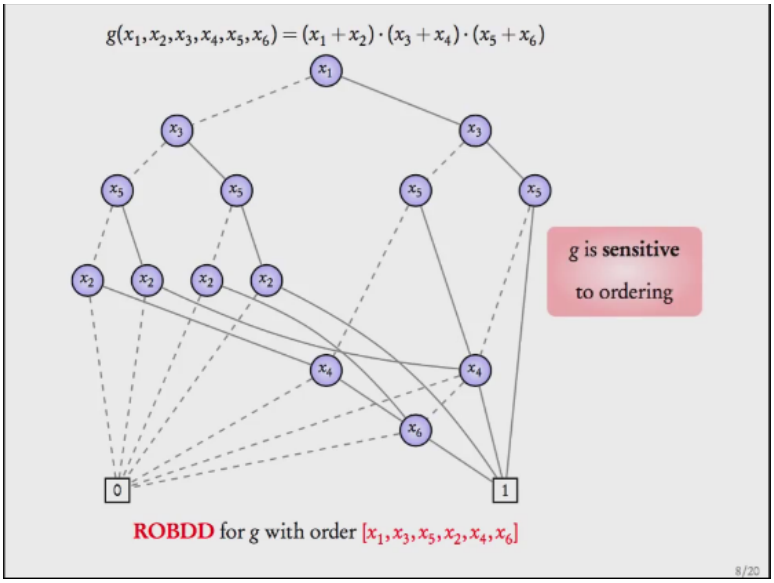
Model Checking
Prof. B. Srivathsan
Department of Computer Science and Engineering
Indian Institute of Technology - Madras

Lecture - 51
Ordered BDDs

We are in unit eleven of this course and this unit is on Binary Decision Diagrams which is data structure for representing Boolean functions. In the last module, we show an introduction to BDDs in this module we will be looking at what are called Ordered BDDs.

(Refer Slide Time: 0:31)



Let us have a look at this Boolean function which takes as input four variables x1, x2, x3, x4 and maps some valuation of these variables to one if an even number is one for example suppose x1 is one, x2 is zero, x3 one and x4 is zero. How many of them have been marked with one, x1 and x3 is one an even of them is one so f of one zero one zero will be one. That's what this definition says.

What about f of one, one, one, zero that will be zero because an odd number of these variables is one. What about f of one, one, one, one? Four of them are one so an even number of them is one so f or one, one, one, one will be one. Now, what about zero, zero, zero, zero none of them is one, still an even number of them is one. So, f of zero, zero, zero, zero will be one since zero is an even number. This is a BDD representing f this is not reduced.

Let me go through some paths. So, if you look at zero, zero, zero, zero is going to one. Zero, zero, zero, one goes to zero. Now, what about this one, zero, zero, zero goes to zero because I think one of them is one. One, zero, zero, one goes to one. So, you can check the other parts. So, this is more or less the True table for this Boolean function f. It represents every path. This is an ordered BDD because you start with x1 then you go to x2 then you talk about x3.

So, once you make a decision for x1 you make the decision for x2 then you make the decision for x3 and then you make the decision for x4 and this is the same in every path. So, this is an ordered BDD and the order is x1, x2, x3, x4. Now, this can be reduced if you look at this look at any x4 either zero child goes to zero and the one child goes to one or zero child goes to one and the one child goes to zero.

So, there are only two kind of trees starting I mean with root at x4 and once you do this reduction you can reduce the number of x3 and x2 and x1. Let us see first there are only two kinds of Trees starting at x4. So, this x4 has it is one child going to zero and it is zero child going to one. This x4 has it is one child going to one and zero child going to zero. These two are different so this x4 keep track of paths that lead's this node after having seen an even number of ones.

So, from here if you see one more one you have seen an odd number of one totally. So, you should go to zero. On the other hand, since you have seen an even numbers of ones if you see a zero again for x4 then you have to go to one. Similarly, this node keeps track of paths that you I mean that comes to this node after having seen an odd number of ones.

So, from here if you see one more one you have to go to one because marking x4 one they will number of one to be even. Similarly, if you see zero you will go to zero again. Similarly, this is x3. This x3 is marking the fact that the paths here have seen an even number of ones. For example, one, one or zero, zero there are an even number of ones. So, if you see one more one you should go to the x4 which sees an odd number of x.

If you see zero you should go to the x4 which has seen an even number of ones. So, if you

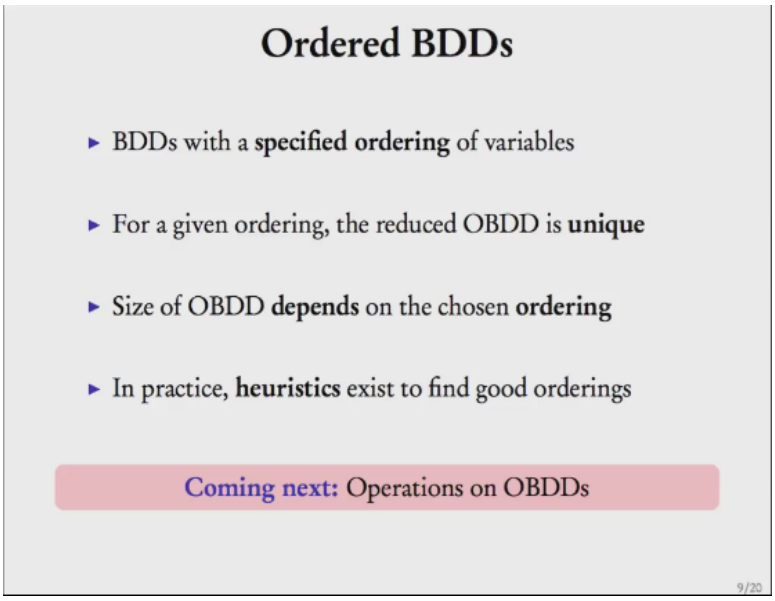
understand this BDD, this would be reduced and with the same order x1 x2 x3 x4 and this succinctly represents this BDD. So, this is the reduced ordered BDD for this ordered BDD. Sometimes we call it RO BDD. Let us now change the order instead of having x1 x2 x3 x4 let us start with the order x3 x1 x4 x2.

However, there is no change in the structure it looks the same. Similarly, when you reduce it the structure will look the same. So, here this x2 maintains parts that come to this so that it has seen an even number of ones this one is the x2 which has seen an odd number of ones. Similarly, this is the x4 which has seen an even number of one so far this is the x4 which seen odd numbers of one so far. So, this BDD is reduced, this is the RO BDD for the same Boolean function f.

So, no matter what order we choose the RO BDD has the same structure the same number of nodes and edges. So, this Boolean function is said to be not sensitive to the ordering that we choose for representing the BDD. However, consider the function g over six variables like this defined like this x1 + x2 dot x3 + x4 dot x5 + x6. Now if you choose this order x1, x2, x3, x4, x5, x6 and you construct the ROBDD it will look like this.

You can you can check it. However, if you a have choose a different ordering x1, x3, x5, x2, x4, x6 then the BBD the ROBBD reduced order BBD will have these many modes. You would not be able to reduce it further. This ordering give us a very succinct representation. However, with the ordering we get a huge ROBDD. So, this Boolean function g is sensitive to ordering.

(Refer Slide Time: 8:17)

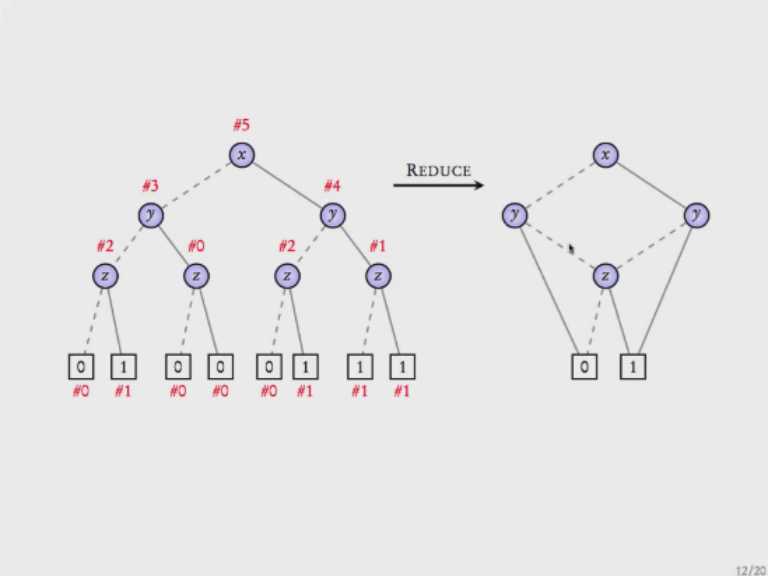


So, in summary ordered BDDs are BDDs that come with the specified ordering of the variables. And once you fix an ordering the reduced OBDD that you get will be unique after applying the reduction rules when you fix the ordering and apply the reduction rules you will come to a Canonical OBDD. However, the size of the OBDD for a Boolean function depends on a chosen ordering.

As you show, there is an example g depending on the order that we choose to get different sized OBDDs. The moral of the story is that the size of the OBDD depends on the chosen ordering and in practice there are some heuristics which will give us some good orderings. It will tell given a Boolean function it will tell us if you choose this ordering you will get a nice OBDD. There are some heuristics however in general finding the right ordering is a difficult problem.

Now, so we have seen what OBDDs are now let us try to come up with algorithm for OBDs. In particular, we will look at reduction algorithm and this union dot at compliment.

(Refer Slide Time: 9:53)



So, let us first start with a reduction algorithm. So, in the last module we show how to apply some rules repeatedly to get a reduced OBDD. But now we will give you a more structured algorithm for reduction of an OBDD. Suppose this is an OBDD, how do we reduce it? First step, for every zero leaves give an ID zero and for every one leave give an ID one. So, this is an ID that we are giving. Now look at the next higher node.

If it is zero child and one child have the same ID, then use the same ID for this. However, in this case both of them are different zero child has a different ID then it is one child then you have to give a new ID to this. Just follow the steps later on you will realize what this is leading to. Look at the next node if it is zero child and one child have the same ID then use the same ID here. However, that is not the case.

Now look if there some other node with the same label such that its left or rather if zero child and one child has the same IDs as this one. Look at the zero child it has ID zero. Look at this nodes zero child it has ID zero. Look at this nodes one child it has ID 1. Look at this nodes one child it has ID one. So, you use the same ID for this as well. Now, look this node. Does its zero child and one child have the same ID? No. So, you have to create a ID for this.

Look at this, both the zero child and the one child have the same IDs. So, use the same ID for this node. Finally, for this see it is zero child and one child. They do not have the same IDs. So,

create a new ID. Now what? For every ID have a node all these will reduce to one node. Take the first node with that ID that you show. And now four has a one edge to two. So, just mark it. For ID 3 take the first edge that you show with ID 3 it is only one.

Similarly, for ID 4 take the first node with the ID you show x1 and for zero and one just take one representative. Now, for this node it is zero child goes to hash zero in that it is one child goes to hash one. Now, for this node its zero child goes to the node with ID 2. Sorry, the one child goes to node with ID 2 will have the zero child goes to zero. Similar here if zero child goes to three and one child goes two –this will be the reduced OBD you can check it for yourself if you have applied the rules you would have got this one.

Let us see one more example. So, this is the reduced OBDD. Let us see one more example first step for all leaves give IDs for all zero leaves hash zero. For all one leaves give hash one. Now, go through every node step by step. If the left and right child have the same ID, then use the same ID otherwise create a new ID here you have to create a new ID. Now, look at this both of them have the same ID.

So just use hash zero, look at this before you think of creating a new ID check if there exists other node which has the same children. I mean same IDs. Yes, this one has the same ID which is then so use the same ID. This is just saying that this sub tree is the same as the sub tree. Now look at this both of them have the same IDs to use that ID 1. Now, so this will just tell you that this is useless decision point from here you can just go to one.

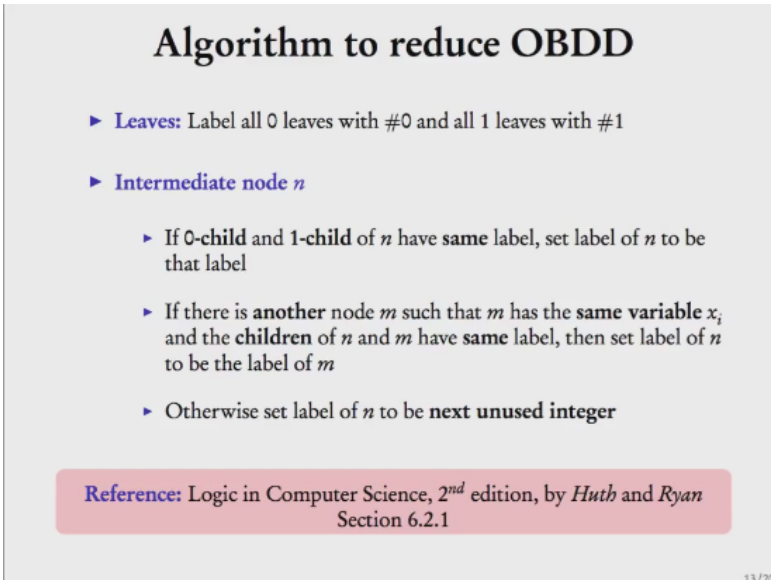
That is what this is more organized way of doing it now. Now, let us continue it. Y do they have the same ID, no. So, use that give a new ID 3, look at this do they have the same ID, no. Is there some other node with the same IDs for children? No. So, put hash board. Look at x do both the children have the same IDs, no. And there is no other x node so they give a new ID for this node. Now, for every ID just have one node which is the first node that you visited.

For zero, you have node for one you have one node. For two you take the first node you visited. Similarly, there is only one z because the zero is represented here. The one is represented here

and the two is represented here. The three is represented by this. Four by is this and five by this and draw the edges accordingly. Five point zero goes to three and one goes to four that is what this is.

Three on zero goes to two which is this one and on one goes to zero, which is this one. This four on zero goes to two. If this is two and on one goes to one this is one and two one zero goes to zero and one goes to one. So this was the same BDD that be used in module one and we got the same reduced BDD. These are reduced so these are also ordered. So these are reduced OBDs.

(Refer Slide Time: 16:59)



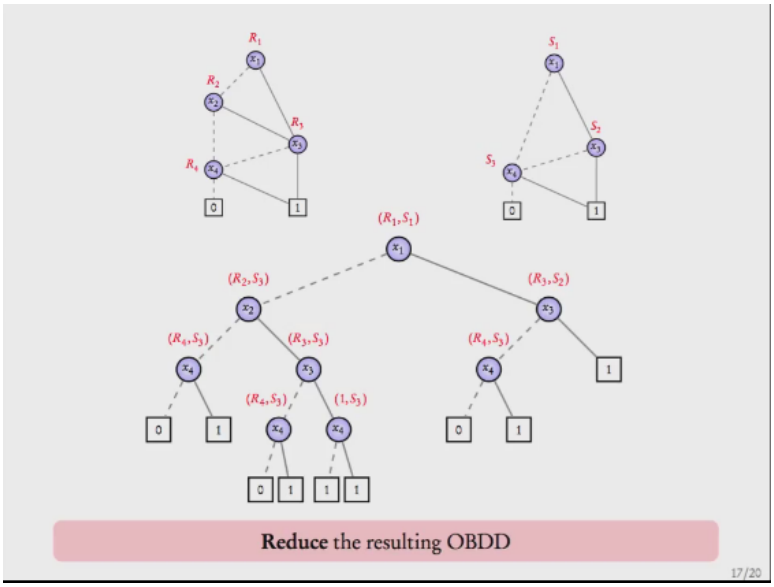
So, let me summarize the algorithm to reduce OBD given an OBDD what you do? First label all zero leaves with hash zero and all one leaves with hash one. For an intermediate node n, if both the zero child and one child of n have the same label, set the label of n to be that label. The label of the child. Otherwise, if there is another node n such that m has the same label xi and the children of n and m have the same label, then same ID that is what I meant then set the label of n to be the label of m.

So, here that what we do so there is another node such that both its children have the same labels. The labels of children of this z are the same. So use the same ID. Otherwise use the next unused integer. And this will give you a labeling and once you do that just take one node corresponding to each input, each ID. Here the label has been used for ID and this is explained in section 6.2.1

of the book that we are using as a reference for this.

So, what we have seen so far we have seen what OBDD are we have seen that Boolean function would be sensitive to the ordering that we choose and we have seen even an OBDD an efficient algorithm to reduce the OBDD that goes through every node only once. So starting from the bottom it goes stil the root labels everything and then does reduction by picking one node for every leaves. Hope this algorithm is clear.

(Refer Slide Time: 19:32)



Now, let us see an algorithm for doing the union of two OBDs. Suppose I give you two OBDs and I want you to add them that means each OBDD represent a Boolean function. Now I want to compute the OBDD for the Boolean function of just the union of these two. How do we efficiently do it? Let us start with just Binary Decision Trees or just Truth Tables. How did we do for truth tables? We looked at every path and then we just added them right.

So, let us do it step by step, choose node x. So, we are here and we are here when you move left here and you move left it means that you are choosing zero for x. When you move left here it means you are choosing zero for x in this group table, just put y on both. Now when you move right here put the node y so this is signifying the fact that you are moving right in both. Now keep doing this. Now you have made a choice for x and y.

Now, when you chose zero for z you see what happens in this path zero, zero, zero is zero here. Zero, zero, zero is zero here. So in the sum you should just do for this path zero, zero, zero you should just put zero. Now what about zero, zero, one. So, zero, zero, one goes to one here and zero, zero, one goes to zero here. So one plus zero will be one. Now what about this path zero, one, zero so zero, one, zero goes to zero here. Zero, one, zero goes to one here.

So zero plus one will be one. Similar, you can continue. So, finally we would get what we did with the truth table. These are two representations of truth tables and this is the representation of the truth table of the union of these two. So, we just added up the leaves and since both of them were of the similar structure we just had to look at corresponding leaves. Now what if one of them is a BDD, instead of do you think all the leaves and all the intermediate nodes be it the reduction.

We got the reduced OBD now what will do? Let us name these nodes for convenience. This one is R1, R2, R3, R4, R5, R6, R7. S1, S2, S3 so let us start with R1 S2 we are here we want to take corresponding valuations in both. Now take the left child you do a zero both of them go to y so go to R to come up S2. So far we have seen. Now, when you do the right side you go to y here however you go to z here what does that mean.

That means that since it is an ordered Binary decision guide them. It just means that the value of y is immaterial for this OBD. So no matter what we choose from this point only the value of y here matters. The value of y here does not matter. Let us see it in a more clear way. Let us continue look at this y R2 we are at R2 here and we are S2 here let us take the left child or rather the zero child.

You go to R4 here and what do you do here you go to zero from S2 on a zero you go to zero now what does this mean? This means that once the value of x is zero and the value of y of zero no matter what z is the Boolean function associates it is value to zero. So in this part no matter which path you choose you just have to add zero because the decision for z should be made only here. Here there is no need so let us continue what about y.

We are at R2 S2 what about one child when you take one you go to R5 here and one here. Similar, this says that in this decision points when you take y to be one the value of z is immaterial. So you have you are at R5 here only the decision here matters. Now, continue when you take the left child of R4 you go to zero now you just add zero here you know that when you do zero, zero, zero here you get to zero.

And when you do zero, zero, zero because since the value of z is immaterial if you do zero, zero, zero you get to zero just add zero and zero. See this thing is just way of maintaining this fact. Now what about the right child? You just look at the value here and add zero to it, one. Now from this node no matter what you take from R5 you just have to add one to it. One, one so this portion is done. Now let us continue from R3 we are here at S3.

Now this is y and this is z this says that no matter what you choose for y here it does not matter. So, how do we encode it here? You just keep S3 to be the same when you take the zero child you go to R6 here but you nesting S3 you make 10 you get result. Similar, when you take a one child in this component you go to R7 but here you stay in S3 it just says that till you reach z. I mean essentially this decision point does not matter for the second component.

Now, here you can take the left child from both because both of them are z when you take zero from the both of them it goes to zero. When you take one from both of them you have to go to one because one plus one is one. When you take zero from here from R7 and from S3 when you take zero you go to one and when you take one from R7 and one from S3 you go to one.

See essentially this procedure is just a way of adding up the leaves when this is given as the BDD and these things will encode the facts that y is unnecessary here from here z is unnecessary and so on. Let us see one more example suppose these two are BDDs and now I want to do a union of them. So, this represents a Boolean function this represents a Boolean function. Now, I want the ROBDD representing the Boolean function of the union of these two.

Let me start from R1 S1 both are x1 so let me take the left and right child from both of them. When you take the zero child so I will use the left and zero child interchangeably I hope you

understand. So, when you take the zero child from here and from here you go to R2 and S3 however S3 has S4 and R2 has S2. So, this is the ordering x1, x2, x3, x4 since x2 is smaller in the ordering just put x2 here.

This one just says that the x2 and x3 are immaterial when you choose the value of x1 be zero. Now you take the right child you go to R3 here and S2 here however both are S3 so it is fine. Now look at this you are at R2 and you are in S3 now R2 is x2 and x3 is x4 so just take left from here and keep this same.

Now, the left of this is R4 goes to x4 just move S3 down now when you take the right child you go to R3 with labelling x3 here because till x3 is less then x4 so keep the label to be x3. Now come to R3 and S3. Let us continue you are at R4, S3 here and both of them are x4 modes so you can take left child from both of them and zero plus zero will be zero. When you take right child from both of them one plus one would be one here.

Now, let us come to this R3 S3 still since S3 is x4 and R3 is x3 you have to just take the left from this node. So you take left you go to R4 just drop S3 now you have reached x4 in both places when you take the right child you go to one noted like this and S3 is still in S4 now. Now this says that when you take this decision x4 does not matter you directly go to one however if S3 isn't S4 so you have to mark x4 here.

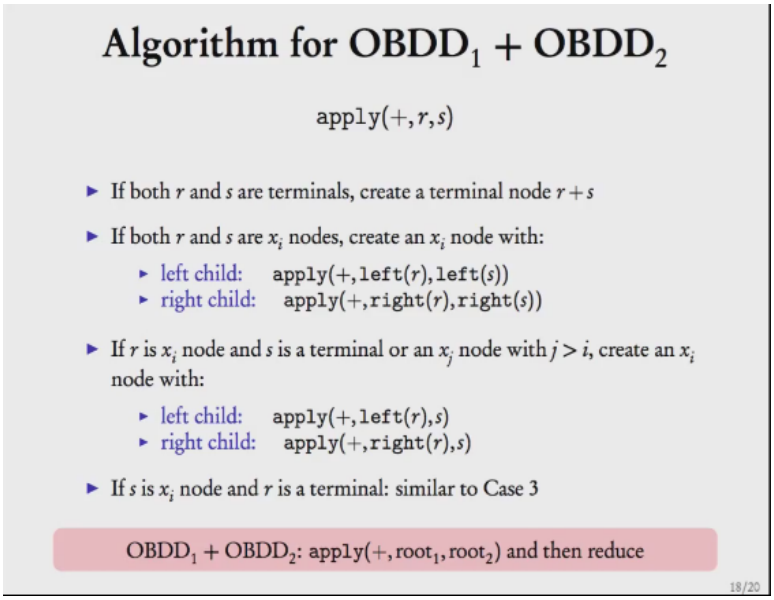
Now look at this node you are at R4, S3 when you take the left child finally even both of them are x4 you can take left child from both you get zero and zero both of them are x4 now. One and one would be one. Now look at this now you are at x4 and no matter what you choose we know that here on the first component it is going to just one. So, when you take the left child zero plus one will be one, one plus one would be one here.

Now, let us come here. Here we have remembered that this one is R3 and this one is S2 since both of them are in x3 you take children from both of them. So this ones left child goes to R4. This one left child goes to S3 and both of them are in S4 so it is fine. This ones right child goes to one and S2 right child goes to one as well. So one plus one would be one here. Now this is in

R4 S3 both of them have the same variable.

So you take the left child zero plus zero is zero and then right child one plus one is one. Now, let me write this algorithm in the next slide and this will become clear. Why the way? The OBDD that you get might not be reduced. So what you should do is once you get this OBDD you need to reduce the resulting OBDD using the reduction algorithm. So, let us now write down the steps for this union algorithm

(Refer Slide Time: 33:28)



So, let us now write down the steps for this union algorithm. Let call it apply the plus operation on nodes r and s. You start from some r and s so here r and s would be R1, S1 you start from the root nodes what do you see if both –so this is a generic algorithm. This is a generic function which takes two nodes to sub trees. Now if both r and s are terminals, just create a terminal node r + s.

If both r and s are xi modes create an xi mode with left child being apply on the left of r and left of s and the right child be apply of, you run the apply algorithm on right of r n, right of s. Let me say what it means in algorithm. Firstly, when you have terminals when you have apply of zero and zero you just take the sum. Suppose you take apply of R3 and S2 both of them have the same variable and what does the function do?

It creates a new variable x5 new node with variable x3 create a left child and the right child and what is this left child? You take the left of both so this becomes R4 S3 and here you took the right of both. So you did the apply on the left child of both of them and here you did the apply on the right child of both of the –here since both of them turned out to be terminals you just take the operation that is what this says.

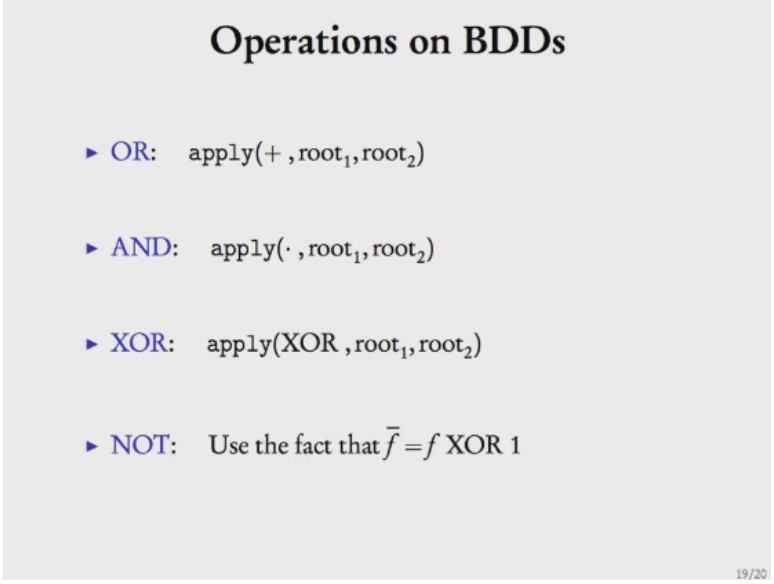
Now if r is in xi mode and x is a terminal or xj node with j bigger then i then create an xi node where you do the same thing however on the s component do not take any children. Just bring the s down. So how did we did that here. So look at his R2 S3.

R2 was an R2 node and x3 was an x4 node which is bigger than x2 so you created an x2 mode now on the left child you just took the left of the x2 node and the right of x2 node and just wrote x3 as it was. You did not take children of them. You just took children of the lower variable that is what this says. If R is xi and s is a terminal highest in some sense or an xj node with j bigger then i create an xi node with left child equal to you took the left child from the just the xi node.

And you kept the s node the same and the right child from the xi node and s node the same. Similarly, if s is an si node and r is a terminal or an xj node with j bigger then i. Do the same thing you just do apply of r, left of s that will be the left child and you take apply of r, right of s. So, this is symmetric to this case. So given an OBDD1 and OBDD2 you apply I mean you run this algorithm on root one which is the root of OBDD1 and root2 which is the root of OBDD2 you will then get an OBDD then you can reduce it.

The underline concept of this algorithm is simple finally we want to make sure that on every enter of the truth table I mean we just need to take corresponding increase in both the OBDDs and this procedure is a way of doing it methodically. Please go through that example once again and make sure that the idea is clear to you.

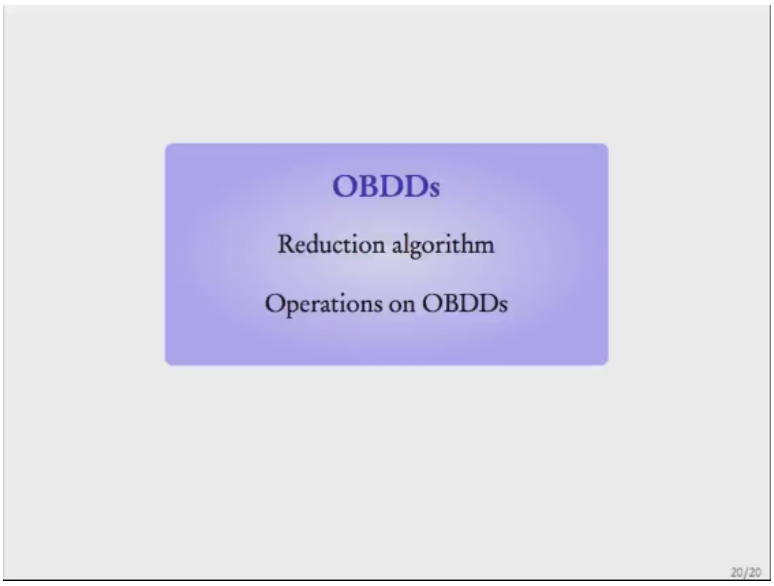
(Refer Slide Time: 38:15)



So, whatever we show was for plus however it will work the same way for dot. Just that in the terminal instead of doing plus we will do dot. Similarly, if you want to do an XOR instead of doing the dot we will do XOR. What about NOT, given an OBDD how do we get an OBDD corresponding to the NOT of the Boolean function. So, here we can use a nice fact.

This is left as an exercise to you check that any Boolean function the bar of any Boolean function is just f XOR 1. This is essentially what you do. You take the truth table of f and in every entry just do XOR. So if there was one, one XOR1 will be just zero if it was zero, zero XOR1 will be one. I have given you the answer. So bar is f XOR 1 and we apply algorithm. So, just by knowing the reduction algorithm and the apply algorithm we can perform all operations on OBDs.

(Refer Slide Time: 39:39)



So, that is the summary of this module. So for some reason we are interested in representing Boolean functions in an efficient way and for that we use OBDDs and after reducing them we get what are called ROBDDs for in this module we show an algorithm for reducing OBDDs and we also show an algorithm to do the union, dot and the node for OBDD. In the next module, we will see how we can represent transition systems using OBDDs.