

**Embedded Systems - Design Verification and Test**  
**Dr. Santosh Biswas**  
**Prof. Jatindra Kumar Deka**  
**Dr. Arnab Sarkar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Guwahati**

**Lecture – 23**  
**Binary Decision Diagram**

Hello everybody. Welcome back to the online course on Embedded System Design Verification and Test. So, till now we have seen about the verification and we have discussed about the CTL model checking. Today we are going to introduce one new tool called Binary Decision Diagram which will help us to represent our system in a compact way and that some tools or technique binary decision diagram will be used for verification purpose also. So, we are going to discuss about the binary decision diagram today.

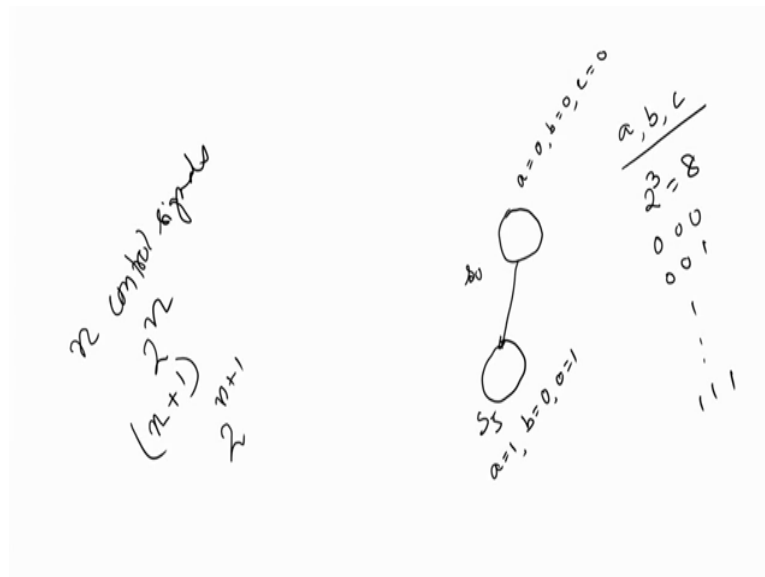
(Refer Slide Time: 01:07)

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

Now, we have discussed about a model checking algorithm and after discussing or after going through the model checking algorithm, what we observed that the complexity of the model checking algorithm is polynomial in the size of the state machine and the length of the formula, ok. So, size of the state machine; again the state machine depends on the number of control variables or number of working variables that we have in the system. So, the inherent problems with the model checking algorithm is the state space expression problem. So, to control the state space expression problem, BDD or Binary Decision Diagram is going to

help us up to certain extent.

(Refer Slide Time: 02:03)



Now, what is the state space? You just see that if we are having 3 variables a b c say control variable of whatever it may be, then we know that we are having total 2 to the power 3. Total 8 different combination of a b c they varies from 0 0 0 0 0 1 to 1 1 1 so, total 8 possible combinations. So, this possible combinations are generally treated as the state of the system. That means, state of the system is all together 8 that means I can say that this is my state s 0 where the values of the signal variable a b c equal to 0. That means, a equal to 0, b equal to 0 and c equal to 0.

Similarly, so from s 0 we are having a transition tool say s 5, ok. In case of s 5, I can say that the values of the control variable is a equal to 1, b equal to 0 and c equal to 1. So, that values of the control variable basically talk about the present state of the system and if you look into this scenario, then we are going to have total 2 to the power 3 which is equal to 8 different states.

Now, for that if I am having n control signal, then total number of states space or the complete state space will be your 2 to the power n, ok. So, if n is 10, we are going to get 1024. If varies 11, we are going to get 2048 like that ok

So, you just see if we are going to increase the control variable, number of control variable the number of states are also going to increase, but in some system all states may not be

reachable. What does it mean? That means, all configurations are not vary for that particular system and we basically are concerned about the reachable state space.

Now, in that particular case say in my design due to some constraint I am increasing the control variable from  $n$  to  $n + 1$ . In that particular case, the total number of states will be your  $2$  to the power  $n + 1$ . So, that means the state space expression is exponential in nature, ok.

So, if and if you look into that model checking algorithm, it is nothing, but the graph traversal algorithm, and the system is represented with the help of (Refer Time: 04:41) structure which is a graph and we are going to apply our model checking algorithm on that particular model. So, it is nothing but the graph traversal algorithm. So, if we having more number of nodes, then algorithm is going to take more steps though it is polynomial in time ok, but still for more states we are having more problems because we have to have more memory space in our machine also to encode all the states.

Now to get our or to content this particular problem now we are going to look another tools, another approach to represent our system; which is known as your BDD Binary Decision Diagram.

(Refer Slide Time: 05:29)

**Binary Decision Diagrams (BDD)**

- Based on recursive Shannon expansion

$$\underline{f = xf_x + x'f_x'}$$

- Compact data structure for Boolean logic  
can represents sets of objects (states) encoded as Boolean functions
- Canonical representation  
reduced ordered BDDs (ROBDD) are canonical

$f_+(x=1)$   
 $f_+(x=0)$

So, in a nutshell how we are going to talk about the BDD or how what is the basic principle of the BDD? So, BDD is basically based on recursive Shannon expression. So, we can

express all the Boolean expression or we can expand it with the help of Shannon expansion.

So, Shannon expansion is given here like that if  $f$  is a function of some variables, then it can be expanded with the help of the participating variable. So,  $x$  is the participating variable for this function  $f$ , then  $f$  can be expressed,  $f$  is equal to  $x f$  of  $x$  plus  $x$  bar  $f$  of  $x$  bar. So,  $f$  of  $x$  basically says that it is the functional value when we are going to put  $x$  equal to 1 and  $f$  of  $x$  bar basically say it is the functional value 1, we are going to put  $x$  equal to 0.

So, if  $x$  equal to 1, then 1 into the values of the function with respect to  $x$  equal to 1 when  $x$  equal to 0, then  $x$  bar become 1 1 dot functional values of the function with respect to the variable values  $x$  equal to 0. That means we are taking care of all the possible combination when  $x$  is equal to 0 and equal to 1. So, this is the basis of our Shannon expansion and to construction of the BDDs, we are going to take help of this particular Shannon expansion.

Now, what is BDD? It says that it is a compact data structure for Boolean logic. That means if we are having any Boolean expression, it can be represented with the help of BDD. Again the state space that we are having can be encoded with the help of an Boolean expression, Boolean logic expression and that Boolean logic expression can be converted or can be represented as BDD. So, it is compact data structure for Boolean logic.

What is the advantage of this particular BDD representation? It says that we are having a canonical representation of an expression of a Boolean expression. If the representation is your reduced ordered BDD, Reduced Ordered Binary Decision Diagram ROBDD, at that time we are going to get the canonical representation and what does it means. That means, we are going to get an unique representation or unique BDD or given Boolean expression.

We are going to look into it, we are going to discuss about it, but you have to very careful about that it talks about the reduced ordered BDDs. So, if it is a simple BDD, then we may not get canonical representation. So, what does it means? We may have several structure for the same Boolean function, but if it is ROBDD, we are going to get unique representation.

(Refer Slide Time: 08:33)

### Shannon Expansion

$f = ac + bc$

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

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

$f = af_a + a'f_{a'}$   
 $= a(c+bc) + a'(bc)$

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

$ac + abc + a'bc$   
 $= ac + a'(bc)$   
 $= ac + bc$

Now, that Shannon expansion just I am going to say how the Shannon expansion is going to give us the proper functional value. Already I have explained that we have taken care of both, all the possible cases x is equal to 0 and x is equal to 1. Now, in the particular case you just see that I am taking small example f is equal to a c plus b c. So, this a Boolean expression having 3 variables a b c.

Now, f a bar until I am going to get a bar, if I take the value of a equal to 0, that means if I put a is equal to 0, then this f a bar will become b plus b c because 0 into c will become 0 and 0 plus something is equal to that value and similarly if I put f a, it is nothing, but the function value at putting the value of a equal to 1. So, in that particular case since a equal to 1, we are going to get c plus b c. So, this is the x functional value when we say x equal to a.

Now, what is the given function it is saying that f is equal to a into f a plus a bar into f a bar. That means, we are going to get a into f a. That means, a into b c plus b c and a bar into f a bar is a bar into b c, ok. So now if I expand it then what I am going to get a c plus a b c plus a bar b c so, this is this is equivalent to a c plus. If I take a bar comman, then I am going to get b c plus, sorry I did a mistake. So, if I take b c comman, then I am going to get a plus a bar and plus a bar is equal to always 1.


So, I am going to get a c plus b c. So, this is the given function. So, this given function can be expanded with the help of this particular Shannon expansion. We have expanded it with the help of the variable a. Similarly we can we can look for the expansion of this particular

function with respect to say variable b or c and eventually the Shannon expansion is going to retain the given function. So, this is the basis of our BDD construction. So, we are using Shannon expansion to construct our BDD.

(Refer Slide Time: 11:07)

**Binary Decision Tree (BDT)**

- Binary Decision Trees are trees whose non-terminal nodes are labeled with Boolean variables  $x, y, z, \dots$  and whose terminal nodes are labeled with either 0 or 1.
- Each non-terminal node has two edges, one dashed line and one solid line.
- Dashed line represents 0 and solid line represents 1.



Now, first before going to BDD we are going to talk about BDT, Binary Decision Tree and from that we will say how we are going to get the BDD. So, what is the BDT? The representation is like; binary decision tree are trees whose non-terminal nodes are labelled with Boolean variables  $x, y, z$  and whose terminal nodes are labelled with either 0 or 1 this is one thing. So, it is having non-terminal nodes I can say some of the non-terminal nodes like that.

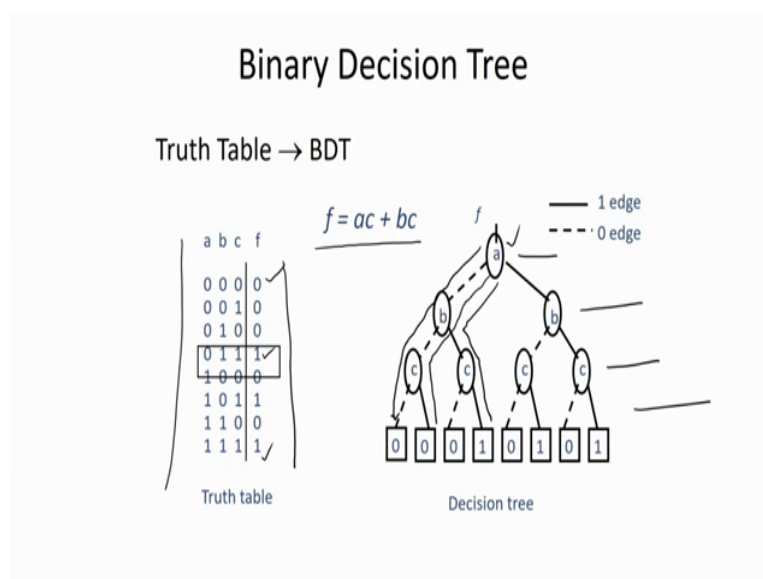
So, if I am having a function of  $a, b, c$ , then this non-terminals node will be either marked with those particular variable  $a, b$  and  $c$  and along with that it is having terminal nodes. So, terminal nodes is denoted by this particular box and if have the values either 0 or 1, it says that if it is 0, that means the functional values is 0 with respect to this particular evaluation and if it is functional values is 1, then we are going to say this is the terminal node 1.

So, now each non-terminal nodes are having two edges; one is dashed line and second one is your solid line and dashed line represent 0 and solid line represent 1. What does it mean say a is a non-terminal node. So, we can say that we are having a dashed line to this particular non-terminal nodes  $b$ . So, in that particular case since this dashed line indicate that this is my value 0 or I can say that evaluation of this particular variable at that particular point is 0.

Similarly it is if we having the outgoing edge which is my solid line, so it says that the evolution of this particular variable is 1 if I am going to traverse this particular part, ok.

So, what we have in BDT Binary Decision Tree, we are having nodes, two types of nodes; terminal nodes and non-terminal nodes. We are having edges, solid edges and dashed solid is going to indicate that the evolution is 1 and dashed is going to indicate that the evolution is 0. Again terminal nodes is going to give me or terminal nodes are going to give us the functional value for 1 or it may be 0. So, these are the component of BDT.

(Refer Slide Time: 13:39)



Now, look for a simple example say I am taking the same function  $f$  is equal to  $a \cdot b$  plus  $b \cdot c$ . So, we know that if we are having many Boolean function that can be represented with the help of truth tables. So, this is the functional or the functional values of the given Boolean function. So, it is said that if  $a$  is equal to 0,  $b$  is equal to 0,  $c$  is equal to 0, then  $ac$  will become 0 and  $bc$  will become 0 and eventually my functional value is 0.

If  $a$ ,  $b$ ,  $c$  are 1, then  $a \cdot c$  will give me 1,  $b \cdot c$  will give me 1. So, 1 plus 1 will give me 1. So, this is the truth table representation of Boolean function. Now, while I am going to draw the binary decision tree for every variable, we have to see the decision whether values of that variable is 0 or 1. So, that is why that from variable  $a$ , I am having either variable  $a$  can take value 0 or 1.

So, after getting the values of  $a$ , now we have to look for the values for  $b$ . Again the

evaluation for b may be either 0 or either 1 after taking c, after taking the decision on b, we are going to take a decision on c, again c may be either 0 or either 1. Now, if I traverse say a equal to 0, b equal to 0 and c equal to 0, in that particular case I know that function and variable is 0. So, this is the terminal node. So, when a equal to 0, b equal to 0, c is equal to 0, my functional value is 0, it is coming to this particular terminal node 0.

Similarly, if I say that a equal to 0, b equal to 0 and c equal to 1, then again my result is 0. If I say that a equal to 0, b equal to 1 and c equal to 1, then my functional value is 1, a equal to 0, b equal to 1, c equal to 1, my functional value is 1. So, it is terminal node at this particular.

Now, this is the representation of BDT, but if I just look into that BDT we do not we are not getting any advantage of benefit because this truth table is also exponential in nature. Now, for 3 variables we are having 8 entries. Now, if I increase one more variables say if we are going to work with 4 variables, then we will be having 16 different entries. So, it is increasing exponentially

So, similarly if I look into this particular BDT, here also I will find that if I am working with 3 variables, then I will 3 levels of non-terminal nodes because we have to see whether these are 0s or 1. Now, if I have one more variables, then what happen I am going to get one more level of non-terminals node and after that I am going to get the terminal nodes. So, for 4 variables, the number of terminal nodes will be your 16 and it will have 4 level. So, again the BDT is exponential in nature. Just for construction purpose we are showing, but excess with BDT we are not getting any advantage, this still having the nature of exponential expansion.



(Refer Slide Time: 17:05)

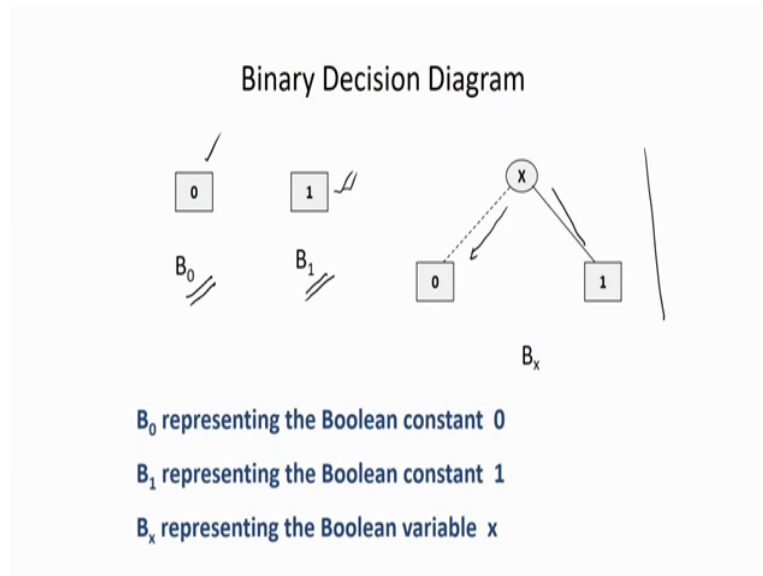
## Binary Decision Diagram

- 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

Now, from BDT now we can talk about the binary decision diagram which is similar to BDT except that BDT is a tree, but BDD is a graph or it is a dag directed acyclic graph and other notions are similar. We have terminal nodes; we have non-terminal nodes, ok. Terminal nodes are marked with the functional value 0 or 1, non-terminal nodes will be marked with the variables and every non-terminal nodes have having two out going as dashed out going as indicates that the value of that particular variable is 0 and the solid line indicates that the value of that variable is 1. So, instead of tree, we are having a graph. It is a DAG Directed Acyclic Graph.

So, we can now see how we can construct a BDD.

(Refer Slide Time: 18:07)



Now, when we talk about the BDD, then we are having 3 primitive BDD. We are having two constant values; 0 and 1. So, these are basically the unknown node. So,  $B_0$  is the BDD for constant 0 is finally, represent that with  $B_0$  and it is a terminal node with the value 0. Similarly,  $B_1$  is the BDD representation of constant value 1 and it represent with this particular terminal node 1 and similarly for any variable  $x$ , we have this particular representation  $B_x$ . It says that BDD representation of that variable  $x$ .

So, we know that since it is a Boolean variable, it can have it can take two values either  $x$  equal to 0. So, it is going to indicate with the help of this particular dashed line and the value of  $x$  is 0. So, it is finally in the in this terminal node and similarly,  $x$  can take one. So, it is represented that with this particular solid line and this is terminal 1. So, these are the primitive BDD actually we have in our system.

(Refer Slide Time: 19:19)

### Shannon Expansion $\rightarrow$ BDD

$f = ac + bc$

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

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

Now, just see we are going to look for the construction of the BDD. So, if we are going to have that same function  $f$  is equal to  $ac$  plus  $bc$ , then  $f$  a dash is equal to  $bc$  and we say this is the function  $g$  and when  $f$  a is equal to  $1$  or  $a$  is equal to  $1$ , then  $f$  of  $a$  equal to  $1$  is equal to  $c$  plus  $bc$  which I am saying that this is the function  $h$ , this is the function  $g$ .

Now, what it says that now  $a$  can have values  $0$ . When  $a$  is having value  $0$ , that means we are coming to another terminal node with the evaluation of  $0$  and in this particular non-terminal node, it is going to represent the functional value of  $g$  and what is  $g$ ;  $g$  is nothing, but  $bc$ . Similarly when the values of  $a$  is equal to  $1$ , then we are coming to this particular terminal nodes, and what equation of Boolean expression we are representing in this particular terminal node is the function  $h$  which  $c$  plus  $b$   $h$ .

Now, you just see that loop nodes is going to represent the given function  $f$ . Now, after knowing the evaluation of that particular variable which is indicated in this particular starting node and here we are talking about  $a$ . So,  $a$  can have either  $0$  or  $1$  when we are having  $a$  equal to  $0$ , then we know that the functional value is  $bc$ , we are indicating it with  $g$ . So, this particular node is going to talk about the evaluation of the function  $g$ . Similarly this one is going to talk about the evaluation of the function  $h$ .

Now, after coming to this particular point, now we are having function of two variable  $b$  and  $c$ . Now, we have to see the evaluation of the function with respect to  $b$  and  $c$ . Now, when for function  $g$ , we are talking about that we are looking at the evaluation with respect to  $b$ , so

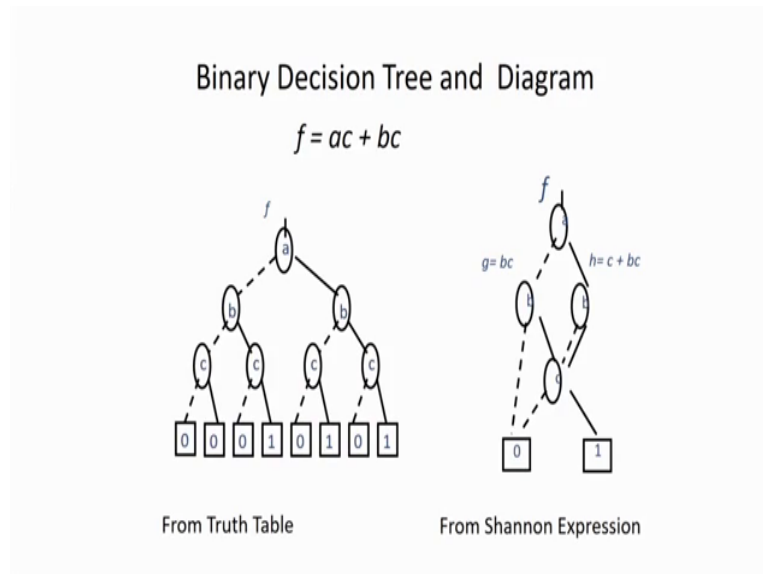
these terminology is b and when it is your b equal to 0, that functional value is 0. So, we are getting this particular dashed test and coming to this particular terminal node 0.

When d equal to 1, then the functional value is c so, we are coming to this particular node and what is the function over here we are going to have the variable c. Similarly if I am going to now look for the function a, if function h if b is equal to 0, the functional value is c. So, it is coming to this particular terminal node c when c plus bc when b equal to 1, again we are going to get c so, with 1 also it is coming to c.

Now, the remaining function that we need to evaluate is having the constants of variable c only. Now, we know that if c is equal to 0, the functional value is 0. If c is equal to 1, then the functional value is 1. So, now finally we are going to get this particular construction or BDD c equal to 0, it stands to 0, c equal to 1 that one.

So, when we use the Shannon expansion, then the BDD representation of this function is this one. So, it is having 4 terminal nodes and sorry 4 non-terminal nodes and 2 terminal nodes.

(Refer Slide Time: 23:03)

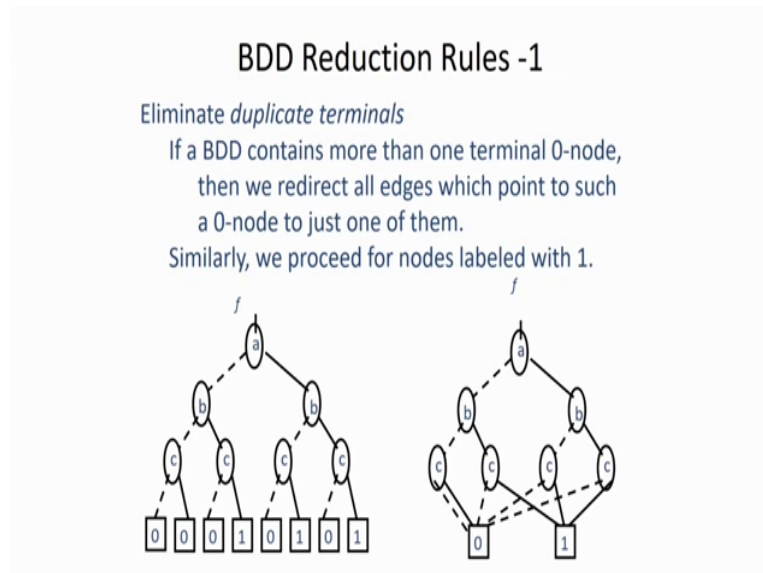


Now, we see same function we are having so, for BDD representation we have these representation where we are having 8 terminal nodes and 7 non-terminal nodes. When you use the Shannon expansion, we say that we are getting 2 terminal nodes and 4 non-terminal node. So, just here I am trying to give an idea that if we use BDD, then for most of the Boolean function we are going to get an compact representation, ok. We will see the

discussion.

Now, we have one, we are having these things. Now, it may have a scope to further reduce this particular BDD or BDT. Now, we will see what the reduction rules are.

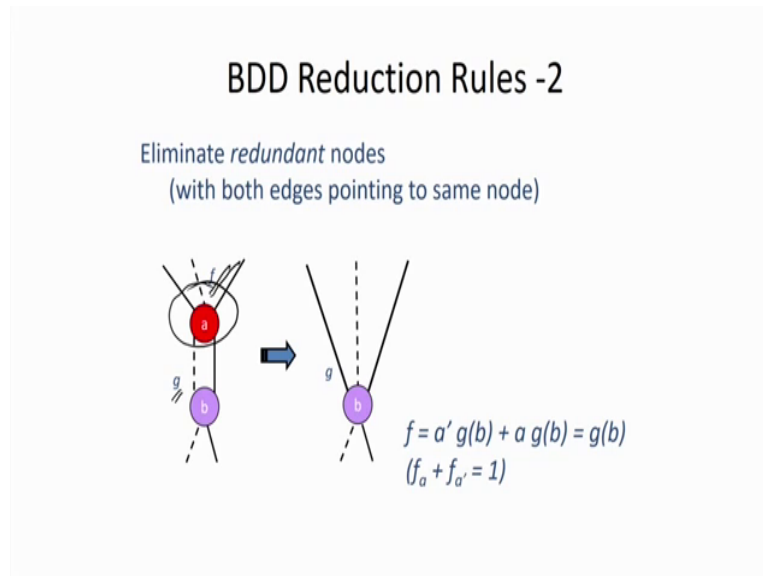
(Refer Slide Time: 23:59)



So, first reduction rule it says that eliminate the duplicate terminals. So, because the terminals are having either values 0s and 1, we are having multiple terminal nodes and multiple terminal nodes with value 0 and with values 1. So, they can be merged. We can represent all 0s by one terminal nodes and 1, all one by one terminal nodes

So, that is why we said if this is the BDT, then what will happen I can have such type of construct where all zeros are merged together and all ones are merged together. So, we are getting this and accordingly we are redirecting all those particular edges which is going to confirm to the original redirection original evaluation or original elevation. So, this is the rule 1. If we having multiple terminal nodes, we can merge them together, ok.

(Refer Slide Time: 24:25)

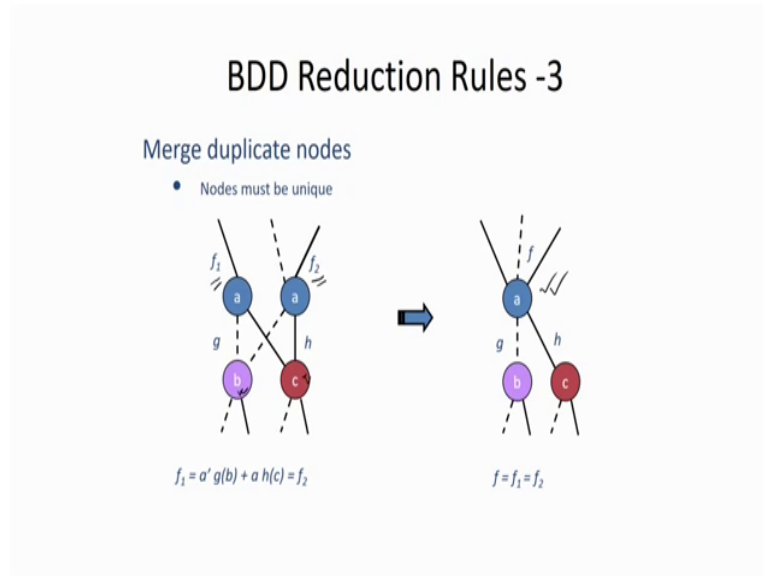


So, BDD reduction rule 2 which says that eliminate the redundant nodes. So, what is the redundant node? So, for this particular node we just see, so this is a; that node a what will happen if I take the evaluation of a is equal to 0, it is coming to this particular terminal node, non-terminal node way. When I take the evaluation of a is equal to 1, again it is coming to the same non-terminal node b.

Now, at non-terminal node a, we are giving the evaluation of function f and f b. We are giving the evaluation of function g. So, you just see whether it is a equal to 0 or a equal to 1. In both the cases, we are going to evaluate the function g in the next level. So, that means decision on a slightly redundant means it is now independent of this particular variable a in this particular evaluation.

So, now since for what a equal to 0 and a equal to 1, we are having the same function g. So, we may eliminate this particular redundant node. So, checking the values of edge redundant at that particular point, it says that elimination of the redundant nodes, this is the second node and rule 3 talks about the merge duplicate nodes.

(Refer Slide Time: 26:31)



Now, we have to see what are the duplicate nodes and how we are going to merge them. So, node must be unique. So, in this particular case just consider this particular scenario in this particular non-terminal node, we are evaluating the function  $f_1$  and in this particular non-terminal node, we are evaluating the function  $f_2$ , ok.

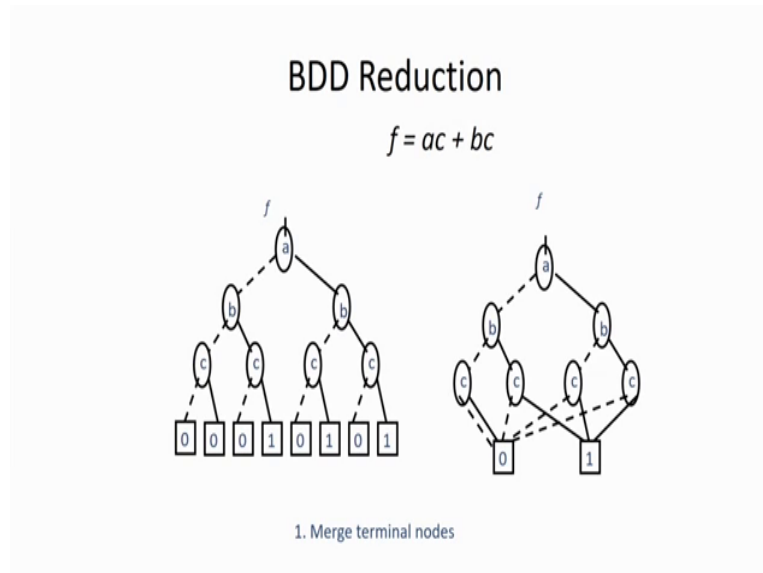
Now, when  $a$  equal to 0, we have to evaluate the function  $g$  in node  $b$  and when  $a$  is equal to 1, then we have to evaluate the function  $h$  with node  $c$ . Similarly if  $a$  is equal to 0, we have to evaluate the function  $g$  and  $a$  is equal to 1. We have to function  $h$ . Now, just see the nature of these two non-binary nodes of almost similar, or I can say it is same because for  $a$  equal to 0, we are going to have the same function  $g$  for  $a$  equal to 1, we are having the same function  $h$ .

So, in that particular case what will happen, we can merge these two nodes to one node and after merging it what will happen that outgoing edges are having the similar behaviour, but all the incoming nodes, these two, these two nodes will be the incoming nodes to this particular merge node  $a$ , ok. So, we can merge duplicate nodes or duplicate non-terminal nodes. So, this is the BDD reduction rule number 3. So, these three rules can be used to reduce our BDD.

So, first rule can be first rule will be used only once because we are having all the terminal nodes at leaf level. All 0s will be merged to one 0 nodes and all ones will be merged to one 1 node. So, it will be applied only once, but rule 2 and rule 3. So, removal of redundant nodes and removal of duplicate nodes can be applied several time because first you remove one redundant nodes, then what will happen we are getting some nodes which can be merged

together. After merging some of the nodes, you will find that again we are getting some other nodes as a redundant node. Now, we can remove that redundant node also. So, they can be applied several times until and unless we could not apply any other rules for this reduction, ok.

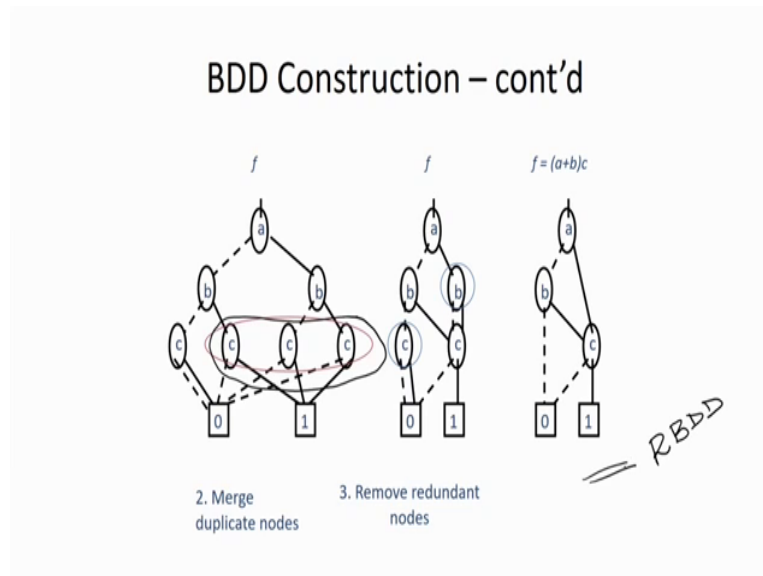
(Refer Slide Time: 29:11)



So, in that particular case what will happen if I cannot further apply the rule 2 and 3 to the BDD that we have achieved, then that BDD is known as the reduced BDD of the given function. Now, for this particular function  $f = ac + bc$  we know that this is the BDD and this BDD can be first merge, these all zeros to one node and all ones for another node.



(Refer Slide Time: 29:47)



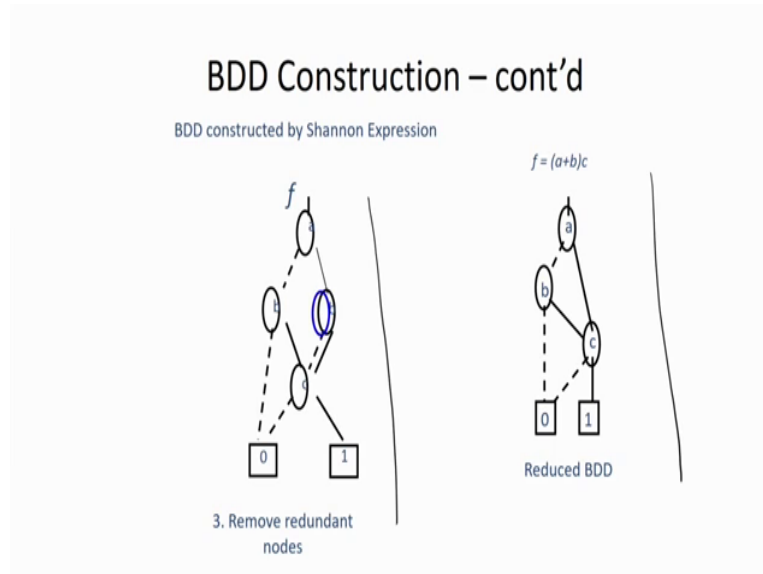
Now, after having these things, now you observe what can be done now here, here you will see their observation these 3 nodes, ok. They are having the similar behaviour, but other  $c$  nodes is having different behaviour because for both 0 and 1 it is zero, but for this tree node for 0 evaluation is 0, for 1 evaluation is 1, ok. So, this is slightly different.

So, now what we can do? We can merge those particular node and we are coming to this particular point, ok. Now, after this what we are getting you just see now we have merged duplicate node. Now, we are going to get some redundant nodes. What are the redundant nodes? These two are redundant nodes because you just see for both  $p$  equal to 0 and  $p$  equal to 1, it is coming to that node  $c$  here also for  $c$  equal to 0 and 1, the functional value is 0.

So, we can remove those particular redundant node. Now, finally we are going to get this BDD. This is the representation BDD representation of the given function and it is the reduced one. So, we say this is the reduced binary decision diagram because we cannot further reduce this particular BDD.

So, we are taking a particular function and from BDT, we are coming to RBDD, Reduced Binary Decision Diagram.

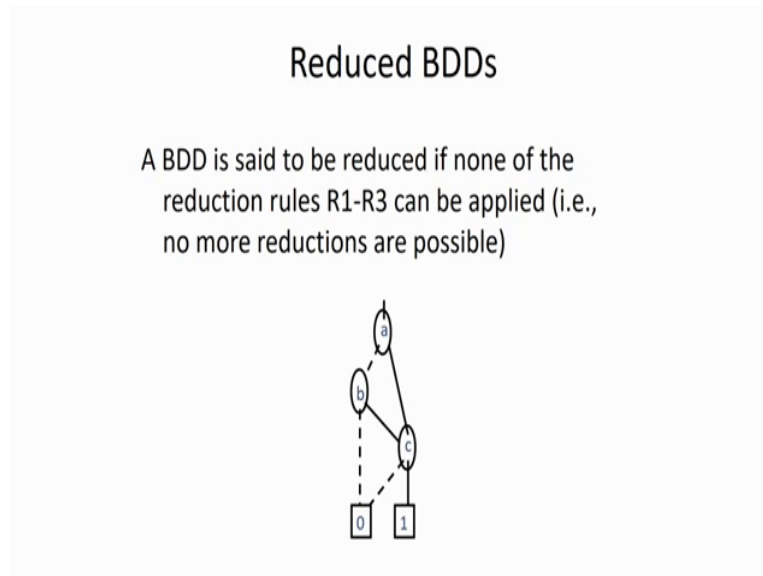
(Refer Slide Time: 31:25)



Now, the same function I think we have constructed with the help of Shannon expansion. So, in that particular case when I apply this particular Shannon expansion, we get this particular structure, ok. Now, here you will find that this node b is redundant node for because for both b is equal to 0 and 1, I have to look for the evaluation of variable c, so that b can be removed. So, what will happen after removal of this particular redundant node, I am going to get each particular structure.

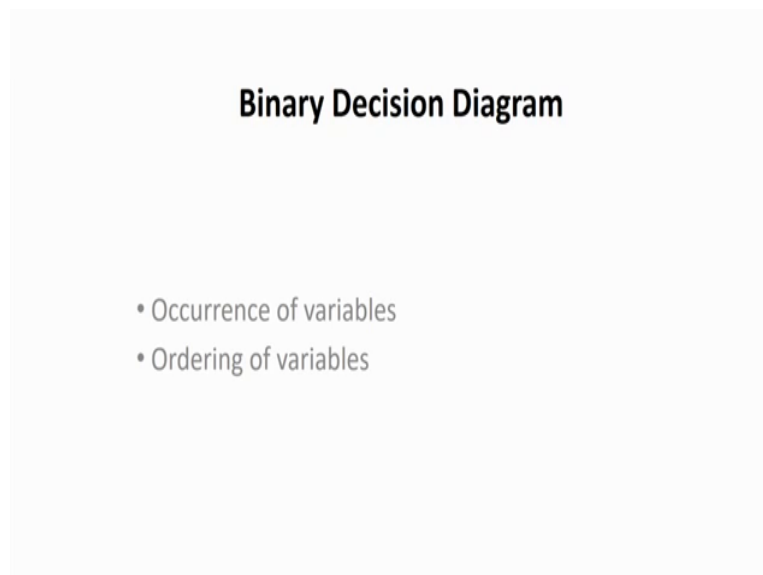
Now, you just see whether you are using BDT or Shannon expansion, we are going to get the same reduced BDD whether always the reduced BDD is having the same structure. My answer is no because it depends on some other criteria also. We will see that criteria also what is that particular other criteria, but anyway whatever we have achieved, it is reduced BDD because we cannot further reduce. That means we cannot further apply the rule removal of redundant nodes and merge of duplicate, ok. So, this is the reduced BDD and for both the approaches, we have achieved same structure, same BDD or same reduced BDD and why we have got the same BDD and it will be clear to you in that particular discussion, ok.

(Refer Slide Time: 32:59)



So, what is the reduced BDD? Already I have said that BDD is said to be reduced if none of the reduction rule R1 to R2 can be applied to this no more reduction are possible. So, here we cannot apply anymore rule. So, this is the reduced BDD of the given function.

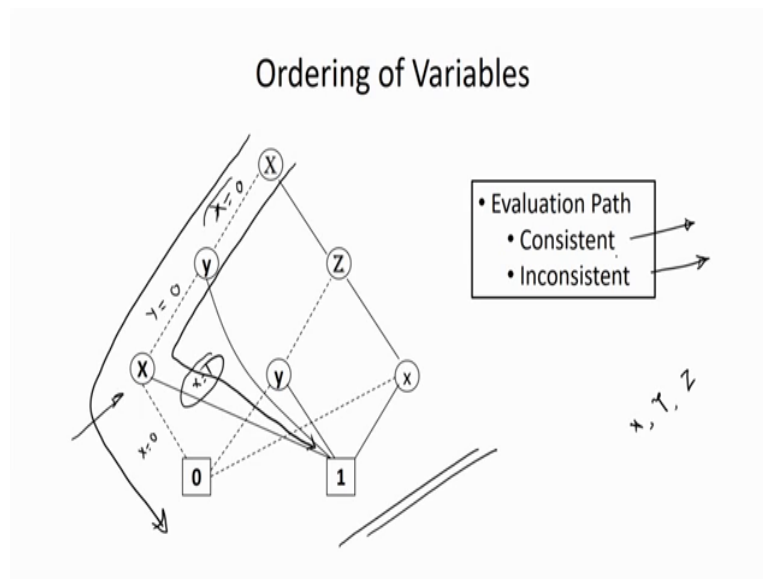
(Refer Slide Time: 33:26)



Now, when you discuss about the BDDs and when you have given the definition of the BDDs here, we are not mentioning anything about the ordering of the variables. That mean in which order variables may appear and occurrence of the variable, how many times that variable can occur in the BDD say x s of variable, then how many times x can occur or in how many

levels  $x$  can occur, we are not mentioning anything about that issues while defining the BDD.

(Refer Slide Time: 34:05)



So, with respect to the given definition of BDDs, we have to accept this one also an BDD of some given function, ok.

Now, in this particular case you just see here I am having key variable  $x, y, z$ . So, first I am having the evaluation of  $x$ , then when  $x$  equal to  $0$ , I am having an evaluation of  $y$ . When  $x$  equal to  $1$ , I am having the evaluation of  $z$ , with respect to  $z=1$ ,  $z$  is equal to  $0$  and  $z$  is equal to  $1$ . Again  $x$  is appearing over here, ok. So, as per the definition of so, with respect to the given definition of BDD, this is a so with respect to the given definition of BDD or representation of BDD.

Now, if we do not restrict the occurrence of a variable in a path, then we are going to face some problem. So, due to that problem now we have to have the notion of those particular path. We are having now two kind of evaluation path. One is known as your consistent path and other one is your inconsistent path.

So, when we are going to consider a evaluation path, always we have to consider the consistent evaluation path only. We have to ignore the inconsistent evaluation path. Now, what path will be the inconsistent one? If you look into this particular example, you just see in this particular evaluation it says that  $x$  equal to  $0$ . When  $y$  equal to  $0$ , we are coming to this particular point, ok.

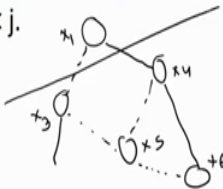
Now, when we are taking decision on  $y$ , again we are relooking into the evaluation with respect to  $x$ . So, it says that if  $x$  is equal to 0 over here or  $x$  equal to 1 over here, so in that particular case when we are coming to this particular non-terminal nodes you just see that already we have taken a decision on  $x$ . This is  $x$  equal to 0 here. Again I am taking a decision or looking for evaluation of  $x$ . It is  $x$  equal to 0 and  $x$  equal to 1 so, if I follow this particular path, then I can say it is a consistent path because in this evaluation the values of  $x$  is written  $x$  0, but if we consider the other path, then this path is treated as an inconsistent path because in a particulate evaluation that variable values will be either 0 or 1. It cannot have both the values.

So, in this particular case what we are going to see some point of time we are saying that  $x$  is equal to 0, at some point of time saying  $x$  is equal to 1, ok. So, in that particular case, this will be the inconsistent path. So, when I am going to look for the functional values of the function that is represented by this particular BDD, then we have to ignore this particular inconsistent path. There may be several inconsistent path. So, you should not consider those particular inconsistent path for evaluation of this particular function or evaluation of this particular function used only consistent path, ok. So, these are the issues of consistent path and inconsistent path in BDD.

(Refer Slide Time: 37:13)

### Ordered BDDs (OBDDs)

- Let  $[x_1, x_2, \dots, x_n]$  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  $[x_1, x_2, \dots, x_n]$  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  $i < j$ .



So, to avoid this particular inconsistent path in BDDs, now we are coming to the notion of ordered BDD, ok. This is called ordered BDD. What is an ordered BDD? In an ordered BDD,

we are going to follow a particular order of the given variables, ok. We are having a set of variables say  $x_1$  to  $x_n$ , then we are going to create an ordered list of variables without duplicating or without duplication. That means, all variables are unique and we are going to construct the BDD and let the BDD all of this variable occurs somewhere in the list, ok. So, in BDDs this variable will occur somewhere in the construction.

Now, we say that B has an ordering  $b_1, b_2, \dots, b_n$  up to  $x_n$ . Sorry B has an ordering  $x_1, x_2, \dots, x_n$  if all variable labels of B occurs in that list and for every occurrence of  $x_i$  followed by  $x_j$  along any path in B, where  $i$  is less than  $j$ . Now, what it says if I am having this non-terminal nodes, whatever path you are saying that if this is the ordering is your this one,  $x_1$  to  $x_n$  if it is your  $x_1$ , this is your  $x_3$ , this is your  $x_5$ , sorry  $x_4$ , then  $x_5$  like that. So, we can have something like that. So, in that particular case we are coming over here.

Now, what is the label of this particular node, it must follow this particular ordering. So, in the ordering it says that already we have using  $x_1, x_3, x_4, \dots$ . Till now we have not used  $x_2$ , but the label of this terminal node cannot be  $x_2$ . It may be after  $x_5$ , it may either  $x_6$  or may be  $x_7, x_8$  like that. If the variable  $x_2$  is present in this particular function, then  $x_2$  have to come somewhere over here which appear before  $x_3$  and  $x_4$  or may be  $x$  that I greater than 2. So, this particular ordering we are going to maintain while constructing the BDD, then we said this is the ordered BDD and what is the order it is  $x_1, x_2$  to  $x_n$ .

(Refer Slide Time: 39:55)

### Impact of the chosen variable ordering

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

Now, impact of the chosen variable ordering now we are talking about the variable ordering.

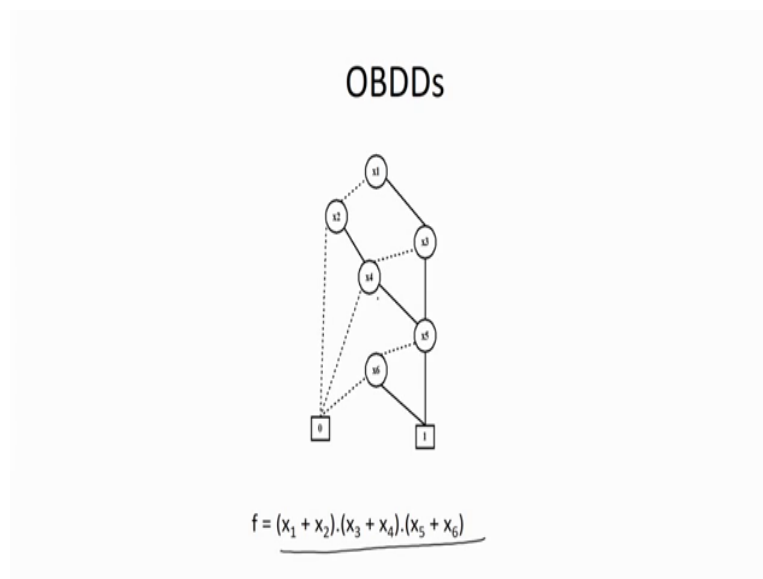
Now, when we are going to use this particular variable ordering what ordering you are choosing depends on that we are going to get different structure. So, structure is not unique if it is our order are different. So, in this particular case just consider a simple example that we are considering this particular Boolean expansion, also Boolean expression.

So, in that particular case either I can have the ordering like that  $x_1 x_2 x_3 x_4 x_5$  like that or maybe we can consider this particular Boolean ordering variable ordering  $x_1 x_3 x_5$ , then  $x_2 x_4 x_6$ . That means, all odds are here are coming at a beginning and then, all the even but here we are using this particular expression.

So, in that particular case it says that if I use this particular ordering, then total number of nodes will be your  $2^n + 2$ , but if we use this particular variable ordering that total number of nodes become  $2^n + 1$ . So, there is a tremendous impact of the variable ordering, but to find out the proper variable ordering to get the optimal BDD representation of even function is a hard problem. We do not have the proper algorithm, but we use some heuristic to contain the size of the BDD, ok. So, depending on the variable ordering, the construction is different and number of nodes will be different.

So, in this particular case one is saying that it is polynomial, but second one becomes exponential.

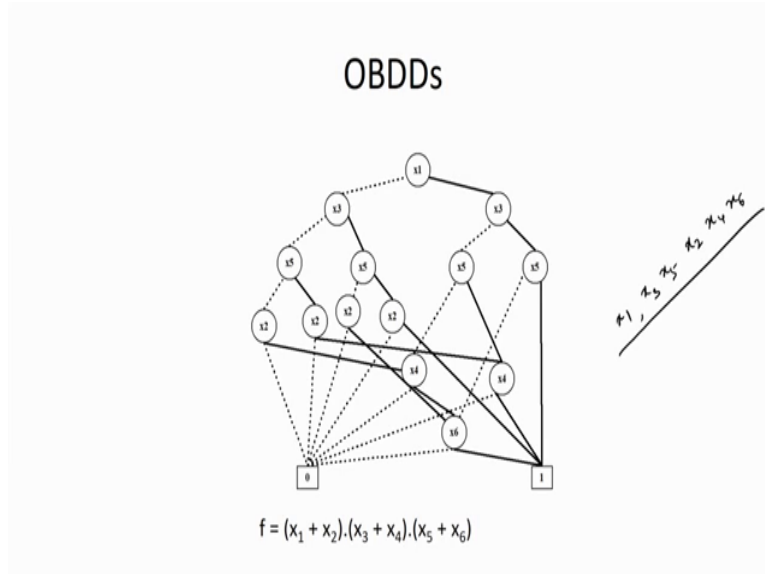
(Refer Slide Time: 41:45)



So, this is the construction as I am saying you can check it. So, if this is the given function of

6 variables and the variable order that I am using  $x_1 \times 2 \times 3 \times 4$  like that  $x_1 \times 2 \times 3 \times 4$ , then this is the BDD representation of this particular function.

(Refer Slide Time: 42:05)



But same function if I use the ordering  $x_1 \times 3 \times 5 \times 2 \times 4 \times 6$ , then we are getting another representation which is having more number of nodes which is exponential in nature of the number of variables. So, variable order variable ordering is also have importance by constructing BDD, ok.

(Refer Slide Time: 42:35)

### Reduced OBDDs (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 seen reduced BDD, we have seen ordered BDD. Now, if we are having an ordered BDD and if it is a reduced one, then we say this is your Reduced Order Binary Decision Diagram and we write ROBDDs, ok. So, in case of ROBDDs, variables are following a particular variable ordering and secondly, it is a reduced one. We cannot further reduce it and in that case we are going to say this is your ROBDD.

(Refer Slide Time: 43:09)

### 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.

So, now what is the property of ROBDD? The reduced BDD or ROBDD representation of a given function is unique. That means, if you represent a given function with an ROBDD, Reduced Order Binary Decision Diagram, then orders you are going to get an unique representation. So, that is why we say that ROBDD is a canonical representation of a given function because in case of canonical representation, we are going to get an unique expression. So, here also the BDD representation is unique or reduced ordered binary decision diagram.

Now, second it says that what it says that let  $B_1$  and  $B_2$  be reduced ordered BDD with compatible variable ordering. So, what is that compatible variable ordering? That means, whatever variable ordering we are using for  $B_1$ , same variable ordering we are using for  $B_2$ , then in that particular case we are having an identical structure and it is 2 BDDs. We are saying it is the representation of the same function and both the BDDs are having compatible variable ordering. That means, both are having the same variable ordering, then the structure is same.

The order in which we apply the reduction does not matter. So, I am saying that I can apply the reduction rule first removal of redundant nodes, then the merge of similar nodes ok, but we can apply in the reverse way also, but finally we are going to get the same BDDs. So, that is why since its representation is unique, we are going to say it is a canonical form if we are going to use ROBDDs.

(Refer Slide Time: 45:11)

### 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  $B_2$ .

We can consider now two different functions  $f_1$   $f_2$  and say  $B_1$  is having  $B_1$  is the BDD representation of  $f_1$  and  $B_2$  is the BDD representation of the Boolean function  $f_2$ . If the ordering of  $B_1$  and  $B_2$  are said to be compatible if there are no variable  $x$  and  $y$  such that  $x$  come before  $y$  in the ordering of  $B_1$  and  $y$  comes before  $x$  in the ordering of  $B_2$ , ok. So, that means they should not have different variable ordering. In that particular case, we are going to say that this is having the compatible, ok. That means, these two BDD representation of these two functional compatible if both of the BDDs are going to follow the same variable ordering.

Now, we are having several operations actually.

(Refer Slide Time: 46:07)

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

Handwritten notes and diagrams illustrating the apply algorithm. On the left, "apply op, Bf, Bg" with arrows pointing to the text. On the right, "f = ab + c", "g = b + ac", a circled "f + g =", and "Bf + Bg" with "f + g = Bf + Bg" below it.

Now, how we are going to do with the help of your BDD say I am having a function  $f$  say  $a b$  plus  $c$  and I am having another function say  $g$ , where it is your say  $b$  plus  $ac$ , then what will happen always I can perform the operation say  $f$  plus  $g$  or I can perform  $f$  dot  $g$  like that. So, these are the Boolean operators that we have. Now, we are going to get some representation, some function over here and here we know that this function  $a b$  plus  $c$  can be represented that with the help of BDD. I say that this is the BDD representation for function  $f$  is your  $B_f$ , I say it is better to write.

Similarly, the BDD representation of the function  $g$  is a  $B_g$ , ok. Now, can we perform this operation directly on the BDDs? One way is to you perform the operation, get the Boolean expression then represent that Boolean expression with the help of BDD, ok. This is one approach, but all if already we have the BDD construction, can you use those BDD to get this particular function  $f$  plus  $g$ . That means, we can say this is may be  $B_f$  plus  $B_g$  can get a similar effect. Yes, we have a method for that where we can perform the Boolean operation on BDDs and this algorithm is known as apply. Apply is the common method and this apply is having 3 parameters, apply op  $B_f B_g$ .

So,  $B_f$  is the BDD representation of function  $f$ ,  $B_g$  is the BDD representation of function  $g$ , but in that particular case both BDD should have compatible variable ordering. If the variable ordering is different, then we cannot use this particular apply method. So, in that particular case we can use this apply op  $B_f B_g$ . Now, if I want to perform this particular  $f$  plus  $g$ , then

what I can say apply, I can use this particular algorithm apply plus B f and B g, ok. So, this whatever BDDs we are going to get, this BDD is going to give me the, give us the BDD representation of this particular operation f plus g, but we have to remember that it has both the BDDs must have a compatible variable ordering. It may be reduced or it may not be reduced, but they have to be ordered and secondly, both BDDs must have the compatible variable orders, ok.

(Refer Slide Time: 49:25)

### Operation on OBDD

**Algorithm apply**

Application of  $apply(op, B_f, B_g)$  will give an 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.

$ROBDD \triangleright B_f$   
 $ROBDD \triangleright B_g$   
 $\downarrow apply(x, B_f, B_g)$   
 $OBDD$   
 $\downarrow reduce$   
 $ROBDD$

Now, after application what will happen is, we are going to get a resultant BDD. So, this resultant BDD will also have the same variable ordering whatever variable ordering we have for B f and B g, but the resultant BDD may not be an ROBDD, ok. So, we are talking about the ordered BDD or I can talk about ROBDD say B f is the ROBDD representation of function f and B g is the ROBDD representation of x after performing this operation say apply plus B f B g resultant BDD will be your ordered BDD.

That means, the resultant BDD is also going to have the same variable ordering that we have for B f and B g, but it may not be the reduced one. So, after that what we can do, we can apply this particular reduction method which to get the ROBDD, ok. So, this reduced algorithm can be implemented because we have the rules tree reduction rule. So, we are having an implementation of that particular reduced algorithm also, ok.


(Refer Slide Time: 50:53)

### Operation on OBDD

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

$$f = \bar{x} \cdot f[0/x] + x \cdot f[1/x]$$
$$g = \bar{x} \cdot g[0/x] + x \cdot g[1/x]$$

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

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


Now, how you apply algorithm works, ok? So, the apply algorithm works under Shannon expansion, ok. So, if I am using two function *f* and *g*, now this is the Shannon expansion of *f* and this is the Shannon expansion of *g*. Now, when we are going to perform any operation, then you just see that you can have these components separately and these components separately and it says that evaluation of function *f* and *g* with respect to *x* equal to 0 and here we say that evaluation of function *f* with respect to *x* equal to 1. So, finally these two components are coming over here and other two components are coming to the other side, ok.

So, in BDD you just see that *x* equal to 0 is having some evaluation, *x* equal to 1 is also having some evaluation. So, this is say *B<sub>f</sub>* and say this is your *B<sub>g</sub>*. So, this the construct and say if both are that variable *x*, then you just see that for *x* equal to 0, we are going to have this particular path for *x* equal to 1 we have this particular path, ok.

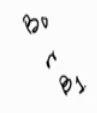
So, now we can use apply algorithm on to given BDDs and that apply algorithm works with respect to Shannon expansion. So, whatever resultant BDD, we are going to get it will give us the correct evaluation of the operation ah. Already I have mentioned about these things.

(Refer Slide Time: 52:41)

### Operation on OBDD

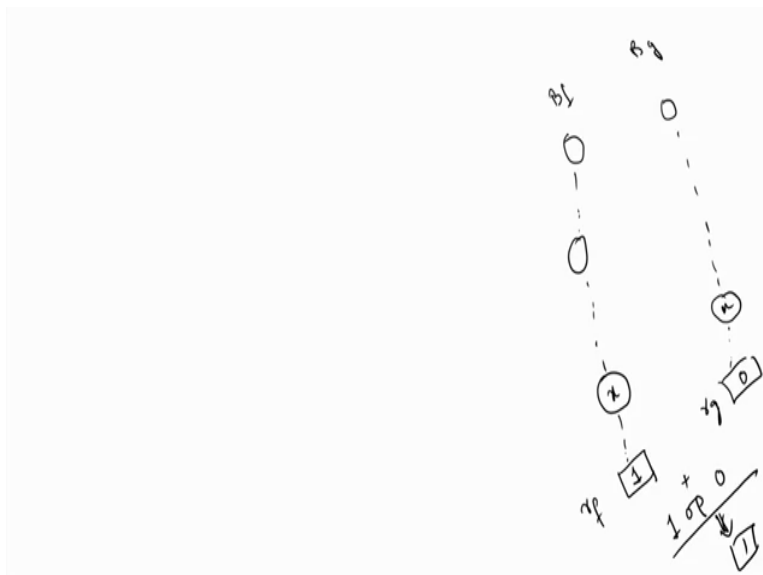
Algorithm apply(op, B<sub>f</sub>, B<sub>g</sub>)

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



Now, how algorithm works? So, if  $r_f$  and  $r_g$  are terminal nodes with labels  $l_f$  and  $l_g$ , respectively compute the value  $l_f \text{ op } l_g$  and the resultant OBDD is  $B_0$  if the value is 0 or if it is  $B_1$ , otherwise. Now, what does it means?

(Refer Slide Time: 53:03)



So, I am having some construct finally say this is the  $x$ ;  $x$  equal to 0, it is 1. Similarly this is  $B_f \text{ op } B_g$ . So, this is also, so I am going to say  $x$  and say this is 0.

Now, what it says when both  $r_f$  and  $r_g$  are terminal nodes. So, this is  $r_f$  and  $r_g$  if both are

terminal nodes, then you apply the operation 1 of 0 on the terminals, and you create a terminal nodes depending on this particular operation say if it is a plus operation, then 1 plus 0 will give me 1. So, the terminal node will be 1. So, if it is a terminal node, either we are going to get the BDD B 0 or BDD B 1 depending on the result of the operation.

(Refer Slide Time: 54:13)

### 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  $apply(op, hi(r_f), hi(r_g))$ .

Similarly, now for the remaining cases, if both our terminals it is fine, but what are the remaining cases that we are having? So, if both nodes are your  $x_i$  node ok, so that means both are having same node  $x_i$  and we are going to perform operation over here. So, in that particular case the resultant BDDs whatever we are going to get, we will create an  $x_i$  node and this  $x_i$  may have either 0 and 1 0 and 1. So, in that particular case, it says that we are going to create an  $x_i$  node and with the dashed line with apply op lo of  $r_f$  and lo of  $r_g$ . So, this is the node  $r_f$  and  $r_g$  then and a solid line with apply op hi  $r_f$  hi of  $r_g$ . So, in this particular case with dashed line, we are going to get this particular node with solid line 1.

So, what is the functional value over here? It says that apply operation lo of  $r_s$  and lo of  $r_g$ . That means, this is basically low value and this is the lo value you just see what we are having over here and accordingly we are going to construct a node and this is high, high accordingly we are going to do these things.

So, what is low and high? Basically this is the true function low end and high end given a non-terminal nodes  $n$  in BDD, we define low of  $n$  to be the node pointed to the dashed line from  $n$ . Similarly, high end is the node pointed to the solid line from  $n$ . So, we are going to

have those particular low and high and accordingly, we are going to perform the operation with respect to these two nodes here, ok. Below this particular path, we will see this thing.

(Refer Slide Time: 56:33)

### 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  $apply(op, hi(r_f),$   
 $r_g)$ .

Now, similarly now another case what will happen we are having something that here we are having an  $x_i$  node which is called say  $r_f$  and I am having an  $x_j$  node in your  $r_j$  say this is your BDD  $B_g$  and say BDD  $B_f$ , where  $i > j$  because both of them are going to follow the same variable ordering. So, what does it mean? When we are coming to this particular point from starting, we are giving some  $x_i$  node and from starting we are getting some  $x_j$  node.

So, when we are coming to this particular point, what does it happen that this may not have since  $i$  is greater than  $j$  before that. That means we do not have any  $x_j$  node, ok. So, that means in this particular evaluation path, it may be independent of  $x_j$  node. So, that is why the  $x_i$  and  $x_j$  is coming together. So, if  $j$  is greater than  $i$ , then what will happen since it is independent of  $x_j$  because we are getting  $x_j$  along with  $x_i$ . So, the resultant function will depend on  $x_j$ .

So, we are going to create an  $x_j$  node and how we are going to create an  $x_j$  node for that dashed line. So, here I am having dashed line and solid line. We apply  $op$   $lo$  of  $r_f$  and  $r_g$  and for solid line  $op$   $hi$  of  $r_f$  and  $r_g$ . That means, since here we remain in this particular  $j$ th node only because we have to see how it is going to be  $f$  when I am going to get this particular  $x_i$   $x_j$  node in this particular BDD, ok. So, that means since it is independent of  $x_i$ , so we keep



or remain with  $x_j$  in this particular BDD, but we will make a progress in this  $x_j$  in the simply manner I can say like that. So, that is why  $lo$  of  $r_f$  and  $r_g$ ,  $hi$  of  $r_f$  and  $r_g$ . So, this is the scenario that we are having.

(Refer Slide Time: 59:07)

### Operation on OBDD

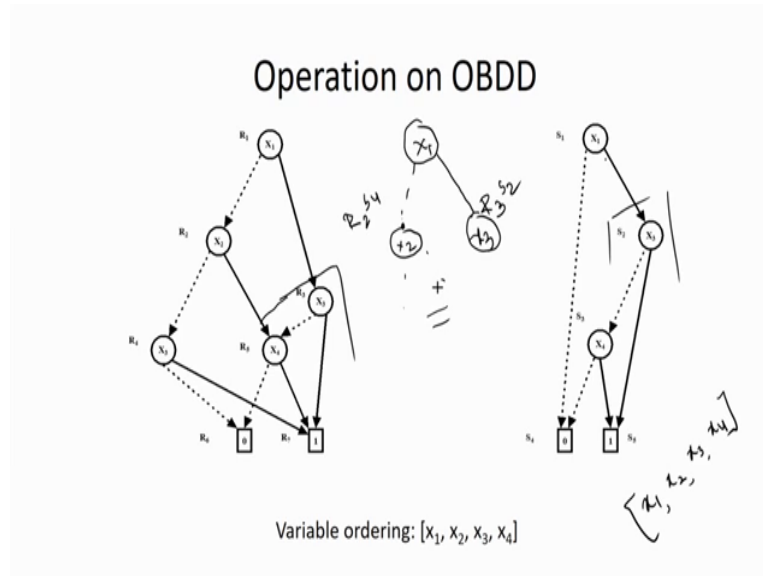
If  $r_g$  is an  $x_i$ -node, but  $r_f$  is a terminal node or an  $x_j$ -node with  $j > i$ ,  
 create an  $x_i$ -node  $n$  (called  $r_f, r_g$ ) with  
 a dashed line to  $apply(op, lo(r_g), r_f)$   
 and a solid line to  $apply(op, hi(r_g), r_f)$ .

This is the reverse of the previous one when I am saying that  $r_g$  is the  $x_i$  node and  $r_f$  is a terminal node or  $x_j$  node. It may be terminal node or it may be an  $x_j$  node also because terminal node will come for below down in the leaf nodes of that, ok.

So, whatever scenario we are having, this scenario is the reverse of the previous one. That means, one is given, then  $B_f$  is independent of  $x_i$ . So, here I am going to get  $f x_j$  node in  $B_f$  and  $x_i$  node in your  $B_c$ . So, this is the reverse of this thing. So, that is why here we are saying that will call it  $x_i$  node where it is  $lo$  of  $r_g$  and  $r_f$   $hi$  of  $r_g$  and  $r_f$ . That means,  $r_g$  is progressing, but  $r_f$  remains at that particular point. So, this situation, this condition is the reverse of the previous one.

So, if you see we might have taken care of all possible cases. One is if their terminal nodes, fine. Secondly, they are having the same  $x_i$  node and third one is they are different one is  $x_i$  node and other one may be  $x_j$  node or terminal node, ok. These are the possibilities that we are having when we are going to construct the BDDs while applying this particular apply method.

(Refer Slide Time: 60:39)



Now, this is the simple example. You can work, you may work with it. Just I am giving the idea say here we are having two BDDs say this is BDD 1 and BDD 2 and it is having variable  $x_1 \times x_2 \times x_3 \times x_4$  and say variable ordering also  $x_1 \times x_2 \times x_3 \times x_4$  because  $x_1$  is coming first, then  $x_3 \times x_4$  here  $x_1 \times x_2 \times x_3$ , then  $x_4$ . So, in this particular path we are following these things.

Now, both are having the compatible variable ordering both the functions are based on this variable  $x_1 \times x_2 \times x_3 \times x_4$ . So, now we can apply, our apply algorithm over here if I am going to perform this particular phase operation. Now, how we are going to proceed? It is a recursive call and we know the start from the root node. Now, both the nodes are your  $x$  node that mean while I am going to construct the BDD, we are going to create  $x_1$  node, ok.

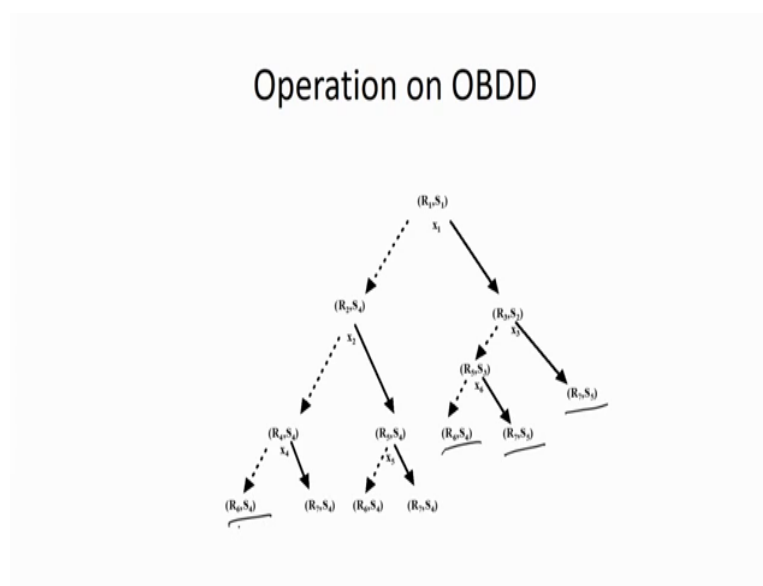
So, when I am going to create an  $x_1$  node, you just see that this is the scenario that we are going to have. That means, I am going to create  $x_1$  node, and here in the low side what I am going to get it is coming to  $x_2$  and if  $x_1$  is equal to 0, it is coming to  $x_4$ . That means, here I have to create a node. Now, I have to work with this particular node. It is nothing, but  $R_2 \ S_4$  and when it is your solid line, then what will happen both are coming to these things. So, what are the work I need to do over here  $R_3 \ S_2$ . Now this is the scenario that we are having. That means, now already I have taken decision.

Now, we have to what write over here. Now, what node I am going to create over here you just see this is  $x_2$  and second one is your terminal node. So,  $x_2$  is coming before that. So, I have to create  $x_2$  node over here because it is independent of  $x_2 \times x_3 \times x_4$  in this particular path.

So, we are going to create  $x_2$  node here and accordingly we will see what we can do actually and similarly, now when I am coming to these things  $R_3$  and  $S_2$ . So, both are  $x_3$  node, then we are going to create  $x_3$  node and apply the method for this particular path, ok.

Now, because from  $x_3$ , we are coming to this particular path so, here I am going to have the evaluation because already we have taken a decision of  $x$  equal to 1. So, this is the way we are going to construct. So, here  $x_2$  node and here  $x_3$  node.

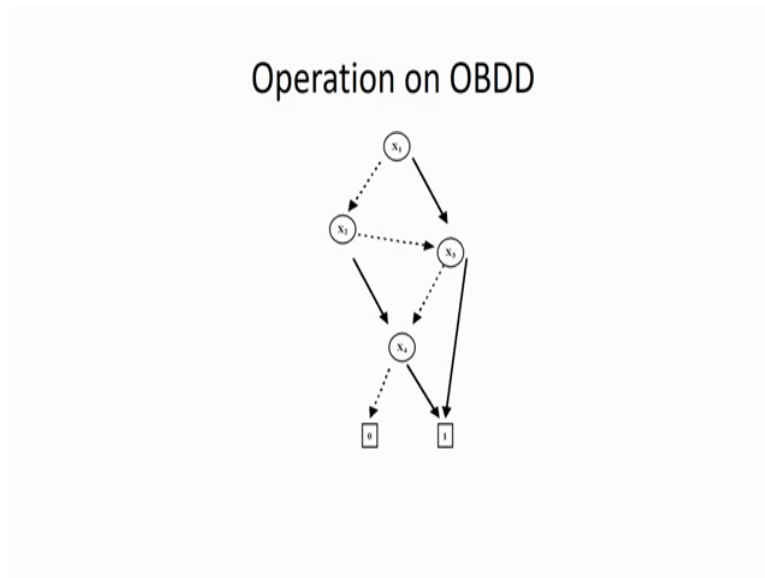
(Refer Slide Time: 63:45)



So, finally we are going to have these things  $R_2 R_4 R_2 R_3 S_2$ , then  $R_4 S_4$  like that we are going to have this is the structure and these are the nodes  $x_1 x_2 x_3 x_4$  like that.

Now, if you see these things, what about  $R_7$ ? All those things you see one example  $R_6 S_4 R_6 S_4$ , these are the terminal nodes. Now, I am going to apply plus. So, that means  $0$  plus  $0$ , it will be  $0$ . So, these are the terminal nodes. Basically now I can apply the operations. So, if I apply the operation, these are the result that I am going to get. So, now you see this is the resultant BDD of after application of this particular apply algorithm, but this BDD may not be a reduced one.

(Refer Slide Time: 64:45)



Now, after that we can apply the reduced algorithm to get the reduced BDD and finally, the reduced BDD is going to have this particular form. So, again it is going to have the same variable ordering  $x_1 \times 2 \times 3 \times 4$ , ok. So, here we can apply or we can perform any operation on BDDs and finally we are going to get an reduced BDDs or reduced order BDDs of those particular operation.

Now, this is the apply is the main algorithm and to get the reduced BDD, we are having all algorithm reduced algorithm. Here I am not going to discuss about it, but already I have mentioned about the rules how to reduce it. Now, we can look for the implementation of the reduced algorithm. So, now apply is the main algorithm where you can perform the Boolean operation on the BDDs.

(Refer Slide Time: 65:45)

### 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  $f$ .

$f = f[0/x]$   
 $f = f[1/x]$

Now, we are having some more x connected algorithm also which are going to help is one is known as your restrict algorithm which is basically the restrict operation which says that the Boolean formula obtained by repressing all occurrence of  $x$  by 0 is denoted by  $f x 0$ . That means, I am going to evaluate the function  $f$  just putting  $x$  equal to 0. Similarly, I can evaluate the function by putting  $x$  equal to 1. So, in that particular case we say that this is the restriction of  $f$ . That means, you are restricting the function or evaluation of the function is by restricting the values of  $x$  to be 0s only. We are please bothered about the functional value 1  $x$  become 1, then we say this is the restriction  $f x$  equal to 0. Similarly, we can have another restriction where we are going to put the values of  $x$  equal to 1.

So, now if I am having a Boolean function, I can replace the variable  $x$  by 0 and we are going to get the resultant function. So, no if we are already having BDD representation of the given function, now how we are going to get the BDD after applying this particular restrict operation. So, this is very simple.

(Refer Slide Time: 67:03)


### Operation on OBDD

$\downarrow \downarrow$   
 $restrict(0, x, B_f)$

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, what it says I can say that restrict 0 x B f. That means, it says that the variable x is restricted to 0. That means, we have least bothered about nor here we are going to say that x is equal to 0. We are not considering or we are not looking what will happen when x equal to 1.

So, in that particular case what we can say that for each node corresponding to x because for every variable x, we are having a node. We may have several nodes. So, we will remove and from BDD and redirect the incoming edges to low of n, ok.

So, if this x is here, then x may have. So, this is the n x may have an either 0 or either 1 and several as may come to this particular point, then what will happen to restrict it since now we are going to restrict n is equal to n. So, we are going to remove this particular part. It is redirect to low of n. So, after removing these things that means all those edges will go. So, whatever that in coming edges are there, that will be redirected to these particular point. So, these are the incoming edges. So, these incoming edges will be redirected to the low of n. Similarly for restrict 1 x B f also it is similar way except that redirection will go to the solid lines, ok.


So, we can do some simple manipulation in the BDDs to get restricts operation and there is another operation.

(Refer Slide Time: 68:57)

### Operation on OBDD

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

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



The diagram shows a function  $f(x, y, z, \dots)$  written in a curved, handwritten style. A circled 'x' is drawn below the function name. Two arrows originate from the circled 'x': one points downwards and to the left towards the text 'x=0', and the other points downwards and to the right towards the text 'x=1'.

Generally we used to perform generally we sometimes relax the constraint on some variable, ok. So, it is called relaxation. Basically if say that if we are some function  $f$  on some variable  $xyz$  like that sometimes we want to see that we want to relax the constraint on say variable  $x$ , we are not concerned about the values of varieties  $x$  equal to 0 or  $x$  equal to 1. So, this is the relaxation. We are relaxing this thing. So, we relax the constraint on some variable  $x$  of a Boolean function  $f$  then  $f$  could be made true by putting  $x$  to 0 or  $x$  to 1, ok.

So, basically what will happen either  $x$  equal to 0 or  $x$  equal to 1, we are not considering this thing. So, we are going to look both the evaluation and we are going to have this thing.


(Refer Slide Time: 69:57)

### 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, this is the expression basically we are going to have and we say there exist some  $x$  where we are going to relax the constraint. So, in that particular case the expression will turn up to be like that because in Shannon expansion how we are going to get  $\bar{x} f$   $x$  is replaced by 0 plus  $x f$  is like that  $x$  is replaced by 1.

Now, that constraint is  $q$  is mentioning either  $x$  equal to 0 or  $x$  equal to 1 because  $\bar{x}$  will become this is 1 into something and  $x$  equal to 1. This now we want to relax the constraint on this particular variable  $x$ . So, the function Shannon expansion only we are getting it. So, this is that if we are going to relax on the constraint of  $x$ , then the evolution of the Boolean expression will become these things  $f$  of  $x$  replaced by 0 plus  $f$  of  $x$  replaced by 1.

Now, say if we are having the BDD construction of this function  $f$ , how we are going to apply this particular relaxation and this relaxation can be applied repeatedly also.

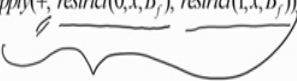


(Refer Slide Time: 71:11)

## Operation on OBDD

Algorithm exists

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

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


So, now in that particular case you just see that I know that here I am saying the evaluation of  $x$  equal to 0 and restrict evaluation of  $x$  equal to 1. So,  $x$  may be either 0 or 1, we are restricting it. So, we are getting OBDDs.

Now, we are going to apply this particular plus over here. So, this is basically what we are getting. Our result will be or resultant BDD is going to give us this particular expression. So, we can use this particular apply algorithm to get the relaxation on some variable  $x$  and for there exist some  $x$  have that means, we are going to relax the constraint on this particular variable  $x$ , then whatever resultant BDD we are going to get that can be constructed with the help of this particular apply operation. So, these are some basic operation that we will be needing while going to perform some operation on BDDs or in particular order BDDs or ROBDDs; reduced order BDDs.

(Refer Slide Time: 72:13)

**Operation on OBDD**

Algorithm exists

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

$\exists x_1 . \exists x_2 . \dots . \exists x_n . f$

This exist algorithm where the relaxation can be applied repeatedly up variable after variable, so finally we can apply these things first we can relax on x and some other variable like that. So, finally we are going to get one expression which will give you the relaxation or functional value of the given function on the relaxation of those particular variables.

Now, I think these are a basic fundamental see common to work with BDD and to understand about the principle of BDDs. So, what we have discussed basically we have seen the construction of BDDs, then we have talked about the reduced BDD, then we have talked about the ROBDDs Reduced Ordered Binary Decision Diagram and ROBDDs are going to have a canonical representation of a given function with respect to that ordering or with respect to the ordering that we are considering.

(Refer Slide Time: 73:19)

## Questions

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

Now, just some questions just see what I am saying. First question do we get any advantage of using BDD binary decision tree? We have explained something we have discussed about these things. You just see whether we are going to get any advantages out of it may not be because it is an expansion in nature. Secondly, to get ROBDD we need not to start from the BDD, we can start from the Shannon expansion itself.

While constructing the BDD, it is required to start from BDT. Already I have mentioned may not be required the definition of BDD does not restrict the occurrence of a variable in any number of times in a path, so that it may lead to consist inconsistency. With example I think we have discussed about these issues. Now, we can look into it is reduced BDD of any function is unique. If reduced BDD of any function is unique excess no because to get an uniqueness, we have to mention about the ordering of the variable also. If you change the ordering, we may get different structure already. We have mentioned one example.

(Refer Slide Time: 74:33)

### Shannon Expansion → BDD

$f = ac + bc$

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

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

Again you can see another example that same example that we are having motivating example that we are discussing over here. So, in that particular case, we have seen that if I am going to take function  $f$  is equal to  $a c$  plus  $b c$  and if we are using the variable ordering  $a b c$ , then we are having this particular structure ROBDDs for the given function  $f$  this the ordering is  $a b$  and  $c$ .

(Refer Slide Time: 75:07)

### Shannon Expansion → BDD

$f = ac + bc$

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

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

Now, if I change the ordering, then what will happen ok, you can go through it. Now, what is the ordering? First we are saying that we are going to first having an evaluation on  $b$  on

evaluation of b, I am going to get this function g and h, then on g and h, it will be evaluated that on c after having the evolution of c, then I am going to have the evaluation of a. That means, in this particular construction we are going to the variable or we are having the variable ordering b c a, ok.

So, now after that what will happen what I can get this, this now you just see that finally we are coming down to these things after using this particular Shannon expansion is there, can you have any scope of reduction over here? No, we do not have any redundant nodes at that particular point. We do not have any duplicate nodes also because duplicate nodes may be possible with respect to this is only because for variable a, we having only one node for variable b, we having only one node.

So, with this particular variable ordering, we have got this structure. So, you just see that structure is different when we have a different variable ordering. So, ROBDDs are not having a unique representation, but with respect to a variable ordering, we are going to have an unique representation. So, if you use this particular variable ordering, always you are going to get this particular structure only whatever way you are going constructing. So, ROBDD is having an unique representation, ok.

With this I will end up my lecture today. In next class, we are going to see some use of those BDDs or in particular ROBDDs with respect to model checking algorithm and finally, we will see the implementation of model checking algorithm where you are using the BDD representation to represent our system.

Thank you all.