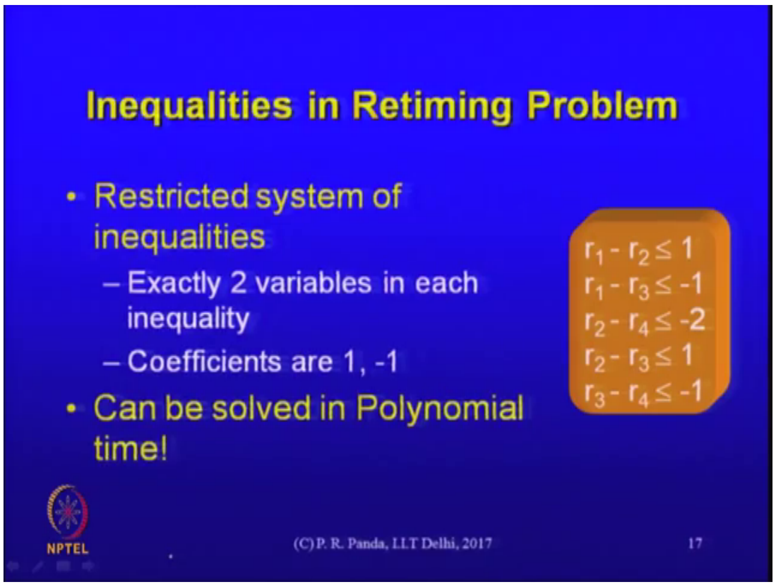
# Synthesis of Digital Systems

**Dr. Preeti Ranjan Panda**
**Department of computer Science & Engineering**
**Indian Institute of Technology, Delhi**

## Lecture – 19
## Efficient Solutions to Retiming & Introduction to Logic Synthesis

(Refer Slide Time: 00:23)



### Inequalities in Retiming Problem

- Restricted system of inequalities
  - Exactly 2 variables in each inequality
  - Coefficients are 1, -1
- Can be solved in Polynomial time!

$r_1 - r_2 \leq 1$
$r_1 - r_3 \leq -1$
$r_2 - r_4 \leq -2$
$r_2 - r_3 \leq 1$
$r_3 - r_4 \leq -1$

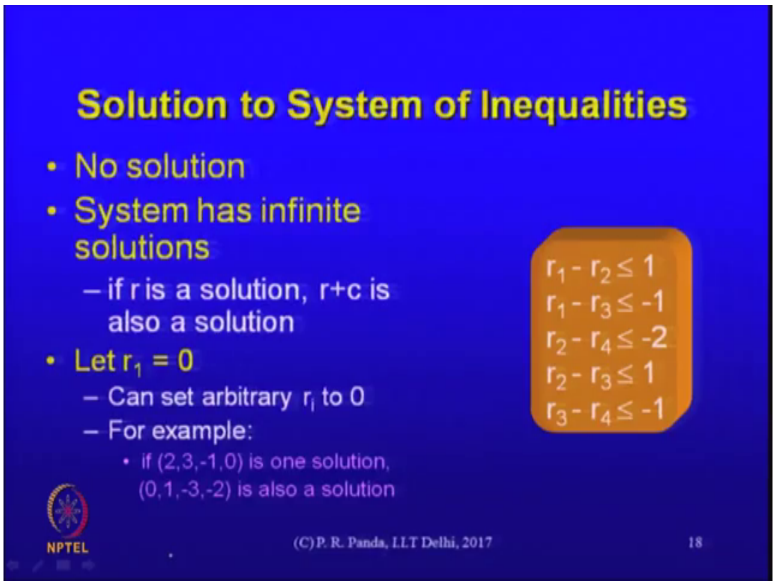
NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 17

This was an example of integer an linear programming problem that we said has some features that actually enable a more efficient solution. What we will look at now is how do you solve it. Efficiency essentially the proof that there is an efficient solution that proof need not be the way, you would actually solve it, this formulation is enough and if you submit this set of inequalities to a standard solver, it will find an efficient solution into this, how it solves it, we will not get into, but let us look at a conceptual proof that this is not a difficult problem, not like the general integer linear program ok.

So, what characterizes this is that you have these coefficients in each of those inequalities, you have exactly two variables and those coefficients are 1 and minus 1; that is the special case for which an efficient solution is known, other special cases are also there for which is known, but this particular special case has a efficient solution.

So, let us go through a set of arguments to ultimately prove that this particular problem is not difficult and in the process we identify; what is the solution to this sort of inequalities ok.

(Refer Slide Time: 01:44)



## Solution to System of Inequalities

- No solution
- System has infinite solutions
  - if r is a solution, r+c is also a solution
- Let $r_1 = 0$
  - Can set arbitrary $r_i$ to 0
  - For example:
    - if (2,3,-1,0) is one solution, (0,1,-3,-2) is also a solution

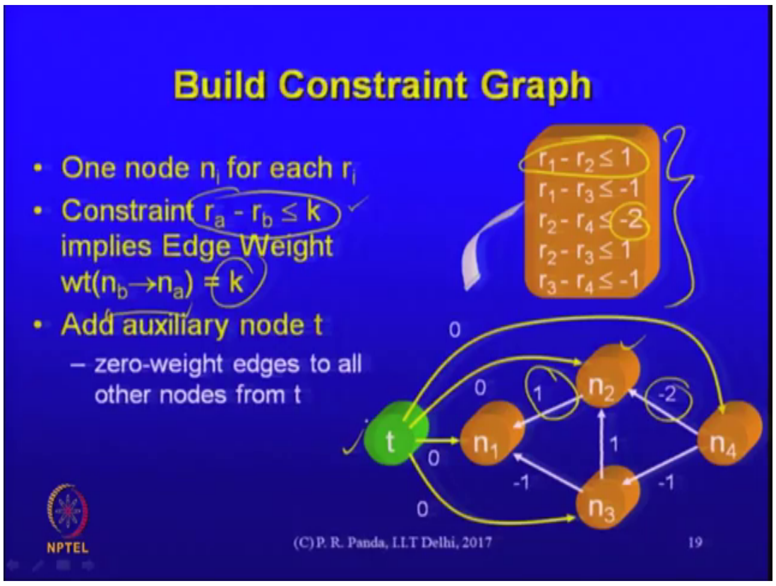
$r_1 - r_2 \leq 1$
$r_1 - r_3 \leq -1$
$r_2 - r_4 \leq -2$
$r_2 - r_3 \leq 1$
$r_3 - r_4 \leq -1$

(C) P. R. Panda, I.I.T Delhi, 2017 18

So, what are the possibilities? Anyway for this set of inequalities, it is possible for you to set up these inequalities such that there is no solution that should be easy, you can set up some pairs that are inconsistent with each other and you end up with no solution, it could also be that the system has an infinite number of solutions that can you see that if you have one solution, then you can generate any number of solutions from there.

Since all are of the kind x minus y is equal to constant, if you increase all the rs by some fixed constant both r 1 and r 2 are there, all of these are increased by the same constant, then they would respect the same constraint as long as there is one solution, you can generate an infinite number of solutions as a special case, you can actually set an arbitrary r i to 0, why not we said that; I can add or subtract anything, any constant specifically, I can subtract whatever is the value of r 1 from all the other rs and I would get a valid solution.

(Refer Slide Time: 03:00)



## Build Constraint Graph

- One node $n_i$ for each $r_i$
- Constraint $r_a - r_b \leq k$ implies Edge Weight $wt(n_b \rightarrow n_a) = k$
- Add auxiliary node t
  - zero-weight edges to all other nodes from t

$r_1 - r_2 \leq 1$
$r_1 - r_3 \leq -1$
$r_2 - r_4 \leq -2$
$r_2 - r_3 \leq 1$
$r_3 - r_4 \leq -1$

(C) P. R. Panda, I.I.T Delhi, 2017 19

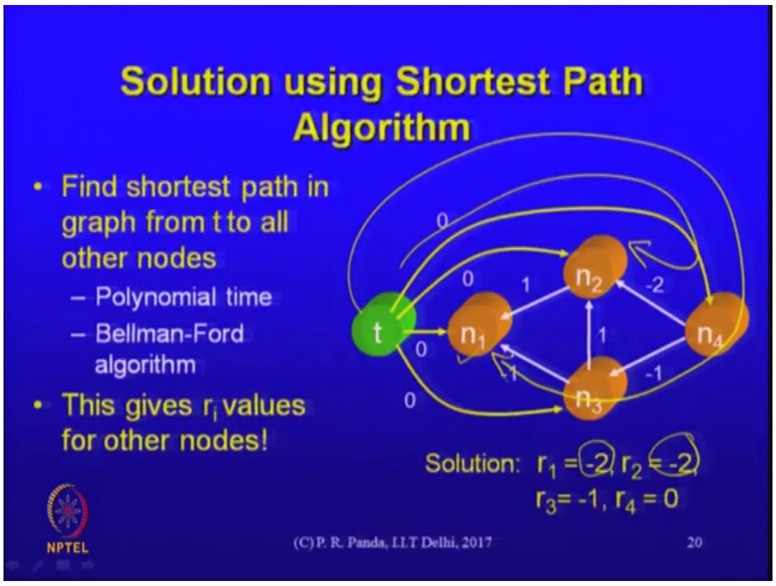
Let us go through a set of arguments with a new graph, yet another graph that is constructed specifically for this proof ok. So, this is just a set of conceptual arguments like we said. So, I have this set of constraints, of course, each of those constraints is of the kind r a minus r b less than or equal to k where k is a constant ok. So, you build a graph in which there is a node for every r i value ok.

And corresponding to every single inequality, there you introduce an edge in that graph in the following way if that constraint is r a minus r b is less than or equal to k, then you have an edge from node b to node a with the weight being k, why is it this way is not obvious, but this is just a construction for us to prove something later on ok.

So, if I have r 1 minus r 2 is less than or equal to 1, I have an edge from n 2 to n 1 with the weight being 1 if it is a negative number, then the corresponding rate here is a negative number, the construction is clear, this is first, second, you add an auxiliary node to the graph t. And from there, you introduce 0 weight edges to all the other nodes, why we will just see, but this is a construction it turns out that it does not violate some property that we will use later on, but it enables us to find a solution.

So, that is what we have done, we constructed the graph and you added one more node t from where you just introduced edges to all the other nodes weights are there on all the edges so on. These are new edges; I just put away to 0.

(Refer Slide Time: 05:27)



## Solution using Shortest Path Algorithm

- Find shortest path in graph from t to all other nodes
  - Polynomial time
  - Bellman-Ford algorithm
- This gives $r_i$ values for other nodes!

Solution: $r_1 = -2$, $r_2 = -2$, $r_3 = -1$, $r_4 = 0$

Now, you find the shortest path in the graph from node t to all the other nodes in the graph ok, let us look at some examples from t to node n to what is the shortest path, remember shortest path can be negative, right. So, actually that shortest path is;

Student: Minus 2.

Yeah, minus 2, where does it come from? That is the path right 0 and minus towards you add up all the edge weights along the path that is the cost of the path that is the path length similarly to n 1 the shortest path is.

Student: Minus 2.

Minus 2, why is that?

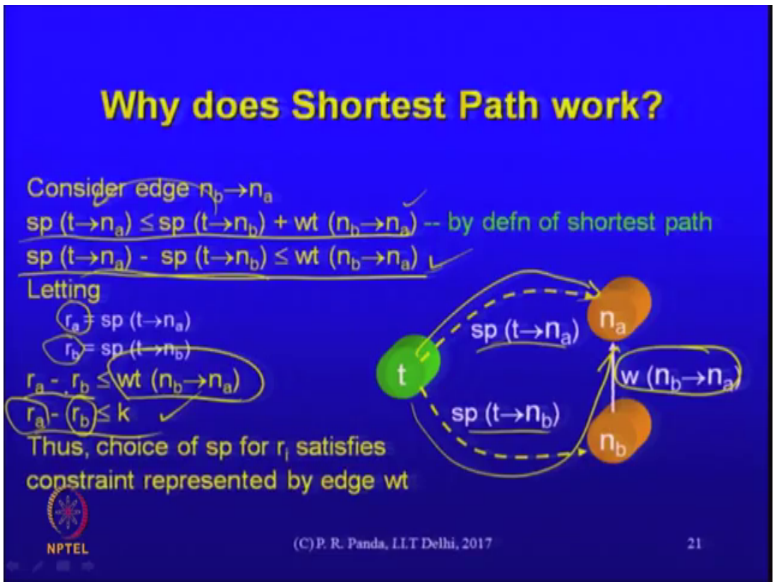
Student: n 4 2, n 3 2.

Right, right; So, that is the path; so, you find the shortest path from t to all the other nodes.

Student: Ok.

The assertion is that this shortest path length actually gives the desired r i values for all the nodes in the graph shortest path length from t to all the nodes in the graph gives us the desired r i values that respect the constraints that we had set up ok, why it should do that? Let us argue in the following way.

(Refer Slide Time: 07:12)



## Why does Shortest Path work?

Consider edge $n_b \rightarrow n_a$

$sp(t \rightarrow n_a) \leq sp(t \rightarrow n_b) + wt(n_b \rightarrow n_a)$ -- by defn of shortest path

$sp(t \rightarrow n_a) - sp(t \rightarrow n_b) \leq wt(n_b \rightarrow n_a)$

Letting

$r_a = sp(t \rightarrow n_a)$

$r_b = sp(t \rightarrow n_b)$

$r_a - r_b \leq wt(n_b \rightarrow n_a)$

$r_a - r_b \leq k$

Thus, choice of sp for $r_i$ satisfies constraint represented by edge wt

(C) P. R. Panda, I.I.T Delhi, 2017 21

Consider that there is some shortest path we have found from t to node a and I have separately found a shortest path from t to node b ok.

And that length is given by the s p of that path from t to node a whatever the part is, right, shortest path by definition must mean that this path length must be less than or equal to this path length because that is also an alternative path to node a. So, if that other path was shorter, then that would be the shortest path. So, that is the assertion. Here, it could be equal also because in fact, that path here could be the shortest path right path through n b; that might be the shortest path in which case, it will be equal perhaps.

But otherwise that other path must be shorter than this one that includes this edge from n b to n a. So, this must be true, right, this is just the definition of the shortest path, otherwise, it would not be the shortest which means if I just take this to the left, I have the shortest path t to n a minus the shortest path t to n b is less than or equal to the weight of that edge.

Student: Ok.

Let us just the rearranging taking that term to the left ok. Now if I say r a is the shortest path from t to n a, r b is the shortest path from t to n b, what was this weight? Anyway;

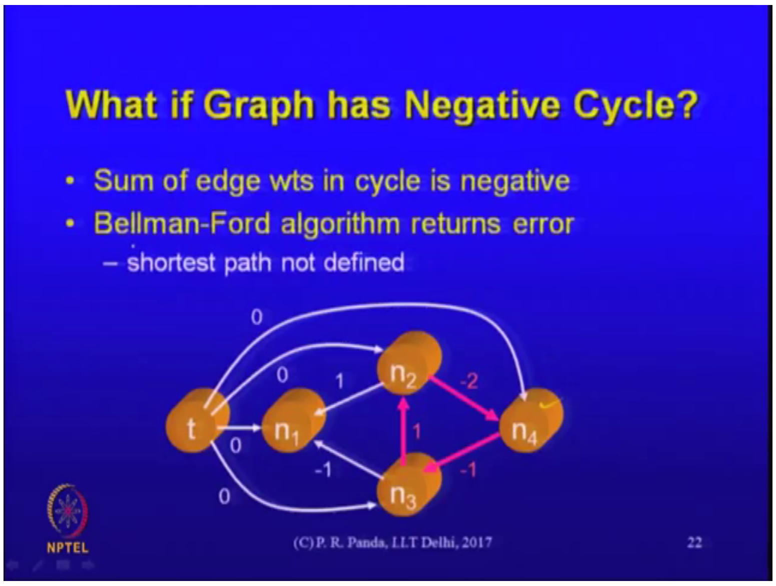
this weight was the kth that came from the constraint, right. So, if I have found the right shortest path from t to all the nodes, this is some arbitrary node n a and n b if I have found the shortest path and since, this is respected by the shortest path, I would have r a minus r b, if I just choose r a to be the shortest path that to a and r b to be the shortest path from t to b, then I would have r r a minus r b is less than or equal to this which is just a rewriting of this equation.

But this is nothing, but k; the shortest path argument just enables us to or give that r a minus r b that particular inequality is satisfied, I have an edge here for every constraint that I had every; in equality that we had set up, right, there is a corresponding k, there is a corresponding each of those inequalities results in one edge being added to the graph and therefore, this argument just tells us that if you just choose the r a and r b to be just the shortest path in this augmented graph, then you have a set of values for the r is that respect all the constraints.

Student: Sir, how is this graph portray the node solution scenario in?

I will come to that, but if there is a solution, this actually gives us the set of. In fact, it gives us all the r is because it is essentially the shortest path from t to all the nodes, right. So, we would have an r vector retiming vector for all the nodes that is valid, once, we are able to find the shortest path like this, any questions about this argument? This is a fairly simple.

(Refer Slide Time: 10:48)



## What if Graph has Negative Cycle?

- Sum of edge wts in cycle is negative
- Bellman-Ford algorithm returns error
  - shortest path not defined

There are situations when the set of inequalities does not lead to a feasible solution. In fact, this shortest path algorithm also sometimes does not return a feasible solution. In fact, we will show that these two are actually the same situation shortest path does not work when will it not work.

Student: Sir, when is if it is positive number;

If what is a positive number?

Student: I mean we are calculating the shortest from t to any node.

Right.

Student: And if to say in this case, if that minus 1 s 1 n (Refer Time: 11:23) was not there.

Ok

Student: If you say, it was plus 1, then the shortest path are able to return a 0.

Yeah, just ok, not a is a return value 0, for some nodes that is fine why not.

Student: Sir, that graph you have shown here is the issue common loop.

Yeah, the there are some times when the shortest path does not if you allow negative edges, then what might happen is that you may have a cycle in which the path length is negative possible, right, what happens if you have a cycle with a negative sum of path lengths and you are trying to find the shortest path.

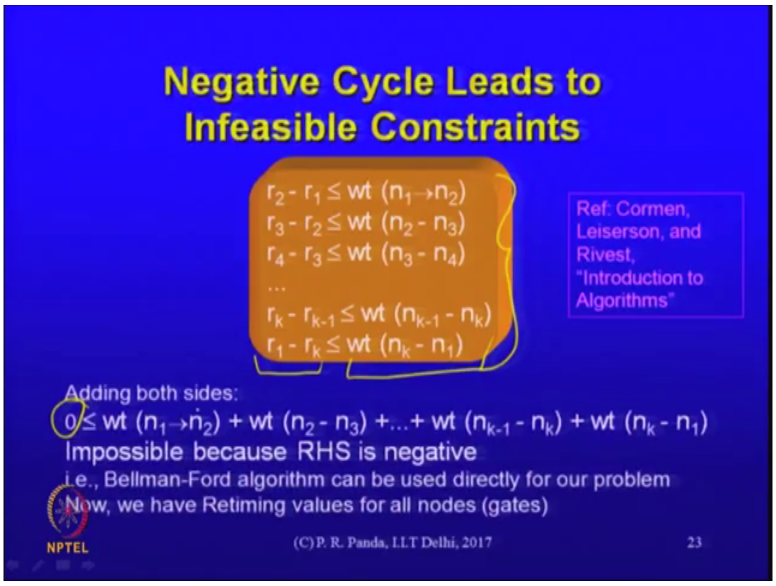
Student: Keep on going.

Yeah, if the shortest path is say, you arrive at one of those nodes, you can keep going around that cycle and the path length can keep reducing. So, that the shortest path is not properly, you find that the algorithm that people use for finding the shortest path single source multiple destined all destinations shortest path is what is called a Bellman Ford algorithm. This is a variation; there is a Dijkstra's algorithm that finds the shortest path between a pair of nodes.

But this, Bellman Ford is a way of efficiently finding the shortest path from one source to all the destinations, this is what you would use for this problem because we need the shortest path to everything, but that algorithm will return an error if such situations are there in which you have a negative cycle ok.

So, essentially shortest path is not defined, the algorithm might not actually return a valid solution, but what we will argue? This is exactly the infeasibility case of the set of inequalities in the following way.

(Refer Slide Time: 13:18)



## Negative Cycle Leads to Infeasible Constraints

$r_2 - r_1 \leq wt\ (n_1 \to n_2)$
$r_3 - r_2 \leq wt\ (n_2 - n_3)$
$r_4 - r_3 \leq wt\ (n_3 - n_4)$
...
$r_k - r_{k-1} \leq wt\ (n_{k-1} - n_k)$
$r_1 - r_k \leq wt\ (n_k - n_1)$

Ref: Cormen, Leiserson, and Rivest, "Introduction to Algorithms"

Adding both sides:
$0 \leq wt\ (n_1 \to n_2) + wt\ (n_2 - n_3) + \ldots + wt\ (n_{k-1} - n_k) + wt\ (n_k - n_1)$
Impossible because RHS is negative
i.e., Bellman-Ford algorithm can be used directly for our problem
Now, we have Retiming values for all nodes (gates)

(C) P. R. Panda, I.I.T Delhi, 2017 23

If there is a negative cycle, what does it mean? As far as the set of inequalities is concerned, this is our set of equations ok. So, negative cycle means you start from r 1 to r 2, r 2 to r 3, r 3 to r 4 and so on; r k minus 1 to r k, finally, r k 2 r 1 that establishes the cycle and let us say, we found as a cycle with a negative sum of edge weights what does it become?

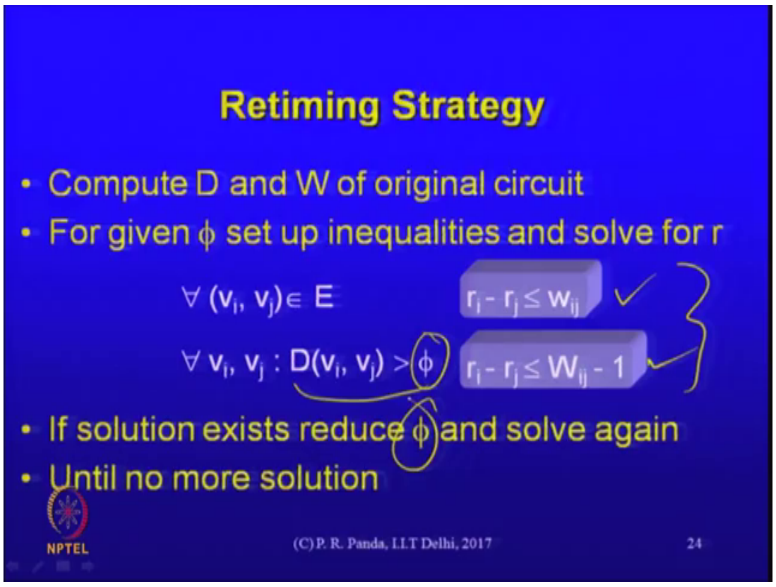
So, if you just add all the left hand sides and the right hand sides, in this set of inequalities, you have 0 on the left hand side because every r i is appearing once with a positive coefficient plus 1 and once with a minus 1. So, this is 0, sum of all of these.

In fact, we are saying is negative, but if there is a negative cycle leading from this set of equations then you actually have a negative value here. So, essentially what you are having is if you add up everything, you have 0 less than or equal to a negative number that is not possible, right.

So, essentially that Bellman Ford algorithm can be used directly for our problem, if it fails, it means that there was no solution anyway to the set of inequalities. So, you just run the single source shortest path algorithm to that augmented graph and what we have as output is the set of retiming values for all the nodes.

So, that is all was the retiming problem completely or there is something else still to long take care of it you got the r i values, what else is there? All r i values tell us number of registers that should be moved from the output of every gate towards the inputs right is there something else to do or problem solved.

(Refer Slide Time: 15:35)



## Retiming Strategy

- Compute D and W of original circuit
- For given $\phi$ set up inequalities and solve for r

$\forall (v_i, v_j) \in E$ $r_i - r_j \leq w_{ij}$ ✓

$\forall v_i, v_j : D(v_i, v_j) > \phi$ $r_i - r_j \leq W_{ij} - 1$

- If solution exists reduce $\phi$ and solve again
- Until no more solution

(C) P. R. Panda, I.I.T Delhi, 2017 24

Let us see, what is the overall retiming strategy? I compute those D and W matrices, this is a two dimensional matrix, right to both D and because from every node to every other node, if there is a path. So, i to j, V i j captures that W i j, similarly is the path weight right.

So, you compute those matrices for that phi, you set up the inequalities of this nature. So, this set of inequalities always shows up, remember this arises from the need to have the number of flip flops on the resulting circuit after retiming being greater than or equal to 0, this one is a conditional inequality, sometimes, you have it, sometimes you do not, if it turns out that the D i j is greater than phi, then you would have it. So, anyway between these two, you have a bunch of inequalities which you would then solve.

Let us say with their strategy that we just indicated by the way that proof that we went through of augmenting that graph with a new node and then running shortest paths, you do not necessarily solve it in that way, we just use that argument just because the shortest path computation is a known algorithm, you might not actually use sort as part to solve it, there are other ways to solve it, but this argument was essentially a polynomial time optimal solution to the retiming problem.

So, you can see that if I just apply that I have an optimal set of solutions of course, I may have an infinite number of solutions, but I have one set of solutions that is optimal as long as optimality is defined in terms of finding a retiming vector that satisfies the clock, there is one thing that is still not addressed, we can find all the rs. So, we first determine the set of inequalities after determining the set of inequalities, we can go ahead and solve it using whatever mechanism.

But efficient solutions are there, optimal efficient solutions are there, the question though is who gave the phi input, this is an input to this whole algorithm. In fact, the set of inequalities are actually determined by that condition and that condition involves phi. So, it is pretty much an input.

So, if our problem statement was given a clock period, find a retiming vector if it exists, then it is enough what we have, but that might not be the problem statement problem statement might be given the circuit, find the best clock period and what is the best clock period for which you can find some retiming vector that satisfies that timing.

So, if that is the case, then I still have to worry about how to generate the phi because this I started off with some phi right, but I do not know maybe a better phi is there, right I found a solution given a phi, we outlined a mechanism for finding the solution, but that still does not tell us how to find the clock period itself, right, it must be that if you keep reducing the clock period after some time, you will not find a feasible if you make it too small, then you will not find any feasible solution ok.

So, that outer loop is still there in the retiming which is I have to find a way of reducing the clock period that is the input to this strategy until there is no more solution. So, what strategy can we take? You can take a simple strategy that just decrements by some fixed constant right by some small constant, you do have to be careful about the number of times you iterate through this solution because you do not want to run it, many times, even though it is efficient, phi can be a real number right, it is not phi is not an integer not necessarily an integer.

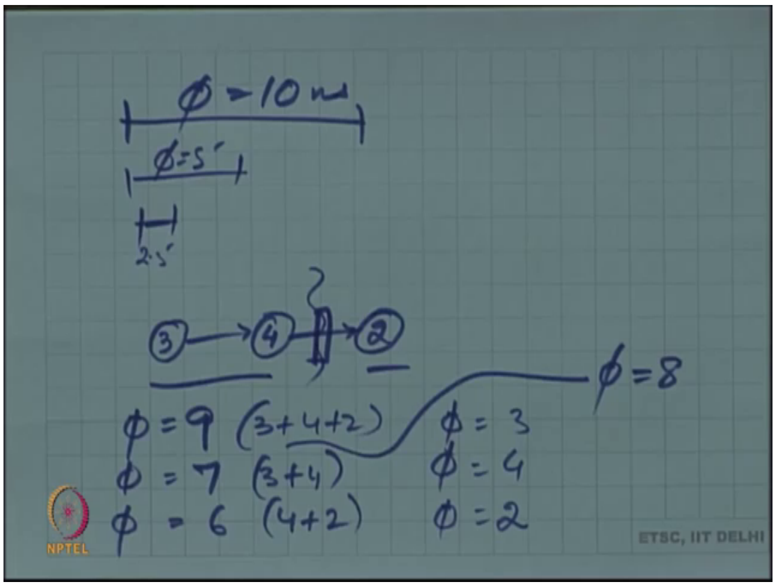
So, we can do a little bit about it which is how to minimize the number of times I set up the equations and solve the equation solving, it is efficient that we have a good algorithm for, but there is this question of how many times I will solve it.

Student: Sir, we can reduce the time by a large number, so that in the first equation, we get annual solution and then we can apply some binary sets kind of strategy (Refer Time: 20:04) the time.

Yeah.

Student: So, that case (Refer Time: 20:09).

(Refer Slide Time: 20:13)



Let us say, first try with a large phi equal to 10 some such and for that I am able to find a solution a binary search kind of a strategy suggests itself as an obvious mechanism for solving this if ten works, then I can try to make it 5 right. So, if the 5 also works, then I can try to make it 2.5 and so on, when do I stop.

Student: So, there has to be two constraints; one is number of iterations, the other is there can be another one.

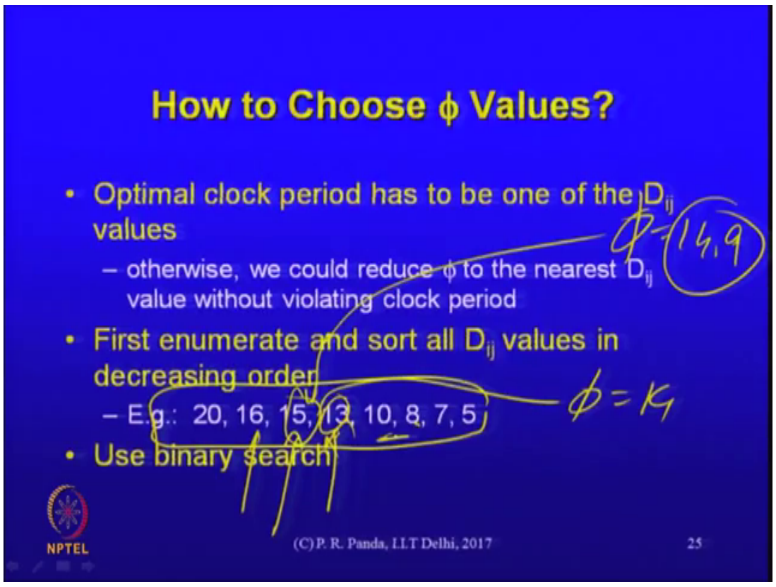
Looking at the netlist, there ought to be a minimum that you can easily figure out what would that minimum be.

Student: Set of (Refer Time: 21:08) delay total delay.

Certainly, each of those gates has a combinational delay, the clock period can be less than that right whatever is the max of all those combinational delays of the gates the clock period had better be more than that because that gate, of course, you need enough time for that combinational circuit to work, we are not doing the retiming in a way that we are moving them into the blocks into the gate, we do not do that; right.

So, that certainly gives bounds and imposes a lower bound on this, but as it turns out, you could actually be more efficient than that.

(Refer Slide Time: 21:48)



## How to Choose $\phi$ Values?

- Optimal clock period has to be one of the $D_{ij}$ values
  - otherwise, we could reduce $\phi$ to the nearest $D_{ij}$ value without violating clock period
- First enumerate and sort all $D_{ij}$ values in decreasing order
  - E.g.: 20, 16, 15, 13, 10, 8, 7, 5
- Use binary search

$\phi = 14.9$

$\phi = 15$

(C) P. R. Panda, I.I.T Delhi, 2017

25

This argument can be extended from the gate lengths to actually the path lengths; the path lengths, we actually computed already those D i j values are the path lengths, what do we need at the end whatever this solution is we can argue that the phi can be equal to at least the simplified way in which this problem is formulated it has to be one of those path length values.

Student: The best value critical; sir the best value.

So, it is like this, I went through the D i j matrix and these are my different path lengths, they are distributed all over, we can argue that the clock period that we choose must be one of these values because if it was not; say I found a solution for phi equals 14, I just tried 14 and it turns out that I managed to find a solution.

If I can find a solution for phi equals 14, I must be able to find a solution for phi equals 13 also because there is no path length between 14 and 13, right, any path that you come combinational paths that you look up well these are the critical combinational paths, right, the D i j, if there is no D i j between 14 and 13 and 14 is passing, then it must be that the critical value is this between 14 and 13, I need not search because there is no path that is actually adding up to that much.

So, if I just list all the values here then. In fact, I can still do my binary search, but only an among these values of D i j the point is there is a binary search, but that binary search is not over the real domains that was causing confusion, right, I I do not know when to actually stop when to start and so on, but it could be that I just take this set of these delay values and the binary searches is just there, right.

So, it is a relatively small number of iterations should be enough for us to converge into the appropriate phi; of course, other hints might be there from the circumstances of the design.

Student: So, basically you are saying take the mid (Refer Time: 24:20) of like 20, 16, 16, 15, 15, 13 and then try to get.

No, it is a this is an array, right you just do this is an array, you start with 13, if 13 fails, then you go here, if 16 fails 16 passes 13 fails, then you go here the standard way you do binary search on it.

Student: But we want to go between 15 13 also, right.

Why we just argued that you should not that our solutions are exactly at these points you do not want to explore the space between 15 and 13.

Student: Ok, that is not very clear.

What I am saying is if there were a solution where I had phi equals 14.9, right, I tried 14.9 and I found a solution, but in fact, there is no path length there that is adding up to 14.9, it must be that I would succeed even at 13,, right because remember there is a path length of 15. So, if 14.9, I am able to find a solution, it means that I am able to find a solution that is less than 15; that 15 is not what is determining my clock period, the next path delay that determines the clock period is 13, right.

Therefore, the range in between need not be searched, it is like if you had nodes with delays 3 4 and 2, what are the possible clock periods at which the circuit can be run one is I can say phi equals 3 plus 4 plus 2 9 that is the solution other is it could be phi equals 7 from 3 plus 4, right, it could also be phi equals 6 from 4 plus 2 or phi could also be 3 phi can be 4 phi can be these are my only path lengths, what we are arguing is that I cannot have an optimal solution of phi equals 8 for this circuit.

Student: It will not be the count of k.

If I can find 8, it must be that I can lower it to 7 and it would still be ok.

Student: Ok.

Right, why because let us say that was the solution putting a resistor here was the solution for phi equal to 8, the path lengths are such that you could reduce it to phi equal to 7 and it is still phi right point is there is no need to test the numbers in the interval between two successive paths lengths remember this is a little bit of an idealized argument, there are other numbers that we need to take care of there is the propagation delay of the flip flops the set up time of the flip flops those are ok.

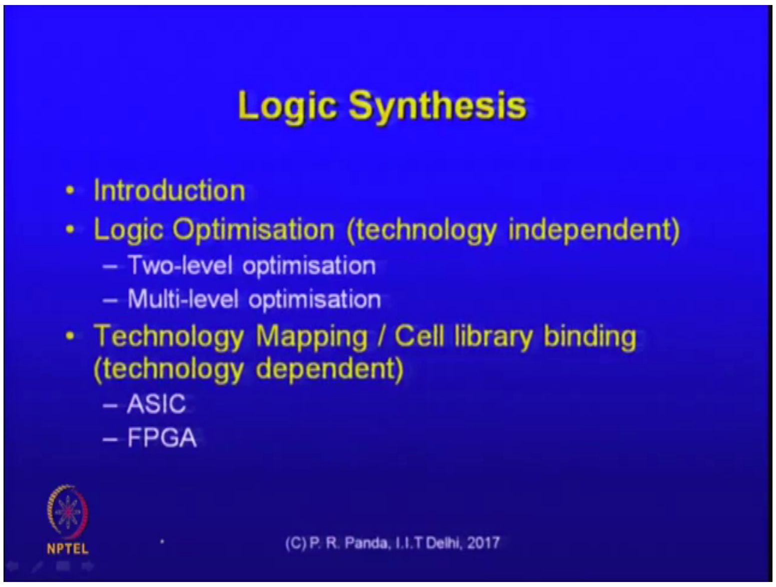
Those do need to be handled, but the way you can handle them is that you leave a certain margin for them and this analysis is in term by excluding those margins and finally, you can add up the set of time and so on to the path delays, right, that completes the retiming problem, this is an instance of a synthesis problem for which an optimal solution is there and can be efficiently found.

This also is pretty deterministic and ready straightforward right, then the outer loops of course, the inner solution mechanism itself is a polynomial solution that gives the optimal answer, but the number of iterations is also small and unlike some of the other problems where we said we will just use a heuristic because it is a difficult problem this problem is not difficult, right.

There are other issues which are common issues to all of synthesis those still show up here which are of the kind that the delays are really estimates of delays rise times are different, fall times are different, geometry is not taken care of all of those which are common to many other steps of synthesis those, of course, are still there, but this particular problem, the way it is formulated a simplified version of a problem, but this problem. In fact, has an elegant and attractive solution.

With that; let us move on to the next topic a little lower in abstraction detail that is logic synthesis, this is a topic that we have some understanding of that logic minimization is something that we have been exposed to in various ways earlier.

(Refer Slide Time: 29:41)



## Logic Synthesis

- Introduction
- Logic Optimisation (technology independent)
  - Two-level optimisation
  - Multi-level optimisation
- Technology Mapping / Cell library binding (technology dependent)
  - ASIC
  - FPGA

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

How to do it, efficiently, we may not have studied, but certainly how to do it, how to represent this and at least a visual solution involving the k map, you might have already seen that. I am essentially representing a truth table of all the values of the inputs and trying to come up with a solution that minimizes the number of literals or indirectly reduces area optimization for area by minimizing the number of literals in a Boolean algebraic expression.

So, we start off from there, we will first look at some ways of representing our designs, then we will look at essentially two phases, the logic synthesis can be thought of as consisting of initially a logic optimization phase that is technology independent the technology independent means that it does not matter what particular ASIC library or FPGA or whatever the target architecture is, we can do some logic optimization anyway that is at a little abstract level the sort of k map of optimization that we have seen belongs to this level.

We do not really care how you will finally, implement it, but we go through that came up optimization in a way that you just try to minimize the number of literals hoping that it translates to a minimized circuit no matter what gates you use assuming of course, that they are reasonable standard cells that have already been designed.

Still the strategy would be different based on whether you are targeting a two level architecture or a multi levelled architecture that we had already seen in the context of the FSM encoding right. So, it is the same thing.

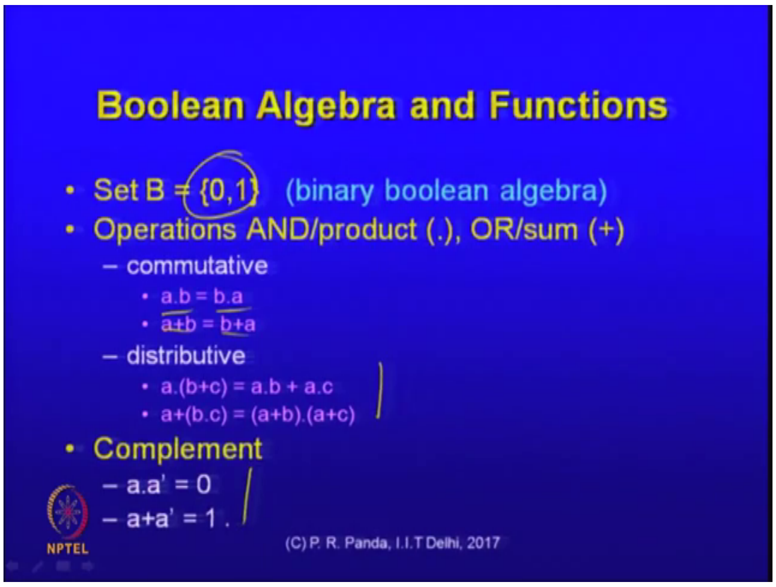
So, that much you do need to know, but within that if you know two level, then there is a certain kind of optimization if you know, it is multi level, it should be the different kind of optimization that is one, the second step would be a technology mapping or a cell library binding both are kind of equivalent to each other. This would take us from an optimized result from the first phase into something that is now generating an optimized netlist for the particular target architecture that we have in mind.

Here it may be a particular ASIC library or a specific FPGA because those numbers that we use are specific to a particular library area numbers delay numbers and. So, on if you change the libraries and all of those numbers will change and therefore, the selected netlist might be different optimized netlist might be different in the first phase we do not worry about all that this is again in the interest of simplification of the passes we just try to minimize the number of literals for example, in an area you know optimization.

Without necessarily taking into account the area of the NAND gate area of the inverter and so on, the overall synthesis step is divided into these two phases the first of which is a little abstract of course, logic synthesis itself is more detailed compared to the other synthesis steps that we have seen, but within that two you can think of a little higher level of abstraction and a lower level.

So, the lower level would be taking data from the library elements and optimising in those dots ok.

(Refer Slide Time: 33:32)



## Boolean Algebra and Functions

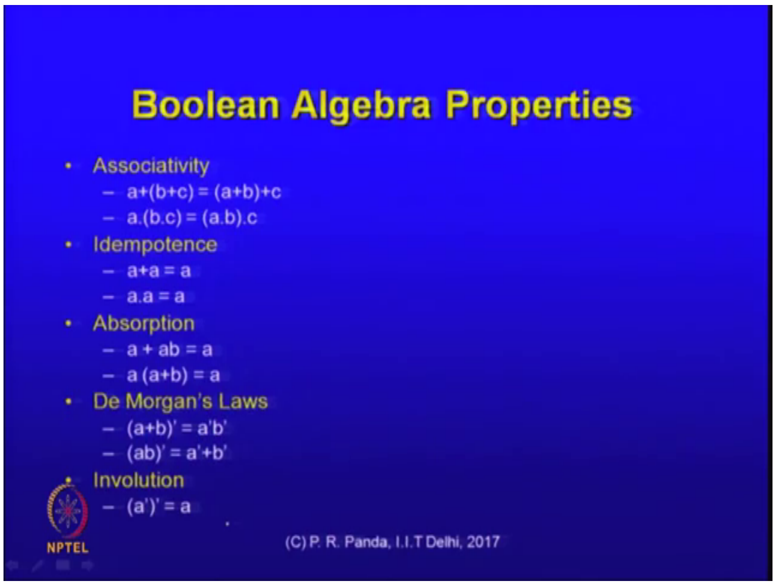
- Set B = {0,1} (binary boolean algebra)
- Operations AND/product (.), OR/sum (+)
  - commutative
    - a.b = b.a
    - a+b = b+a
  - distributive
    - a.(b+c) = a.b + a.c
    - a+(b.c) = (a+b).(a+c)
- Complement
  - a.a' = 0
  - a+a' = 1 .

(C) P. R. Panda, I.I.T Delhi, 2017

And before we can get along with synthesis, let us just do a quick recap of Boolean algebra, in case you have forgotten some of it, this B being a set of 0 1 refers to what the rest of the algebra is built in terms of variables that take values only 0 and one operations are defined AND, OR and so on.

There are these properties of each of those operations commutative means that a dot b equals b dot a, a plus b is b plus a, there are rules for distribution there are rules for complement.

(Refer Slide Time: 34:07)



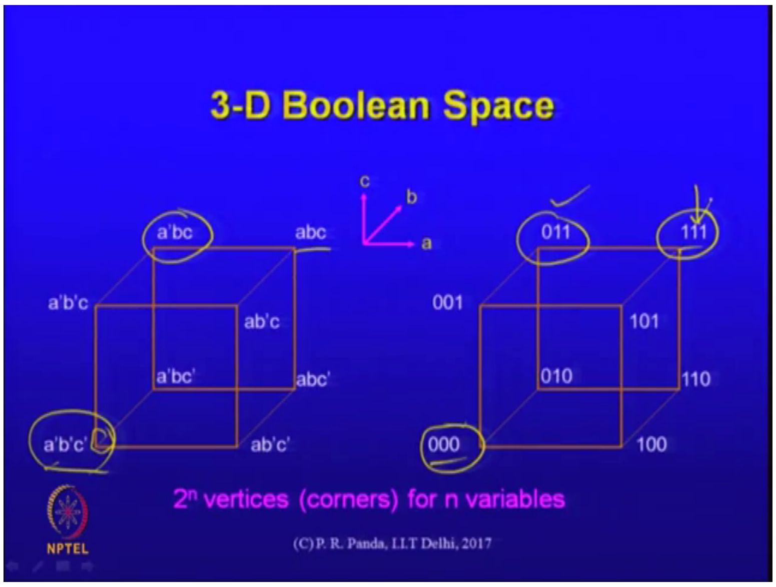
## Boolean Algebra Properties

- Associativity
  - a+(b+c) = (a+b)+c
  - a.(b.c) = (a.b).c
- Idempotence
  - a+a = a
  - a.a = a
- Absorption
  - a + ab = a
  - a (a+b) = a
- De Morgan's Laws
  - (a+b)' = a'b'
  - (ab)' = a'+b'
- Involution
  - (a')' = a

(C) P. R. Panda, I.I.T Delhi, 2017

There are a number of other properties that should just be a quick recap associativity means, it does not matter whether you pick up b plus c, first or a plus b first when you have such a sequence of operations absorption these rules ought to be familiar De Morgan's laws are to be familiar. So, these this is all familiar space.

(Refer Slide Time: 34:29)



## 3-D Boolean Space

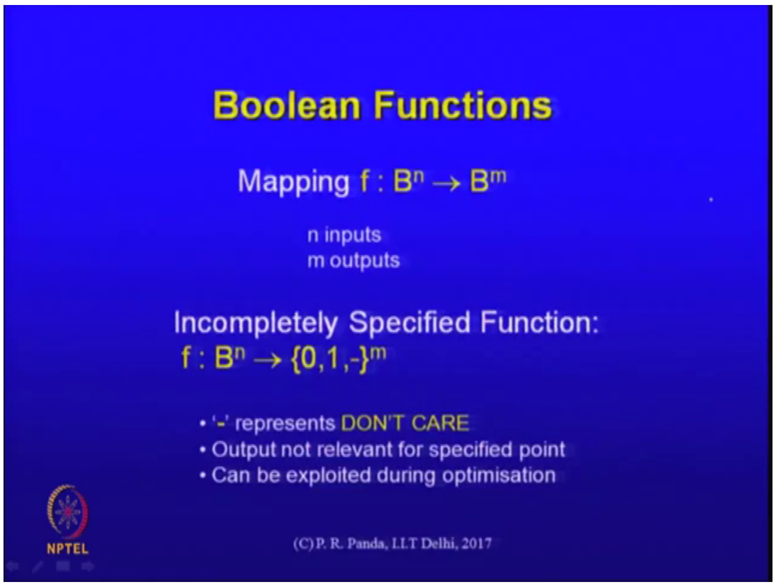
$2^n$ vertices (corners) for n variables

(C) P. R. Panda, I.I.T Delhi, 2017

Let us introduce some conceptual representations to enable us to argue about specific synthesis optimizations later on let us define a 3-D Boolean space the 3 arising from just as having a function of 3 variables. So, assume I have a b c as my 3 variables, then you can imagine a cube in which every vertex or every corner corresponds to 1 min term, if I can label it with 0s and 1s. So, one of this is 0 0 0 that is 1 0 1, this is 0 1 1 and so on.

There are 8 vertices in this 3 dimensional cube and in some straightforward way, you can map a min term on to each one of those vertices 0 0 0 means that all the a b c are there in complemented form all ones means that I have a b c uncomplimentary and some node like 0 1 1 means that it corresponds to a prime b c. So, the position here refers to the variable a, b or c, you can see how to generalize that if there were n variables then you would have an n dimensional space, all right.

(Refer Slide Time: 35:55)



## Boolean Functions

Mapping $f : B^n \rightarrow B^m$

n inputs
m outputs

Incompletely Specified Function:
$f : B^n \rightarrow \{0,1,-\}^m$

- '-' represents DON'T CARE
- Output not relevant for specified point
- Can be exploited during optimisation

(C) P. R. Panda, I.I.T Delhi, 2017

Then the Boolean function in general is a mapping from a vector of size n to a vector of size m where the function has n inputs and m outputs that is what that function is still.

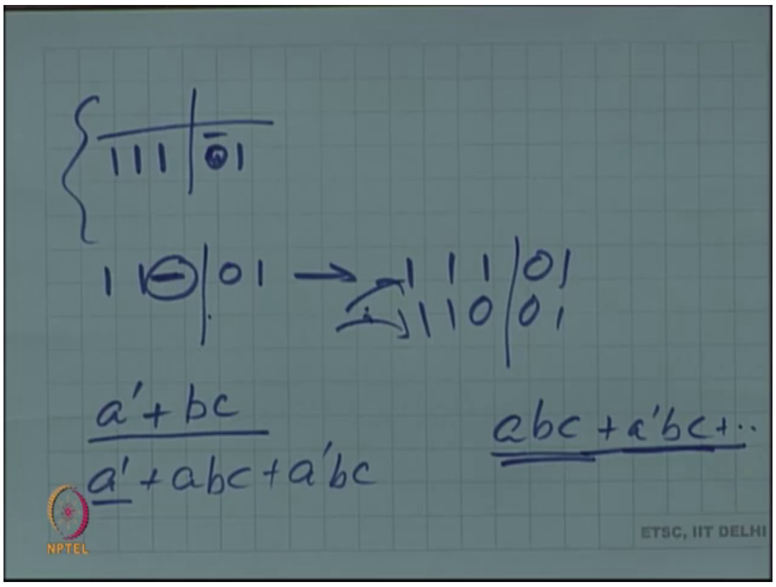
Here we can also introduce do not care in completely specified function means that that do not care could be there on the output, it is not just a Boolean which is just 0 and 1, but every one of those m values on the output could also be do not care.

How about on the input, should I extend this to also have do not care on the input, do I have any more information, for example, in this case, where I have a do not care on the output also, I do not care either on the output is that giving us any more information than if I were to have only Boolean values on the output.

Student: Yes, you can reduce the literals column cubes.

Yeah, it is giving us something more by saying that I do not care about a particular output value for an input combination I am giving some flexibility to the tool and say 0 is ok, one is also that is different from me insisting that I must put a 0 or a 1 there, right. So, in some ways, it is actually more information it is a different function, but on the input side.

(Refer Slide Time: 37:45)



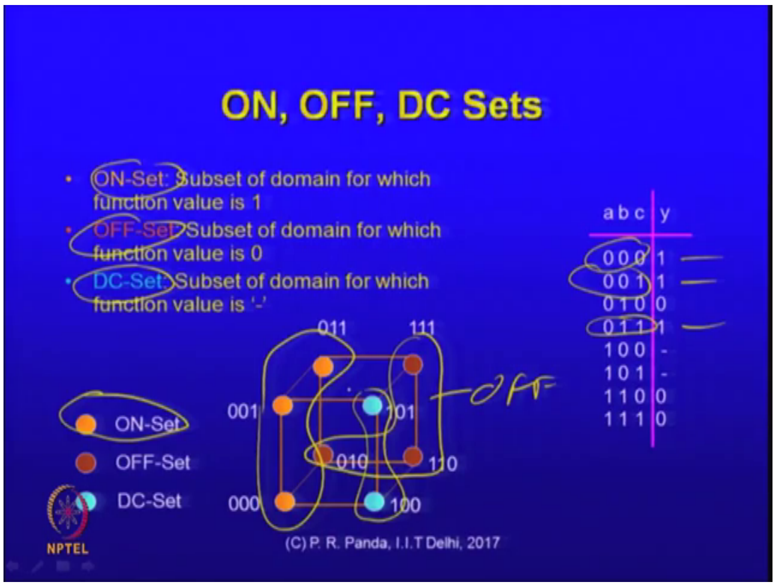
So, the input vector; if it was like that let us say, I have an output vector like that instead, if I said 1 1, this the output is 0 1 am I giving any more information, this was a truth table, right function means that for every input combination, I should specify the output combination that is a truth table in a truth table if 0 here is replaced by do not care, it means something technically, it could be a different function, right on the input side if I replace a 1 or a 0 by a do not care, what am I essentially doing.

Student: I am saying that the output function is now depending only 2 variables instead of 3.

Well, the equivalent way of looking at it is this is nothing, but I am saying 1 1 1 0 1 and 1 1 0 0 1, this is a convenience for me by just saying, this is do not care, I am essentially referring to two rows in the truth table, but I have not given you any more information these two are the same right. So, this is a way of compacting the truth table compacting the input, but I am really not telling much more about function itself.

On the other hand, putting the do not care on the output, I am saying something about the function. So, that is why I have still be on the domain side, it is a core domain where it is essential that I should put it do not care, let us define a few other things.

(Refer Slide Time: 39:25)



## ON, OFF, DC Sets

- ON-Set: Subset of domain for which function value is 1
- OFF-Set: Subset of domain for which function value is 0
- DC-Set: Subset of domain for which function value is '-'

| abc | y |
|---|---|
| 000 | 1 |
| 001 | 1 |
| 010 | 0 |
| 011 | 1 |
| 100 | - |
| 101 | - |
| 110 | 0 |
| 111 | 0 |

● ON-Set
● OFF-Set
● DC-Set

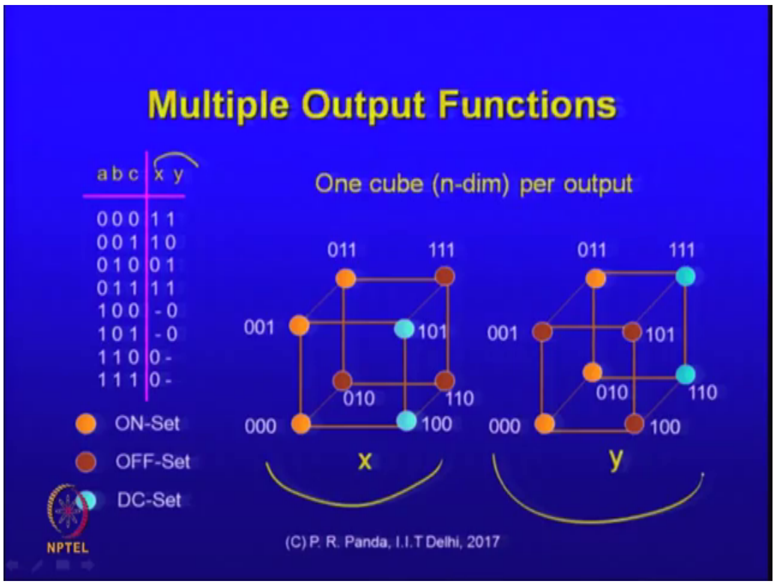
(C) P. R. Panda, I.I.T Delhi, 2017

We are talking about these vertices on that n dimensional space right, I am using 3 dimensional examples, all through, this discussion is more or less synchronized with the logic synthesis discussion in the textbook, all right and this representation is also from there ok.

Let me define 3 sets an on set and off set and a do not care set so on set is the subset of the domain for which the function value is 1 ok. So, those are all of these points I have 3 of these rows having a 1 at y. So, 0 0 0 at 0 0 1 and 0 1 1 that is these 3 points are in the onset of the functions, similarly, offset consists of a 3 points ok.

So, that is off and I have these two points in the do not care set every point in that space every vertex in that space belongs to one of these sets.

(Refer Slide Time: 40:46)



## Multiple Output Functions

| abc | xy |
|---|---|
| 000 | 11 |
| 001 | 10 |
| 010 | 01 |
| 011 | 11 |
| 100 | -0 |
| 101 | -0 |
| 110 | 0- |
| 111 | 0- |

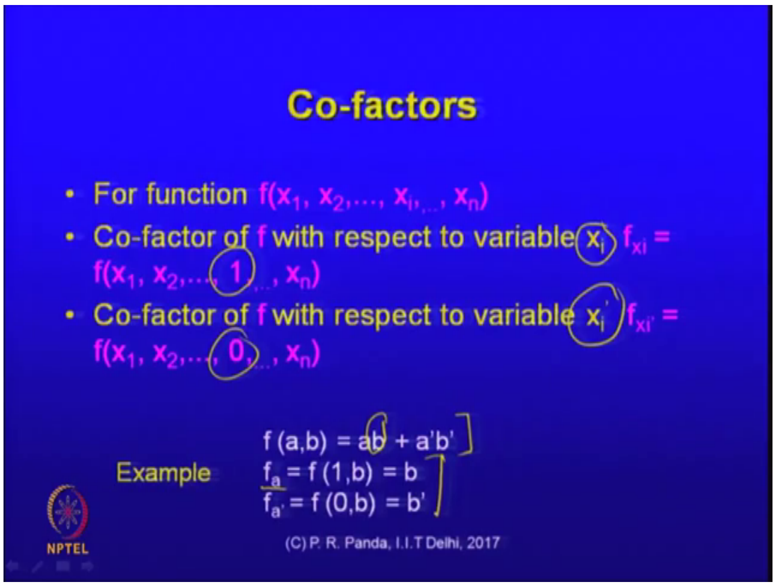
| | |
|---|---|
| ● | ON-Set |
| ● | OFF-Set |
| ● | DC-Set |

One cube (n-dim) per output

(C) P. R. Panda, I.I.T Delhi, 2017

I may have multiple outputs right a function may have more than one if yes, then I just have a different cube, it is an n dimensional cube in general drummer, I have a different cube for each output for x, I have a cube and for y, I have a different cube ok. So, keep that in mind that is all that is there, we will use that in various ways later on.

(Refer Slide Time: 41:11)



## Co-factors

- For function $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$
- Co-factor of f with respect to variable $x_i$ $f_{x_i} = f(x_1, x_2, \ldots, 1, \ldots, x_n)$
- Co-factor of f with respect to variable $x_i'$ $f_{x_i'} = f(x_1, x_2, \ldots, 0, \ldots, x_n)$

Example

$f(a,b) = ab + a'b'$

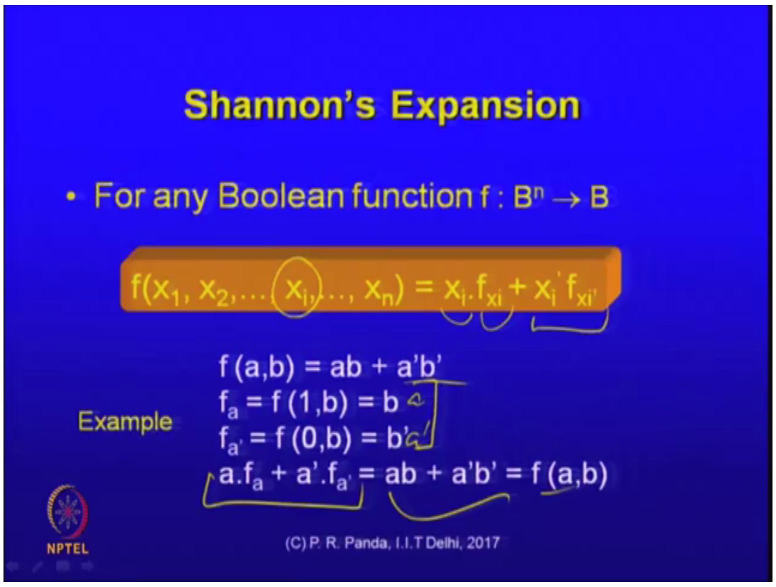
$f_a = f(1,b) = b$

$f_{a'} = f(0,b) = b'$

(C) P. R. Panda, I.I.T Delhi, 2017

Let us introduce the idea of a cofactor. So, given a function of n variables x 1 to x n the cofactor of f with respect to a variable x i is obtained by just substituting x i equals 1 in that function and the cofactor with respect to x i prime is obtained by just substituting 0 wherever x i occurs in that function ok. So, example is if this was my original function and cofactor of f with respect to a is if you substitute a equals 1 right. So, if a equals 1 means you get b from here and this becomes 0. So, you have b f a prime is if you substitute 0 here which will pick up b prime b those are mine cofactors is just a simple definition.

(Refer Slide Time: 42:21)



## Shannon's Expansion

- For any Boolean function $f : B^n \rightarrow B$

$$f(x_1, x_2, \ldots x_i, \ldots, x_n) = x_i . f_{x_i} + x_i' f_{x_i'}$$

Example

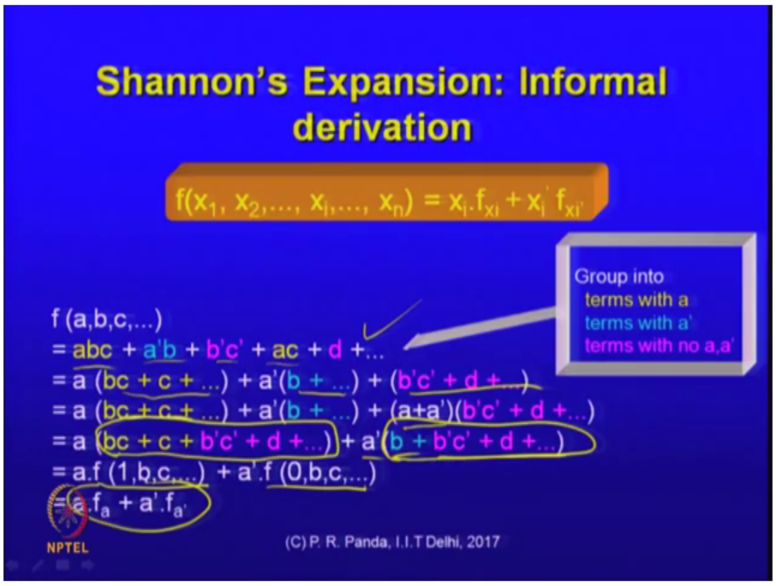
$$f(a,b) = ab + a'b'$$
$$f_a = f(1,b) = b$$
$$f_{a'} = f(0,b) = b'$$
$$a.f_a + a'.f_{a'} = ab + a'b' = f(a,b)$$

(C) P. R. Panda, I.I.T Delhi, 2017

Let us look at one standard result called Shannon's expansion, any Boolean function of n variables can be expressed like this x i, you take any one of those variables x i times cofactor of f with respect to x i plus x i prime times the cofactor of f with respect to x i prime ok, in that example that we just saw this was the function, we determined the cofactors with respect to a and a prime, if you just substitute a f a plus a prime f a prime, you would get essentially this is multiplied by a that is multiplied by a prime you get back the original function itself.

Why this is true in general it is not hard to see we can informally derive it in the following way.

(Refer Slide Time: 43:17)



## Shannon's Expansion: Informal derivation

$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i . f_{xi} + x_i' f_{xi'}$

Group into
terms with a
terms with a'
terms with no a,a'

$$
\begin{aligned}
& f(a,b,c,\ldots) \\
& = abc + a'b + b'c' + ac + d + \ldots \\
& = a(bc + c + \ldots) + a'(b + \ldots) + (b'c' + d + \ldots) \\
& = a(bc + c + \ldots) + a'(b + \ldots) + (a+a')(b'c' + d + \ldots) \\
& = a(bc + c + b'c' + d + \ldots) + a'(b + b'c' + d + \ldots) \\
& = a.f(1,b,c,\ldots) + a'.f(0,b,c,\ldots) \\
& = a.f_a + a'.f_{a'}
\end{aligned}
$$

(C) P. R. Panda, I.I.T Delhi, 2017

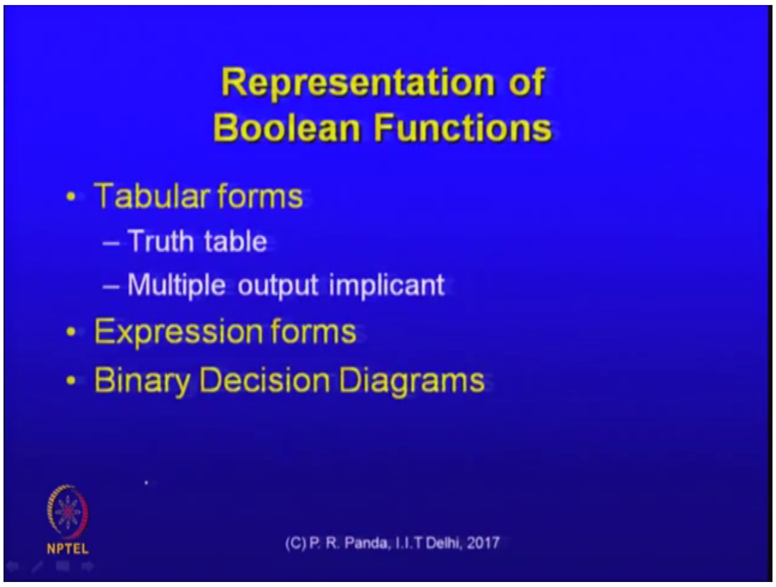
Suppose, you have some function of a large number of variables and I would like to pick up one of them, let us say these cofactors with respect to a I am trying to pick up then this expression is coloured in the following way the yellow terms are those that have an a in them, the blue terms are those that have an a prime in them and the rest of term the pink terms have neither a nor a prime ok.

So, if I group them, I have a times all of these plus a prime times, all of these arising out of the yellow and the blue terms and these that have neither and a nor and a prime, let me put them into those expressions in the following way, let me multiply them by a plus a prime nothing changes because that is 1. So, then if I regroup the terms, I essentially I am multiplying a by all the yellow terms plus all the pink terms right.

And similarly a prime is also multiplied by the blue terms and all the pink, the pink terms go in both the places, but what is this expression anyway, this is nothing, but the cofactor of that original expression with respect to a you drop all the a prime terms right and you drop as from the ones that have a that is the yellow terms and the pink terms since they have neither a nor a prime they would show up in both the cofactors right. So, that is what I have here.

Similarly, this is the cofactor of f with respect to a prime and so that is what I have. So, that is right, my you can see this is an informal, but you can easily generalize this into a formal proof if you wanted to this is a result that is used at various places in logic synthesis. So, it is as sort of a very fundamental result ok.

(Refer Slide Time: 45:30)



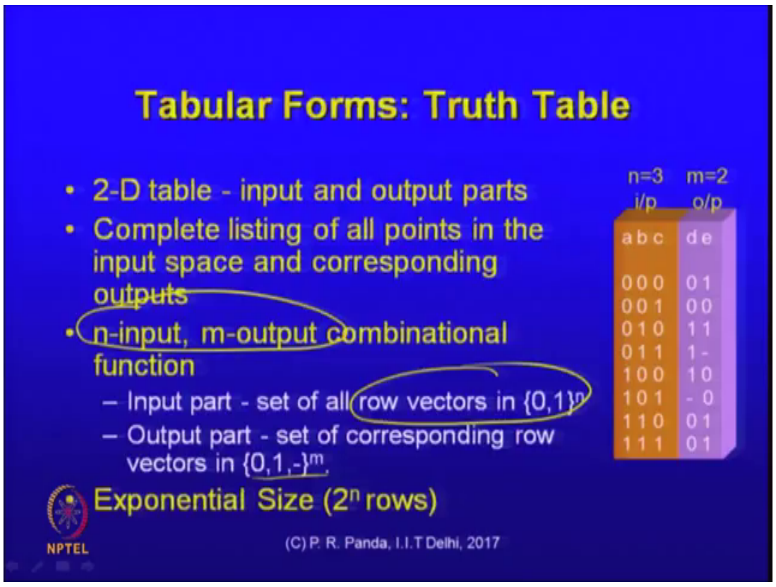
## Representation of Boolean Functions

- Tabular forms
  - Truth table
  - Multiple output implicant
- Expression forms
- Binary Decision Diagrams

(C) P. R. Panda, I.I.T Delhi, 2017

Let us quickly move on to representation of the Boolean functions starting off with the tabular forms because those are the ones that are familiar truth table is essentially a two dimensional table in which there is an input part.

(Refer Slide Time: 45:42)



## Tabular Forms: Truth Table

- 2-D table - input and output parts
- Complete listing of all points in the input space and corresponding outputs
- n-input, m-output combinational function
  - Input part - set of all row vectors in $\{0,1\}^n$
  - Output part - set of corresponding row vectors in $\{0,1,-\}^m$
- Exponential Size ($2^n$ rows)

| n=3 i/p | m=2 o/p |
|---|---|
| a b c | d e |
| 0 0 0 | 0 1 |
| 0 0 1 | 0 0 |
| 0 1 0 | 1 1 |
| 0 1 1 | 1 - |
| 1 0 0 | 1 0 |
| 1 0 1 | - 0 |
| 1 1 0 | 0 1 |
| 1 1 1 | 0 1 |

(C) P. R. Panda, I.I.T Delhi, 2017

And an output part; we have already seen this, this is just a recap, what characterizes the truth table is that it is a complete listing of all the points or the vertices of the input space every row corresponds to one of those vertices in the cube and if you have an n input and m output combinational function how many rows are there in the truth.

Student: Exponential.

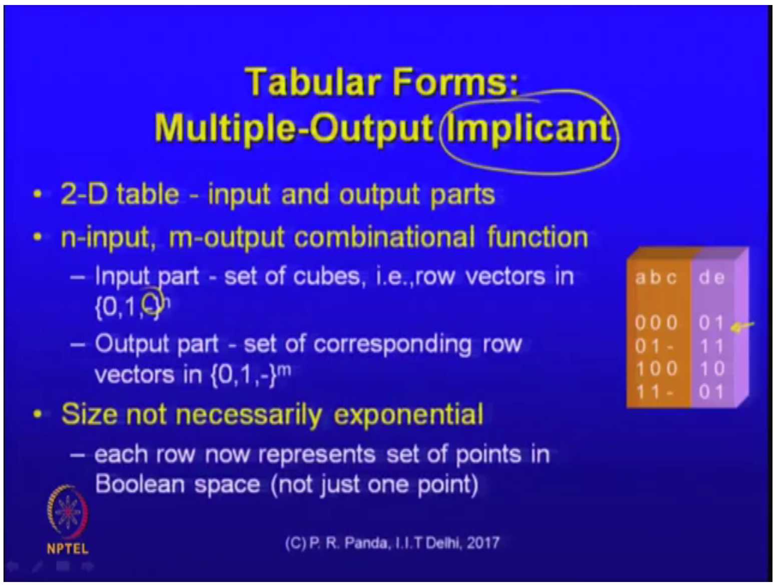
Exponential in;

Student: 2 n n.

In n; right the number of inputs is what it is exponential in terms of right does not matter how many outputs are there, right. So, all the row vectors that you could compose of 0s and 1s of length n is what shows up there, each becomes a separate row and output part, we will say, it can have 0, it can have 1, it can also have do not care. So, it is a vector of size m n. So, this of course, is a nice way to visualize small functions problem.

Of course is that explicitly doing it and manipulating the truth table becomes inconvenient when the number of variables becomes large , but still that is the first thing you can start off with.

(Refer Slide Time: 47:01)



## Tabular Forms: Multiple-Output Implicant

- 2-D table - input and output parts
- n-input, m-output combinational function
  - Input part - set of cubes, i.e.,row vectors in $\{0,1,-\}^n$
  - Output part - set of corresponding row vectors in $\{0,1,-\}^m$
- Size not necessarily exponential
  - each row now represents set of points in Boolean space (not just one point)

| abc | de |
|---|---|
| 000 | 01 |
| 01- | 11 |
| 100 | 10 |
| 11- | 01 |

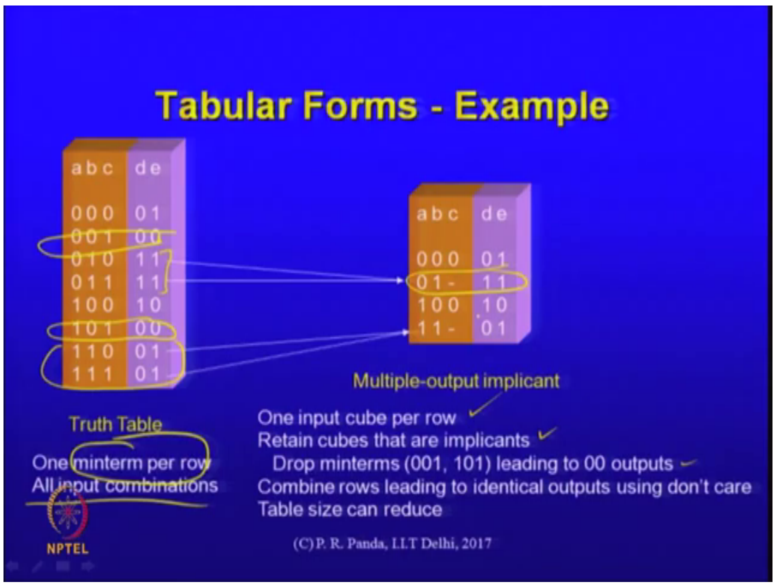
(C) P. R. Panda, I.I.T Delhi, 2017

You can generalize it in the following way still a tabular form the Multiple-Output Implicant is a sort of a compacted version of the truth table itself this is also two dimensional table there is an input part and an output part, but the input part you are allowed to have a do not care which could be used to drop some of your terms if you can properly find those combinations, you look at two rows and if the results are the same, then you could combine them with the do not care some way ok.

Output is still same, but being an Implicant here means that if there are rows for which all the outputs are 0 then we drop it we will define formally Implicant later on, but a cube is an Implicant, if it leads to an output of one n for at least one of the outputs if all of them are 0,, then we we drop them that is a way of reducing the size of that table this size is not necessarily exponential the worst case, it is exponential.

In fact, you have functions that have an exponentially large number of Implicants there is nothing you can do about it, but often it could be that this could be used to compact the specification.

(Refer Slide Time: 48:36)



## Tabular Forms - Example

| abc | de |
|---|---|
| 000 | 01 |
| 001 | 00 |
| 010 | 11 |
| 011 | 11 |
| 100 | 10 |
| 101 | 00 |
| 110 | 01 |
| 111 | 01 |

Truth Table

One minterm per row
All input combinations

| abc | de |
|---|---|
| 000 | 01 |
| 01- | 11 |
| 100 | 10 |
| 11- | 01 |

Multiple-output implicant

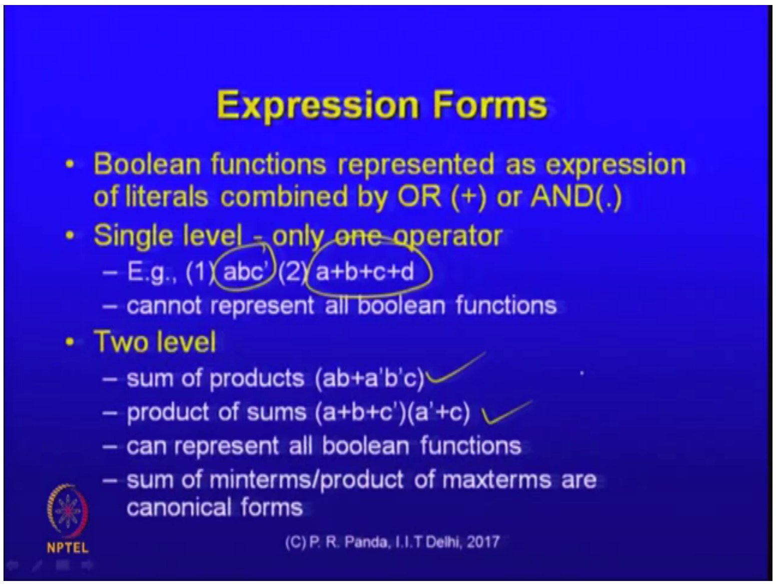
One input cube per row
Retain cubes that are implicants
Drop minterms (001, 101) leading to 00 outputs
Combine rows leading to identical outputs using don't care
Table size can reduce

(C) P. R. Panda, I.I.T Delhi, 2017

So, those are just the two tabular forms that we look at one is truth table you have one min term per row and all the input combinations , but in that multiple output Implicant table, you have one input cube per row that is not necessarily a min term per row because many min terms can be combined into a single cube you retain those cubes that are Implicants means that you drop this cube and you drop this cube because the output is 0 0 all the outputs are 0, then you drop it from the table.

Then you can see whether there are rows that actually lead to the same outputs. So, you have two rows with outputs being same and I have 0 1 0 and 0 0 1 that could be compacted into 1 row in that table same thing here. So, that is a way of a tabular representation that sometimes at least can avoid blowing up of the number of rows with the number of entries are essentially the size of the data structure that you are using to capture the function.

(Refer Slide Time: 49:54)



## Expression Forms

- Boolean functions represented as expression of literals combined by OR (+) or AND(.)
- Single level - only one operator
  - E.g., (1) abc' (2) a+b+c+d
  - cannot represent all boolean functions
- Two level
  - sum of products (ab+a'b'c)
  - product of sums (a+b+c')(a'+c)
  - can represent all boolean functions
  - sum of minterms/product of maxterms are canonical forms

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

So, those are the tabular forms let us move on to other representations expression forms are ones that are common, of course, you can have Boolean functions represented as expression of literals that are combined by some operators or an and it could be a single level expression which means that there is only one operator.

So, literal can be a Boolean variable or its complement it, both are a single literal single level means that you have only one operator that is operating on possibly several literals. So, that is one example this is one example, but this is too simple, it cannot represent all Boolean functions we certainly need to be able to represent all the Boolean functions.

Two level is of course, something that we should know, it could be a sum of products, it could be a product of sums this is good enough for us to represent all Boolean functions, of course, sum of min terms or equivalently product of max terms are what are called canonical representations canonical representation means that given a function, there is only one way to represent that sum of min terms you can see a given Boolean function can be represented using many different expressions.

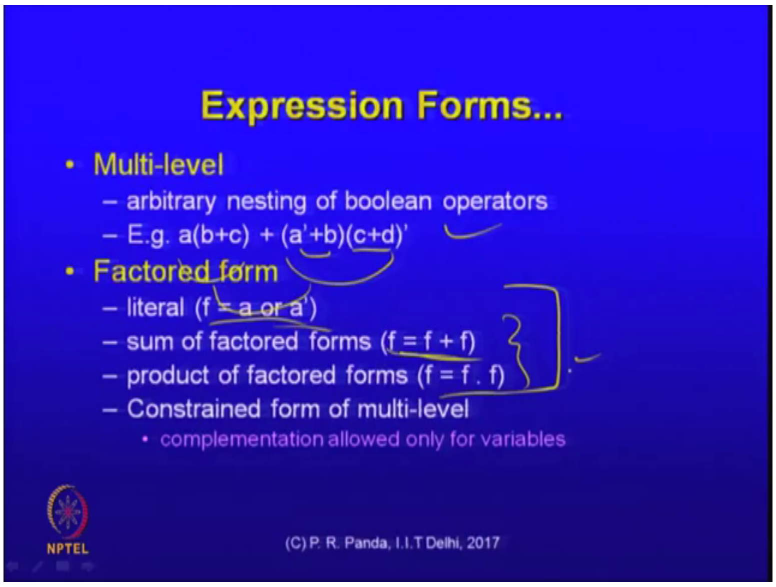
But in terms of min terms, it is still the same right. So, your expression could be a prime plus b c right this is a more compacted form that expression could be written in other ways also for example, this can be a prime plus b c can be expanded as a b c plus a prime b c, right and similarly this can be expressed.

So, there would be many different ways of writing that same function, but what we are saying is that if I reduce it to only a sum of min terms form which is all the variables are there either complemented form or in the non complemented form and so on, then there is only one way to represent every function that is obvious right because that essentially corresponds to taking the set of on points in my truth table or in my 3-D space or n dimensional space it is just a set.

Of course, the order in which you write these could be different, but this is essentially a set of min terms that we have taken a set does not have ordering. So, there is only one way to represent this max term is the equivalent thing, if you do not use some of product, but you use product of sums. So, both of these are canonical all the other forms are not necessarily canonical that multiple output Implicant table that is not necessarily canonical truth table is canonical.

Of course, but the other one that was showed this one is not necessarily connected you could actually have different tables that are derived from the same truth table.

(Refer Slide Time: 53:03)



## Expression Forms...

- Multi-level
  - arbitrary nesting of boolean operators
  - E.g. a(b+c) + (a'+b)(c+d)'
- Factored form
  - literal (f = a or a')
  - sum of factored forms (f = f + f)
  - product of factored forms (f = f . f)
  - Constrained form of multi-level
    - complementation allowed only for variables

(C) P. R. Panda, I.I.T Delhi, 2017

Expression form could also be multi level multi level, essentially means that you have an arbitrary nesting of Boolean operators, you can have plus operator you have and operator to the result of the plus you have invert operated operating on the result of a plus and you have this being a different function, you have combined in different. So, any level of hierarchy could be there could also have a factored form that somewhere in between where what is allowed is here.

So, you can have a literal that literal is a variable or its compliment, you can have a sum of factored forms, you can have a product of factored forms, this is a grammatical specification f equals f plus f means a new factored form is generated from two other factored forms by applying the plus operator, it could be a product.

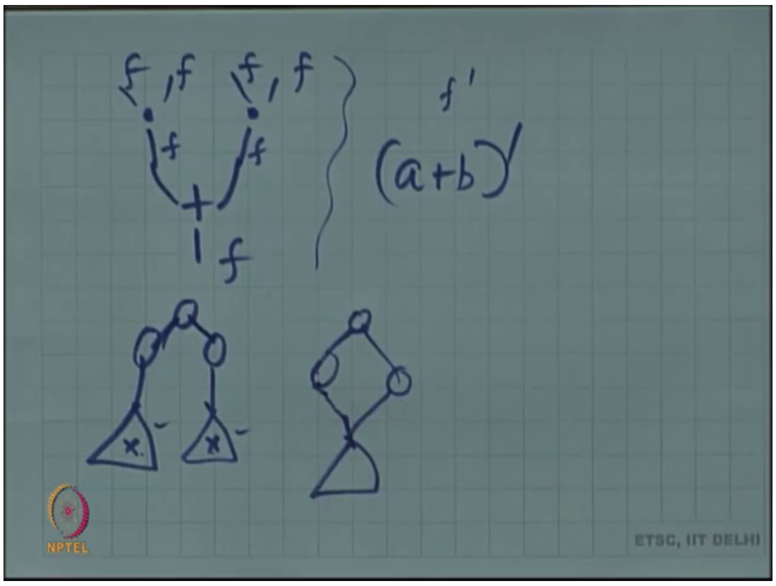
This is a constrained form of that general multilevel representation in what way, if this were the only way for me to generate expressions, how is it a constraint form, what is it that you can do here that you cannot do here, you can have compliments, you can have sum of things, you can have product of things which allows hierarchical composition or not of any depth of that expression.

Student: We (Refer Time: 54:51) multi level we cannot write.

Yeah, we are just trying to see what is the difference how is this more constrained multi level is the most general form. So, how is this different if I say that you are allowed a literal a literal is the factored form some factored forms is a factored form product of factored forms is also a factored form what have I left out.

Student: Sum of products of factored form.

(Refer Slide Time: 55:21)

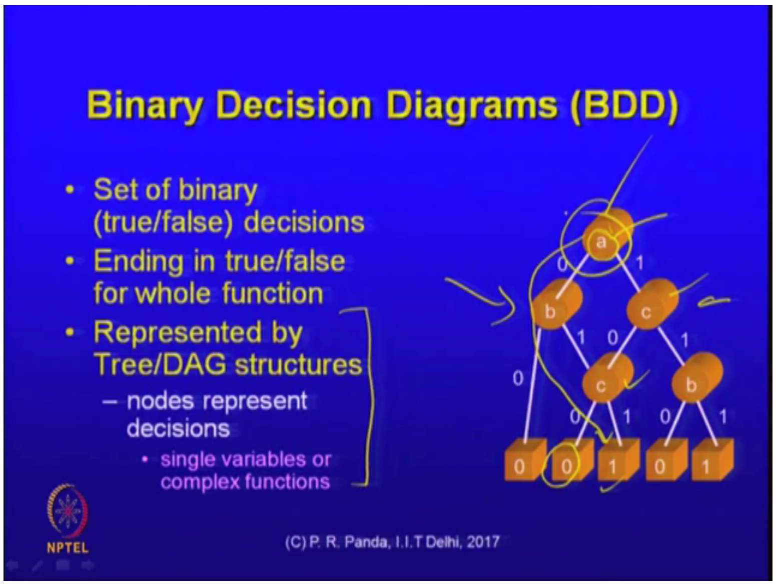


No, to product of two factored forms is also a factored forms ok. So, that is also a factored form plus of two factored forms is also a factored form. So, this is fine.

Student: f into f dash.

Yeah, see f prime is not there which means a plus b prime is not a factored form literal can be complemented, but an expression cannot be more complex expression cannot be complement. So, anyway that is one way, it is a simplified version of the multi level logic, this is also complete in the form of you can express any Boolean function in factored form simply because sum of products is fine.

(Refer Slide Time: 56:13)



## Binary Decision Diagrams (BDD)

- Set of binary (true/false) decisions
- Ending in true/false for whole function
- Represented by Tree/DAG structures
  - nodes represent decisions
    - single variables or complex functions

a
0
1
b
c
1
0
1
0
c
b
0
1
0
1
0
0
1
0
1

(C) P. R. Panda, I.I.T Delhi, 2017

So, sum of min terms is fine and that is good enough for us to capture anything these are all trivial data structures that we have seen.

So far, the binary decision diagram is one on which we can spend some time this is a data structure that was designed to capture Boolean functions in a somewhat non trivial way let us define the structure itself and some very interesting properties that we can study of these BDDs binary decision diagrams.

This diagram consists of a set of decisions the value being true or false the value of an expression or value of a variable being true or false essentially is what we mean by that decision, this is such a structure each of the nodes corresponds to an evaluation of something what that thing is we have to say, but we will label it with that expression an expression is evaluated and if we are at a node we evaluate that expression.

If it is true, then we go one level down and evaluate whatever is the expression at that node if the result were false then we go down and whatever is the expression there we evaluate. So, there is a series of decisions that we take at each step, we do an evaluation and based on it is true or false, we go down one path or the other of this diagram ultimately, we end up with a leaf level which is essentially true or false. So, that is all that is there if that path is false then the value of that function that is captured by the data structure is false for those expressions that we evaluated along the way.

So, you can see that if I attach a variable name, there what I am saying is if I follow that path down right if a were 0 and b were 1 and c were 1, then the value of that function is 1; that is what we are trying to capture, but in general a decision diagram leaves this unspecified though you can have a there you can also have a more complex expression there, we will constrain the definition for it to be more meaningful, but a decision diagram in general could be anything binary just means that there are two results 0 and 1, but you could have more complex expressions there and that is also fine.

We will try to avoid that we will try to come up with a subset that is more meaningful, but this is what is a binary decision diagram at the end of sequence of decisions you have the result true or false and the true or false means that the function value is true or false for that path that you took this is represented by a tree or a dag kind of a structure.

Student: Tree kind of any operator kind of length between the literals.

Everything has to be implicitly captured by these decisions, but we will see later that if I have an operation what does that become in the industry, but this is an an expression essentially, of course, the data structure is meaningful if any Boolean function I am able to represent in this data structure, right.

Student: This if for this is for one output if I have multi valued output, then I will have different trees for each of the output.

Yeah, we will get that, but this is just a single.

Student: All paths will become a sum to give the output in this. So, if I have;

We may be jumping ahead, but what we would actually do is we build one structure, but each node would correspond to some function, right. So, if I had more than one output, I would still build one structure, but the root would have something some other node would have something.

So, I will try to capture all of them in one structure why one structure because we may want to reuse some of the decisions, but we will get there. So, it is represented in the tree as you can see this is the root of that tree well whether it is a tree or a dag essentially there is a direction that is implied in the structure which is you traverse this structure from top to;

Student: Bottom.

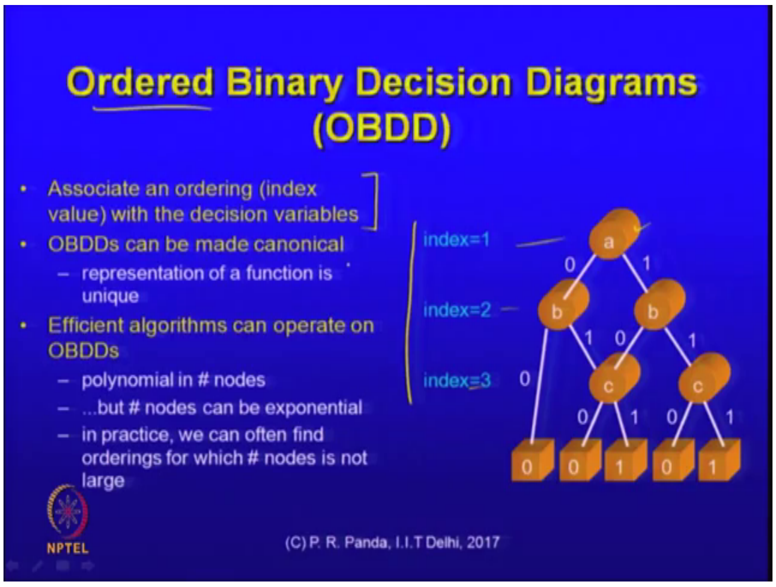
Bottom dag or a tree, it is just that you start from the top, but let us say if you have a structure like this and it turns out that these are actually identical.

So, this decision diagram in some obvious ways can be thought of as a tree, you start from the top two directions, then you get to the next one two directions. So, it is a tree kind of a structure in general, but you can end up with some of these sub trees that are actually identical to each other, then you could argue why have two different instead the structure should be such that you have only one tree this is just the reason, why it is not in general a tree structure, but a dag structure it is because there are common sub trees that you might want to;

Student: Exploit.

Exploit equivalent tree is of course, also there, but if they are there in order to minimize the size of the data structure, we would possibly end up with a dag structure in this tree, what we are formally doing is the nodes represent decisions that is fine, but for us it could be a single variable or a complex function, again, we are just talking about a general binary decision diagram there are some other decisions that we can make first.

(Refer Slide Time: 62:04)



## Ordered Binary Decision Diagrams (OBDD)

- Associate an ordering (index value) with the decision variables
- OBDDs can be made canonical
  - representation of a function is unique
- Efficient algorithms can operate on OBDDs
  - polynomial in # nodes
  - ...but # nodes can be exponential
  - in practice, we can often find orderings for which # nodes is not large

(C) P. R. Panda, I.I.T Delhi, 2017

index=1
index=2
index=3
a
b
b
c
c
0
1
0
1
0
1
0
0
1
0
1
0
0
1
0
1

Let us define an ordering of the binary decision diagrams what we are doing is you have your variables a, b, c suppose, it is a function of 3 variables that we want to design then you associate an ordering with these decision variable. So, these decision variables in this BDD are nothing, but our Boolean inputs to the function and you impose an ordering where you say a corresponds to index 1 b corresponds to index 2 and c corresponds to index 3 that is where the ordering comes from in the name.

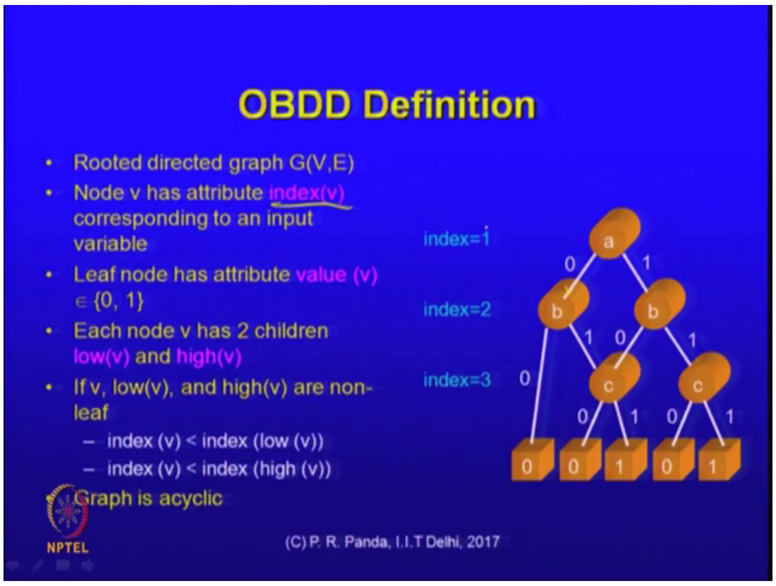
So, we are moving in the direction of making that structure canonical that is why these limitations are being imposed, but the one thing is let us get out of this mode of any expressions and instead the each of these nodes is associated with a variable one of the input variables in this case, it can be a or b or c these OBDDs can be made canonical which means that if you have one function you have exactly one representation.

We need to do something more to that, but the ordering is essential if for that same function you change the ordering then you would have a different BDD structure. So, an ordering is defined in this representation there is an implicit ordering that it is defined in terms of an ordering. So, why do these efficient algorithms can operate on the ordered BDDs. So, we are moving in the direction of minimizing the size of the data structure, but also improving the efficiency of whatever manipulation we want to do later on with that data structure.

So, those algorithms can be polynomial in terms of the number of nodes. So, it is good to have the smaller number of nodes for that structure, but of course, some functions may be such that your number of nodes may be exponential anyway that is just the general case that a typical function may be compact in terms of its BDD representation OBDD means you have an index that is associated with the variables, you do not necessarily need a node at every stage after b if b were one, then you moved in one direction you move to a different node if b were 0, you are directly saying that the output is 0, we did not go through another level of nodes there.

But that is all right; that is just the way that function is essentially, we are saying that if a is 0, then b is 0, then irrespective of the value of c the result is 0 on the other hand, if a is one and b is 0, then depending on the value of c the result is n 0 or 1.

(Refer Slide Time: 65:03)



## OBDD Definition

- Rooted directed graph G(V,E)
- Node v has attribute index(v) corresponding to an input variable
- Leaf node has attribute value (v) ∈ {0, 1}
- Each node v has 2 children low(v) and high(v)
- If v, low(v), and high(v) are non-leaf
  - index (v) < index (low (v))
  - index (v) < index (high (v))
- Graph is acyclic

index=1
index=2
index=3
a
b
b
c
c
0
1
0
1
0
1
0
1
0
1
0
0
0
1
0
1

(C) P. R. Panda, I.I.T Delhi, 2017

That is just the informal specification n, but let us just formally define it that OBDD is a rooted directed graph, I did not put the directions here, but implicit in this figure is that we are moving from the top to the bottom ok. So, it is a directed graph where a node v has an attribute of an index every node has an index ok.

So, node a is associated with index 1. So, that index tells us the ordering of that variable yeah.

Student: In terms of data structure implementation. So, the left most path which is a b and then we go to 0.

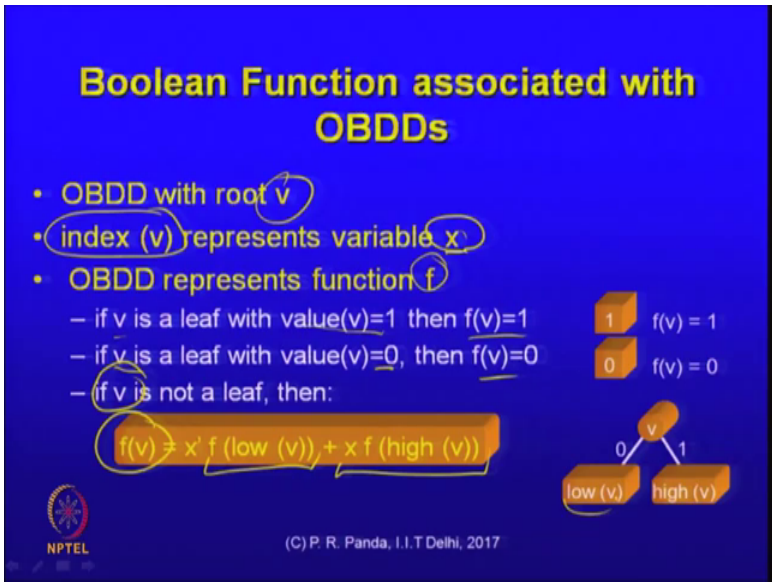
Yeah.

Student: So, 0 should not be at index a or we need to have an empty node there.

Indices are associated with variables this is not an optimized representation yet this is just the definition. So, far, but the indices are is associated with only variables right here is how I define the OBDD the leaf node has an attribute called a value.

So, there is a value that is associated with these nodes and for the leaf nodes the values are 0 or 1 correspond to true or false each node v has two children, we call it low of v and high of v and if a node v; it is low and the high these are like the children the low and the high if all of these are non leaf nodes, then this holds the index of v is less than the index of both its children that is what is imposing the ordering in the OBDD that the graph is acyclic because of that, right.

(Refer Slide Time: 66:49)



## Boolean Function associated with OBDDs

- OBDD with root v
- index (v) represents variable x
- OBDD represents function f
  - if v is a leaf with value(v)=1 then f(v)=1
  - if v is a leaf with value(v)=0, then f(v)=0
  - if v is not a leaf, then:

$f(v) = x' f(\text{low}(v)) + x f(\text{high}(v))$

1
f(v) = 1
0
f(v) = 0
v
0
1
low (v)
high (v)
NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

Once you do that there is an OBDD that has a root node v that index of v essentially represents some variable x is one of the variables right. So, that OBDD represents a function f in the following way if v is a leaf with the value of one then that function is one remember the root is v.

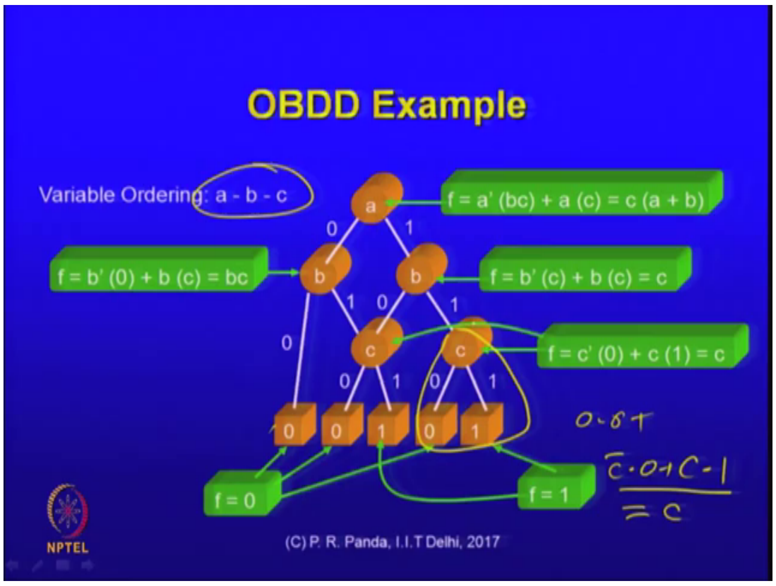
So, this is just a special case of a inductively defined function if v is a leaf with a value 0, then the value is 0 these are the two base cases of the induction, but if v is not a leaf then the value associated with v is remember the variable associated with v is x right. So, the value associated here is x prime times f of low of v lower v is the low child and x times f of high of v which is the high child.

You can see this an application of the Shannon's.

Student: Shannon's.

Expansion right; So, essentially what you have at low would be the cofactor with respect to x prime and what you have at high would be the cofactor associated with x.

(Refer Slide Time: 68:01)



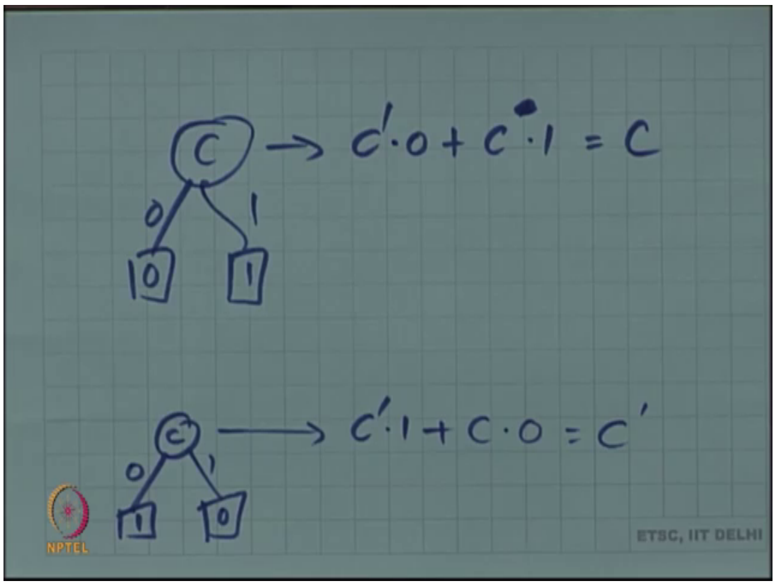
Yeah, just a quick example here for this graph with the variable ordering a b c which means that first a equals 1, then b, then c, the function associated with each of these leaves is clear the 0s means false, the 0s means true for this sub tree.

So, for every node there is a Boolean function that we can associate for this, I have what c bar or c prime times 0 plus c times one c is a variable remember. So, that function would be what?

Student: c.

Would be c itself, how do I represent c prime.

(Refer Slide Time: 68:57)



c
0
1
0
1
$\rightarrow c'\cdot 0 + c\cdot 1 = c$
c
0
1
1
0
$\longrightarrow c'\cdot 1 + c\cdot 0 = c'$
NPTEL
ETSC, IIT DELHI

So, this much is just see what would c prime be this function is c prime times 0 plus c times 1 that is how would I set up c prime.

Student: Swap the 0.

Yeah.

Student: Swap the values.

Yeah I just swap the two leaves. So, just c is still there there's no complementing on those variables there is just variables right, but I just have one 0 then the associated function is c prime times one plus c times 0 that is 0 ok. So, that is my function for c you can see that this sub tree is identical to their sub tree, therefore, that function is the same function what is the function for this I have f equals b prime c plus b c right if it is true then it is c the result is c itself.

What about the function for this, I have b prime times 0 plus b times c that is b c. So, there is a function that I can associate with every one of the nodes once I come here. So, if this is b c and this is c, then I can go here and derive a as a prime times b c is this plus a times c which is this finally, the function that is corresponding to that entire structure is a c plus b c ok.

Let me stop here and I will continue in the next class.