INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

NPTEL

NPTEL ONLINE CERTIFICATION COURSE

An Initiative of MHRD

VLSI Design Verification & Test

Prof. Jatindra Kr. Deka

Department of CSE

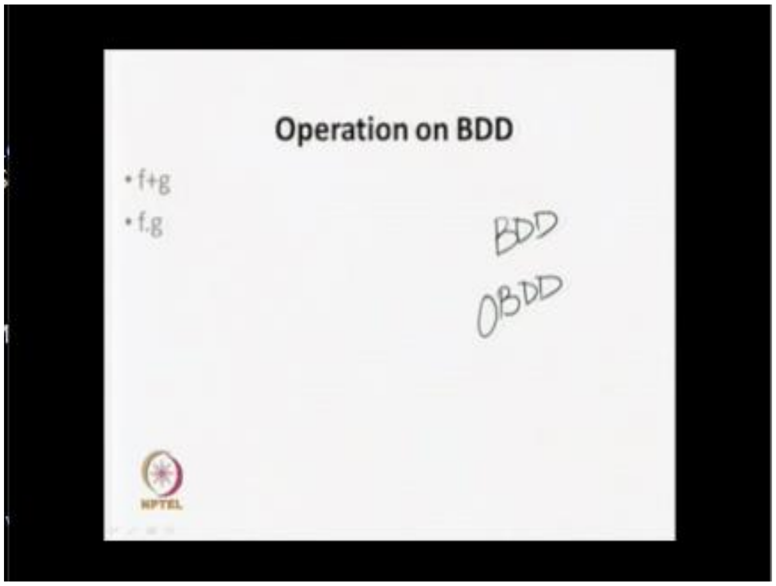
IIT Guwahati

Module VI: Binary Decision Diagram

Lecture III: Operation on Ordered Binary Decision Diagram

Okay we are discussing about a data structure called binary decision diagram and this find this diagram is used to represent any Boolean function okay. So today we are going to see some operation that can be performed on ordered binary decision diagram.

(Refer Slide Time: 01:01)



Operation on BDD

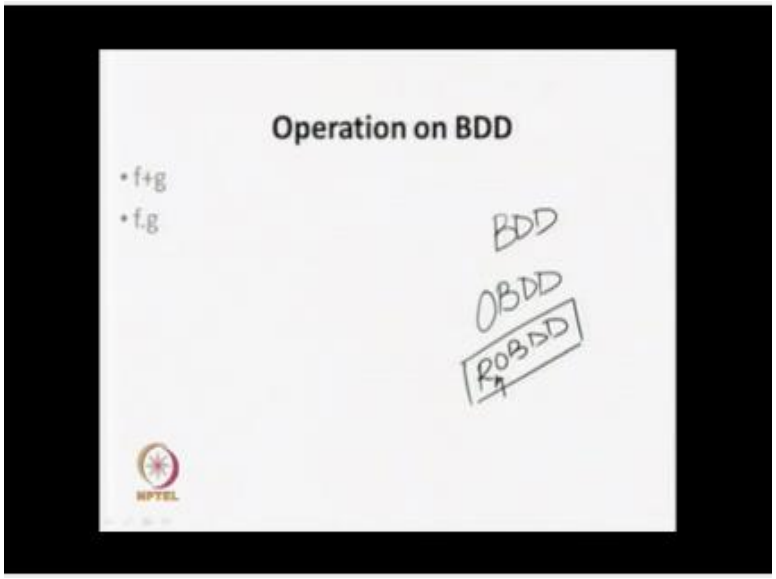
- f+g
- f.g

BDD

OBDD

Because, we already discussed about BDD we introduce what is your BDD Binary Decision Diagram then we have talked about OBDD Ordered Binary Decision Diagram. That Binary decision diagram we are going to maintain a particular ordering of the very at use, okay.

(Refer Slide Time: 01:34)



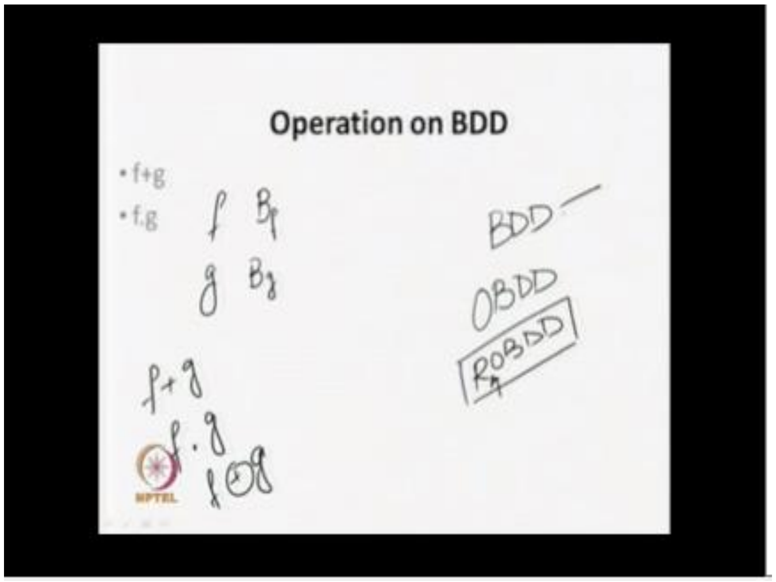
Operation on BDD

- f+g
- f.g

BDD
OBDD
ROBDD
NPTEL

And after that we have talked about ROBDD Reduced Ordered Binary Decision Diagram that means it is the reduced form ordered, it reduced BDD of a given particular Boolean function which follow a particular ordering of the very a force. And we have seen all property they have done ROBDD defined function is always unique and it is a canonical depression of a given Boolean function.

(Refer Slide Time: 02:15)



Operation on BDD

- f+g
- f.g

$f \quad B_f$

$g \quad B_g$

$f+g$

$f.g$

$f \oplus g$

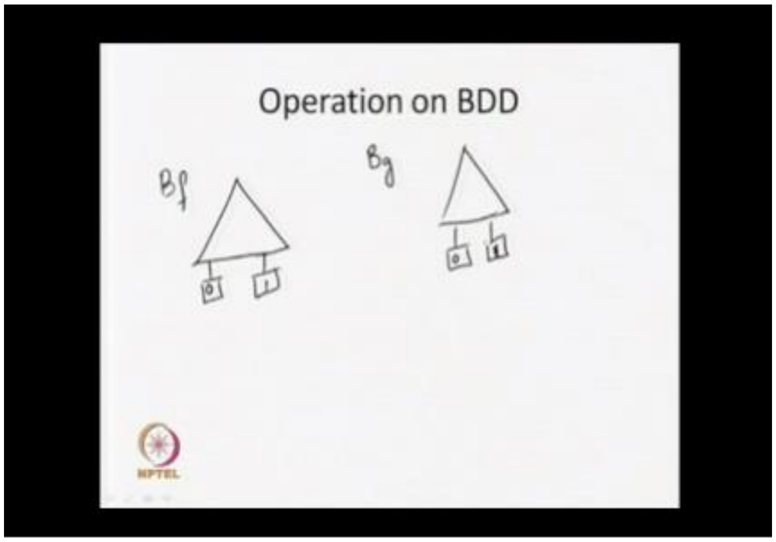
BDD

OBDD

$\boxed{\text{ROBDD}}$

So with a particular variable ordering we are going to get a unique structure you need a BDD structure for that particular function, if it shares the BDD sorry if he sends the variable ordering that we may get another structure. Now so I am having a two function f and g then the BDD represents B_f for BDD function and for f is $B_{g,}$, okay. Now what happens on the operation f+g or perform the operation of f.g like that f+(XOR Operation)g.

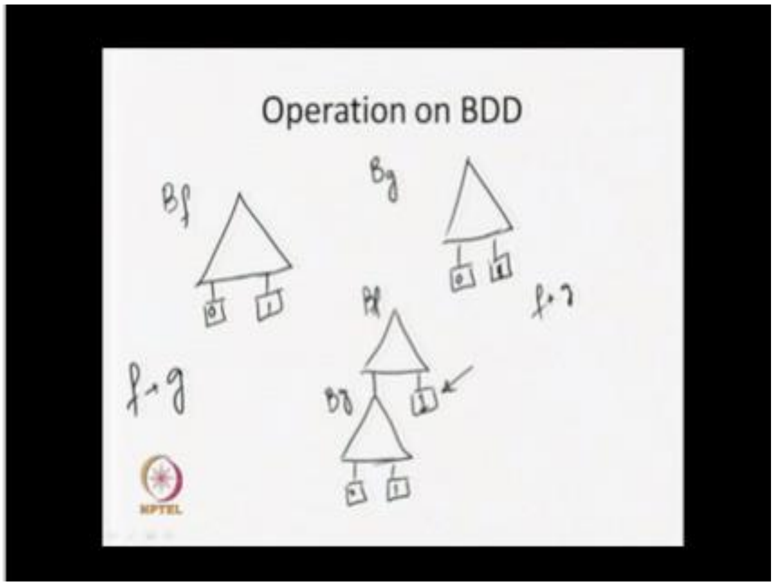
(Refer Slide Time: 02:53)



Operation on BDD

Well we have talked about BDD Binary decision Diagram that we have and that them we have send the problem those who have problem on BDD also. So what do we have seen in the particular case, you have seen that if I give a function BDD of f and BDD presentation are of b_f then b_f I can say this is a structure and eventually you are having the terminal node 0 and 1. Similarly I am having a node B_g of a function g and I can say that I am having a BDD representation of this function and these two are the terminal nodes, okay.

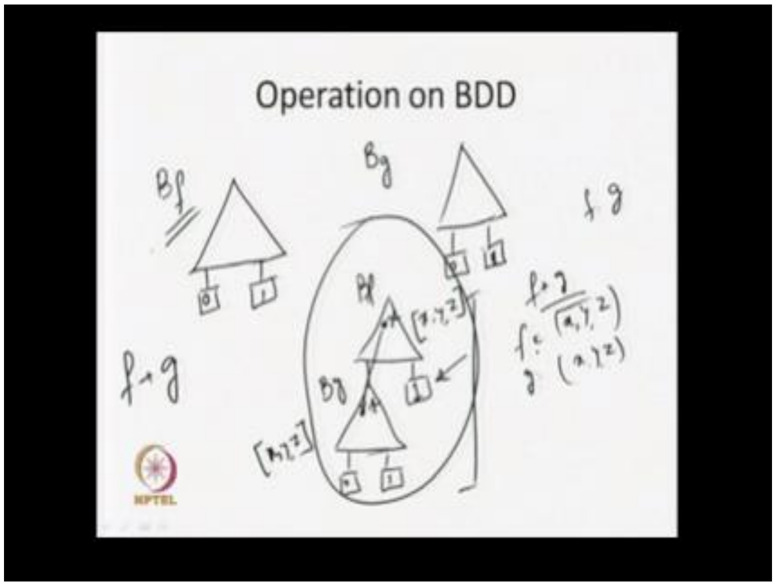
(Refer Slide Time: 03:57)



Well I am going to say f+g and what will happen we can pop on to the BDD what we can say that b_f so if you function f is 1 , then f+b=1. If f is 0 then what will happen will we have the loop for the illusion of your function g. So in that particular case what happens you have seen that in the particular 0 node we are going to put this particular BDD of b_g and 0 and 1. If f is1 then we are going to get functional value as 1 and if f is 0 and we are going to look for the what is the illusion of that it is going to depending on that I am going to get the functional loop of f+g.

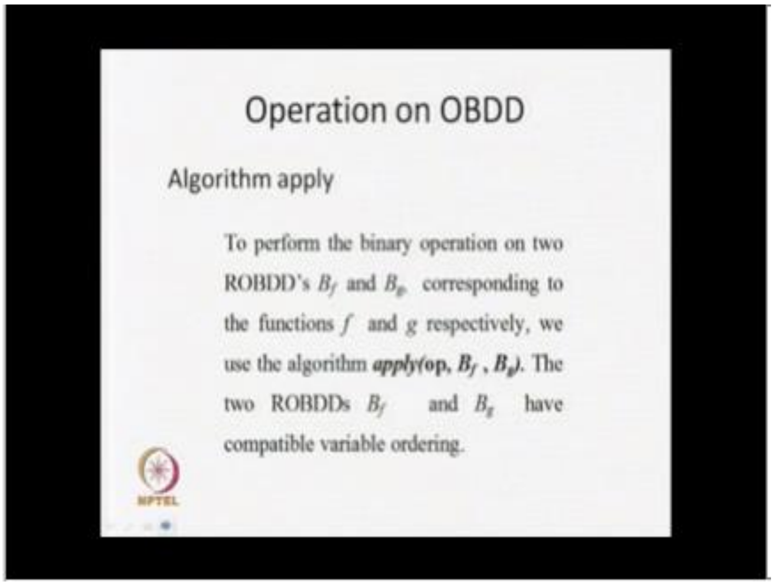
Similarly, we can go for f.g also ,but what is the problem we are going to get here user have it a function same body F-(x,y,z) similarly G-(x,y,z) which is having a variable x,y,z . The user said BDD f is an ordered variable of design diagram that is going to a variable ordering of x, y,z. Similarly that b_g is also a variable ordering set x,y,z , but when the constant at this particular BDD in that particular case what will happen we are not going to get this ordering of the variable x,y,z.

(Refer Slide Time: 05:12)



Somewhere here x is appearing and somewhere here also x is appearing at one particular part x is appearing twice, which is of your ordered binary decision diagram. So that means if you are simply going to construct a BDD for f+g by taking with the b_g and b_f. Then we are not going to get an ordered BDD that means the result in BDD is not an order. So what can simply we can do is cannot use this particular operation we use ordered BDD which have to handle it differently to test this lecture we are going to see how we can perform those particular Boolean operation on the BDD is are particular ordered binary decision diagram ,okay.

(Refer Slide Time: 05:40)



Operation on OBDD

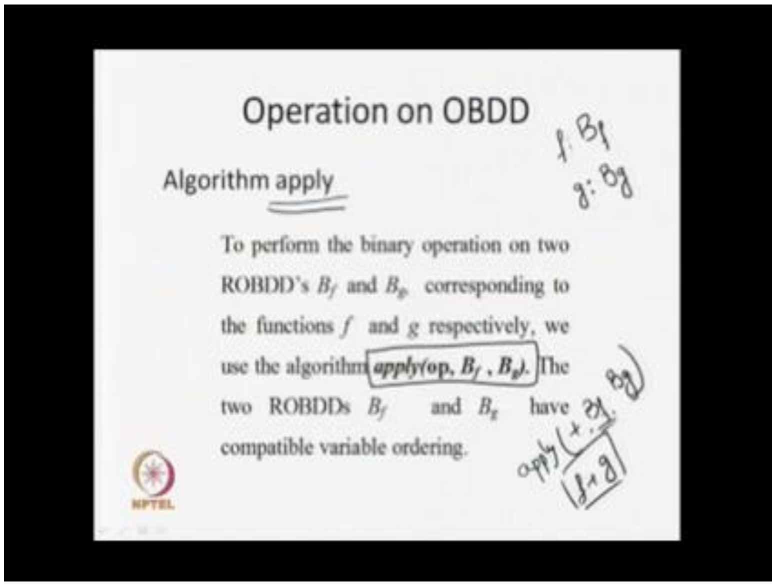
Algorithm apply

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.

So we just see that what we going to see in this particular case. Any kinds of operation we are going to use an algorithm called apply. This apply algorithm will be used to perform any Boolean operation on BDD. So when we apply these things on BDD we are going to get some result in between is going to give us the evaluation of the operation in this particular Boolean operator. So the apply algorithm is going to take f of B_f and g B_g.

Then the algorithm apply will take this particular form it is having three parameter apply operation Bf and Bg it can give any binary operator over here and to BDD. So Bf and Bg are BDD represents f and g if I use this particular function apply (Bf.Bg) in this particular case what is happen went to perform the all operators on Boolean function f and g . So f+g and after that we are going to represent BDD or ROBDD.

(Refer Slide Time: 07:07)



Operation on OBDD

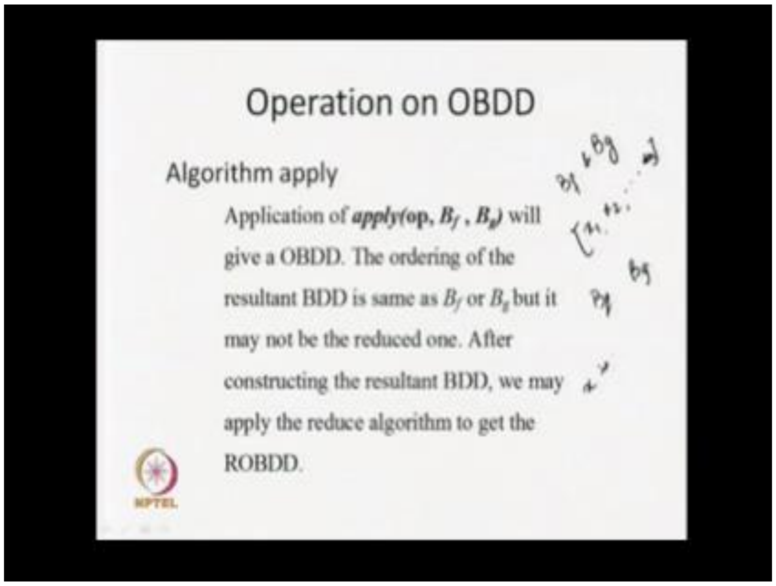
Algorithm apply

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 $\boxed{\textbf{apply(op, } B_f \textbf{, } B_g\textbf{).}}$ The two ROBDDs B_f and B_g have compatible variable ordering.

f : Bf
g : Bg
apply(+, Bf, Bg)
f+g

In this particular case, we are having the BDD represents a Boolean function f and this will represent of g. So we will use this particular apply algorithm perform this particular edition. Similarly use or we can use XOR operation like that in this particular operation and this apply algorithm is going to perform or give us the BDD which is going to represent the function f+g or f.g .

(Refer Slide Time: 08:32)


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.

Now while I am going to talk about a f, g or Bf in BDD it is an ordered binary decision diagram and these two binary decision diagram is compatible in ordering. So what does it mean in case of compatible variable algorithm both ordering x1, x2 like Xn the both Bf and bg suite following ordering, okay. So that is what is said mean what the compatible variable ordering say if I am having two variable ordering in Bf and Bg.

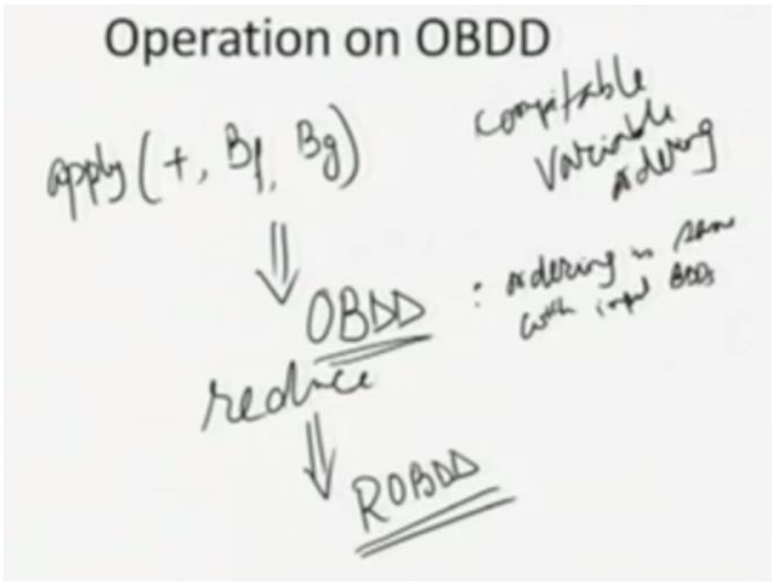
So in Bf if x is appearing before y in the ordering then depth should be satisfy in Bg also, x must appear before y in Bg also. For all variables if it satisfies this particular requirement we can send that this is the variable ordering for these two BDD’s. So one primary requirement for this apply algorithm is massed up compatible ordering. So one primary requirement for use of this apply algorithm is that both the BDD’s are variable ordering.

So when we use this particular operation we are going to give 2BDD and Bf and Bg and just said that ROBDD reduced binary decision diagram. After application of this particular operator we are going to get to perform the task Bf and Bg, if this particular apply algorithm will return it will give me an ordered OBDD which represent f+g. And the ordered of the variable is a ordering of Bf and Bg. So the result in BDD is also an ordered BDD. Then BDD is the ordering of the two inputs in BDD's.

But after application of this particular apply operator of whatever BDD we are getting in may not be reduced on, okay. So after apply algorithm what we can do we can use the reduced algorithm to get the reduced ROBDD reduced ordered binary decision diagram. So look what happens we are going to use apply algorithm + Bf and Bg compatible variable ordering the resultant of this variable will give an ordered binary decision diagram so in this particular case the ordering is sent with input BDD.

Whatever order you are having input between in BDD is going to give the same ordering we are going to get an BDD the same ordering but the ordered may not be a reduced one after ordering this and so. To this particular resultant between these we are going to use the reduce algorithm. In last class, I have discussed about this particular algorithm's after this previous particular algorithm whatever BDD we are getting it will be your ROBDD reduced ordered binary decision diagram.

(Refer Slide Time: 11:53)



Operation on OBDD

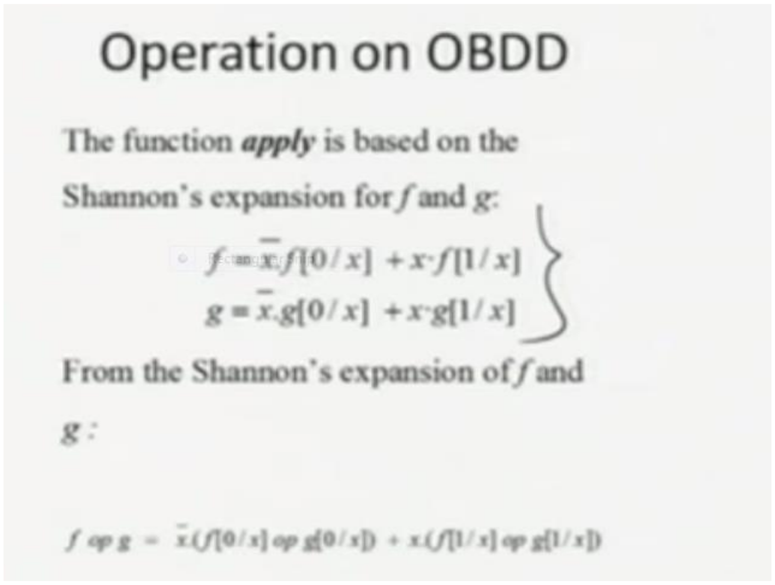
$$\begin{tikzcd} \text{apply}(+, B_f, B_g) \arrow[d, Rightarrow] \\ \text{OBDD} \arrow[d, Rightarrow, "\text{reduce}"'] \\ \text{ROBDD} \end{tikzcd}$$

compatible variable ordering
: ordering is same with input BDDs

If we are having two functions at f and g, we can depression these two functions with the help of to our abilities and basic requirement is that the ordering of this to be this must be some. That there should be compatible for variable ordering after application of this particular apply algorithm going to get a ordered BDD which represent the particular function f of g, okay. That is the resultant did we use to the reduced one so we are going to use the reduced algorithm to get the ROBDD of this particular output.

Put OBDD; now how are you going to you known how you are going to perform this particular apply operation, okay. So this is very well I can base on hours and on expansion on which I have already mention the construction of BDD depends on the expansion of a Boolean function.

(Refer Slide Time: 12:30)



Operation on OBDD

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

$$\left.\begin{aligned} f &= \bar{x}.f[0/x] + x \cdot f[1/x] \\ g &= \bar{x}.g[0/x] + x \cdot g[1/x] \end{aligned}\right\}$$

From the Shannon's expansion of *f* and *g* :

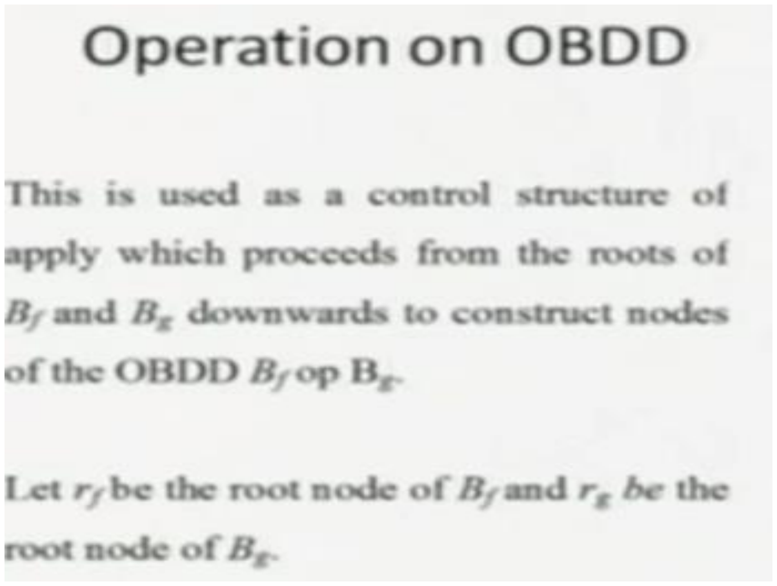
$$f \text{ op } g = \bar{x}.(f[0/x] \text{ op } g[0/x]) + x.(f[1/x] \text{ op } g[1/x])$$

The Shannon expansion it is representing with the help of this expansion of functions can be replaced f=xbar f[0/x]+x.f[1/x] and g=xbar g[0/x] +x.g[1/x]. We are going to evaluate g by replacing x=1 so these are the two Shannon expansion of our given function or we are going to perform any operation f of g this part we are going to perform half operation on the evaluation upon the representing x/0 and evaluation of function g is replacing by x/0.

So we perform this particular operation on this half function because already we have taken a decision of x=0 similarly for x=1 and I can going to look for this particular thing is the operation. While constructing a BDD we are doing the applied Shannon expansion to the regard expansion, So that sends the Shannon expansion will be used for performing the operation in this particular applied algorithm. So from root node we are going to perform and going down to the leaf node step by step

It is some sort of your top down approach we are going to start with the root node which out of given BDD’s traverse it down in top down fashion eventually we are going to get the root node, or terminal nodes or leaf nodes. Once you are coming particularly to this leaf node or terminal nodes then, what will happen in terminal nodes we are going to get terminal values so in this particular case we are saying that f of g when we are coming down to terminal nodes. Then in case of terminal nodes you are having the value either 0 or1.

(Refer Slide Time: 14:39)



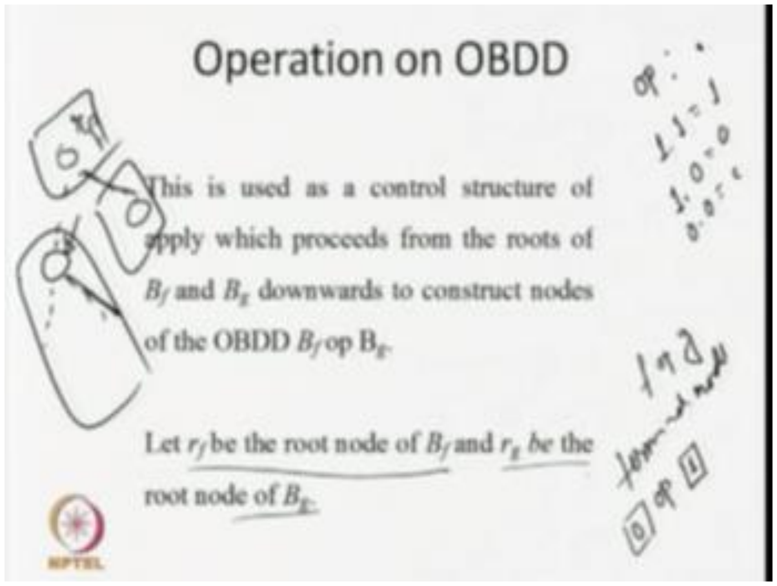
Operation on OBDD

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.

Now when I am coming to this particular family nodes that I can apply this particular operation because I know that my operation is yours a lot and I know that 1.1=1 and other combination 1.0=0 or may be 0=0 so like that we are going to construct a result and BDD and when we are going to get the terminal nodes that we can eventually applied operation on those particular terminal nodes.

(Refer Slide Time: 16:20)



Operation on OBDD

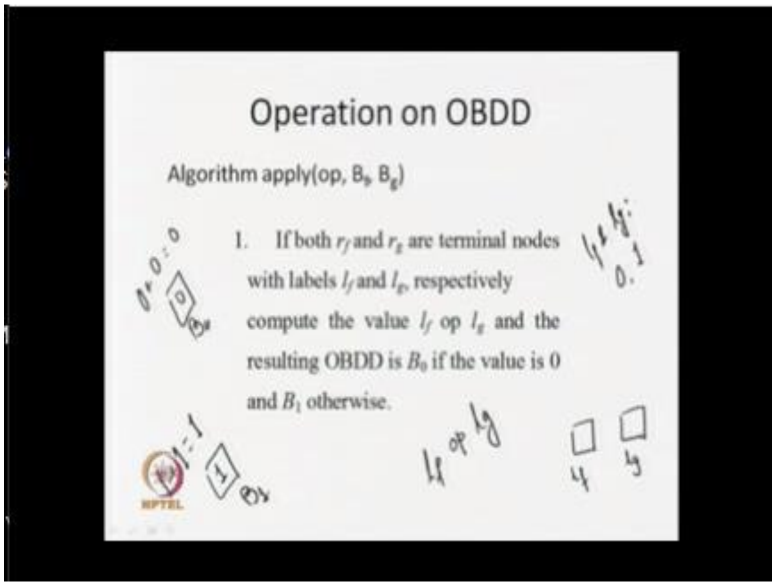
This is used as a control structure of apply which proceeds from the roots of B_f and B_g downwards to construct nodes of the OBDD B_f op B_g.

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

So we are going to do with we cannot simply let Rf be the root node of all or BDD bf and Rg be the root node of BDD Bg that one you can say this is my root node Rf and we are having that BDD something like that, okay. Now I can apply any levels once I take the decision of your look for this particular loop called to this particular to sub details. That is sure when I am coming then I will say that this is this is my root node Rf. So that is why in general I am saying that let rf be the root node of Bf and Rg be the root node of Bg.

Now, what your first step apply algorithm Bf and Bg keep both Rf and rg are terminal nodes which level it with Lf and Lg, okay. So now first condition is using it when we are coming to the terminal nodes this is your two terminal nodes two levels this is your Lf and Lg. Lf and Lg these two can have 0 and 1 because these are the terminal nodes and because and these are the fellow.

(Refer Slide Time: 17:41)


Operation on OBDD

Algorithm apply(op, B_f, B_g)

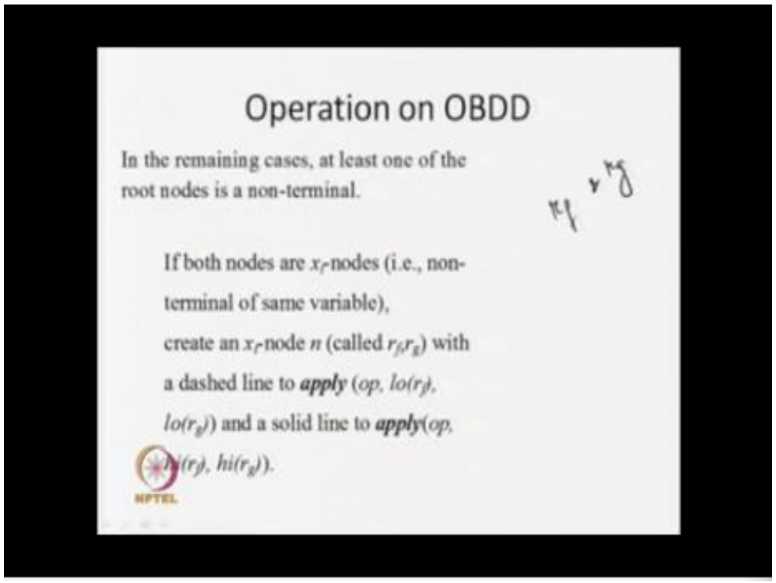
1. If both r_f and r_g are terminal nodes with labels l_f and l_g, respectively compute the value l_f op l_g and the resulting OBDD is B_0 if the value is 0 and B_1 otherwise.

0+0=0
B_0
1+1=1
B_1
l_f + l_g
0, 1
l_f op l_g
l_f
l_g

Them compute the Lf of Lg that means in this particular when you are coming to distance and going to perform the Lf operator and Lg and the resultant between the b 0, if the result is zero or it is b 1, if the result is your one that mean I can say that 1+1=1 so in that particular case I am going to get the BDD 1 which is your B1. Similarly if it is 0+0=0 then I am going to get the resultant of 0 which is the BDD .While we both the nodes are the terminal nodes may I the well vision of this function.

So we can apply this on the terminal nodes and the resultant will be B 0 of B1. This is the first case we are going to consider both the terminal nodes okay. Now if it is a not terminal node then what will happen at least one of the nodes will be non terminal nodes, and I am going to talk about the Rf and Rg if both are not terminal nodes then atleast one of them will be a non terminal nodes.

(Refer Slide Time: 18:14)



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, $hi(r_f)$, $hi(r_g)$).

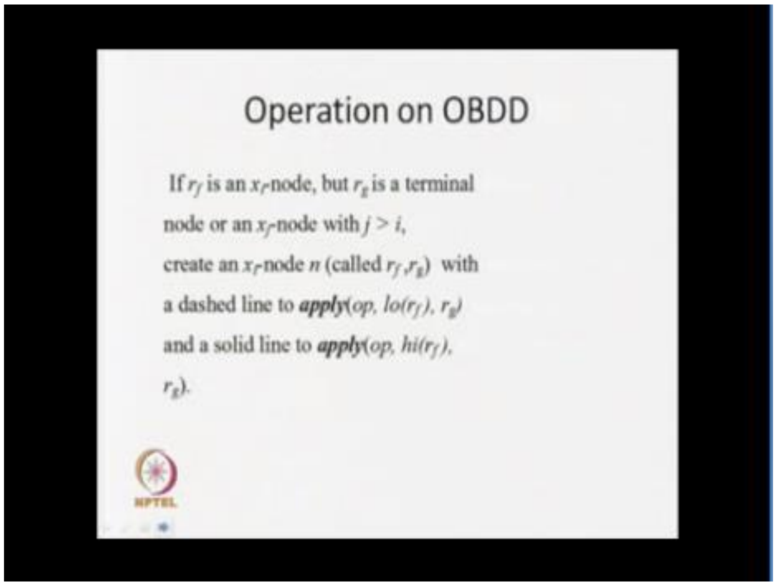
Now first cases we are going to consider the both nodes are xi nodes, both are xi nodes means both is are your both nodes are your terminal nodes and they are labeled with their variables xi okay. Now we are going to create a node xi that mean, since now both the function are depends on this particular variable xi. You are going to create a node xi and we level this particular node or we are going to call this particular node Rf and Rg because in my given BDD's one node is Rf and second node is Rg.

So we are going to level a node called Rf and Rg, and since both this particular nodes are labeled by xi we are going to construct a xi node. And after that when I am this xi node this xi node is to one is this and one is your solid. Xi=0 or xi=1. In this particular case we are going to apply this particular operation for your this line of lf, lo and lo (rg) apply (op,hi(rf),hi(rg)). Basically this is my rf node and I have noted two function already we have defined.

One is your lo(rf) and second one is your hi(rf) what you have written is basically rf node that is pointed by the dest line in case of lo say if it is said as my say pq thatlo (rf) written p and hi(rf)will be written as q, that means now when I am going to perform the operation on this particular rf and rg makes the but looking into the relation we have to take this is on this two nodes p and q.

So we are applying this things of lo(rf) and lo(rg) and in solid line applied of (hi(rg),hi(rg)), so whatever we are having is what is this dest line and solid line what is the decision we have to look on this particular details. So that is why we are saying that you are going to apply the operations of lo (rf), lo(rg) and incase of hi9rf) and hi(rg). So if both are leveled with this then we are going to follow this particular node.

(Refer Slide Time: 21:15)



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).

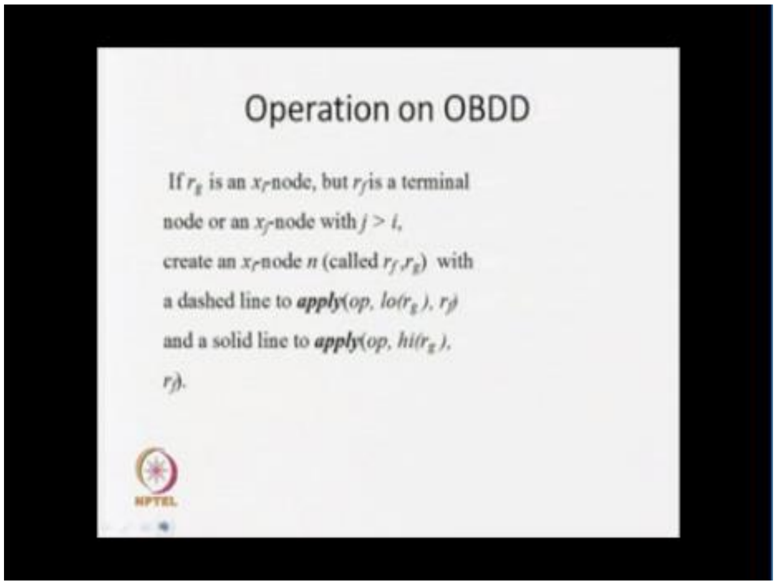
On the other hand in the second case, what will happen we may have one non terminal node or one terminal nodes. So this particular node rf is being an xi node and rg is an xa node where j>I , so in were we are traversing it from top to bottom when we come to this particular scenario one is xi and one is xa, j>I physically it says that this particularly BDD bg independent variable on the computation part because traversing it from top to bottom and uniformly you are going down.

In one part we are getting xi and second part we are not getting xi we are getting xl, so what does it means that means your bg is independent of xi in that particular evaluation part. So for that what will happen now that the resultant BDD may now be depended on xi because one of the functions is depended on xi so we are going to create a xi node at that particular point and similarly it will be labeled by rf and rg because these are my root nodes.

And similarly we are going to construct this particular two nodes then what will be the position you just see that we are going to call apply this particular op(lo(rf),(rg)) because erg is independent of, of your xi. So that means I have to again look for down I am going to get xi in that time I will see what will be the situation.

So we are going to apply rf and rg, because rg is your independent of your I and in case of your solid line we are going to apply (op,hi(rf)) so this is the way I am going to construct my resultant BDD. So here rg of an xi node but rg is a terminal node or it is an xl node, here I am talking about an xi node j>I or it may happen that rg may be your terminal node also. So we will see in that case what will happen, so this is the second scenario.

(Refer Slide Time: 24:08)



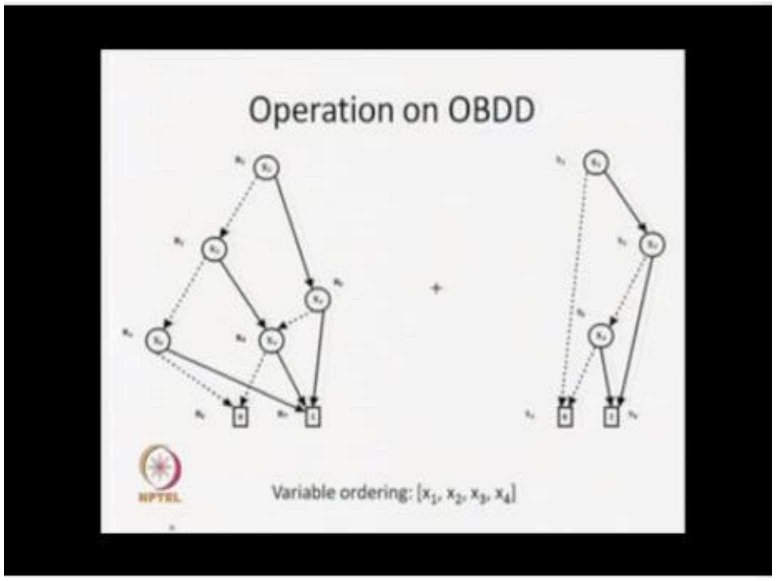
Operation on OBDD

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).

And third generation is that I am using if rg is the xi node but rg is a terminal node on xn node of j>1. So this basically this is symmetric to this particular condition only but now situation is different what you are saying that rf it is a xn nod eor terminal node. And rf sorry rg is an xi node and here it says that j >I. so this is similar situation like the previous one but in that particular case we are saying that rf is an xi node but now we are saying that rg is an xi node.

So this is the similar situation, so here in the similar way we are going to treat it. So In op(lo(rg),(rf)) and apply op(hi(rg)(rf)) this is symmetric so that is why we are taking in the previous case we are taking lo(rf)(rg) now we are going to take lo(rg)(rf). So these are the rules we are having over here.

(Refer Slide Time: 25:12)

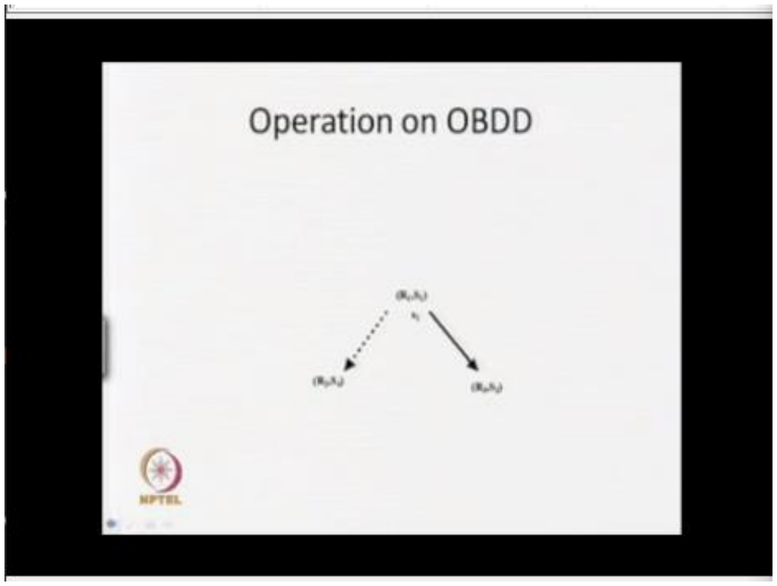


Now we are just going to see that now we are going to take two ROBDD’s so we are going to traverse it from top and it will be a top down approach, if both are terminal node then we are going to apply the operation of and it will be 1(1) or 0(0) or 1(0) or 0(1). In all other cases they will be your non terminal nodes once it is appearing in both the terminal nodes are having the same label, we seen what we are going to do.

In the second case one is a non terminal nodes second may be a terminal node or another non terminal node but they are labeled is not set, so you have seen how it is going to or how you are going to handle this. So this apply algorithm we are going to now we are going to with the help of an example, so we just see that it is one BDD of function Bf and say this is the BDD of function g we can say that Bg okay. Both are having the same variable x1…x4 okay.

And that variable ordering is x1, x2, x3, and x4 by looking into this construct of this two BDD we can see that they are having the compatible variable ordering they are following the function of ordering. Now I am going to perform the adding +g okay. This is that I am saying that this BDD +that BDD is going to give me a plus g. Now what we are going to do to start from the top I consider these two are my root nodes this is your rf and rg.

(Refer Slide Time: 27:24)



Operation on OBDD

Now you are having the r1,r2,r3,r4,r5,r6,r7, s1,s2,s3,s4,s5, okay. Now see what we are going to construct what we are going to take this particular root node okay, Now I am going to take and we see that both are having the same labeled x1 and x1. So we are going to create an x1 now, so lo(r1) is going to r1,r2 and lo(x1) going to as for. Similarly hi (r1) is going to x2 and hi(s1) is going to s2 that is why we are constructing this particular structure r1,s1 which labeled with your x1.

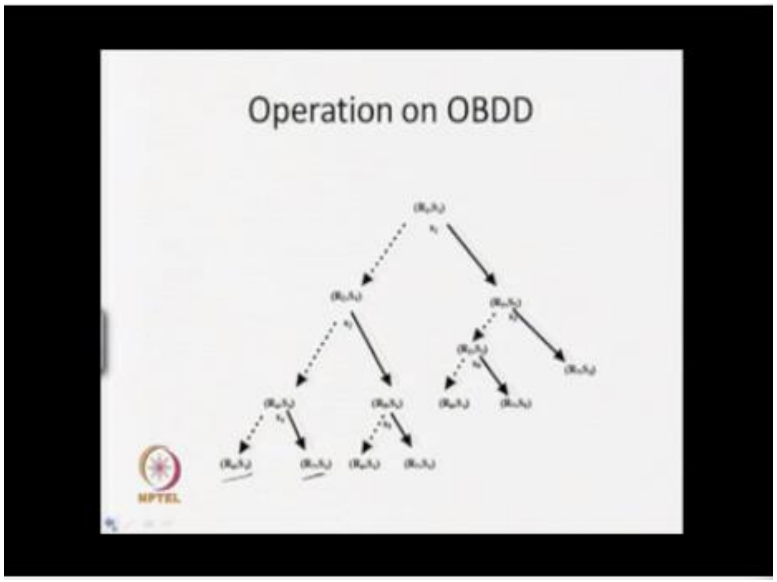
Now it is r1, x2, x4 and r3, s1 okay. So this is the scenario that we are having, so we are coming to this particular thing be x1. Now after that what will happen on that means I am going to look for or if I am going to follow this particular desk line this will be my root and this one will be my another root node okay. Now I am having this partial structure, in this particular case now 2 is a root node and your s4 is a node, so one is your non terminal node second one is your terminal node.

So in this particular case for what we are going to do will for desk line will follow this lo(op) this one and it will remain as it is or your hi1 will follow the structure and it will remain as it is. So in this particular case look in which we are going to apply this particular rule already have discussed that lo (rf)(rg) hi(rf)(rg). So we are going to bring to get this structure form from r2, s4 we are getting r4 as 4 and r5 as 4 okay.

Similarly, if we look into the other side for1 so this is my, root node and this is my root node. So we are going to construct an s2, r3, so this is your s2, r3 and these are the rest line when it is coming to r5 so both on non terminal nodes and we are going to get an x3 node over here because we just see that, is your x2 and this is your x3 this is also x3 and so I am going to get an x3 node over here.

And we are getting this line to solid line. Now r4, s4 and r3, s4 and r5, s3 and r2, s6 in this particular case now, by considering these as root node this one will be one root node and this one will be another root node. So we are going to construct this line solid line. Similarly here we are getting this thing so; we are going to have this scenario. Now we are coming up to this particular labeled.

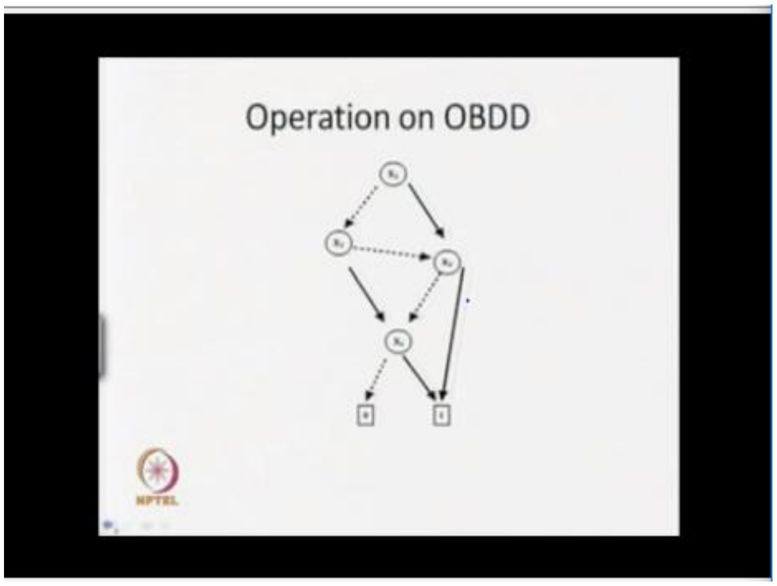
(Refer Slide Time: 30:28)



Operation on OBDD

Now when we are constructing from top r1,s1 we are starting it and coming down now you shall see that those particular nodes whatever we are getting over here all this thing are non terminal nodes now I am going to apply the binary operator on this particular terminal nodes okay. Now while I am applying this particular terminal node you just see that 0=0 will be 0, something like that 1+0 will be 1. So basically uses this is your r4 s4 , r7 s4 okay, r7 is 1 and s4 is one so 1+1 it is going to give me 1.

(Refer Slide Time: 31:23)


Operation on OBDD

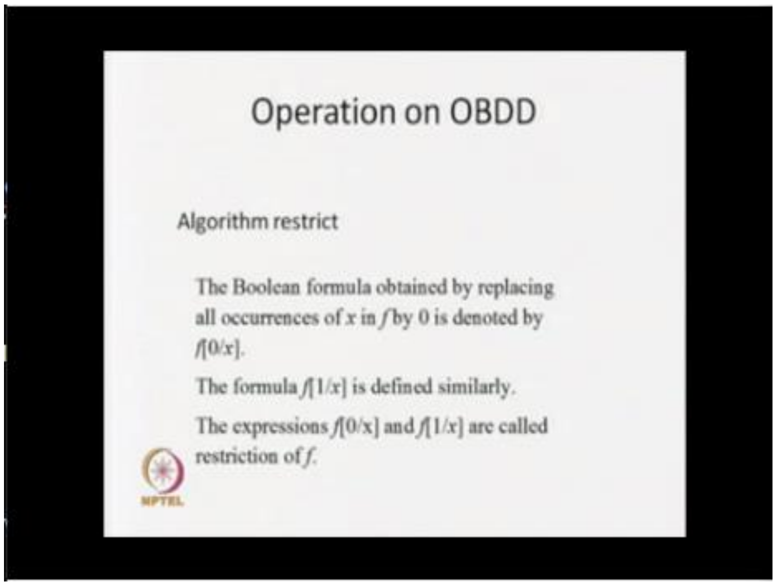
So in this particular case I am going to get 1, so I eventually I am getting this particular structure okay. Now this structure what will happen now I can construct a BDD on this particular structure and getting say this is your x1 this is a rest line and this is your solid line, and in this line I am getting x2, In solid line I am getting x3 okay. In this particular rest line I am getting x4 and in solid line I am getting this particular x5 okay.

Like that I am getting this rest line and this solid line, and then solid line will come to 1. So like that I can construct a BDD so I am getting a BDD and this is basically OBDD, I am getting this OBDD from this particular structure. Now after getting this particular ordered binary decision diagram we can say that it may not be reduced one. After that we are going to apply this reduce algorithm and after applying of this particular reduce algorithm, we are getting this particular structure ROBDD that we are getting.

So eventually we are getting ROBDD for f+g and the ordering of the variable is same with the ordering of a bf and bg this is basically x1, x2, x3, x4 where bf is the BDD or ROBDD of f and bg is the ROBDD of g, so what happen we have used this particular apply algorithm + bf bg and we are getting this particular ROBDD's for f+g.
So like that if we are having ROBDD's we can always use some binary operation on this two structure and we can eventually get the ROBDD's to represent this particular Boolean operation

okay. So this is one important algorithm and we will be using this algorithm while going to loop for the users of ROBDD.

(Refer Slide Time: 33:42)



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 $f[0/x]$ and $f[1/x]$ are called restriction of f.

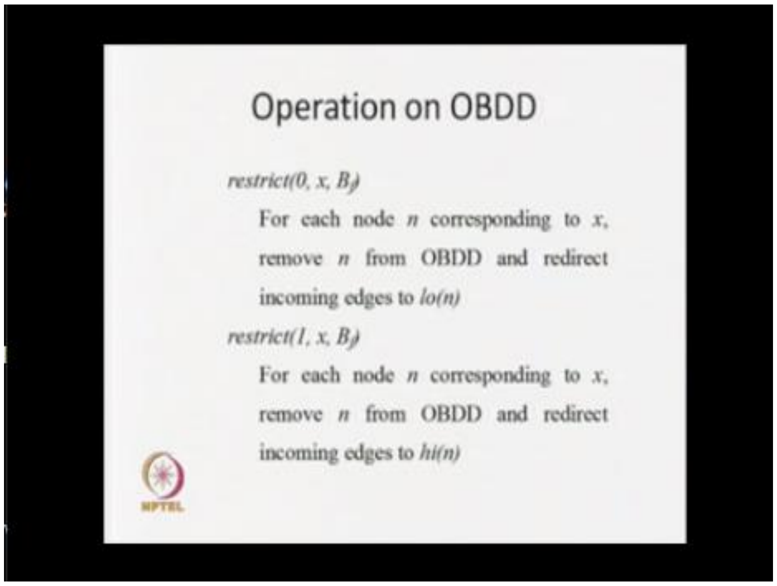
Now another issue is going to see that the apply algorithm, now we are going to look for another issues that a Boolean formula obtain by impressing all operands of x by f, x or in a 0 that means if I am having any Boolean function say f=xy+x'z+xyz. Something like that I am having the Boolean function so using this depends on the evaluation of x y and z, x may be either 0 or1, similarly y and z. But we can say that what is the valuation of this particular function of all x/0, so this is basically denote by f all x replace by 0 what is the functional value.

Similarly we can say that functional if I replace all x by 1, that means we are putting a restriction on one variable I am saying that x will always be 0 only or we will say that x will always be 1 only which means which you just look for the fellow of this particular function depending on the other variable. So this is the way we can look into it and this is basically known or restrictions of f okay.

We are going to say that we are following to restrict the function with respective some variables I am having a variable function which depends on several variables but I am saying that restrict that particular function for one particular variable to be 0, or restrict the function for a particular variable to be 1 only. So we are going to say this is the restriction for the particular variable function f. So we are going to have restriction of this function f is replaced by 0, x is replaced by 0.

Now if are having BDD representation of f say bf, whether we can directly restrict it or not, we can apply this particular restriction on it or not. So I know that if I am going to restrict it for x=0 then find something like that it will be your z only because x will be 0 and other terms will be 0. Similarly, if I am having the BDD representation of this Boolean function but can I apply this particular restriction on that particular Boolean function or not.

(Refer Slide Time: 36:12)



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)*

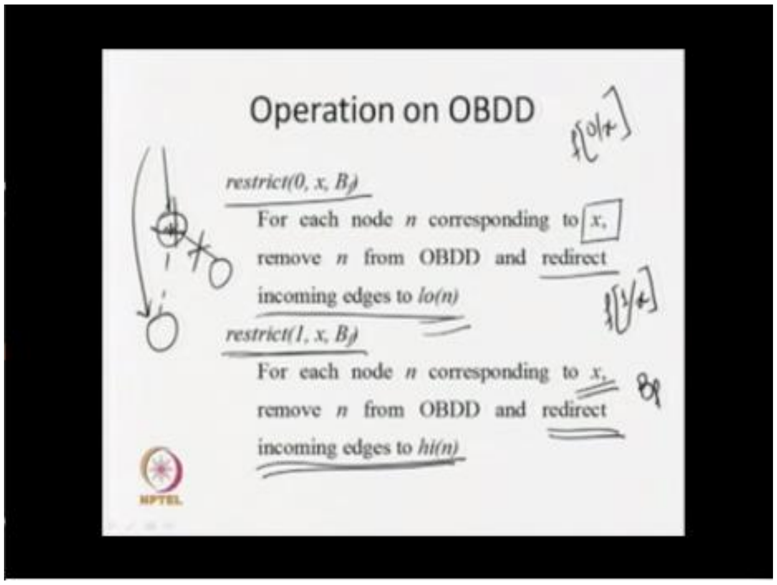
So it is very simple so what happens we are going to said that this is restrict 0, x, bf basically nothing but a replacing all variable of all x to be 0 and restrict 1, x, bf this is nothing but restriction of this Boolean function f replacing of all function x/1. So if I am having the BDD representation of this particular Boolean function of bf it might be a ROBDD also. Then with the help of this restrict operation or the restrict algorithm what we can do we can construct the f x replaced by 0.

So what we are going to do in this particular case we are going to look all notes corresponding to this particular label x okay. So we are going to look for all corresponding notes that label x and will remove all the all such n, and will redirect the incoming as lo(n) because why we have redirecting the incoming as to just lo(n) because we are restricting x to be 0.

So basically, if I am having a x node over here and said these are your low and this is your high okay, so we may some other. When we are restricting x to be 0, so basically if I am x node over here else and said these are your low and this is your high okay. So you may some other so when I am going to restrict x to be 0 that thing then x will be always 0, so this evaluation we are not going to consider.

So in this particular case we are going to remove this particular node and whatever incoming you are having all those will be redirected to this particular node. Similarly, we are going to use this particular restrict 1, x, bf so this one is similar. Now we are going to remove that entire particular node labeled at x and we are going to redirect the incoming as it too high of x. So this is where we can construct ROBDD's or all restrict operation also.

(Refer Slide Time: 38:33)



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)

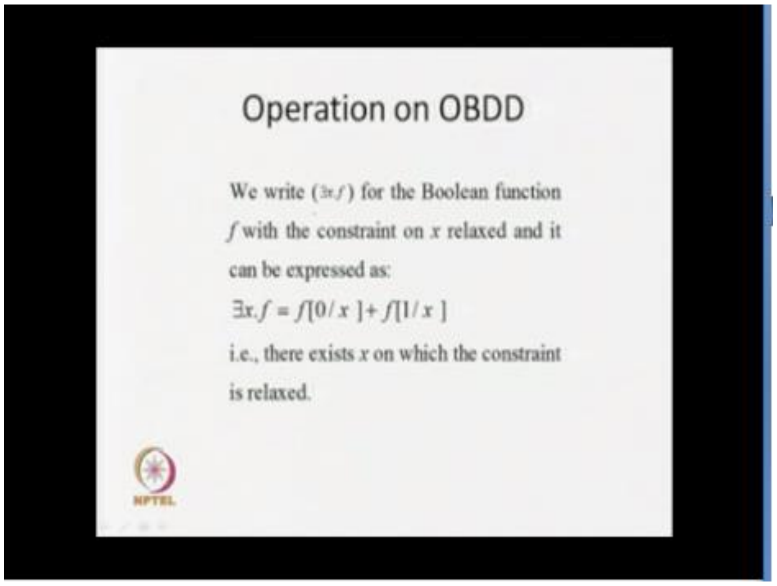
For each node *n* corresponding to *x*, remove *n* from OBDD and redirect incoming edges to *hi(n)*

While restricting the function x will be 0 and restrict the function x=1 okay, so we can if we are having a BDD representation of a Boolean function we can construct the restrict of 0, x, bf by moving the entire x from the particular BDD and redirecting the incoming access to the low of access okay. Similarly, for restrict 1, x, bf, so we are talking about both some functions say I am having f(x, y, z) = xy+ yz.

So in this particular case what will happen it is a function depends on this variable, so we are saying that sometimes we restrict that particular function on a particular variable that they will be either equal to 0 or that they will be equal to 0. Sometimes we want to release the constant on some variable, what is this particular function the function will be one provided x=1 and y=1 or y=0 and z=1. So the function will be 1 provided it is a particular constant.

Sometimes we want to release the constant on some variable said, you just look into a particular function or look into the function that is derived from the this particular n function, were it is independent of one particular variable that means we are going to release the constant of some variable x for this particular Boolean function x. That means the function x of evaluation 1 either x=1 or x=0, whatever be the value of x their functional that it depends on other variable.

(Refer Slide Time: 40:31)


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:

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

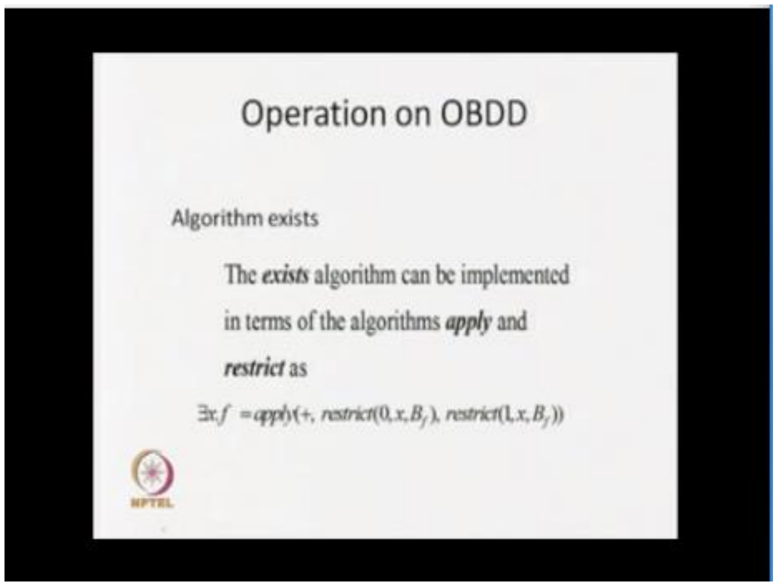
i.e., there exists x on which the constraint is relaxed.

So in that particular case we are going to see we are releasing a function on a particular variable okay. Now, generally we write this thing with ∂xf we are releasing this particular function on this particular variable x that means we are said that there x is the variable x where the constant is released okay. Now this is expressed as with this expression, here x is that as we are going to release the constant of x of this particular function. This is f either x will be replaced by 0 or x will be replaced by1.

That means x can, now we are going to look for the functional value of this function, so this it is very simple what is the Shannon expression basically we are going to see that x=0,then we are going to say that evaluation of the f with respective 0 or x=1. Now I want to release the function f, x=0 or x=1 then we are not considering the fellow of x. So we are taking the two x(bar) and x, so we are saying that f(x)=0 +f (x)=1.

So whatever may be the value of x, now going to look for the valuable function of the given function releasing on the particular function of variable x, we are saying that there are exist the x on which the constant is released. And we evaluate the particular function with the help of these expression something we release the function of the some variable. So this is the way we are going to, so with this we are going to expansion we can evaluate the particular relaxation.

(Refer Slide Time: 42:19)



Operation on OBDD

Algorithm exists

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

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

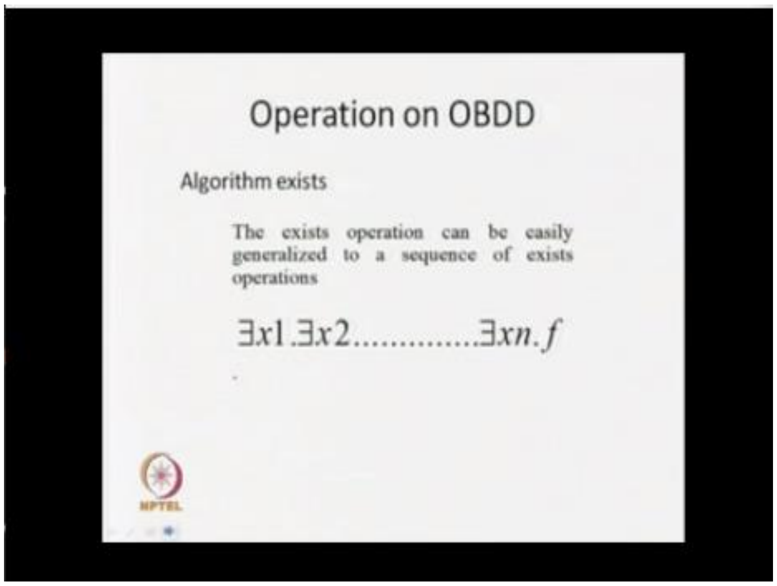
Now if we are having the BDD representation of this particular variable function whether we can use this particular BDD or not, so if we are having the BDD representation bf of a given function f then how to find the relaxation of a given variable x. All ready we have seen this restriction on operation on restrict algorithm, with the help of this restrict algorithm restricting the function on to a particular evaluation x=0 or x=1.

But in case of exists releasing on a particular constant may be either 0 or it may be1, so that is why we can evaluate this particular ∂xf all for this particular expression using this particular restrict 0, x, bf and restrict x=to be 0 and restrict x to be=1.So in case of realization we are going to set up x may be either 0 or 1, so we are using this particular OR operation plus operation.

So we are using apply (+, restrict (0, x, Bf), restrict (1, x, Bf)), now you just see that we are using this particular restrict x to be 1 restrict x to be 0.So if I am saying ROBDD or Bf and Bg after restricting I am going to get one OBDD's and second is restrict operation is going to give me another OBDD okay. But the variable ordering to result in OBDD will be same with the particular Bf now we are going to apply this particular person plus in the apply algorithm, so the resultant will be another OBDD.

So I can say that this is your b1 and say this is b2 so I am getting OBDD b1 and OBDD b2 after performing this chance I am going to get another OBDD b3 it not be a reduce one so the resultant BDD used to reduce the algorithm so I am going to get the ROBDD. So now the order of this particular variable sends the particular BDD of f. So this is the way we can find out the restriction on some variable.

(Refer Slide Time: 45:08)



Operation on OBDD

Algorithm exists

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

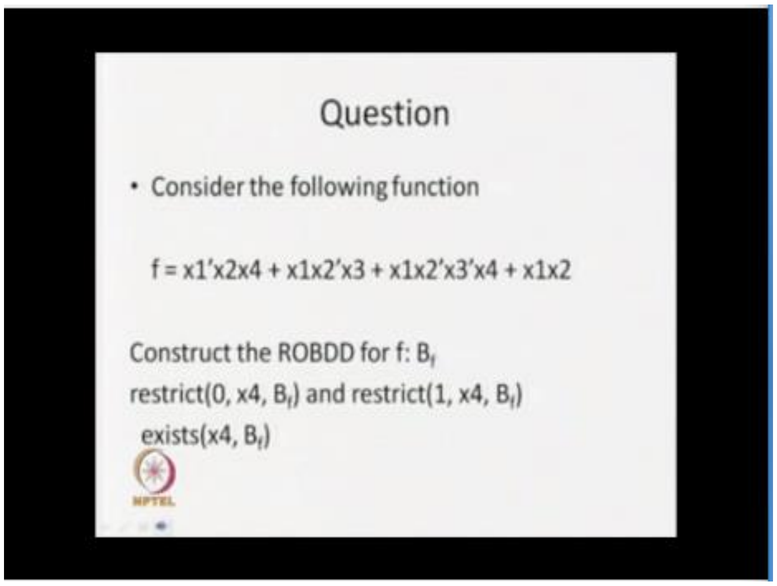
$\exists x1.\exists x2..............\exists xn.f$

So this way we can find the restriction of the of this particular given function f, now this particular x is the operation that can be nested also or it can be given as a sequence also. Basically we can say that if I am having an function f on some variable x1 x2 something like that xn. So what will happen if there x1 that means we are releasing that particular function on variable x, so we are going to get one function f which is now independent of x.

Now if we are getting gone function what will happen to do this particular function again I can put a relation on the x2,that means whatever resultant function I am getting I am going to put an on this x2 okay. So again I am going to get a Boolean function again I can put a relaxation on this particular Boolean function say on the variable x, like that we can keep on relaxing it some function we can use as a sequence.

So we are going to use this particular expression that is going to release on say xn xn-1 like that, so what will happen if f is going to represent the OBDD bf so this particular result there exists fx1 will be another OBDD so to this particular OBDD I am going to release an x2 I will apply on this set of operation over here, it may be another OBDD. So like that I can keep on going apply this particular operation on this particular resultant OBDD okay.

(Refer Slide Time: 46:46)



Question

- Consider the following function

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

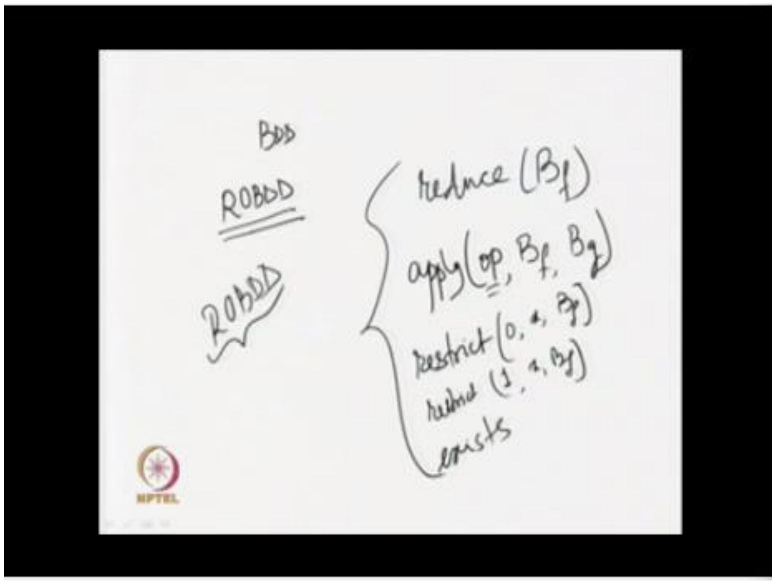
Construct the ROBDD for f: B_f
restrict(0, x4, B_f) and restrict(1, x4, B_f)
exists(x4, B_f)

So we have seen some algorithm till now, so first what happens we have seen that BDD that eventually have come to the ROBDD okay now what will happen we are going to get an unique representation of this particular ROBDD of a given function. So now to walk with that particular ROBDD representation we have seen some algorithm, so what are the algorithms the first we have talked about reduce, with the help of this particular reduce algorithm we can get a reduction of a given variable say Bf.

We have seen another algorithm, which is your apply with this particular apply we can use any binary operation on two BDD's BF and Bg and we have seen that this particular apply person is going to give us an BDD which is going to talk about the f (g) and one main constant on this apply algorithm is that Bf and Bg would have computable variable ordering.

And whatever resultant BDD we are getting that BDD will also have similar ordering with Bf and Bg but the output BDD may not be reduce on, so we can use apply this reduce algorithm to get the ROBDD. We have seen another things algorithm which is your restrict where we can restrict the function on some variable to be 0 or 1. So I can say that restrict (0, x, Bf) and restrict (0, 1, Bf) okay.

(Refer Slide Time: 49:05)



BDD

ROBDD

ROBDD

reduce (B_f)

apply$(\underline{op}, B_f, B_g)$

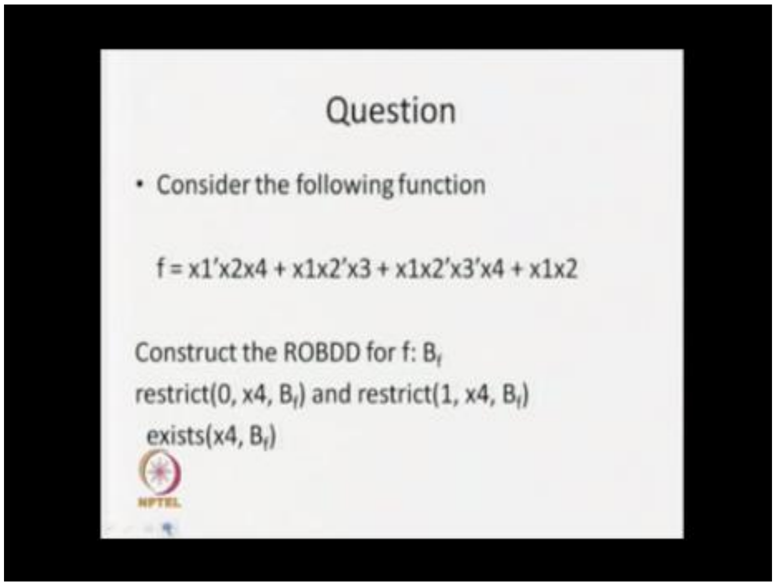
restrict$(0, x, B_f)$

restrict$(1, x, B_f)$

exists

So this is the restrict operation that we are getting over here and we have seen another algorithm which is x is basically we have going to release the constant of the given variable or it may be a series of variable. So these are the algorithm that we have seen which can be directly used on your ROBDD and resultant you are going to get another ROBDD as a result where the ordering of the output will be same as the input.

(Refer Slide Time: 49:08)



Question

- Consider the following function

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

Construct the ROBDD for f: B_f

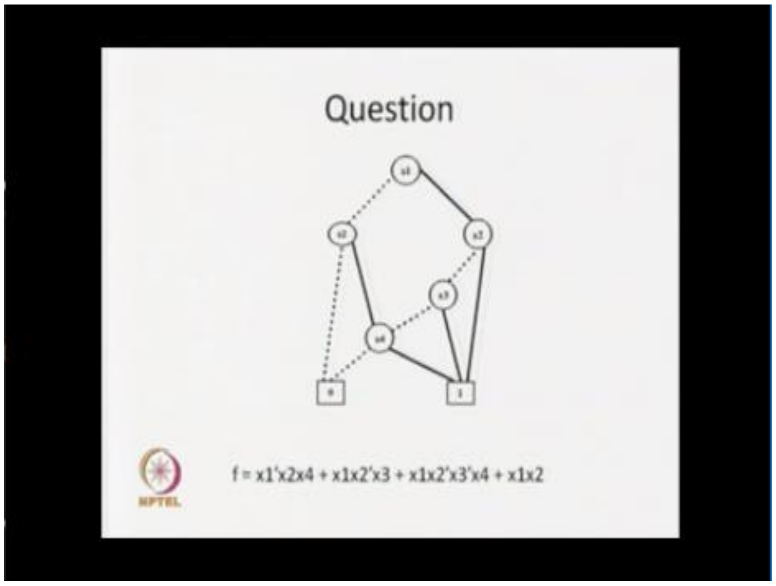
restrict(0, x4, B_f) and restrict(1, x4, B_f)

exists(x4, B_f)

Look into the simple question about here I am saying that consider this particular Boolean function and I am expressing a Boolean function. Now construct the ROBDD's of this particular function f: Bf. So I am giving a function F which depends on x1, x2, x3, x4, first we are going to construct the ROBDD for Bf then look for the structure of the restrict (0, x4,Bf) that is your are restricting x4 to be 0 or restricting the Bf where x4 to be 1.

That means you are restricting the function F where x4=1 also construct the BDD's and after you will look for this particular function exists(x4, Bf) that means you are releasing the constraint on the variable x4 for in the given function x okay. Just say what the Bf BDD look like, you can use the Shannon expression to construct the particular BDD here I will just give you the resultant BDD, okay.

(Refer Slide Time: 50:25)



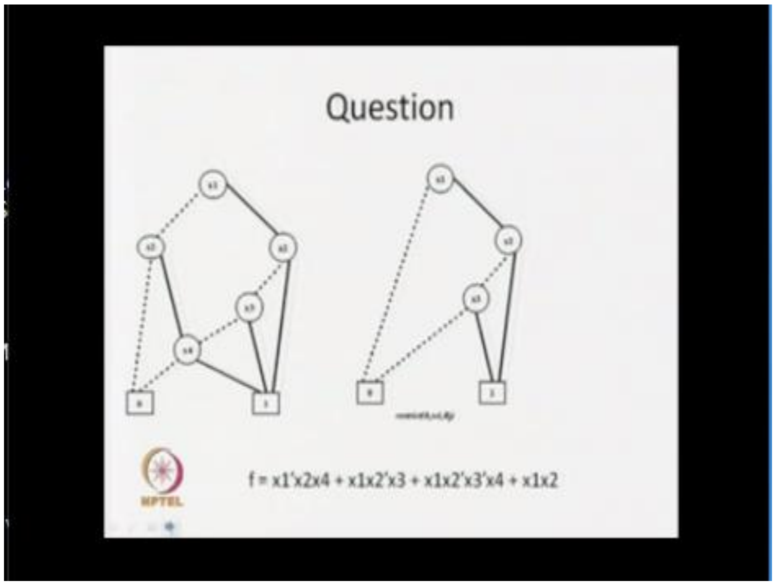
Question

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

So this is the ROBDD for this given function and the ordering that we are going to consider that is your x1, x2, x3, x4 okay. By using the Shannon expression you can construct the OBDD and after that apply the reduce algorithm to get the ROBDD and eventually you will get this particular structure. I am saying that with a given variable ordering we are going to get the unique representation.

If this is your ordering you are going to get this structure and the first term x1’ x2 x4, x1=0, x2=1, x4=1 so the function will give me the fellow 1. Similarly x1 x2’ x3 it will give me evaluation 1, x1 x2’x3’x4 and this is another one evaluation. X1x2 is the OBDD presentation of this particular given function and we can check that restrict node be applied here. So we can say that this is the ROBDD of a particular given function.

(Refer Slide Time: 51:52)



Question

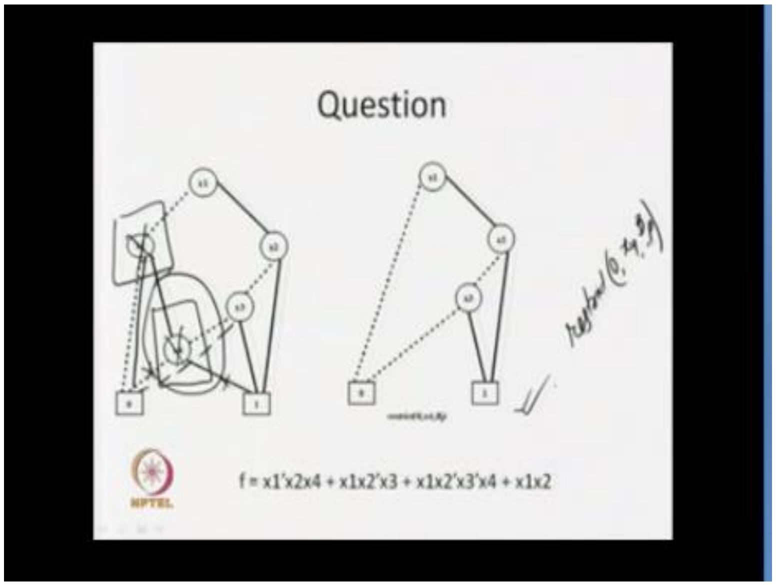
$f = x1'x2x4 + x1x2'x3 + x1x2'x3'x4 + x1x2$

Now what I am talking about as I was saying that apply the restrict operation of your restrict(0, x4, bf) so what is the rules we are going to restrict the function on a x4 so we are going to look for all x4 nodes, and we are going to remove this particular x4 node found this particular BDD’s. So we are going are going to remove this particular x4, when x4 is removed it is output is also removed, so this two will be removed.

And now we are going to restrict x4 =0, so we are going to say that once x4 all incoming would be as restricted redirector to this particular desk line, this desk line is redirect into to this particular 0 that means this desk line will be directed into 0 to this particular 0 and x4=0 and redirected to 0 then the solid line will be particular to be 0. So that means after removing a these particular position we are getting this particular BDD.

And if you look into these particular, these may not be reduced on now we are going to look for whether the reduction can be possible or not or you will find that you will find out this x2 is a redundant node, so you can remove this particular R x2 is also this is the application of the algorithm so eventually you are getting this particular BDD.

(Refer Slide Time: 53:28)



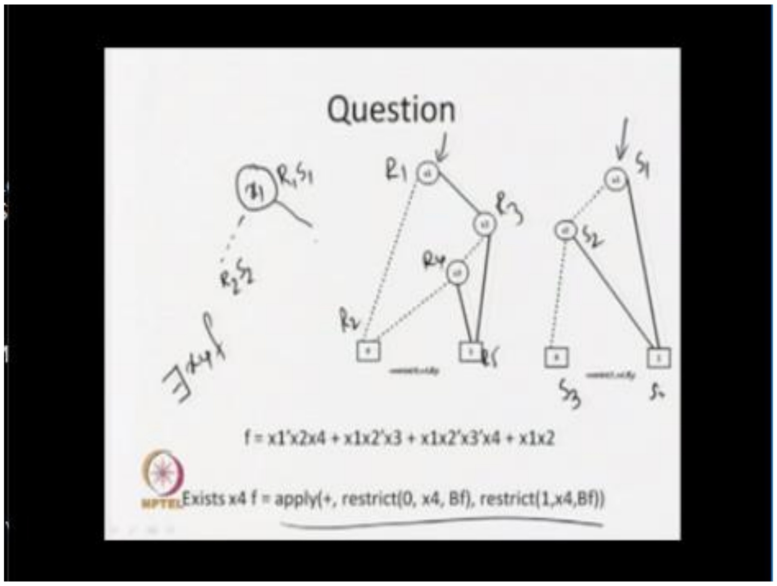
X1is 0 then it is coming to 0 , x1=1 then going to take decision on x2 than 0 and 1 then going to take decision on x3 this is the ROBDD for restrict (0, x4, Bf). Now similarly, I took construct the BDD for your restrict (1, x4, Bf),so in this particular case what will happen similar rules you are going to apply and what we are going to do we will remove this particular x4 an d whatever incoming we have that will be redirected to this particular 1 as this particular x4.

So that things that 1 will straight to come is desk line come over here, so we are removing this, so whatever resultant BDD getting it may not be reduced on just try to see the rules, now when I am coming to this particular x3, then we are getting there we are getting the redundant test we can remove it. After removing it what will happen that incoming will just be redirected to over here, and again I am going to look into the x2 and find out this is another redundant test we are going to remove it.

So after removing it there solid line will be redirected to this particular one, so eventually I am getting this particular structure. So this is basically structure for you restrict (1, x4, Bf), so we are this node for this particular BDD for this given function and you now getting the BDD's or OBDD's or if I can say that ROBDD's are restrict (1, x4, Bf) restrict(0, x4, B4). Once I am giving this particular two BDD's then one I can look for the exist there exists x4f that means and it is going to evaluate this thing with the help of   this particular expression apply (+, restrict (0, x4, bf),restrict(1, x4, Bf).

So that means I am having this two operation for the restrict 0 and restrict 1 what will happen the root node is your x1 node, in that particular case I am going to create an x1 node and so what is the labeled that this is your r1, r2, r3, r4, r5 so this is your s1, s2, s3, s4. So I am going to create an x1 node and I am just calling your r1 as 1. Now this desk line will go and we are going to construct a solid line, so the desk line will go low (r1) and this is your (r2) and this is your low (s1) and (s2).

(Refer Slide Time: 56:44)



Question

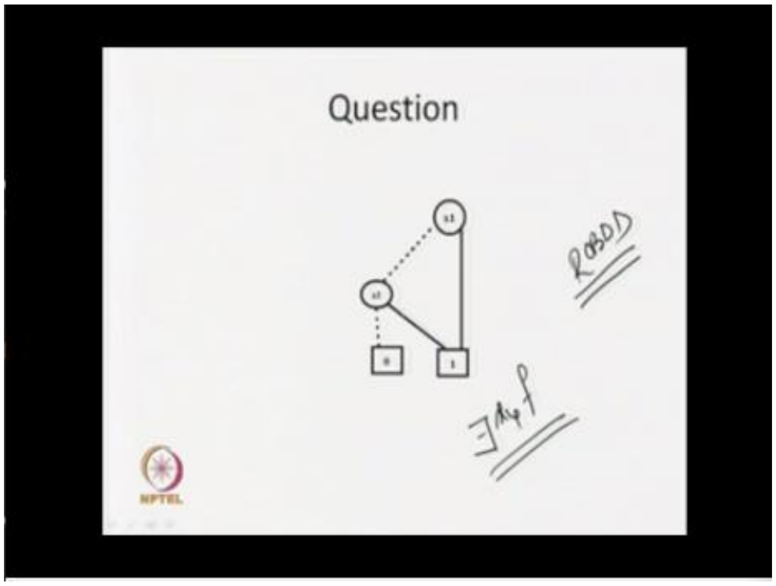
f = x1'x2x4 + x1x2'x3 + x1x2'x3'x4 + x1x2

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

In your solid line hi (r1) is your (r3) and hi (s1) is your (s4). Now we just say that is your r2, s2 and r3 and s4, now we say that r2 terminal node and s2 is a terminal node that means no since it is a non terminal node so we are going to create a node x2. Similarly, here r3 is a non terminal node by s4 is a terminal node. So we are going to x2 nod eat that particular node now similarly I have to look for the solid and desk line.

So like that we can now use this particular apply algorithm and construct this particular BDD's okay. Now after that we are going to get a resultant BDD it may not be a reduced one and after that apply the reduced algorithm the ROBDD's or this particular x is x4f. Now what will the structure look like, so you can find that eventually I will get this particular ROBDD.

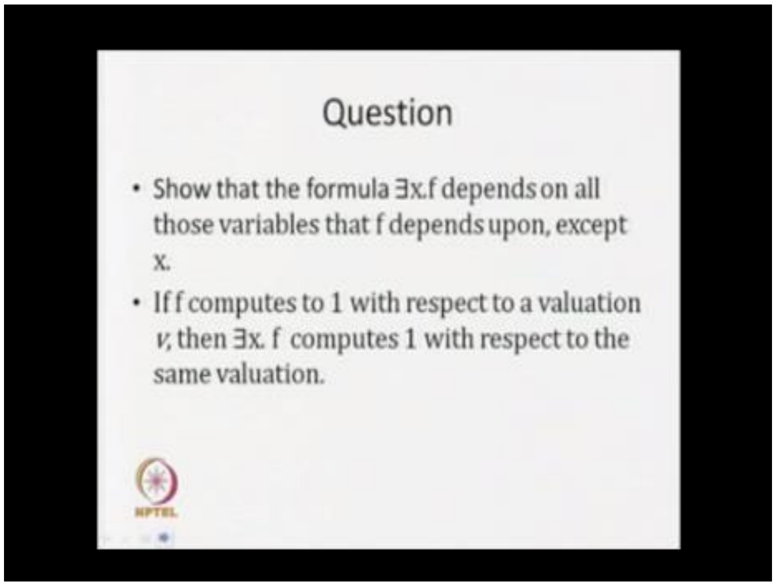
(Refer Slide Time: 58:17)



Question

You test it that this is your there exists this x4f, the ROBDD represents this on this of the particular given restrict of x4f okay so what will happen after getting this BDD you try to construct it and what about you are going to get you apply this algorithm and you will get this particular ROBDD. Now just have a quick look on this particular problems of question.

(Refer Slide Time: 58:38)



Question

- Show that the formula ∃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 ∃x. f computes 1 with respect to the same valuation.

Show that the formula for every f depends on all those variables that f depends upon expect x, so this function f it depends on the some variable x1 to xn somebody like x y z like that. There exists x f what does it mean it relaxing this particular function on some variable. So now what is it asking that there exists of that depends on all those variables that depends on x. Now we are just trying to getting the filling what there exists whether this is correct or not.

If it is correct you can stabilize it or if it is not correct then also you can stabilize this or for the given example for the first one is a simple one. Second one is the E (f) computes to one with the evaluation if I say f is a function x, y, z, w like that I can have the evaluation of this particular once and say x=1 y=0 z=1 answer w=0, so with respect to this evaluation we are going to have some evaluation of this function.

So if this is complete this once then what is that saying is then they are exists f computes on with respective to the evaluation of this function. So if this evaluation complete once then what is they are saying that there are exists a compute on with respect with to this evaluation. Now what we have to do that there exists f compute 1 with respected to some evaluation. I thought you might of got some fillings Yes in this it is going to compute once, why because they are exists x.

In this particular case we are going to release the constant of x whether x=0 or x=1 it is intimated if for a particular evaluation f computes to 1 then there exists x, f will also compute 1, with this particular same evaluation because there exists x and the of this particular x=1, 1, 0. When x is 1 it is going to give you a on this side also it is also given 1 okay. So with this I will conclude my lecture here today.