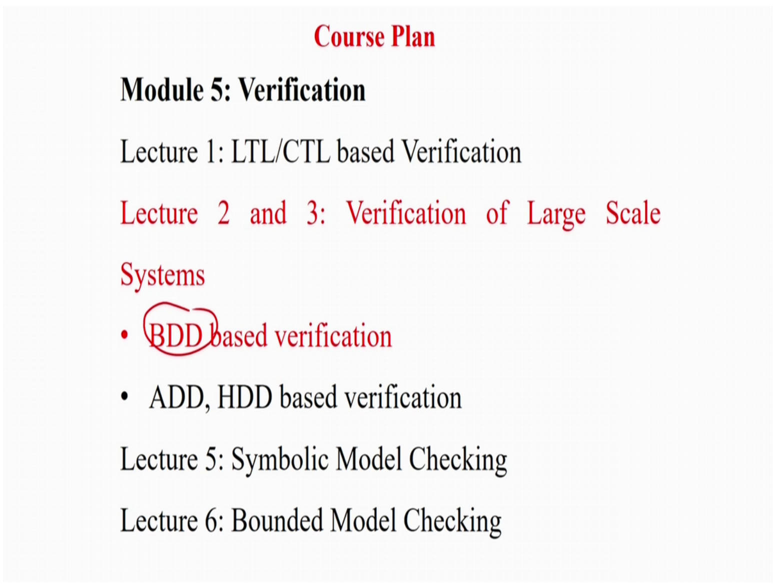
Optimization Techniques for Digital VLSI Design

Dr. Chandan Karfa
Dr. Santosh Biswas
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 20
Verification of Large Scale Systems

So, hello and welcome to the module we are discussing, that is on verification.

(Refer Slide Time: 00:31)



Course Plan

Module 5: Verification

Lecture 1: LTL/CTL based Verification

Lecture 2 and 3: Verification of Large Scale Systems

- BDD based verification
- ADD, HDD based verification

Lecture 5: Symbolic Model Checking

Lecture 6: Bounded Model Checking

So, in the last class basically, we had on, in lecture 1, we had basically discussed on very brief and we had tried to, I mean compact everything, regarding LTL and CTL based verification and mainly we emphasised how model checking is done during CTL. So, what was the basic idea we tried to means recapitulate in that class where we are were trying to basically have the means which was mainly required for the prerequisite that is to understand basic model checking before we can go to optimisation techniques for verification.

So, what we had seen in that lecture basically that we have to have a formal model that is a state based model for the system you want to verify and then we there will be some LTL or CTL formulas, which will be basically you design intent; that means, for example, if you have seen that if you switch on the heater or the microwave, after sometime, it should start heating. So, some of the basic design intense safety properties, etcetera with i d in terms of (Refer Time: 01:25) temporal logic formulas and then there are automatic model checking algorithms, which will verify mathematically that, whether the formula is valid in the model or not. So, if I mean it is a very high probability, almost it is mathematically guaranteed that you will get a answer yes, for the model checking, if both the models and the formula are correct.

So, in case, in case there is certain deviation rather you have written a wrong formula or the modelling or the system has an error, that is why the model is not capturing. What you want to implement in that is the model being or the system being implemented is not correct as per the design intent, then the model checker will tell you an error, also it will generate the states and the trace is where, which is leading to the error. So, that you can go and debug and find out the issues.

But then what are the main complexity we have found out? In fact, that lecture we have found it is quite mathematically, means a sound and it is mathematically guaranteed that if the formula and the model is correct then there is no error, all the states will be, will be model checked and it will be told that the formula holds in all the states, then you can be 100 percent sure that the model will work as per the specification then I mean the, I mean there is no need for any kind of simulation, because the design is correct by, correct by mathematical verification, but where the problem lies?.

The problem lies in the modelling itself, because if you have, when you analyse the complexity, if you recall, we found out that the main problem arises, because of the model size, because the complexity of model checking is linear. I mean it is, it is not exponential, it is, I mean much lower, compared to the size of the model that is, it is, it is basically depending on the, on the order of the size of the formula like that, whatever number of different atomic, I mean propositions or the different type of formulas, which you have to verify.

So, the, complexity depends on the, on the length of the formula, because for each of the formula, formula sub parts basically, like always in future something implies etcetera, etcetera. For each part of the formula, you have to label certain states and labelling certain states is nothing, but it is simply a graph traversal. So, that part is not very highly complex. So, where the complexity basically, arises is how you model it, because if they are five variables in the system, the number of states can be two to the power 5, because already we have seen that, the elaborate example of this, microwave controller and in which case we have only 4 or 5 variable and the number of states, we have some 6 or 7.

So, think about practical system, where the number of variables can be 20. There is order of state is 2 to the power 20. So, what actually kills, the whole idea of verification is the complexity of the model, because that is exponential in the number of variables and in the, in the most of the cases, a practical system will have 100s of variable. So, that is where the trucks of complexity comes in, where you require optimisation. Now, throughout this lecture, we will be trying to deal with those complexities right.

The next two lecture said, will be verification of large systems, where first we will see about binary decision diagram based verification and then we will see arithmetic decision diagram, high level decision diagram based verifications, in what case basically BDD is a binary decision diagram, which is nothing, but a very nice data structure. One of the data structure, most beautiful data structure, I have ever seen, which as revolutionised, the whole CAD industry that it has shown, how a simple binary tree can be compacted and all redundancy can be removed and we can get a very nice, data structure, which is very compact actually.

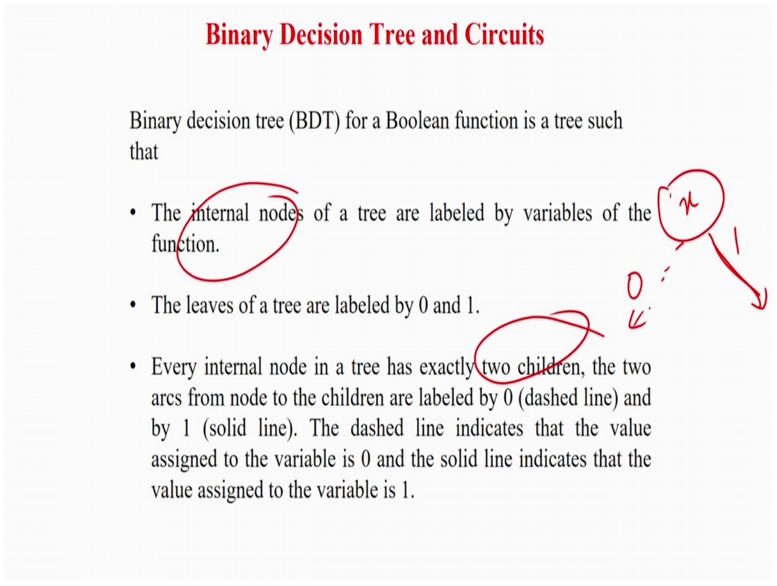
When I say, you should all have a feeling, whenever we say that the number of states is the order of 2 to the power 20, order 2 to the power 30, if they are 30 variable, it is not actually order of 2 to the power 30, it is order is 2 to the power 30, but the real number of states, there is much-much lower than that, because all combinations cannot happen. Because among all the variables, there are only very few meaningful combinations, which are actually required to model.

All other are very much redundant and can be eliminated, but where if you are using a very rudimentary banditry type of data structures or a simple finite state machine based structure and the, all states will be explicitly enumerated reading to the higher exponential complexities, making the modelling itself. So, complex that verification cannot go beyond a toy example. So, if you look at most of the literature, very few of them handle even a system, which is, which is as complex as a breaking system. It will be most of the works, basically try to deal with very small system, have been 10 to 20 variable about systems handling.

About 50 variables are very-very rare and you only find in very-very specific cases in the; if you look at the paper and the literature. So, compared to testing VLSI compared to testing verification is a more difficult problem in terms of complexity, because testing inherently is not exponential inherently. It is such kind of a searches, because we do not require explicit modelling of the, system in terms of states and transitions, but in case of verification, it is the other way round, always have to start with the modelling of the states and a state based, state transition diagram is required.

So, whenever we talk over state transition diagram or a graph exponentially, it actually kills us. So, what we will see called BDD, which will try to compact the representation and then we will see that B C BDD is again a binary representation can. We go for some abstraction like arithmetic or high level decision diagram and then we will move ahead.

(Refer Slide Time: 06:23)



Binary Decision Tree and Circuits

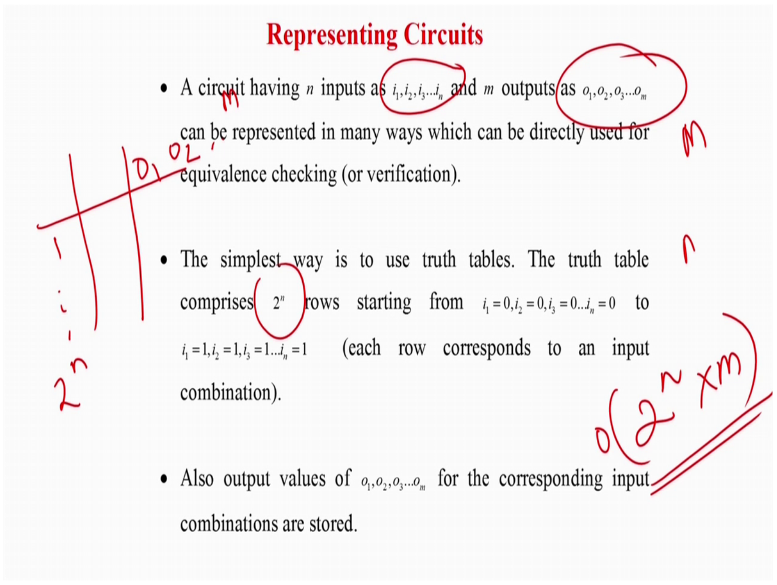
Binary decision tree (BDT) for a Boolean function is a tree such that

- The internal nodes of a tree are labeled by variables of the function.
- The leaves of a tree are labeled by 0 and 1.
- Every internal node in a tree has exactly two children, the two arcs from node to the children are labeled by 0 (dashed line) and by 1 (solid line). The dashed line indicates that the value assigned to the variable is 0 and the solid line indicates that the value assigned to the variable is 1.

So, basically we will start that throughout the, whenever from computer science is coming to picture, we know about the Boolean function binary arithmetic that 0s and 1s. So, basically whenever we want to represent anything you have, a tree is a binary decision. Tree is a very well data structure. So, the internal nodes are the variables of the function and then there are trees with leaf nodes 0s and 1s and each internal node has two child, the, I mean basically you all know that.

So, there will be a variable say x and this will be for 0 and this will be for the 1 variable. So, here we actually represent by the dash line. So, whenever will be representing the 0 for 0 variable, if x is equal to 0, we generally take the left x and if x equal to 1, we go the right x standard data structure, but in this course and most other CAD courses, you will find out that they actually make a dotted line for easy illustration and only problem is that actually, it will blow up that is the only thing.

(Refer Slide Time: 07:13)



Representing Circuits

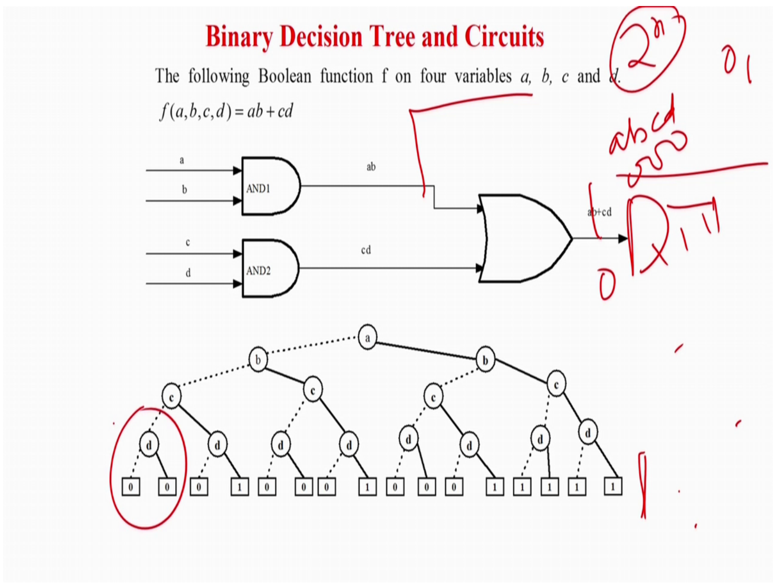
- A circuit having n inputs as $i_1, i_2, i_3 \ldots i_n$ and m outputs as $o_1, o_2, o_3 \ldots o_m$ can be represented in many ways which can be directly used for equivalence checking (or verification).
- The simplest way is to use truth tables. The truth table comprises 2^n rows starting from $i_1 = 0, i_2 = 0, i_3 = 0 \ldots i_n = 0$ to $i_1 = 1, i_2 = 1, i_3 = 1 \ldots i_n = 1$ (each row corresponds to an input combination).
- Also output values of $o_1, o_2, o_3 \ldots o_m$ for the corresponding input combinations are stored.

Otherwise, binary decision diagram, sorry binary decision tree can actually represent any kind of digital systems. Only problem is that the exponential blow up idea is very simple, you take the variables, left child 0 right child 1 and all the possibilities you enumerate and finally, the leaf node will have the values of 0s and 1s. If you take a path from the root node to the leaf node, you will find out that what is the value of the function, if those variable values are taken by the part traverse from the root node to the leaf nodes standard data structure algorithms.

Basically, how a, how do we represent circuits, generally most important one. We study in our digital design is a truth table. So, we have all inputs and basically, we have all outputs in a, in the part of the table that, all one side will be all inputs, and one side will be all outputs and basically, how many rows will be there in the truth table, if there are n variable, there will be 2 to the power n rows 0 0 0 0 0 1 dot dot dot up to 1 1 1, all the inputs will be explicitly enumerated and for each of the inputs, what will be the outputs? We are assuming m outputs. So, that also has to be enumerated. So, how? What will be size of the table, the size of the table will be easily equal to 2 to the power n for all the rows and into m, because, that many number of columns will be there.

So, is some-some order of like this, will be basically the size that is, we will have this many rows and basically, column also basically, I am saying, it should not be, n is the output, n is the input. So, basically your table would look like, something like this. So, you have 1 2 dot dot dot, 2 to the power n. These are all the values and this is O 1, this is O 2. All the outputs will be enumerated up to m outputs. So, generally they say that, they will be 2 to the power n rows, and 2 to the power, and n columns. So, basically that is, what is the size of a truth table. So, basically is nothing, but you explicitly enumerate all the inputs and outputs. So, that is the most fundamental and easiest way of representing a circuit.

(Refer Slide Time: 09:04)



Binary Decision Tree and Circuits

The following Boolean function f on four variables a, b, c and d.

$f(a,b,c,d) = ab + cd$

Then basically the truth table can be easily converted into a binary decision tree like, for example, this is an simple-simple circuit, if you can look at. So, the output function is a b plus c d. Let me just zoom it for you. So, this is a simple circuit a b plus c d and this is the function a b plus c d. Now, if I want to represent it in terms of a truth table. So, we will have a b c d. So, we will start from all 0s to all 1s, because only 1 output. So, it will be O 1 and you can write out the values over here, for all the rows explicitly.

So, this is actually the truth table representation. Simply, it can be represented also as a binary tree, a this was standard data structure. So, we will have the first variable that is a, the second will be b. So, all will be b, 3rd will be c, as you can see, this is 3rd, is the 3rd variable in the sequence is 3 and 4th is d and explicitly enumerate. You say that if a is equal to 0, I go this side, if a equal to 1, I go this side, then basically. So, if a equal to 0, I go this side, then if b equal to 0, I go this side, if d equal to 1, I go this side. Similarly, for this and similarly, for d basically, all the paths, you explicitly enumerate in terms of a tree standard.

Digital tree is there and you get the values like for example, if a equal to 0 b equal to 0 c equal to 0 d equal to 0, the output is basically equal to 0. So, you for that, you have to actually look at this part of the tree, if you can look at. So, a equal to 0 b equal to 0 c equal to 0 d equal to 0. So, this leaf node is marked with a 0. Similarly, you have to enumerate for all the cases, if you count 1 2 3 4, in the leaf node you will be 16 leaf. Leaf nodes for that basically, for, but if you have the values of 1 1 1 and 1 the output will be 1. So, 1 1 1 1, if I put the output will be 1.

So, basically 1 1 1 and 1, the output will be 1. So, for all the individual values, for all the these are actually, these leaf nodes are all the output. All the row correspond to all the rows of the truth table and if you take one path from the root node and if you come to any of the leaf node, this corresponds to 1 entry in the truth table. So, this is basically nothing, but a graphical representation of the basically your truth table.

So, very-very nice graph, you can do lot of reasoning and you know that if the number of inputs are n, the number of leaf nodes will be 2 to the power n and all the node in the non leaf nodes will be 2 to the power n minus 1 standard data structure. These are basically the orders. So, if you have a 100 inputs circuit. In fact, you may practically not, cannot even draw that tree, even if for a 15 input circuit, such trees cannot be drawn. So, that is where actually the complexity comes.

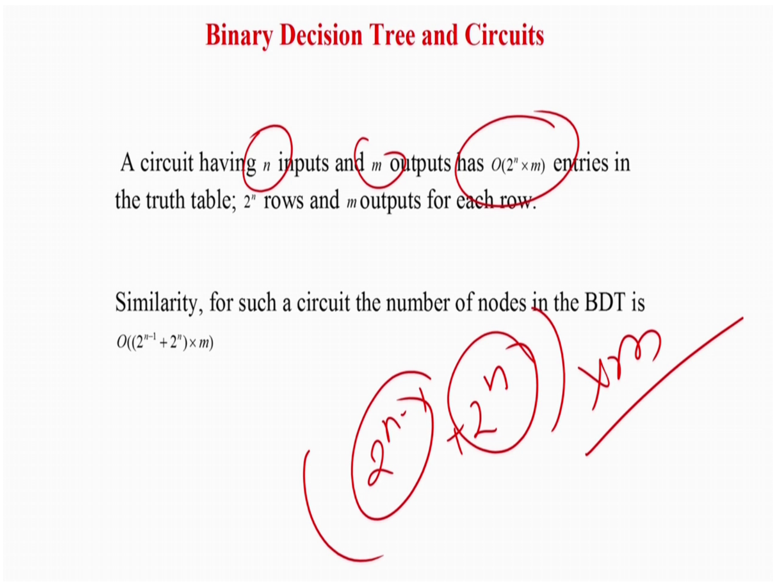
Now, why I am taking about all these, basically this, binary decision tree, etcetera. When you are talking of verification basically, whenever it is a system, basically there are flip flops and then we have state variables and state enumeration, which is mandatory for verification, but state I mean, if you, if you, if you can recollect your sequential circuit design fundamentals, we always have basically a combinational logic, which is the next state function block and then you have a flip flop. So, and then we have a flip flops.

So, we actually try to model the whole finite state machine in terms of a next state logic, that is your flops sorry, we have something like this. If you take something like, this is your basically, your flip flops, the register bank, then you have an input, this is your combinational cloud that is your next state logic block and there are feedbacks from here and also there are in primary inputs. So, this is basically your nothing, but your combination circuit. So, if you have some good data structures to model, this a combinational cloud modelling. The flip flops is nothing, but just enumerating the state values.

We will have an explicit lecture to see how actually binary decision diagrams can be used to represent models, state model, which is the most difficult part in verification. So, first-first you are trying to see how to model this N S F block, that is a combinational circuit using some good data structure. Once you can model this modelling, this state variables is trivial, because there is just flip flop, means just take, take the value from one state to the next state.

This just a state transition that modelling is not difficult, if you can very have a good data structure to model the N S F block, that is the next step, functional block, which are combinational circuit. So, that is why we are mainly dealing with combinational circuits here, where we are explaining binary decision diagram, because from, for the time being, we just take assume that if I can model the combinational circuit in a very nice and efficient manner, we can help the complexity, there then modelling of the flip flops will be trivial right.

(Refer Slide Time: 13:17)



Binary Decision Tree and Circuits

A circuit having n inputs and m outputs has $O(2^n \times m)$ entries in the truth table; 2^n rows and m outputs for each row.

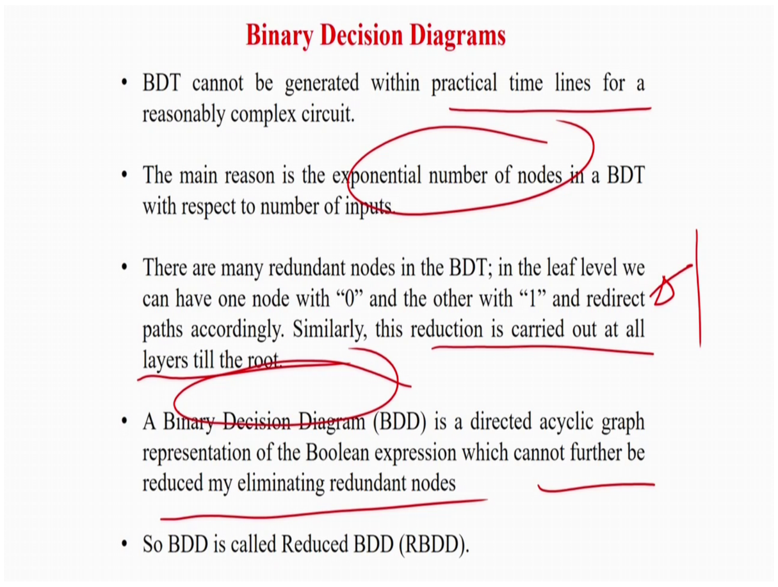
Similarity, for such a circuit the number of nodes in the BDT is $O((2^{n-1}+2^n)\times m)$

$(2^{n-1}+2^n)\times m$

So, the problem here is that, told you. So, that basically the n input rows and m output rows, the order is 2 to the power n into m, basically that is the truth table and for the binary decision diagrams, basically one thing you have to know, know that 1 1 binary tree, binary decision tree will be for 1 output, if there is another output here, may be then another gate will be there, another output will be there, as many outputs will be there, that many binary trees has to be there, because 1 binary tree represents 1 output.

So, basically, the order will be 2 to the power n minus 1 plus 2 to the power n that is the leaf nodes, non leaf nodes into m. Why m? Because there are n, explicit trees required for m outputs. So, huge complexity, so; obviously, nobody can even think about modelling a practical system in terms of a binary decision tree.

(Refer Slide Time: 17:04)

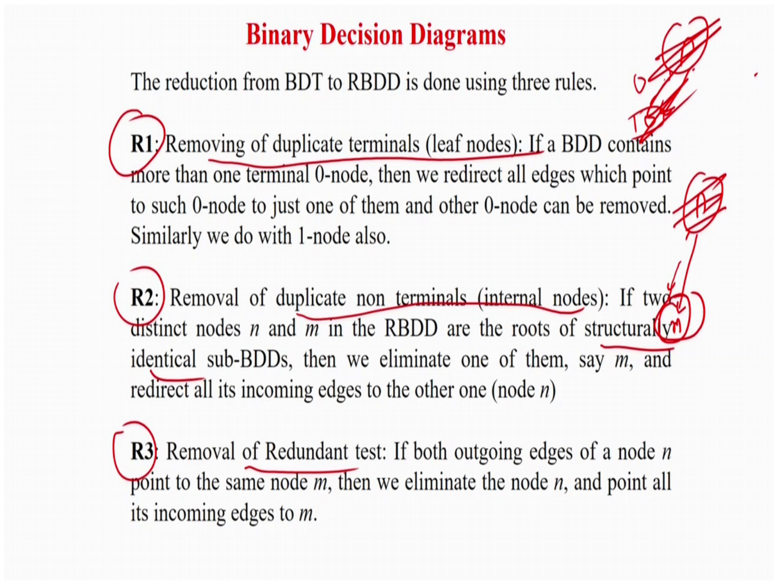


Binary Decision Diagrams

- BDT cannot be generated within practical time lines for a reasonably complex circuit.
- The main reason is the exponential number of nodes in a BDT with respect to number of inputs.
- There are many redundant nodes in the BDT; in the leaf level we can have one node with "0" and the other with "1" and redirect paths accordingly. Similarly, this reduction is carried out at all layers till the root.
- A Binary Decision Diagram (BDD) is a directed acyclic graph representation of the Boolean expression which cannot further be reduced my eliminating redundant nodes
- So BDD is called Reduced BDD (RBDD).

Basically, then something actually which revolutionised, it was by Bryant, R V Bryant. So, we are all thankful to professor Bryant for finding out such a interesting data structure, which actually revolutionised the whole CAD industry. So, binary decision tree cannot be generated in practical time lines for a reasonable complex circuit. Reasonably complex here means even if 20, it is very-very difficult, the main reason is the exponential number of nodes, in a binary decision tree. Now, what professor Bryant observed that, there are lot of redundancies, we have to start eliminating the redundancy.

(Refer Slide Time: 14:36)



Binary Decision Diagrams

The reduction from BDT to RBDD is done using three rules.

R1: Removing of duplicate terminals (leaf nodes): If a BDD contains more than one terminal 0-node, then we redirect all edges which point to such 0-node to just one of them and other 0-node can be removed. Similarly we do with 1-node also.

R2: Removal of duplicate non terminals (internal nodes): If two distinct nodes *n* and *m* in the RBDD are the roots of structurally identical sub-BDDs, then we eliminate one of them, say *m*, and redirect all its incoming edges to the other one (node *n*)

R3: Removal of Redundant test: If both outgoing edges of a node *n* point to the same node *m*, then we eliminate the node *n*, and point all its incoming edges to *m*.

If in, if a child can tell that, if you look at, even a child can tell you, where are the redundancies. So, even a class, I mean junior school student can tell you there are so many redundancies in the leaf node, because leaf nodes can have only two values or, or 0 or 1, but redundancy means there is 1 2 3 4 5 6. You count, there will be around, around say 50 percent of 0s and may be another 50 percent of 1s; that means, basically in this case, may be sometimes, we will have 25 percent, 1 or 25 percent, 0 or some numbers, but actually if you can find out that, very simple is that.

I can write only 1 0 and only 1 1, only 2 nodes. I will have and I will make a redirection for everything. This 1, I will put it over here, because straight forward I can say, there are lot of redundancy in the leaf node. So, I eliminate all, I just keep two of them and redirect the edges like that a 50 percent is 2 to the power n minus n that is the last node complexity will be gone, in just a click.

So, but very interestingly before professor Brant, nobody could think about it. People were just thinking about a complex handling, complexity in different manners were abstraction etcetera, but this practical feature, on this practical observation was the one of the key observations, which actually found out with leaf to the direction of the development of this data structure called binary decision, diagrams quite interesting, but just like as a, as you have seen as I told you like scan chains and many other fundamental, data structures of fundamental algorithms in the VLSI design test.

Verification are based on very-very simple observations, I in this case, there are so many redundancy in leaf node. Eliminate the redundancy, just put 1 0 and 1 node, because in case of, binary, whenever talking about binary system, there can be only 2 outputs 0 and 1. So, throw away all repetitive 0s, put only 1, 1 0 and 1 1 block and redirect your job is done, but then that is only one part of the story. Now, people have found out that, not only in the leaf node, in the non leaf nodes also, there are lot of redundancies. We have to try to eliminate them.

Now, that is what is basically, something called a binary decision diagram. So, this a 0 and 1 can be redirect accordingly. Similarly, this reduction is carried out at all layers, till the root, that is how what is binary decision diagram, you reduce the leaf node done. If you find similar type of redundancies, you go to the top and keep on doing it, we will take examples and find. So, binary decision diagram, the directed acyclic graph, Boolean expressions, which cannot be further reduced by redundant. So, redundancy means, if you can throw away that node and do some kind of redirection, the Boolean function does not, does not get changed.

So, therefore, sometimes BDD people better actually, call it reduced BDD, R BDD basically. So, basically BDD, wherever talk, you sometimes, it is a reduce time, is no longer used directly, but it means that the BDD is did not reduced form, because it cannot be further reduced. Whatever extra or redundant variables or I mean all this duplicate stuffs are there I have, all been eliminated then it becomes a binary decision diagram and surprisingly the compression or mean and as I told you, this is one case of optimisation, where there is no loss in quality; that means, you have a very big data structure, the very big tree.

You eliminate the redundancy and you get a smaller size tree that is the binary decision diagram or reduce binary decision diagram, but interestingly the function does not change, means for whatever sets of 0s and 1s for the inputs, which will give 1, the same case will happen in binary decision tree, as well as for the binary decision diagram, that is for any input. You take any input, your combination, you take both, the trees will give both the decision trees BDD or binary decision tree will give the same answer, that is you are not modifying the function only, we are redundant the, redundant the redundancies.

So, this is one very good idea, I mean fundamental of BDD is that it compresses, but still without any loss or that is without any compromise. It goes for an optimisation and surprisingly the reduction for most of the cases is 99 plus may be in a BDD binary decision tree. You have say 10 million or 1000 nodes, if you go for a binary decision diagram for most of the cases, it may come to around 10 or 20 nodes. Reduction is 99 plus, but some plus only for very few cases like a multiplier algorithm etcetera, there is no big reduction.

But those type of functions are very-very less in most of the general purpose circuits or main general purpose system, the reduction is drastic, that is why I always say that BDD is one of the best data structures I have ever seen, which has revolutionised the whole CAD industry, because all the algorithms now, written I mean, CAD is in terms of BDD after that, we had several advances like add decision diagram high level decision diagram, but the basic fundamental is this, remove the, all the redundancies and try to compress as much as possible.

So, basically three simple rules will revolutionised, one is called the removing R 1, which is called the removing of the duplicate leaf nodes, that is very simple. All, all 0s and 1s, you remove, put only 0 and only 1 and redirected the paths accordingly. R 2 is actually called removal of duplicate, non terminal internal nodes that is basically, away from, not from the leaf above, above the leaf, if there is some redundancies removed. How you do it, it? Any two distinct nodes m and n say, are the roots of structurally identical sub BDD's; that means, say this is 1 root and this is 1 root.

So, the BDD here and this say BDD here are identical, then basically say, may be there is some coming and from there is some other path, it is coming the BDD here and the BDD here are similar, then what you can do? Similarly, you can delete this n and redirect this R 2 m that is, what is the elimination of redundancy from the (Refer Time: 20:00) that is same. BDD is actually obtained from binary decision tree, redundancy, reducing and or eliminating redundancies. These node you do R 2 is that, if you find out some internal nodes or internal sub part of the tree is looking similar, remove one sub part and try to merge the edges such that only one sub common, sub part is written and the other is removed.

And basically finally, R 3 is removing of removal of redundant test. Basically, if both outgoing edges of node n point to the same node n, then remove, n is very simple like this is 1 node called node, called n and they both to point n that is, this one both 0 and 1. If the value of variable n is both 0 and 1 reading to the node called m then you can eliminate n; that means, n whether 0 or 1 is reading to the same child, then basically n is, n is not frequent in case of binary decision tree. What we will do? You will write n 0 sorry, for n equal to 0, maybe it is going to, this is some value and, basically may be 0 0 and n equal to 1, also may be going to same value.

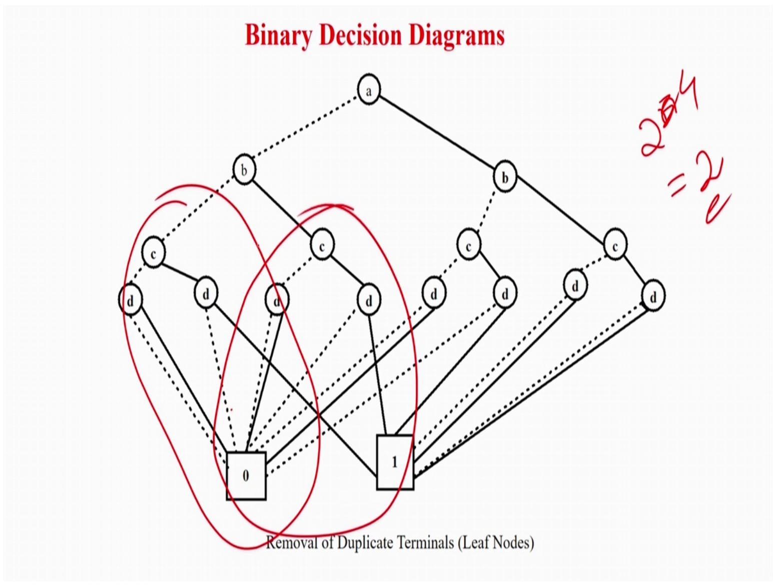
So, in binary decision tree, what you write you have to do in this way, n equal to 0 2 0 n equal to 1 2 1 assuming that basically, it is some kind of a leaf node. So, it is, you write in binary decision tree and, but in binary decision diagram it, if it says that it any node, if it, if it points out sorry. Here, what I was trying to say basically, in case of binary decision tree, if n equal to 1, you get 0, n equal to 0 equal to 0 and if n equal to 1 equal to also 0, then we write it in this way, but actually this is very-very much redundant. So, the rule tells that not only about the leaf node, at any point of time, if you find out that both 0 and 1 is leading to same node, because in case of binary decision diagram will have a single node, you have to just eliminate n and this.

So, it will be called. So, in binary tree, we do not think of any redundancy. We just keep on putting both the conditions left child right, child left, child right, child and it blow up, but here we apply the three rules leaf nodes. We all eliminate, keep only two redirect internally, we search that if there is two common sub BDD' s then retain one of them. Other one you eliminate, redirect the edges accordingly and third is basically, if you find any node, where both the edges outgoing left child and right child is going to the same, other successor node then removing the parent node and redirect accordingly.

So, Brunt had, there are lot of mathematical proves behind is; so, I am not going. So, they have proved that by using this three rules, did you get the minimal structure basically, we will give the reduced BDD sorry, reduced binary decision diagram and you cannot have any further reduction beyond that.

That means, that is the most compact representation of the function, you cannot have any more reduction by applying this three rules or this three rules are enough to, if you take a binary tree and if you apply this rules, ultimately you are going to find out the best solution for, from which you cannot further reduce. Therefore, it is actually called the, basically called the reduced binary decision diagram.

(Refer Slide Time: 22:55)



Binary Decision Diagrams

Removal of Duplicate Terminals (Leaf Nodes)

We will start taking an example. So, I am not going to both. Hence, and forth basically, this is, this tree, I means, I will just remove all this figure for you, anyway all the written nodes, for this course will also be uploaded in the websites. So, this is the binary tree. Now, I actually, I eliminate all this leaf nodes and I keep only two of them. So, if you look at it, I will get something like this, because I have only two nodes for 0 and 1. So, all the remove duplicate nodes have been removed and I get a path like this tree structure, like this. Only one edge we will study. So, it will be easier for you.

Let us try to concentrate on this path. So, c d all 0 leaves to 0 c equal to 0 d equal to 1 leaves to 0. So, basically we will see how it, it, it is getting reflected ok. The left part c 0 d 0 0 c 0 d 1 also 0 that is this part. So, we will see, how it is getting reflected in the binary decision diagram. So, if you look at it this part let us zoom. So, if you can see c equal to 0 d equal to 0 is 0 c equal to 0 d equal to 1 is also 0. So, in the binary tree, we basically had something like, this would not have been, there would have another node, which is 0. Basically, this is the dotted line. So, basically, we will say that c equal to 0 d equal to 0 is 0 c equal to 0 d equal to 1 is also equal to 0.

So, unnecessary, we had one replication. So, now, we have in this case of binary decision diagram by applying the rule 1. We will actually make both the, both the edges, point was single node, because we have only 1 0 and only 1 1. So, now, you can easily look at this tree and all the, you just, draw this, eliminate basically, eliminate all the root nodes. I have only 2 nodes and redirect the edges. I have shown, for one example this tree will be obtained. So, straightway first reduction is coming down from 2 to the power n to 2.

Basically, the leaf node, there was 2 to the power n child, in this case it was 1 2 3 4 that is basically, it was 4 16 from 16 to 2. So, the root nodes, number of root nodes of a binary decision diagram will always the constant, that is equal to 2 or less. Sometimes, it can be also be less that, we are going to see, because if the function always gives 0 or always gives 1, there will be a single node, but it can never be higher than 2, that is what is the beauty.

Now, interesting parts, we are going to do that is R 1. Now, as I told you, we will try to find out common sub graphs in this. So, if you look at the, let us try to look at this sub graph. So, what the sub graph is saying? Let me try to zoom this sub graph. So, if you look at this sub graph, it says that c equal to 0 d equal to 0 sorry, c equal to 0 d equal to 0 is 0 c equal to 0 d equal to 0 is this 1 c equal to 1 d equal to 0 is 0 c equal to 1 d equal to 1 is 1.

So, this is 1 graph. Let us now look at this sub graph. So, another sub graph basically, if you consider, let me eliminate this and take the other sub graph. So, just let me take this sub graph ok, and you see again c equal to 0 d equal to 0 is 0 c equal to 0 d equal to 1 is this c equal to 1 d equal to 0 is 0 and this. So, if you look at these two sub graph exactly look. Similar, c 0, this edge. This edge, this d node and this d node is similar.

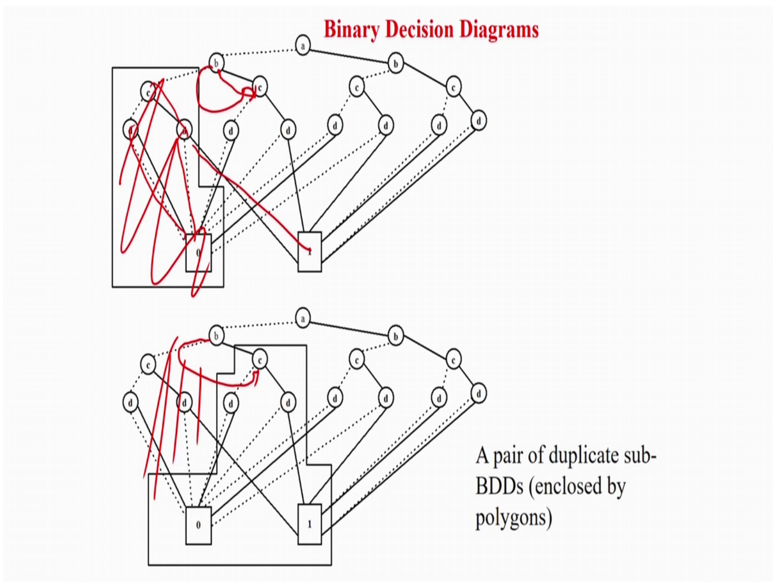
Similarly, this d node and this d node is similar, because 0 means 0 here also 0 means 0 here, also 1 means 1. So, these are the 2, there will be more I am just taking, because I think this part also will anyway. So, sub graphs, which are common, this 1 2 sub graphs or sub BDD, which are common in, which are redundant, because they are exactly common. So, basically one of them can be easily eliminated. This part can be eliminated of course, the leaf node, you have to keep basically.

So, these part I think can be very easily dropped out, because they are very-very similar, we just keep one of them and readjust accordingly; that means, if you are eliminate this, this b only has to be design it over your job is basically, done. Why are you want to keep redundant ? Similarly, if you look at this part CDD, this part, this part of the tree, if you take anyway, let me eliminate the other part. So, you can see, in these two part. Let me study ok. So, if you see c, if you look at sorry, not this. This one, on this part, if you see.

So, this is c equal to 0 d equal to 0. So, c d same path c equal to 1 then d. So, no, no, no this is not common basically.

Because in this case, this two nodes are also not common, because in this case both (Refer Time: 27:32) to one here, the d is different. So, this two are not sub common, but basically you have to find out all such kind of common sub graphs in the tree and eliminating them. So, basically these two are not common, but the because, this d and this d nodes are not equivalent. So, therefore, this two are not common, but this sub graph and this sub graphs are sub BDDs are common.

(Refer Slide Time: 27:53)



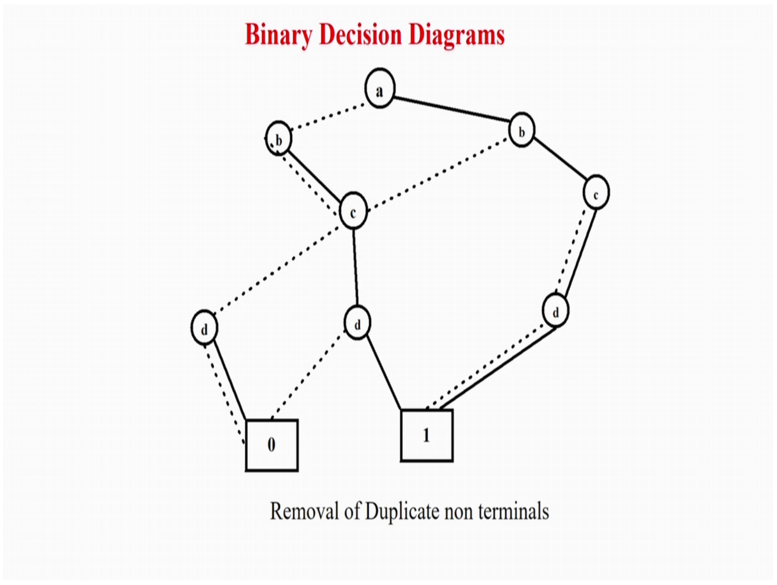
Binary Decision Diagrams

A pair of duplicate sub-BDDs (enclosed by polygons)

So, basically what is being showed in this, a pair of duplicate sub BDD's enclosed by the polygons, this one and I will just sub, I will just zoom it for you, big view. Now, this part and this part are basically common similar. So, what we do? We eliminate it. So, if we eliminate, what is going to happen? There will be only 1 such sub graph and basically, this b will start pointing to c, if you look at it. So, this is what has happened. So, this is your basically, the graph.

So, this is your graph sorry. So, basically this is your graph c equal to 0 d equal to 0 and 1 similar, c equal to 1 d equal to 0 0 and 1, if you look at it, this is, this graph. So, this is the path and in this case c d c equal to 1 d equal to 0 is 0 d equal to 1 is equal to 1. So, basically this is the path, I was talking about and this path. So, this path is what is remaining and as I told you, you have to also add just, because if one of them gets eliminated, the left child of b will also have to start pointing to this edge. So, basically this is gone, d will, this part of d will also start pointing, we will see.

(Refer Slide Time: 29:02)



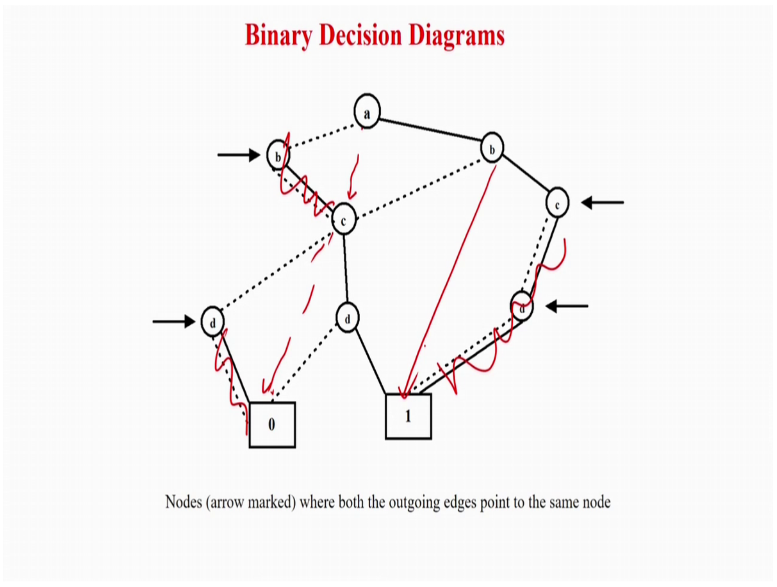
Binary Decision Diagrams

Removal of Duplicate non terminals

So, basically this what has happened, this is your common, sorry. So, basically this is the graph, which is one part. We have retained the other part, we have deleted and this, this part of the b that is dotted line, which as pointing to the sub graphs, that is eliminated will now start pointing over here.

Similarly, you have to, if you search the whole graph, there are lot of such redundant trees and you will find out something from intermediate binary decision tree, which has no redundant sub parts. So, I am not going for all the steps very-very easily. You can try to look out, in this graph, find out the redundant one and is eliminated, basically you can find out. May be there is some kind of duplicate c's over here. So, anyway that you can study and find out.

(Refer Slide Time: 29:44)



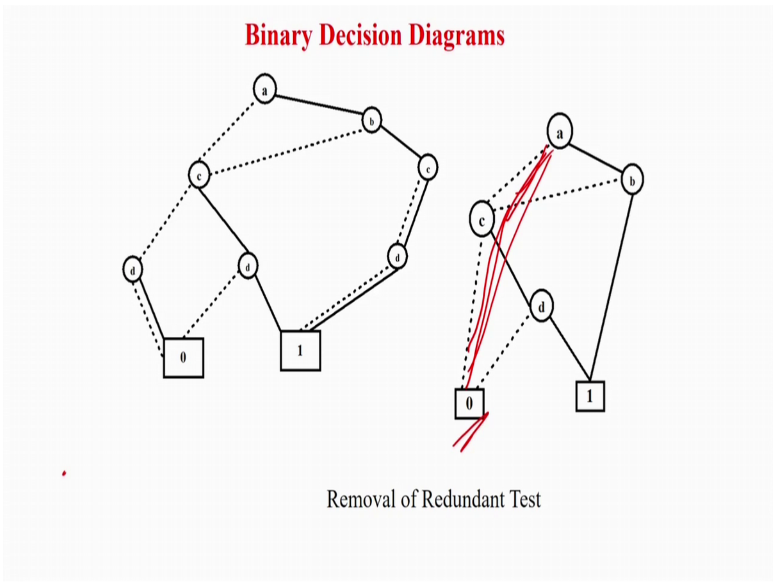
Binary Decision Diagrams

Nodes (arrow marked) where both the outgoing edges point to the same node

Now, basically what now, another rules you have to write like for example, very simple, you can see that these nodes are basically, not required, very trivial. You can see, because both the left edge and right edge are going to same thing; that means, it is a do not care basically, in terms of digital, they are do not cares basically b equal to 0, c equal to b equal to 0 b equal to 1 basically, we are going to same c. In fact, I do not require this at all I can directly bring my a dotted line here.

Similarly, this d is also not required, you can actually directly bring this down over here, this can be eliminated. Similarly, this d and this whole stuff is not required, you can directly bring your, bring down this one over here, because, this nodes are not required, because whenever you have some all both 0s and 1 leading to this. So, they will be it be same thing. So, basically we will have something like this, this part will be eliminated, this part is also not required, you will bring it to here. This whole part is not required directly, you can bring it up to here.

(Refer Slide Time: 30:35)



Binary Decision Diagrams

Removal of Redundant Test

So, it will actually somehow lead to some graph like this, which is the most reduced form, you cannot for that given function, you cannot have any further reduction then this. So, how many from, 16. There is 2 to the power 4 2 to the power 4 plus 2 to the power 3. Basically, 8 plus 16, at some basic around 30 nodes. We have brought it down to 1 2 3 4 5 6. So, from 6 from the order of 30, so, one fifth reduction.

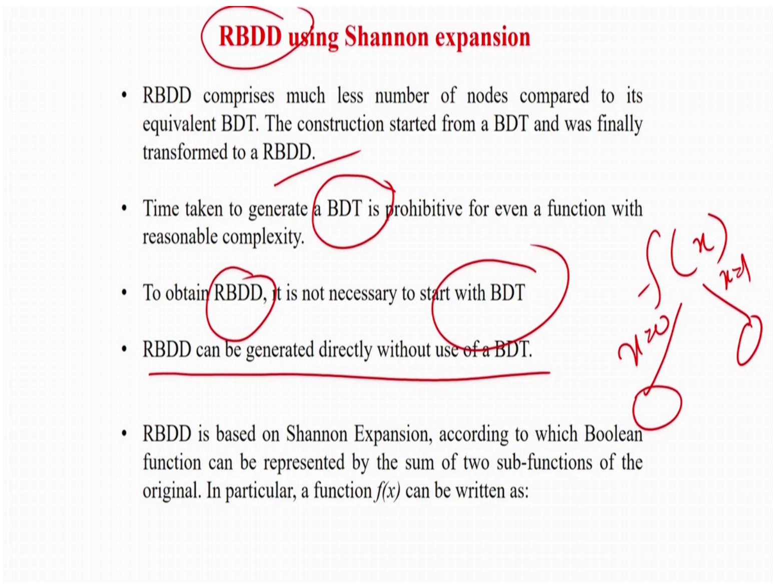
Basically, this is, this ratio will actually again blow up means, this size will start exponentially rising and this rise is basically linear, if you are going for a (Refer Time: 31:16) functions like, if you have some functions like 100 variable. So, this will be in the order of 2 to the power 100, but this 1 will not increase in that order for general most of the functions. It will be around some 300, 400 or may be 1000 or 10000 nodes. So, basically the size of the BDD on the number of inputs do not blow up exponentially first.

Most of the functions or most of the cases, but the denominator, which is the binary tree or exponential function actually blows up therefore, the ratio of compaction is very-very high, when you are using a binary decision diagram as the example shows and the most beauty of it is that, this function and the original BDD functions are not changed. So, whenever like 1 1 example, we had 7, seen that if a equal to b 0 b equal to 0 c equal to 0 d equal to 0, you are, what you getting the value of 0.

In this case if a equal to 0 c equal to 0 is 0; that means, d equal d equal to 0 and b equal to 0 are redundant variables; that means, they are basically do not cares. So, very quickly, you are going to get the solution and that also without any error like we already see that a equal to b equal to c equal to d equal to all 0, the answer was a 0. So, just have to traverse, the BDD a equal to 0 b is redundant. So, you will not find b over here, c is 0, d is not required, you find the answer 0.

So, it is correct basically, BDD's are also one thing, if there some redundant variables in paths, they will all be eliminated, but b is not redundant in this path. So, it is there, but if some of the variables are totally redundant, they are not required in the function, which can be minimised, which we do in terms of, Karnaugh map etcetera. BDD will totally eliminate those variables from the, you will not find immediately and from, from the path, this path example I have shown, shown you that a equal to 0 b equal to 0 c 0 d 0 like this one, like this example.

(Refer Slide Time: 32:59)



RBDD using Shannon expansion

- RBDD comprises much less number of nodes compared to its equivalent BDT. The construction started from a BDT and was finally transformed to a RBDD.
- Time taken to generate a BDT is prohibitive for even a function with reasonable complexity.
- To obtain RBDD, it is not necessary to start with BDT
- RBDD can be generated directly without use of a BDT.
- RBDD is based on Shannon Expansion, according to which Boolean function can be represented by the sum of two sub-functions of the original. In particular, a function *f(x)* can be written as:

Now, this path like a a b c d this path actually, in this path, a and b and d are redundant. So, you can directly have this. So, what is saying is that whatever output the binary tree will give you, binary decision tree will give the same input, will be given by the binary decision diagram. So, they are basically equivalent, but this is very compress version of the binary decision tree and the compression happens, because we are eliminating out all the redundant node.

First, you are eliminating all the leaf redundancies, then you are eliminating all the intermediate redundancies, then you are finding out some nodes, where basically both 0s and 1s are leading to the same child and we are trying to eliminate out those nodes and redirecting the transitions accordingly. So, we are getting a very-very compile structure, that is what is the beauty of binary decision diagram. So, what it means that, if you have a very big circuit in terms of 100s of inputs, the binary decision diagram will be much-much smaller than the binary tree and this binary diagram will be binary decision diagram will be used to represent.

Basically, your state machines and all the model checking, we have done in the state machine labelling, will all be done in terms of binary decision diagrams only. So, the modelling structure will drastically change from explicit states to BDD based representation and the model checking of state labelling etcetera, will be done, not only on the state explicit, state machine. It will be done symbolically on the BDD. So, that will actually improve the scalability or the complexity are drastic fashion. So, what is the step 1 on one side? We have a state machine, which is then, we have some temporal formulas like CTL LTL would be a model checking on the graph, that is actually a very complex problem.

If the, the number of state variables are large, other side is that we will not even go for the state modelling, we will directly represent the state variables or the finite state machine in terms of BDDs and as I, as already shown in this case, it will be extremely compact representation of the whole state enumeration, then you can have very big circuits or very big systems represented symbolically in terms of BDD and you did not explicitly represented then in terms of the state based modelling, then what you do then, all the model checking algorithms like labelling etcetera, will not be done on the explicit state. State based machine, but it will be done on the BDDs resulting in more efficiency in those algorithm and very quickly, you are going to get out the solution.

So, using of BDDs will help for to verify very large systems by representing them symbolically, the term symbolically is used, where you are using verification, everything or representation in terms of decision diagram, binary decision diagrams. Now, one important thing, we are going to study that is construction of BDDs. So, now, what we have seen in the last example that we have a binary tree and then you are making into a BDD. So, very you should you come to your mind that if you have the binary decision diagram or binary decision tree then we are using lot of elimination etcetera, to do it, but the binary decision tree itself is not available for a very large circuit, then how can you make the binary decision diagram?.

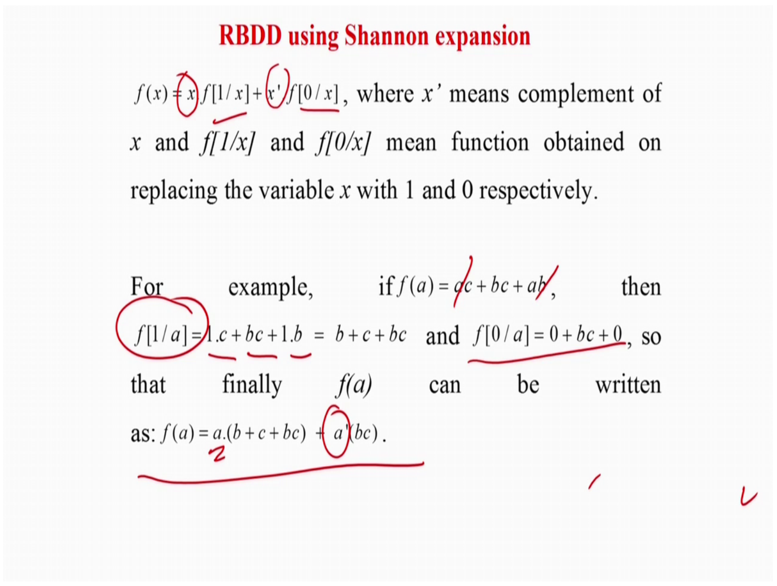
Because you have a very big structure, you are reducing and you are making a binary decision diagram, but if the, how to make that big tree itself, that is the very big question, the idea is that nobody actually does in that fashion. These was shown to explain you the difference between a BDD and a binary decision tree. Basically, people try to bill binary decision diagram from the scratch, that is they will have some small, small tree. They will merge them and try to make up a bigger binary decision tree, because that is impossible that you have a binary decision diagram very big one and you start reducing and making at it that is not possible. Binary decision diagrams are basically built up from atomic gates and they will make the larger binary decision diagrams that we are going to show.

Basically, when brand etcetera, tried to compress at that time, there was no decision diagram. They were looking at binary decision tree and they were trying to find out that, there can be a data structure, which will compress it in this manner. So, therefore, this was a thought process and to illustrate the beauty of binary decision diagram, but practically, when you are going to construct, nobody can construct it in the fashion, which you have seen that you take a binary decision tree and then you reduce and make it, because binary tree itself is not available.

So, what actually we do? We actually construct reduced BDD with the something called a Shannon’s expression. So, as I have told you the binary decision tree is a very small structure time taking to make a BDD is prohibitive. So, to obtain BDD, we need not start with binary decision tree. The need not is a good word, we cannot start with a binary decision tree. R BDD’s can be directly generated without a binary decision tree basically, for that we require something called Shannon’s expression, in which case, whenever you have a function called f of x we will have two paths; one x is equal to 0, then x equal to 1, then this will be 1 BDD, then again we we will keep on doing it.

We will get the finally, binary decision diagram and at every stage of the construction, you have to keep on reducing, you cannot have the whole exponential growth and reduce that is not possible, you take a function, make a binary decision diagram or make a binary decision tree, rather eliminate of redundancies, make it the binary decision diagram and then go for the next level of combinations, which is actually, that is going the binary decision diagrams on the leaf levels.

(Refer Slide Time: 37:56)



RBDD using Shannon expansion

$f(x) = x.f[1/x] + x'.f[0/x]$, where x' means complement of x and *f[1/x]* and *f[0/x]* mean function obtained on replacing the variable x with 1 and 0 respectively.

For example, if $f(a) = ac + bc + ab$, then $f[1/a] = 1.c + bc + 1.b = b + c + bc$ and $f[0/a] = 0 + bc + 0$, so that finally *f(a)* can be written as: $f(a) = a.(b + c + bc) + a'(bc)$.

For example, what we do, if we have a function called f of x? Generally, Shannon expression means, we have one path in which f of x equal x dot x equal to 1; that means, if you have a function which, which we first take out, 1 variable put the replace that variable by 1 and make a sub function inside, then another part is x equal to 0 and the other part is replaced by value of x equal to 0. We gave, get another function and we keep on doing it basically, easy to, this is actually mathematical representation, very easy to explain by an example like, in this function a b plus b c plus a b. So, what we do first, we say that a equal to 1, if a equal to 1 mean you replace the value of a equal to 1 and a equal to 1.

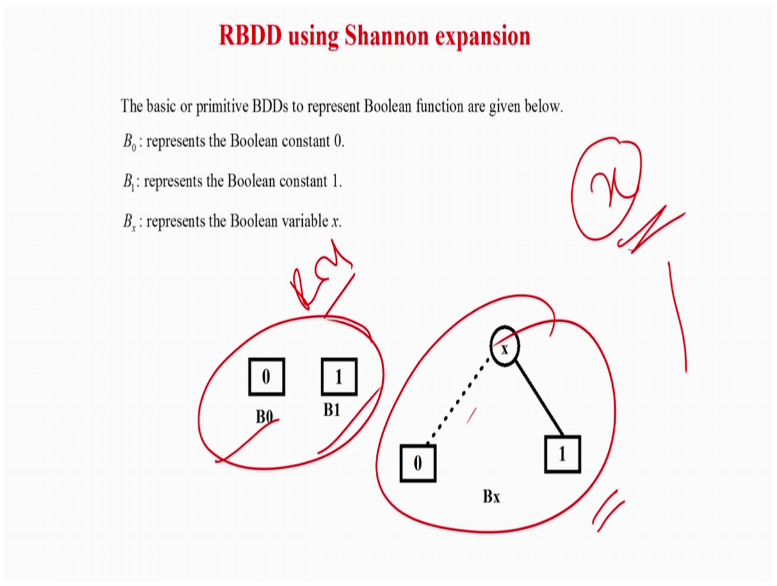
So, you are going to get 1 dot c b dot c 1 dot b that is actually f of a replaced with one that is nothing, but equal to b plus c plus b c. Now, second will be I will replace a with 0s, if you replace all a with 0s. So, this term will go, this term will go only b c will be there. So, this is nothing, but equal to f, this 1. So, finally, we can write this 1 as f of function, a is basically nothing, but a correct into b plus c plus b c and it a bar that is a dot is equal to b c. So, now, again you have to do it in terms of b, do it in terms of c basically; that means, the first node is a.

So, a a 0 will be equal to d c and a equal to 1 is nothing, but equal to the bigger one that is equal to b plus c plus b c and you have to keep on doing it and at the stage, you have to keep on reducing, that is what is the idea of a, binary decision diagram. Construction is the Shannon expression, because even for a very big circuit, whatever you have at least the function, function is nothing, but your design intent, that is a very compact representation.

Whenever we want to convert this functions, to some kind of models were, we can do model checking on algorithmic verification, then we require a graphical or we require a data structure, which can represent it from all the paths traversals etcetera, because function is nothing, but some a b plus b c plus c a some variable combinations. We cannot do much with it, but to go for logical reasoning all this things, we should have represent it in terms of decision tree.

So, if you make a binary decision tree with very large, but with Shannon's expression, we can construct BDD from the basic root level itself. So, that we can get a compress structure and at the every stage. So, basically, how we start about it. So, basically, may be, we have a function first.

(Refer Slide Time: 40:18)



RBDD using Shannon expansion

The basic or primitive BDDs to represent Boolean function are given below.

B_0 : represents the Boolean constant 0.

B_1: represents the Boolean constant 1.

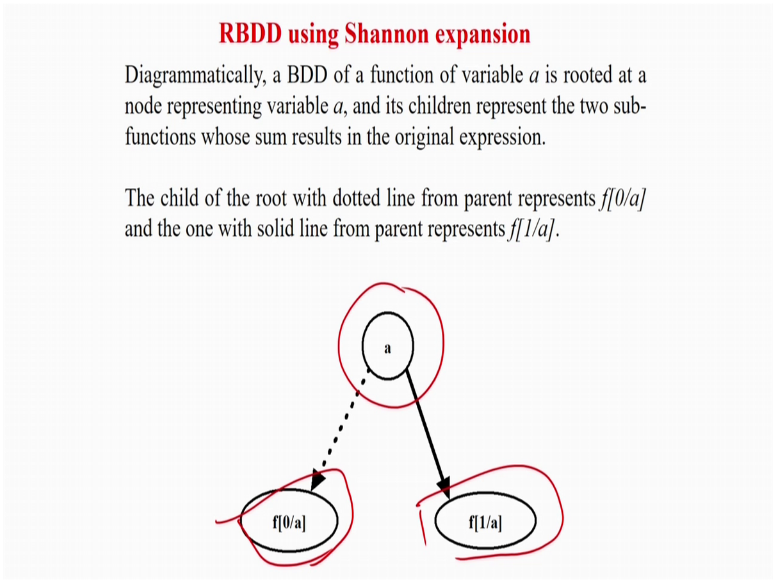
B_x : represents the Boolean variable x.

We start with preliminary function, may be we have some variable called x. So, first we will have the leaf, two leaf node, whenever we try to build a BDD, we start from the leaf. So, there will be only two leafs as we know 0 and 1, then maybe there is 1 variable for all variables, you have. We have to make a structure like this x 0 is 0 x 1 is 1 that is the variable itself. I assume that, that is only by input. So, if you have input x 1 x 2 x 3 x 4 x 5.

So, you will have this for the root, this will have it only ones and how many variables you have, you will all have trees of this nature that is x is 0 means 0 x is 1 is 1 that is representation of x. Now, we will merge them and try to make larger BDDs. So, those things actually I will, I will, we cannot cover in one class, but we will cover in next class. We will see how to make such bigger BDD from smaller BDDs, but what I, what Shannon expression does it takes?.

First, the leaf nodes then all the variables, it is enumerating, because it is very easy to understand that if I want to enumerate x, a single variable in BDD. So, x 0 is 0 x 1 is equal to 1 that is the value of x itself. So, all this leaf label stuff, they will make 2 leaf level and all the variables, they will make then, they will try to merge them and make a bigger model, but at every state, they will reduce.

(Refer Slide Time: 41:27)



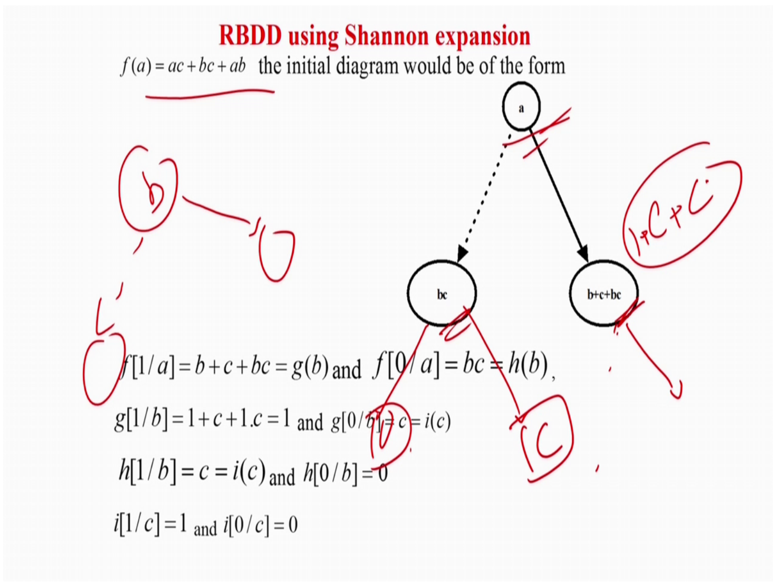
RBDD using Shannon expansion

Diagrammatically, a BDD of a function of variable *a* is rooted at a node representing variable *a*, and its children represent the two sub-functions whose sum results in the original expression.

The child of the root with dotted line from parent represents *f[0/a]* and the one with solid line from parent represents *f[1/a]*.

So, that is actually BDD construction which we will see in details later. So, basically if you say, they represent in some manner. So, if you have a bigger function. So, we will have f of a over here and then we will have this sub function of Shannon that is f, a in that function all a ‘s will be replaced with 0 and what are the value, you will have and this part actually will have all the a’s, which are replaced by 1 and this 1, you actually, you will have to merge them to make up a larger binary decision diagram. We will take all examples, do not worry about it, just, just the theory. I am telling you at present.

(Refer Slide Time: 41:59)



RBDD using Shannon expansion

$f(a) = ac + bc + ab$ the initial diagram would be of the form

$f[1/a] = b + c + bc = g(b)$ and $f[0/a] = bc = h(b)$,

$g[1/b] = 1 + c + 1.c = 1$ and $g[0/b] = c = i(c)$

$h[1/b] = c = i(c)$ and $h[0/b] = 0$

$i[1/c] = 1$ and $i[0/c] = 0$

So, basically we will start from something like this. This is the function a c plus b c plus a b. So, what you are going to have, we are taking a as the leaf node, because they assuming that level of a is first then the next level of node will be b then the next ordering will be c and so forth. So, a equal to 0 means, we have already seen the value of b c. So, let me zoom here, we are having something like a equal b, this was the function a b plus b c plus a b and then a b. This is the BDD Shannon intermediate representation.

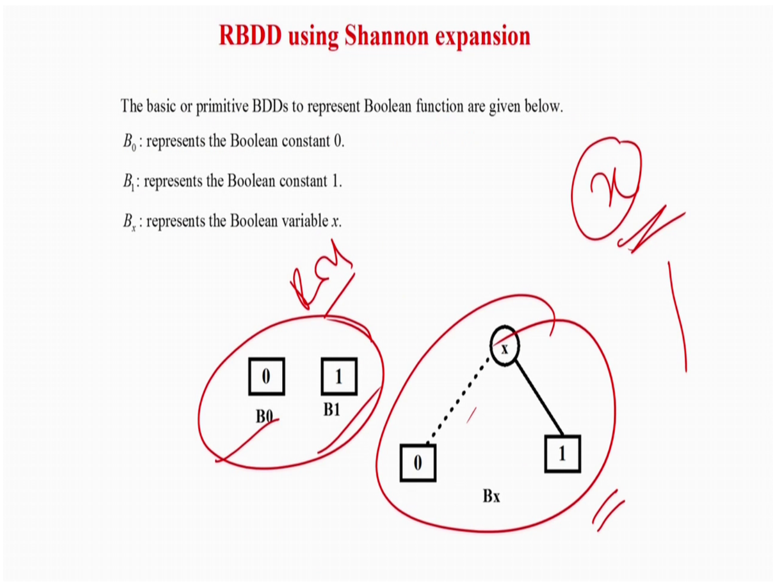
So, b c and then b c plus c plus b c, because if I make a equal to 1, you are going to get the value over here. Next, what they are going to do. Now, same thing, they are going to replace with b sorry. So, this will be 0 and this will be 1. So, if b equal to 1, you are going to get the value of c over here and if b equal to 0. You are going to directly get the value of 0 over this dotted line here. This is dotted line similarly, you replace represent for again for this and for this. So, if b equal to 1 over here then actually, it will be 1 then this one will be equal to b equal to 1.

There is 1 plus c plus b. So, if you look at it, it is nothing, but equal to 1. So, I have to put a 1 over here and if you make b equal to 0. So, in this case, you can find out that the answer is equal to nothing, but the node c right. This is, b plus c plus b c, if I make c equal to 0 sorry, b equal to 0, then this is gone. This is gone, only c will remain. So, this is the next intermediate structure, sorry.

So, same thing is written over here and then you have to. So, only one more node remains that c equal to 0, what happens c equal to 1, what happens. So, basically now, it c equal to 0 means, you are going over here, c equal to 1 means you are going over here and this procedure basically repeats. So, this is one way of forming the BDD's, at the stage you have to eliminate the redundancies.

We will take explicit examples to illustrate all this things, but I think, we have got the idea that how we can go about in forming the BDDs by Shannon's expression. We have a function, we have to select the order that a or b or c or d and keep on explicitly enumerating a equal to 0. This b equal to 0 then again you have to keep on doing it, till you get the fundamentals like, sorry.

(Refer Slide Time: 44:18)



RBDD using Shannon expansion

The basic or primitive BDDs to represent Boolean function are given below.

B_0 : represents the Boolean constant 0.

B_1 : represents the Boolean constant 1.

B_x : represents the Boolean variable x.

Like this was the leaf node and that is the root node and similarly, you will keep on expression and finally, you are going to get the binary decision diagram something like this.

(Refer Slide Time: 44:25)

RBDD using Shannon expansion

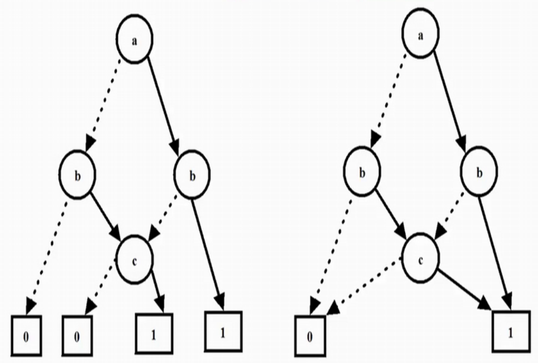


Diagram of the function with redundant nodes

Final OBDD of the function

So, what I told if you make a tree like this as I was discussing like, again in this case, it is b equal to 0 means, it will finally, a 0. So, b equal to 0 means it will be finally, a 0, then if b equal to 1 here means, you have to go for c and then only you can decide if b equal to 1. So, if b equal to 1 means c. So, basically if you look at it b equal to 1 means it is c. So, then again c will decide and this is the leaf structure of c as I have told you c 0 0 c 1 1. So, all the, if you have some leaf structures, all variables in the l leaf variables.

So, it will be represent by 0 and 1 that is nothing, but this, this, this structure, this structure. So, similarly I mean for all others, it can be expressed like a equal to 1 means as I told you b c. So, now, again this will be now, becoming b. So, if b equal to 1. It was nothing, but equal to 1. So, if you see a equal to 1 b equal to 1, you can directly go over 1, that is it was b equal to 1 means c plus 1 plus c plus 1 plus c plus c. So, this is nothing, but equal to 1. So, in this case c is basically becomes a redundant variable.

So, that is why you are writing from a, from b, you can directly have a. So, in this way you are going to get it, but now at, at, at every stage basically, it will add some redundant node, everybody have to finally, you go for at each state. You have to actually go for redundancy elimination, because in this case, you can see there are some redundant nodes in the left. So, use the rule R 1 R 2 and R 3. So, you are going to get this final minimised version.

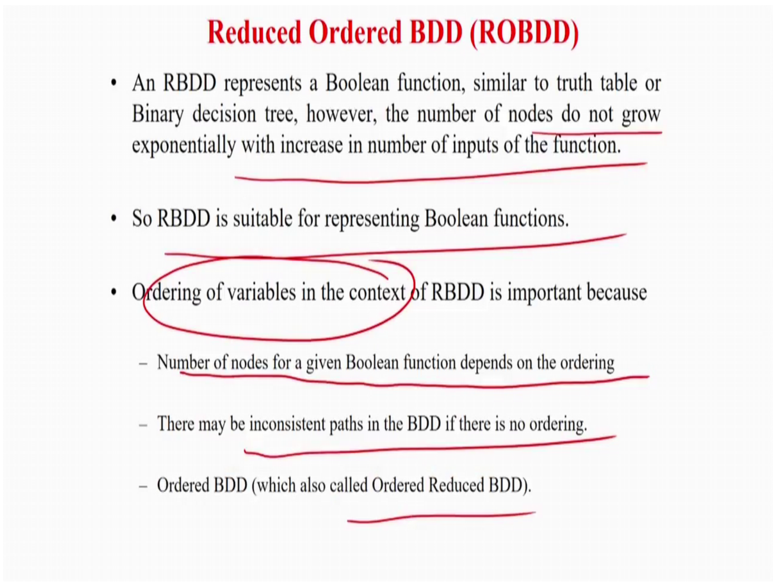
So, this one way of generating the BDD’s using a Shannon's expression that you take a node, then you decide the ordering. So, ordering is a, then b, and then c, then you take a equal to 1 have this sub graph here, have the sub function of a equal to 1 here and b in the left side sorry, a equal to 0, this is the sub function a equal to 1. This is now, you again do it for b, then do it for c and keep on doing it till you get the leaf nodes and every time, you have to see for this redundancy elimination and finally, you are going to get the BDD based function].

So, in this case, you need not actually have to go for something called previously. We were discussing that, you are actually taking a binary decision tree and you are going to make a BDD. So, that is not a way of explicit way of doing it. So, this is a Shannon expression or way of making a binary decision diagram or reduce binary decision diagram from the function itself, but sometimes, what it may happen is that you can have the circuit itself by looking at the circuit. You have to make a binary decision diagram that is another way of doing it, for that you have to know the operations of BDD.

So, operations of BDD is a slightly detail topic, which we will cover in the next lecture, but I think till now, you have got the idea that binary decision diagrams are very compact representation, but you do not require a binary tree, to do it you, if you have the expression, you can use Shannon’s method of doing it an intermediate steps, you have to somehow eliminate the redundancy. Another more interesting way of doing it is a given the circuit itself, how can we do it that also you are going to see in the next lecture, but before that one very important part is actually ordering.

So, what was the ordering that, that what I have said is that first, you are taking a then you are taking b and then you are taking c you can also also have taken the other way round that in this function, I could have started with b equal to 0 and b equal to 1 and then again I could have drawn the whole tree. I could have also started the, to do, it is the size of the BDD same surprisingly no answer is for different variable orderings, the size L drastically differs. So, that is one bad thing.

(Refer Slide Time: 47:42)



Reduced Ordered BDD (ROBDD)

- An RBDD represents a Boolean function, similar to truth table or Binary decision tree, however, the number of nodes do not grow exponentially with increase in number of inputs of the function.
- So RBDD is suitable for representing Boolean functions.
- Ordering of variables in the context of RBDD is important because
  - Number of nodes for a given Boolean function depends on the ordering
  - There may be inconsistent paths in the BDD if there is no ordering.
  - Ordered BDD (which also called Ordered Reduced BDD).

I should say about BDD, because if you doing arbitrary variable ordering the compression may not be very high. If you doing a very good variable ordering then the compression will be very-very good. So, therefore, binary decision diagrams are very sign of binary decision diagram is very much dependent on the ordering. Now, of course, if you want have some kind of finding out, whether two Boolean functions are equivalent or some properties, equivalent checking etcetera.

This same ordering we have to follow, because if you have a function, if your another function is, if you find, want to find out, whether this two functions are equivalent or not, then the 2 BDD's will be identical, that is proved, because the number of variables will be minimum in both the cases, subject to ordering; that means, 1 1 for 1, you cannot take the ordering like a b c and the other, you cannot take the ordering as b c a that is not possible, if you take the same ordering and if 2 Boolean functions are equivalent the same BDD's will appear and for that given ordering.

There cannot be any more reduce size decision diagram, if you are applying the rule R 1 R 2 and R 3. So, that is why BDD's are also sometimes called reduced ordered, binary decision diagram, because ordering is very-very important anyway. It is reduced, because if you apply R 1 R 2 and R 3, you cannot have further reduction, but on the sometime, you appreciate the fact that it is very much dependent on the ordering that, what is the first level, second level, third level, fourth level.

What was the variable orderings, you were using for the Shannon based expression or in any way, you are generating the BDDs in case of binary tree, whatever be the size you take the tree will all be exponential, because everything is means, explicitly mentioned in the leaf node and the sub leaf nodes. So, whether you take b c a or a b c, there will be no change in the structure of the circuit always, everybody will have 2 node, each child will have two nodes and similarly, it will go for.

But in binary decision diagram rather order binary decision diagram, the order actually have a great impact. So, it says that R R BDD represents of Boolean function, similar to truth table binary decision have a the number of nodes, do not grow exponentially with the function ROBDD is very suitable for function that is very good, but ordering of variables in the context of important is very-very important, because number of nodes of a given Boolean functions depends on the ordering that is, if you are bringing a good ordering, the number of nodes in the BDD is less.

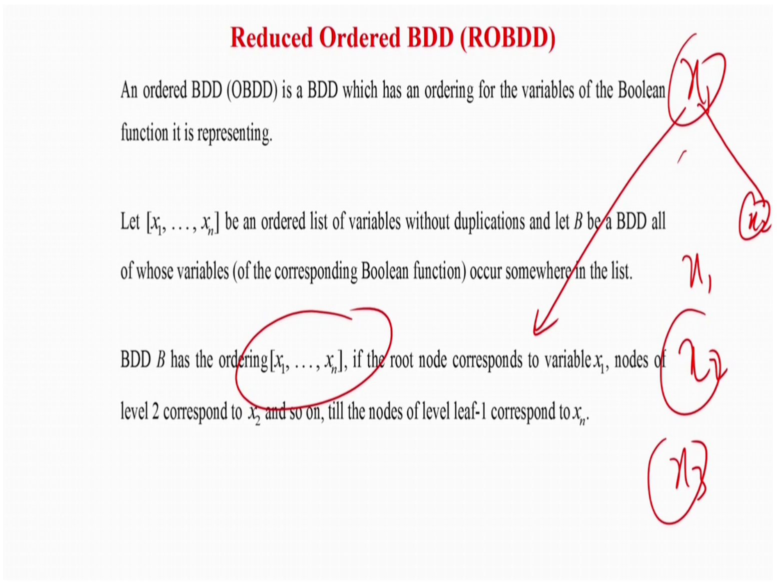
If you take a bad ordering, it will be very high, how to then find out good binary order. There are lot of heuristics and methods available, which we can guess that which is the best or very good binary audition, good binary ordering, they all variable ordering, but if you ask me that to find out the best binary means Booleans means variable ordering then that problem is also very-very difficult problem. Nobody tries to solve that in a very true manner, because there will be 1 or 2 or some, ordering for which will we give the optimal answer or the best answer.

Most reduced version, you are going to get, but to find out that ordering is a very difficult problem to solve and more time taking. So, that will again, you lose out all the interest of having using BDD's in the complexity, but the good thing is that for generally, for a given variable ordering of a good type will, which, which can be found out in reasonably very less amount of time, you are going to get a very-very good compression.

The best compression finding is a very difficult problem, but like from 98 percent compression to 97 percent compression or 95 percent compression. I am very happy, I do not want to go from 95 percent to 97 percent compression by spending hours and days to find out that best ordering, that nobody does, but for typically some most of the orderings, you are going to have a very good compression, that is one good idea, good thing and there are some other issues like there can be inconsistent paths.

If there is no ordering, ordered BDD is also called reduced order between; that means, what do you mean by inconsistency here, because say that I want to find out whether, two binary decision diagrams or two functions are equivalent order, if this same binary ordering is taken then basically, there will be two identical O BDD ROBDD's for this functions, but if the orderings are different, then you will not be able to, able to do the compression, because the orderings will look different. So, that is why, they are saying that for most of the meaningful compression, we always have to order the variables for a given example or a given context for every case, the ordering has to be maintained. So, therefore, you always use this term called ROBDD.

(Refer Slide Time: 51:40)



Reduced Ordered BDD (ROBDD)

An ordered BDD (OBDD) is a BDD which has an ordering for the variables of the Boolean function it is representing.

Let $[x_1, \ldots, x_n]$ be an ordered list of variables without duplications and let B be a BDD all of whose variables (of the corresponding Boolean function) occur somewhere in the list.

BDD B has the ordering $[x_1, \ldots, x_n]$, if the root node corresponds to variable x_1, nodes of level 2 correspond to x_2 and so on, till the nodes of level leaf-1 correspond to x_n.

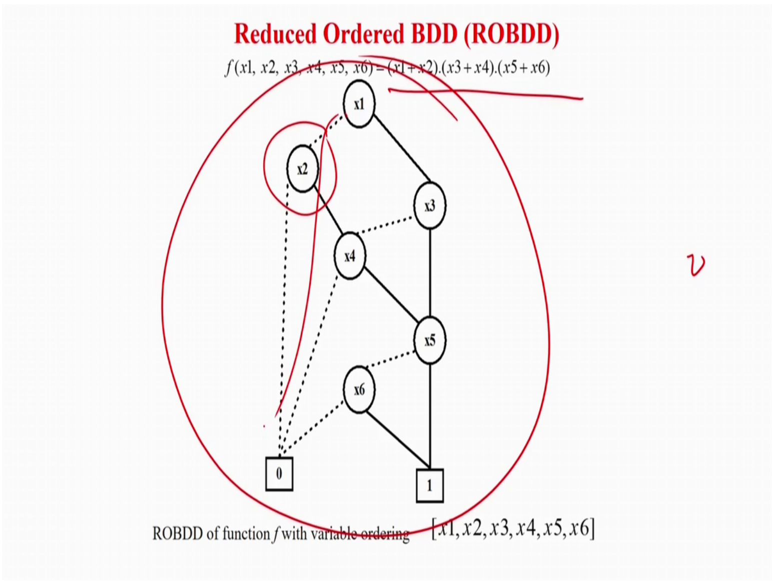
Now, this is mathematical. So, what do you mean by, variable ordering. So, basically let be this n on 20, the order list of variables without duplications, because a b c d e f no variable should be duplicated. So, these are all your input variables x 1 to x n and b let b be are BDD of whose variables occurs, somewhere in the list. So, there is a BDD ok, then there are lot of variables x 1, x 2, x 3 and all of them are happening in some, one may be x 2.

This may be x 2, x 3, this may be x 5 or something like x 3; that means, in your binary decision diagram. All the variables are appearing at some levels there; that means, none of the variables are redundant in at least one path. They are there means, it is the binary decision diagram, when you are drawing, it is not the case that the node x 3 say is no were appearing in the tree, in one path. It may not be there, but in other part, it is there then the BDD has a ordering x 1 to x n, if x 1 is the leaf node and then x 2 will be found out in the second level, x 3 will be found out in the third level and so forth, but the idea is that you may not find out x 2 in all the paths from x 1, in the second level, some path may from x 1, may be directly going to 0 or 1.

In the other part, the second level x 2 will be there; that means, in at least some path the x 2 will be there in the second level that is the difference between binary decision diagram and binary tree. Binary decision tree means every path, you will have x 2 in the second, x 3 in third level and so forth, but here in all the paths. You may not have the respective node at that level, but in at least one path, it will be there and the ordering is fixed; that means, the no path, you will have x 3 first and x 2 second.

That will not be the case, if x 1 x 2 x 3 x 4 is the ordering. So, in all paths or in some path the ordering will be maintained like x 1 x 2 in that, but x 3 may not be there x 4 and x 5. But it will never be the case that there will be a path where, x 1 then x 2 then x 5 and x 3, that will not be there. So, this actually the variable ordering that is root node, that in the second level at some path, there will be x 2 3 means at some path x 3 and so forth.

(Refer Slide Time: 53:50)



Reduced Ordered BDD (ROBDD)

$f(x1, x2, x3, x4, x5, x6) = (x1+x2).(x3+x4).(x5+x6)$

ROBDD of function f with variable ordering $[x1, x2, x3, x4, x5, x6]$

Now, interesting I have taken this is the function x 1. So, this is your function x 1 plus x 2 dot, this one is the function, I have drawn a O BDD reduced order O BDD, the ordering x 1 x 2 x 3 x 4 x 5 and x 6. So, x 1 is there, x 2 is the path of course, you see in this path there is no x 3, x 4 and x 5, but there is a path in which case, the second level x 2 is there similarly, in this path. There is no x 2 over there x 3 is directly coming, but if you look at it, there is one path where x two is there. So, there is be one every path, there will be at least one path, where all the variables are there.

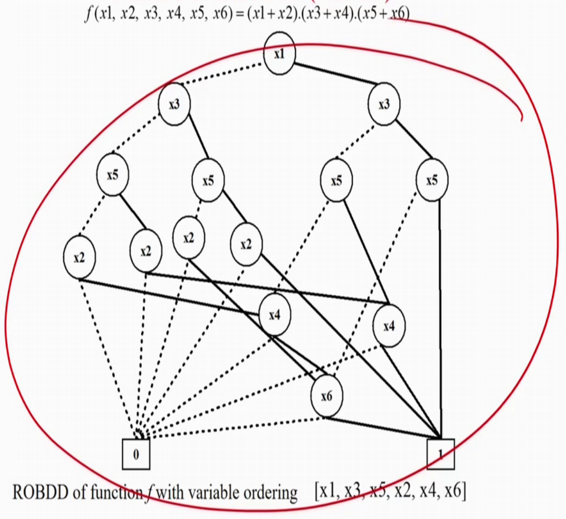
So, there no redundant variables. Secondly, in no path, you will able to find out that x 3 appears first and x 2 appears next, that is not going to be the case, if in some path x 2 is appearing after x 1; that means, in that path x 2 cannot be present. So, in this tree basically, the ordering is x 1 x 2 and x 3 and x 4 x 5 and x 6. So, if you see x 1 x 3 x 4 x 5 x 6 and so, forth. So, as it is very much compress in some path, some of the nodes will not be there that is; obviously, the requirement, when we are going from BDD to binary decision tree to binary decision diagram; so, but in no path, we will find the orders plate.

So, in this case the ordering is this and this is if you draw the binary decision diagram, which you can take as a home work, you will find out that this is the sign of the tree. Now, another binary ordering, this is just the name, I mean, I can change I mean is, x 1. This is, I mean, this is just for this example, it happens that x 1 x 2 x 3 x 4, this serial ordering is going to give you the best one of the very good ordering.

(Refer Slide Time: 55:26)

Reduced Ordered BDD (ROBDD)

$f(x1, x2, x3, x4, x5, x6) = (x1 + x2).(x3 + x4).(x5 + x6)$



ROBDD of function f with variable ordering [x1, x3, x5, x2, x4, x6]

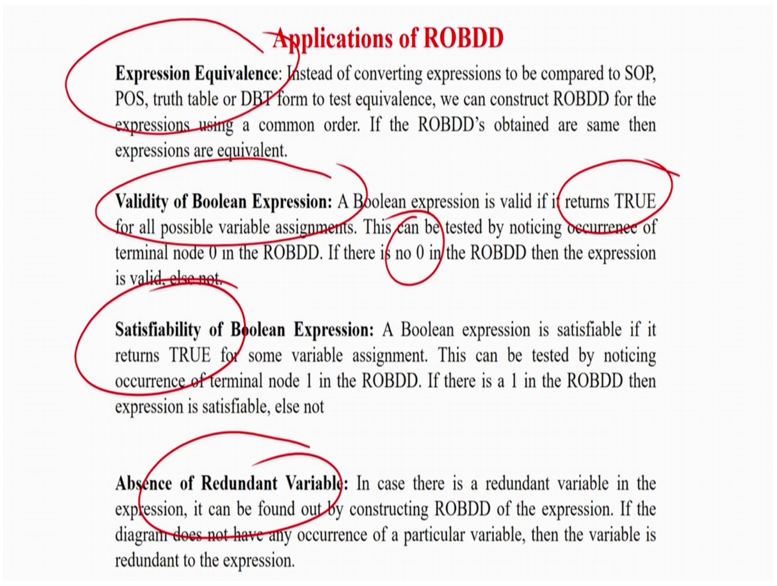
If you take a jumbled ordering like x 1, x 3, x 5, x 2, x 4. Some ordering you will found out that the tree is almost exponential in size. So, do not go by the fact that, that is this function is x 1 x 2. So, I have taken the variable ordering in a serial fashion. So, I am getting a very good structure and if I taking a one bit flip like x 1, x 3, x 5, x 2. I am going to get a jumble order. I mean very big size, it only happens for this example what I try to show is that for the same function, if you take different combinations of ordering, you are going to get very-very different size, tree structures.

So, this one is very compact and this one is very exponential, but this is the, in this case this is the best ordering and this is the worst ordering, both are very-very difficult to find any good typical ordering. You will find, you will be finding that I am going to get a reasonable good compression. There is a standard package called cut package, with actually implements the whole BDD algorithm. In fact, internally, they have nice heuristic to determine good variable orderings not the optimal one.

So, that is what I am saying that people do not try to find out the best ordering, finding the best ordering will actually blow up the problem exponentially and it will again land up to the issues of binary decision tree and all those things. There are good heuristics, which will find out somewhat very-very good variable ordering, where you will get very high compression like 99 percent plus or even higher.

So, we all typically try to find out the good ordering and then we are going to use BDD construction, whether by Shannon's expression tomorrow. We will see some other ways of construction and finally, we are going to lead a very-very compress data structure.

(Refer Slide Time: 56:54)



Applications of ROBDD

Expression Equivalence: Instead of converting expressions to be compared to SOP, POS, truth table or DBT form to test equivalence, we can construct ROBDD for the expressions using a common order. If the ROBDD's obtained are same then expressions are equivalent.

Validity of Boolean Expression: A Boolean expression is valid if it returns TRUE for all possible variable assignments. This can be tested by noticing occurrence of terminal node 0 in the ROBDD. If there is no 0 in the ROBDD then the expression is valid, else not.

Satisfiability of Boolean Expression: A Boolean expression is satisfiable if it returns TRUE for some variable assignment. This can be tested by noticing occurrence of terminal node 1 in the ROBDD. If there is a 1 in the ROBDD then expression is satisfiable, else not

Absence of Redundant Variable: In case there is a redundant variable in the expression, it can be found out by constructing ROBDD of the expression. If the diagram does not have any occurrence of a particular variable, then the variable is redundant to the expression.

So, what are the applications of BDDs is actually first is called expression equivalence that is there are two functions given, if to find out, whether they are equivalent or not how do you do it, we will basically have a typical way same variable ordering try to find out the ROBDDs and if the both BDDs are equivalent same should not be, I call it exact, exact image of it then you are going to say the functions are equal.

Why it will happen, because for a given variable ordering the ROBDD will always be same, you cannot have any further reduction. So, if I have the function a and function b and I take the same variable ordering of course, I will have the most short minimum size tree and of course, they have to be same then there is something called validity of Boolean expression. Validity means basically, a function, if is always true then I said that the function is valid. So, in this case what is going to happen the BDD will have no 0 node sorry, no, yeah there will be no 0 node, in the leaf that node will be eliminated.

All the paths will direct you to 1; that means, for all input cases the answer is going to be a 1. So, that is actually called the validity of Boolean expression, there is also very-very important in some kind of verification cases, because you find out manipulating by function elimination labelling etcetera. The output of the function should always be one that is all case, it is satisfied, there is something called satisfiability and at least in one path. It is satisfied then, that case there should be a node in the leaf level mart one, if there is node one in that; that means, there is some path, which are leading to one; that means, at some places the expression is satisfied.

Very and the very important is finding out of the redundant variables, only the we have seen in the test or a TPG cases that if there are redundant circuit variables then that fault cannot be tested and your a TPG algorithm keeps on looping, trying to find out alternate path to get that fault tested or that fault is non testable, because of redundancy and you kill lot of time in a TPG. So, if you have a Boolean function, you directly want to implement the circuit on this, if is a very big circuit, you cannot use express, you cannot use Karnaugh map etcetera.

So, there can be any chances of, redundancy happening, but if you can represent in terms of BDD, if there are some redundant variables; that means, the variable happens nowhere in the BDD, not even in a single path; that means, that function that variable is a do not care variable, just remove it in the, in the examples, basically x 2 is not means a redundant variable, because in one path, it occurs, but for some functions or in most of the functions, you will find out that there are some of the variables, which will appear nowhere in any of the path; that means, those variables are redundant and you have to eliminate them.

In that case there lot of advantages area reduction testability, becomes easy and so forth. So, these are some of the very-very prominent and important use of ROBDD and the advantages all comes, because of compression and because of ordering. Without ordering only with compression, you cannot do much, because for the same function, if you have different ordering, nothing can be compared and equivalence cannot be changed.

So, once it reduced order, once it is reduced binary decision diagram with a variable ordering given, then the utility actually becomes a great utility and this is actually one of the heart of data structure of all the CAD algorithms. So, whenever you talk of circuits functions state machines, we all represent them in terms of binary decision diagram and again as I told you what we do basically, we take a typical ordering or the 2 or heuristic exist, which give you good variable ordering and you get a typical, very good compression, because for most of this systems we develop the all combinations of inputs are actually not valid or we do not require them and they are do not required to be modelled itself, only a very-very few fraction are meaningful.

So, that is the key idea of going for verification of complex systems that instead of basically, making a large size of state machine explicitly, we will try to represent everything in terms of binary decision diagrams. So, that all redundancies are eliminated and try to model all these, means model checking, algorithms labelling, searching etcetera, in the BDD itself, which is actually called symbolic way of doing it.

Next class, what we will try to do; we will quickly try to see another way of doing generating the BDDs giving a circuit itself, rather than not going for the Shannon's method like, we have a circuit, we have the variables then how can we quickly make a binary decision diagram out of it? Which is, will be done possible through operations on BDDs, then we will also try to see some interesting advanced structures, because anyway in testing, we have already seen that whatever you do, if you remain at the byte level or bit level then nothing can be done, when you have last system like, systems of systems.

So, in this case also we will try to move our self from this binary business to arithmetic business, integer business high level abstraction. So, advanced version of BDD is like arithmetic decision diagram, high level decision diagrams. Also we will try to see in the next lecture, but one thing the philosophy everywhere is similar that, you have to remove the redundancies.

Thank you.