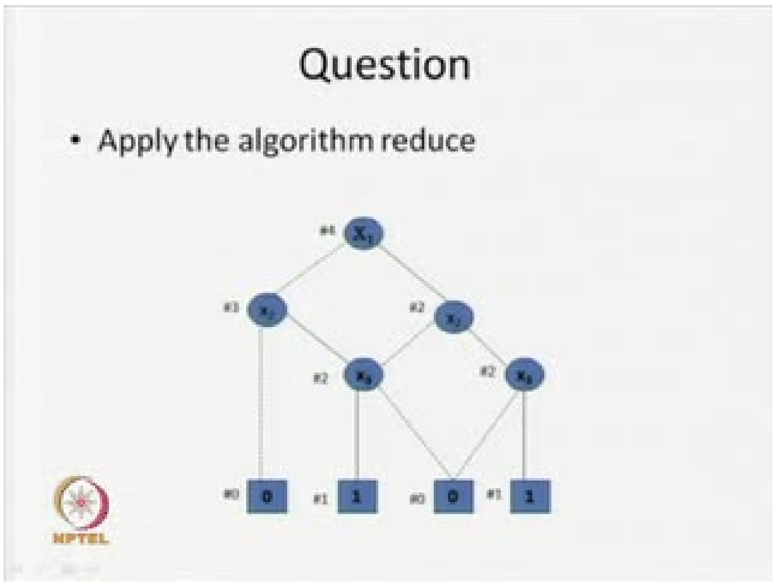
Question

- Apply the algorithm reduce

Now as a one question, I am saying that apply the algorithm reduce to reduce this particular BDD. So I am giving in your OBDD, ordered binary decision diagram and you apply the algorithm reduce to it. So it is having three variable x 1 and x 2, x 3 and the ordering is your x 1, x 2, x 3 so this is the ordering that we are having and this is the given BDD. Now apply the algorithm reduce to reduce this particular B D D. So in this particular case what will happen? First I am going to label these particular nodes.

So, in this particular case what will happen? All 0-node will be labeled with 0 and all 1-node will be labeled with 1 now, where labeled is particular terminal nodes, then we go up. Now in this particular case, when I come to this particular label x 3 so it is 1-node is coming to 1 and 0-node is coming to 0 that means; your sub BDD is not same so we are going to have a order is s 2. And similarly, this is also in x 3,0 is going to 0 and 1 is going to 1. So, these are not same with 0 and 1and secondly these two are having the same label x 2 same variable x 3 and x 3. So sub BDDs are 0 for both the 0, it is going to one terminals for both 1, it is going to zero terminals. So, it will also going to get, going to get the same label with this particular x 3 so both are label with your x 2.

(Refer Slide Time: 53:54)



Question

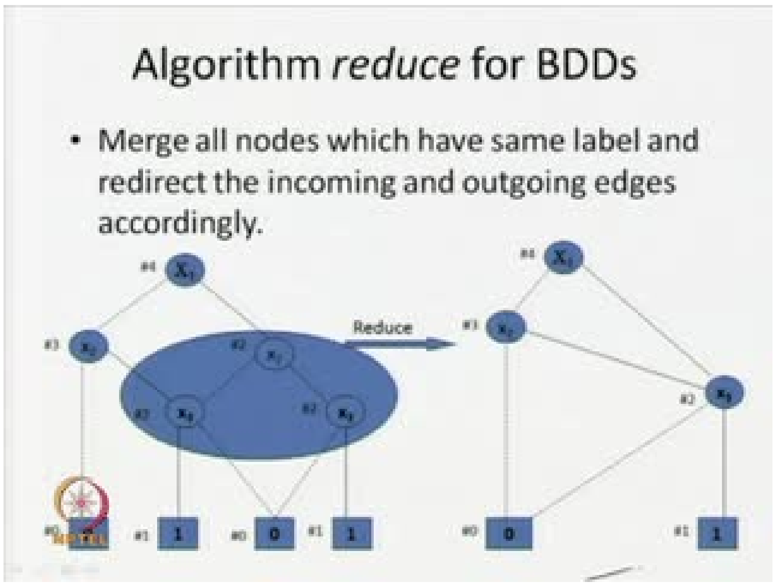
- Apply the algorithm reduce

Similarly, if I am coming to this particular node then if 0 is going to label 0 and 1 is going to label to that means; their sub BDDs are not same so it is going to get a define label so I am going to label it which tree. When I come to this particular label with 0, we are going to this particular label 2 with 1 also we are coming to this particular label 2. So in this particular case, it is going to get the same label with these two nodes, it is going to get x 2.

Now when I come to this particular node, 0 is coming to your tree and 1 is coming to your label 2 so these are different. So in that particular it is going to get a new label and we are going to say that, going to give a label 4. Now this is, now we are starting from the terminal nodes going in a bottom operation and coming to the terminal nodes, this is root nodes and all labeling is over. So in that particular case, now next page is your merging.

So, these three nodes are having the same label so we can merge together and ultimately we are going to get this particular reduced BDD. So merge all the nodes which have the same label so these tree are having the same label, we are going to reduce it to 1-nodes and accordingly; we are redirecting this particular in inputs and output edge. So, ultimately we are going to get this particular reduced B D D.

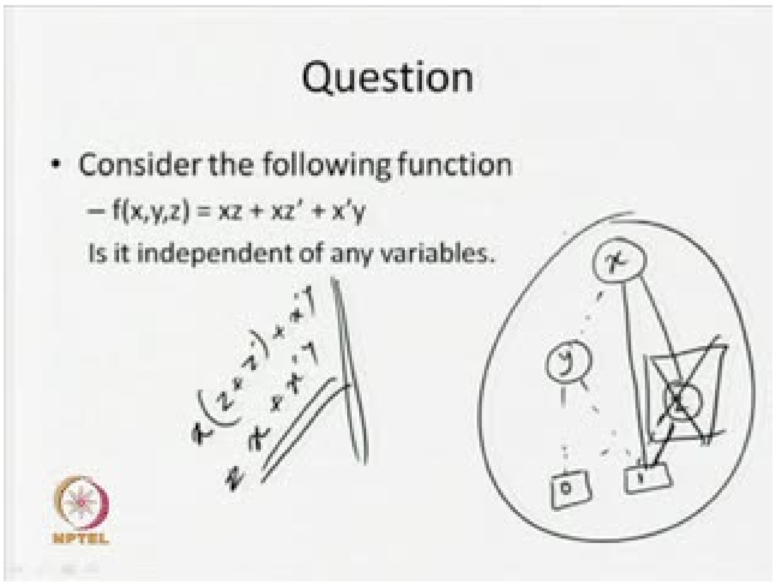
(Refer Slide Time: 55:50)



Algorithm *reduce* for BDDs

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

(Refer Slide Time: 56:10)



Question

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

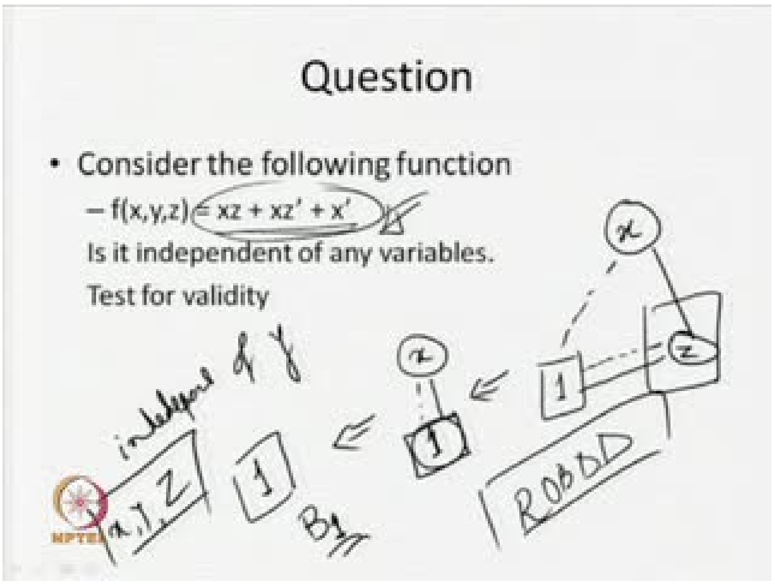
  Is it independent of any variables.

So another question you just see that, consider this particular Boolean function x z plus x z dash plus x dash y, is it independent of any variable? So this is a function x, y, z and it is having involved this particular x, y, z now you have to see whether it is independent of any variable or not. I think we can simply construct this particular BDD so; this is your x if I am having a ordering. So x equal to 0, I have to take decision on y, if x equal to 1, then I have to take decision on z. Now if it is y is equal to 0, then my it is 0 and if y is equal to 1, then I am having 1.

Now similarly, if it is x z then if z is equal to 1 and then what will happen? x equal to 1, z equal to 1. So in this particular case, my functional value is 1 and if x equal to 0 1 and z equal to 0, then one functional value is your 0 sorry, x equal to 1 and z equal to 0 sorry, this is not. So in both the cases, it will be one only because x equal to 1, z equal to 1; it is going to give me 1, x equal to 1, z equal to 0, it is going to be 1.

So in this particular case, I am having this particular redundant test so I can remove this one. So, in this particular case, eventually this is come to these particular points so this is the resultant BDD and in this particular resultant BDD, z is not appearing over here so we can say that, this particular function is independent of set. So this is small function I can, we can do the Boolean manipulation also what I am going to get from this function x z plus z bar plus x bar y that means; what I am going to get? x z plus z bar is equal to 1, so this is x plus x bar y which input (( )) also we can say that, this function is independent of the variable z but if I give a bigger expression, such type of manipulation may not be that simple. So if you have that BDD representation, from this BDD representation we can say that, this function is independent of variable z.

(Refer Slide Time: 58:46)



Question

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

  Is it independent of any variables.

  Test for validity

Similarly, you just look into this particular function very small function and simple function I am giving x z plus x z bar plus x bar, is it independent of any variables? Is it test for validity? Can you look for the validity? When I can say that it is valid, if the resultant validity is your 0s, just try to construct it again. Again I can say that I am going to take decision of your so this is a very simple it is from this equation itself, I can say that it is y is not there so it is independent of y. So I am going to take x x may take value as 0 and x may take value as 1. So when I am going to construct it, when x bar equal to 0, then my functional value is 1. Now when x equal to 1, then we have to take decision on this particular z. Now when z is 1, then x z is going to give me 1 and when z is 0, x 1 z 0 is going to give me 0 so in this particular case I am going to have this thing.

Now eventually what will happen you just see that, now this is a redundant test so I can remove it. So what I am going to get? This is x sorry, this is terminal so I am removing it so 1 is come over here. Now see that x is again this redundant test x is equal to 0 is 1, x equal to 1 so what happen? From here I can say that, I am going to get this particular BDD 1. So what are the BDD I am getting? This is the BDD B 1 that means; for all valuation the function is going to give me 1.

So since my resultant BDD is B 1 so what I can say that, this is a valid function. Secondly, by looking into this particular function, I have said this is independent of y. Now after looking into this particular BDD, what I can say that? It is independent of x, y and z that means; this function is independent of this all these particular three variables. So just see that, by looking into this particular BDD, we can take the decision very easily. So this is a small function I am giving but if I am going to give a bigger expression by inspecting this function, it would be difficult to say whether it is valid? Whether it independent of some variables? But, once we get ROBDD by looking into the structure of this particular ROBDD, we can take many more decision quickly with this I end of my lecture today.

Thank you.