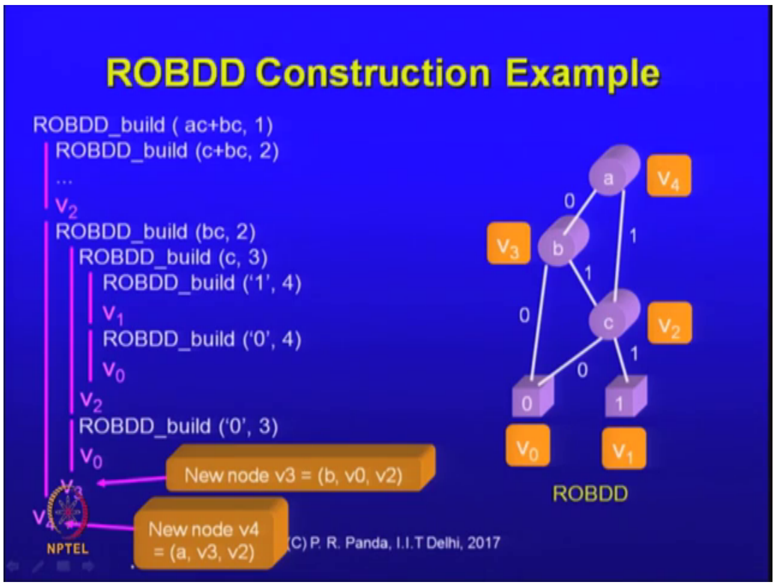
Synthesis of Digital Systems

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

Lecture - 21
Introduction to Logic Synthesis

(Refer Slide Time: 00:24)



ROBDD Construction Example

```
ROBDD_build ( ac+bc, 1)
  ROBDD_build (c+bc, 2)
  ...
  v2
  ROBDD_build (bc, 2)
    ROBDD_build (c, 3)
      ROBDD_build ('1', 4)
      v1
      ROBDD_build ('0', 4)
      v0
    v2
    ROBDD_build ('0', 3)
    v0
  v3
v4
```

New node v3 = (b, v0, v2)

New node v4 = (a, v3, v2)

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

Let us talk about the ROBDD complexity. The algorithm for the construction is reasonably straightforward, but let us take a look at the complexity of the procedure and also the space complexity of the data structure for a given function, how big can the data structure get. It turns out that there are some functions where the ROBDD in spite of it being an optimal structure, a minimal structure can still grow exponential in the number of variables.

You can see that we start from the top, the root has 1 node, the next level can have 2 nodes, the next level can have 4 nodes and so on. There would be some functions where the order blows up exponentially with respect to the number of variables you cannot help it.

But what about the time complexity of this algorithm that we presented the input is a Boolean function, and the other one is an index. It goes through all the levels of the tree; and expands it all the way until you get the leaf as 0 or 1 that is when the termination happens. So, there is something better we can do with respect to the algorithm. Do you see some redundancy in this strategy? You know we can try and avoid possibly.

A bound should be understood we already said that the data structure can grow exponentially large in some cases ok. Data structure cannot possibly grow exponentially large without you spending an exponential time in creating it. How does the data structure become large? It is because we are creating nodes or whatever the structure is we are doing some work and through that it is becoming large or small. So, in the worst case, you cannot help it. If the space complexity is exponential in the worst case, the time complexity in the worst case is also exponential.

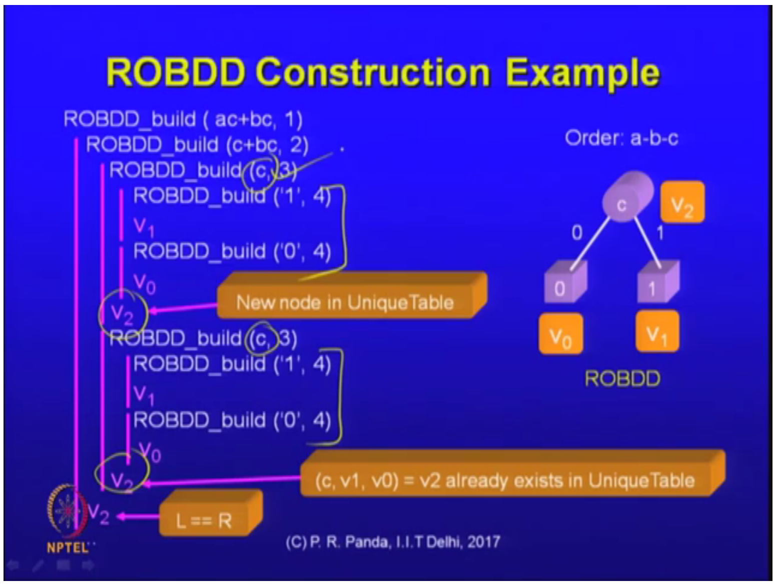
So, we are just saying that for the cases where smaller structures exist, the typical ROBDD is small for a typical function can we do better. The problem we do not want the time to be exponential even if the space is small, so that is where we could possibly do something about it. Again do something means we can try and identify possible redundant computations we are doing along the way. For that example itself you should have noticed there is some redundancy.

Student: So, the v 2 node was (Refer Time: 03:21).

The node is not duplicated. Remember it is created the first time the second time around that node is returned, we do not create a new node, but it is true there is something redundant that is going on there with regard to v 2. This is the second page right we started off with right.

Student: (Refer Time: 03:27).

(Refer Slide Time: 03:40)



ROBDD Construction Example

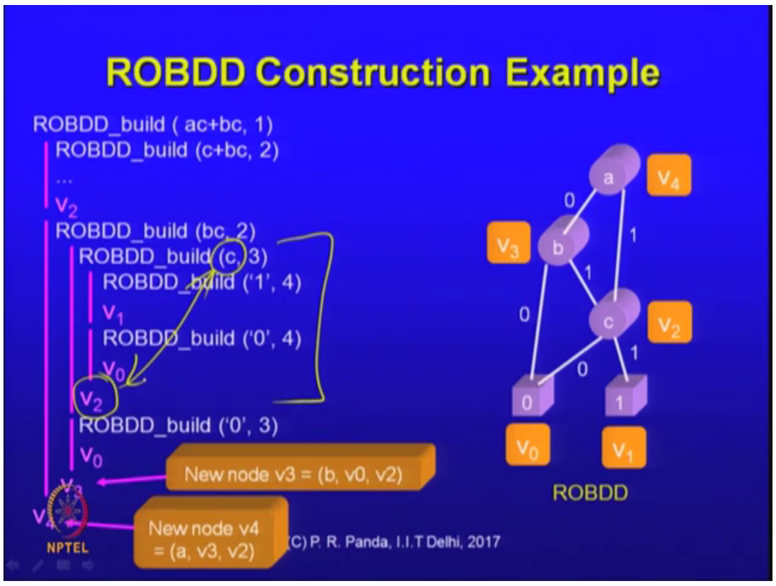
```
ROBDD_build ( ac+bc, 1)
  ROBDD_build (c+bc, 2)
    ROBDD_build (c, 3)
      ROBDD_build ('1', 4)
      V1
      ROBDD_build ('0', 4)
      V0
    V2  <- New node in UniqueTable
    ROBDD_build (c, 3)
      ROBDD_build ('1', 4)
      V1
      ROBDD_build ('0', 4)
      V0
    V2  <- (c, v1, v0) = v2 already exists in UniqueTable
V2  <- L == R
```

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

The v 2 was created for the first time here, but was reused here and also later on. What could we do about this? We want the reuse actually right so but what can we avoid here. We can avoid the function calls this is a function call that is repeated right. We could do something about it by somehow identifying that this function and that function are expected to be the same.

In the general case it is hard to say that, but certainly at that level where there is only one variable or a very simple expression that I should be able to take a look at and say that this function c has already been captured. So, next time I get a c, let me just match against it and return the associated node v 2 that much I could do. Again it is not a failsafe method in the sense that there would be variations that we may not be able to catch.

(Refer Slide Time: 04:58)



ROBDD Construction Example

```
ROBDD_build ( ac+bc, 1)
  ROBDD_build (c+bc, 2)
  ...
  v2
  ROBDD_build (bc, 2)
    ROBDD_build (c, 3)
      ROBDD_build ('1', 4)
      v1
      ROBDD_build ('0', 4)
      v0
    v2
    ROBDD_build ('0', 3)
    v0
  v3
v4
```

New node v3 = (b, v0, v2)

New node v4 = (a, v3, v2)

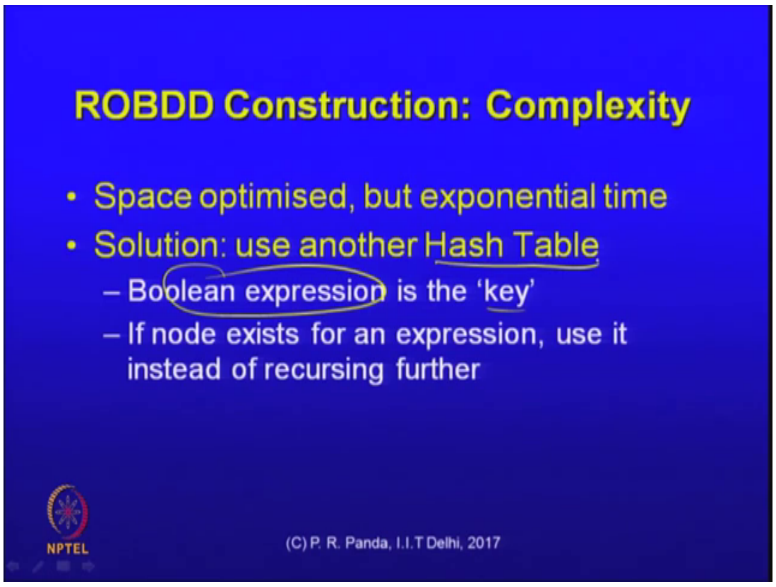
(C) P. R. Panda, I.I.T Delhi, 2017

Now, the question then is how do I find out that the same function has been seen before. But that is one redundancy it is occurring here too right. So, if even if there are some small things that we could do to identify this redundancy and get rid of it, and somehow relate that input with that node, then we could avoid quite a bit of recursion particularly at the lowermost level that is where most of the time is getting spent.

Student: Sir c is wider at.

Right at the bottom there are more calls.

(Refer Slide Time: 05:31)



ROBDD Construction: Complexity

- Space optimised, but exponential time
- Solution: use another Hash Table
  - Boolean expression is the 'key'
  - If node exists for an expression, use it instead of recursing further

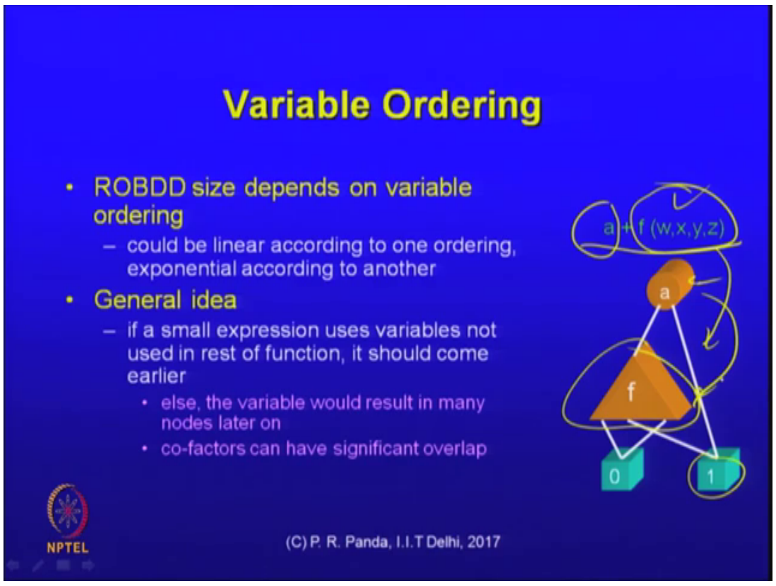
(C) P. R. Panda, I.I.T Delhi, 2017

So, as of now that data structure is not helping us, but we could extend the same idea using a different hash table, where we have to identify some structure for the Boolean expression, but it could even be just a string. For example, if it is just one variable, it could just be whether that variable is associated with an index right.

So, certainly we can take this step that if it is nothing complex, it is just one variable or it is complement, then we should be able to keep structures of some bookkeeping where we just see has been seen earlier or not if it has seen then we just reuse the associated node. But in general that Boolean expression can be treated as a key to a new hash table how to design it is a different question, it cannot be too complex remember a plus b is same as b plus a.

So, I could take a string matching that is all right, but then that will not match correctly all variations will not be caught, but still some would be caught and that also can help and improving the overall runtime of the system. So, idea is that you I used a different hash table, and there I maintain all the expressions that have been seen so far. How do I maintain is a different question we can be simple, we can be more sophisticated in that maintenance. But that Boolean expression itself is the key and if the node exists then I just use the node instead of recursing further ok.

(Refer Slide Time: 07:09)



Variable Ordering

- ROBDD size depends on variable ordering
  - could be linear according to one ordering, exponential according to another
- General idea
  - if a small expression uses variables not used in rest of function, it should come earlier
    - else, the variable would result in many nodes later on
    - co-factors can have significant overlap

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

a + f (w,x,y,z)
a
f
0
1

Let us look at a few other interesting issues regarding the ROBDD. The size depends on the variable ordering by size here we mean the number of nodes in the BDD in the

ROBDD. There is a strong dependence on the variable ordering. You use some ordering, you get some structure. If you use a different ordering, you get a different structure. In general, there is very little in common between those two structures; although we will look at interesting variations later on, but if you just randomly change the ordering then you get a totally different structure.

So, the complexity depends on that ordering complexity as defined by the number of nodes in the structure depends on the order it is possible that you have a linear number of nodes according to some ordering, but according to some other ordering, you could have an exponential number nodes. Here too we could use some heuristics to control the efficiency of the BDD structure.

Let us look at an example like this you have a Boolean function of a, w, x, y, z - five variables ok. And as it turns out you are able to decompose that function into one part here and a second part that does not involve a ok. So, perhaps it is a plus plus some x y plus w z or some such function. If it is like that then you can see that this kind of a function can be represented like this; a is the root node, nobody gave us that order do; we have to determine that order. It is our data structure. So, we have to find what is the best ordering.

But here is an example where it seems as though I am able to do it efficiently. I just have a on top like I make a the root of the structure, then what is the logic? If a is 1, then the result is 1; if a is 0, then the result is whatever that other function is. So, I am able to essentially augment whatever was the BDD for f with just one node and I have the new BDD right. So, the overhead for handling a was very very minimal; it was just 1 node right.

What is the generalization from here, this is a specific example of course, but there is some rule that I could use from.

Student: Do you have any expansion if one of the cofactor comes out to be 0 fixed then.

One of the cofactors turns out to be 0 or 1.

Student: Fix that.

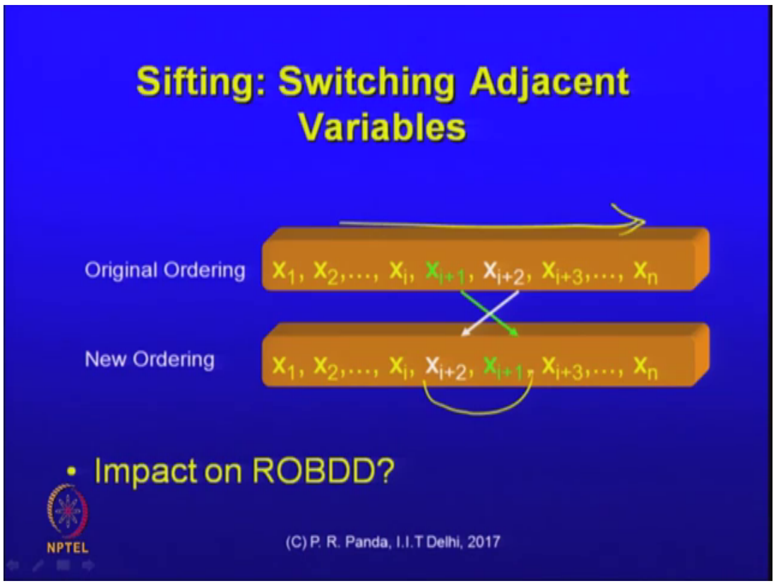
Fix here it is yeah, here it is that. The generalization could be that if in a complex function with many expressions, if there is one small part that uses variables not used in the rest of the function. If I am able to decompose it in this way here we are able to do it because a was different and all the other functions; even though it is a complicated function they were disjoint where one was a small set and the other was the larger set right.

Student: Large.

Then the rule is that those variables in the smaller set here in that set is just a, they should come towards the top of the BDD. If it was the other way around if a came somewhere in the middle or later, then in fact you do not get as efficient a structure, a actually repeats in a number of things because it is there in both the cofactors of w; it would be there as in the cofactor with respect to w, it would be there in the cofactor with respect to w prime also.

So, actually it blows up into many different nodes that could be one simple rule when determining the ordering. Now, we are trying to find since the complexity is a function of the ordering. We are trying to find what is a good ordering rules of thumb is what we are trying to decide here ok.

(Refer Slide Time: 11:41)



Sifting: Switching Adjacent Variables

Original Ordering $x_1, x_2, \ldots, x_i, x_{i+1}, x_{i+2}, x_{i+3}, \ldots, x_n$

New Ordering $x_1, x_2, \ldots, x_i, x_{i+2}, x_{i+1}, x_{i+3}, \ldots, x_n$

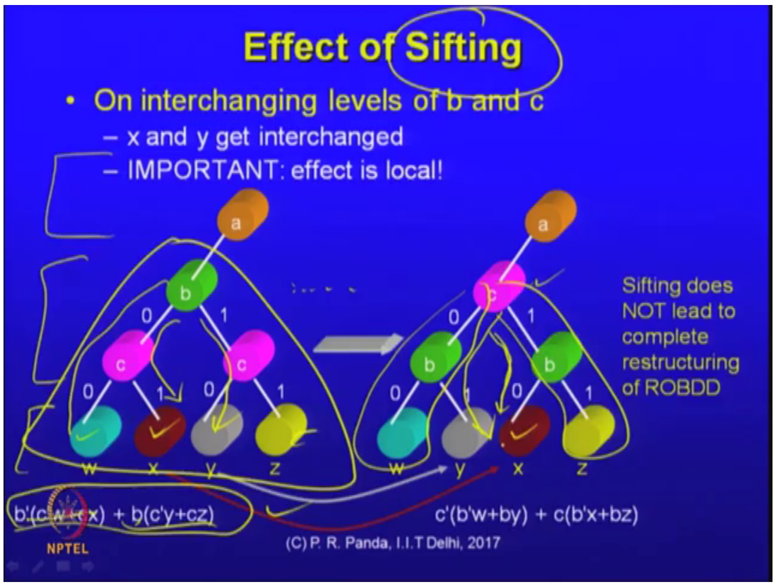
- Impact on ROBDD?

NPTEL

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

That leads to a very interesting transformation of the BDDs that is worth studying. Let us say I have some function of n variables, and there is an ordering that I chose for the ROBDD. Consider a new ordering in which a pair of variables have been interchanged, adjacent pair of variables have been interchanged. Now, the question is what is the impact on the ROBDD when you do that?

(Refer Slide Time: 12:17)


Effect of Sifting

- On interchanging levels of b and c
  - x and y get interchanged
  - IMPORTANT: effect is local!

a
b
c
c
0
1
0
1
0
1
w
x
y
z
a
c
b
b
0
1
0
1
0
1
w
y
x
z
Sifting does NOT lead to complete restructuring of ROBDD
b'(c'w+cx) + b(c'y+cz)
c'(b'w+by) + c(b'x+bz)
(C) P. R. Panda, I.I.T Delhi, 2017
NPTEL

Let us put up simple example of an ROBDD we are talking about b and c being those two variables whose ordering we want to interchange. There is a lot of stuff on top that does not matter. As you know those are not affected by interchanging b and c. There are other things at the bottom each of these here is essentially a sub tree of all the other variables that are there, that is the most general structure you can think of right. Whatever that whole function is that can be expressed as b prime times something plus b times something.

So, of course, there is only one b, where b being the first one in that subtree there is only one b. Actually there may be other bs in the other part of the BDD and that is we are just looking at one function and we are trying to examine the effect of starting with b or starting with c. So, that function that is there here this is the sort of a general view of what that function could be.

And if my leaves here these are the sub trees that come below the cs are w, x, y and z, then do you see that the expression that captures my entire function here rooted at b would be this is b prime times something plus b times something else ok, both of these could be functions of c. So, I now have c prime times something w plus c times something else x. So, my general function is this there is a b prime c prime element, there is a b prime c element, there is a b c prime and there is a b c, each of these is then multiplied by w, x, y and z

What happens if you interchange b and c? In now I will say instead of keeping b as the root what if I have c as the root given that my function is this.

Student: (Refer Time: 14:49) consider all the combinations of b and c the outputs.

This picture is showing all the combinations. What happens if b is 0, and c is 0, what happens if b is 0, c is 1 and so on. It is one of four things only can happen with respect to the rest of the BDD. Depending on whether b c is 0 0 or is 01 or 10 or 11. So, if I now make c, the root then I would have a structure like this. Well, if c is 0, then I have something, but that b would be repeated; earlier I had 1 b and 2 cs. Now, since I am starting from c, there is 1 c and.

Student: 2 bs.

2 bs. But what happens to the rest of the structure, in fact, we can argue that the rest does not change at all.

Student: (Refer Time: 15:44).

At least the there is some rearrangement, but the complexity does not change that because the w, x, y, z still remain w, x, y, z. Why do they remain? You can see that in this new structure what do you need to do? If you were to go down that path which is b was 0 and c was 0 then you would arrive at early same thing has to be preserved in the interchanging. So, if c is 0 and b is 0, I still have w this is the same as earlier. So, w still goes there. What if b was 1 and c was 1, I still have that same path as earlier this thing does not change. But if it was 0 1.

Student: Then b x or more than w change will come.

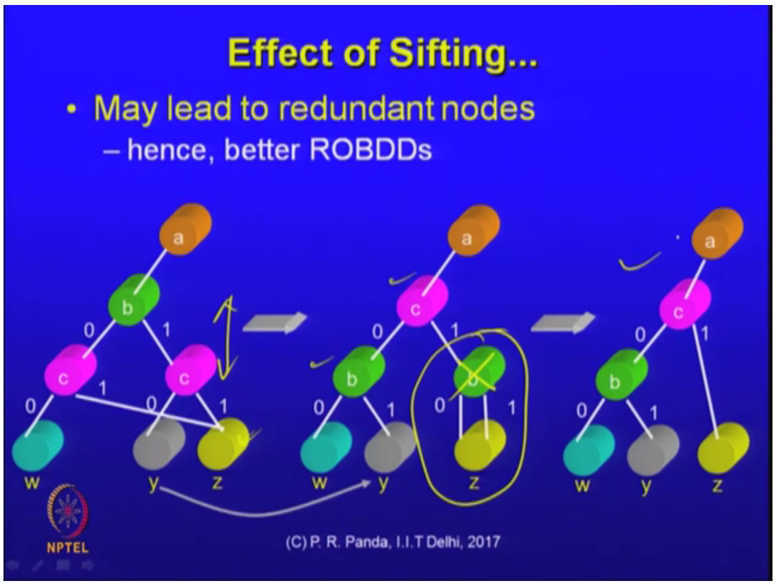
If it is 0 1 that means, that if I interchange the order, then that corresponds to this path where c was 1 and b was 0, so that path actually corresponds to this path. So, what was x here now appears here when you do the interchanging of the two-levels. So, similarly if b was 1 and c was 0, I was at y that corresponds to this path where c is 0 and b is 1. This transformation is called sifting transformation.

And one important effect of this is that there is a rearrangement of the sub graphs that are rooted below the c level right. There is just a rearrangement, but in fact, it does not lead to a complete restructuring of the ROBDD like a random permutation would cause. So, that effect is local. If you want to examine the effect of the interchange, the effect is actually a fairly small effect on the overall structure. Therefore, the implication is that such a transformation good form part of an algorithm that is trying to examine the different orderings to find out what is the best ordering.

Student: Sir, but this is the case when all of the implicants are being used if in a ROBDD we could have not have all the four combinations.

Yes, yes. So, some of them could actually be 0, I did not say what it was, maybe it is a complicated graph, but maybe it is the leaf, yeah it could be anything. The complexity of w, x, y, z, we did not look at. But what we are saying is the irrespective of what the complexity is at the lower levels, the change of two adjacent levels is a small transformation, it is not a big transformation that is all that is the implication yeah, but it is important to understand because we can use that in a in an overall strategy where we are trying to find what is the best ordering because some our rings might be better than others.

(Refer Slide Time: 18:49)



Effect of Sifting...

- May lead to redundant nodes
  - hence, better ROBDDs

. So, so far it is just that the change is local, but it actually could help in discovering more efficient structures. Let us assume something like this. I do not have all the four different sub graphs, but instead if bs was 0, and c was 1, both of these were actually pointing to the same function is possible right. If you would have seen this earlier also in the process of redundancy removal of the OBDDs to generate the ROBDD we were actively trying to look for patterns like this.

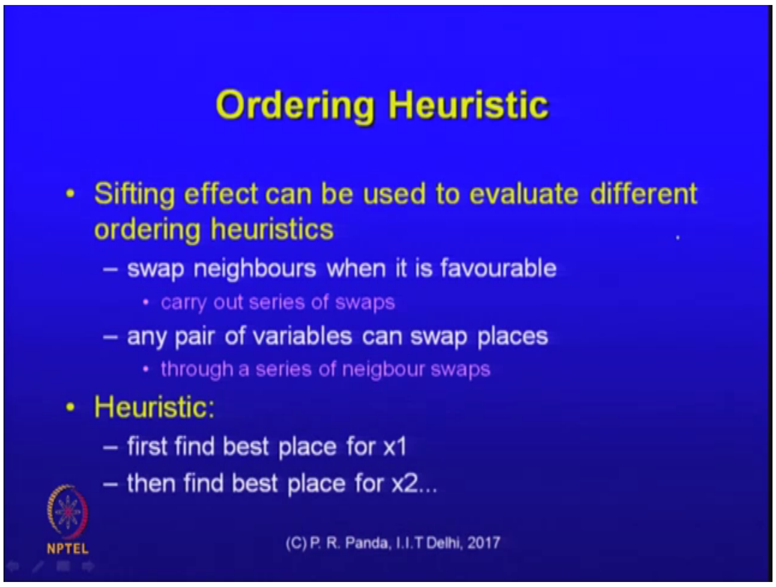
So, if such a structure were there, and now you try to interchange b and c, what happens c goes to the top, b comes here to the next level and the rearrangement of w, y and z will lead to a situation like that like this. You can see that visually also. If you arrive at z, then it does not matter whether the value of b was 0 or 1, both ways you are arriving at z. There is a difference based on what c is, but it is independent of b.

But this structure is familiar right, this pattern if it occurs in the ROBDD, we should actually dump that node, and this is my more efficient representation right. But this comes about in a natural way it is discovered when we just do that sifting optimization, where we look for whether as a result of the sifting that we did interchanging of the specific pair of variables that we are currently trying out, you could actually have a better ROBDD smaller ROBDD because some node might become redundant.

Student: (Refer Time: 20:53) where the selection of sifting can be made intelligent in the sense.

Yeah, we are saying that first sifting is a local transformation second is sifting may lead to a better structure right mainly not necessarily of course, we did not know about this before we actually tried it out it could be that you go in the other direction you find that it is a worse, but it is fine in the process we can retain whatever was the best that we have seen.

(Refer Slide Time: 21:22)



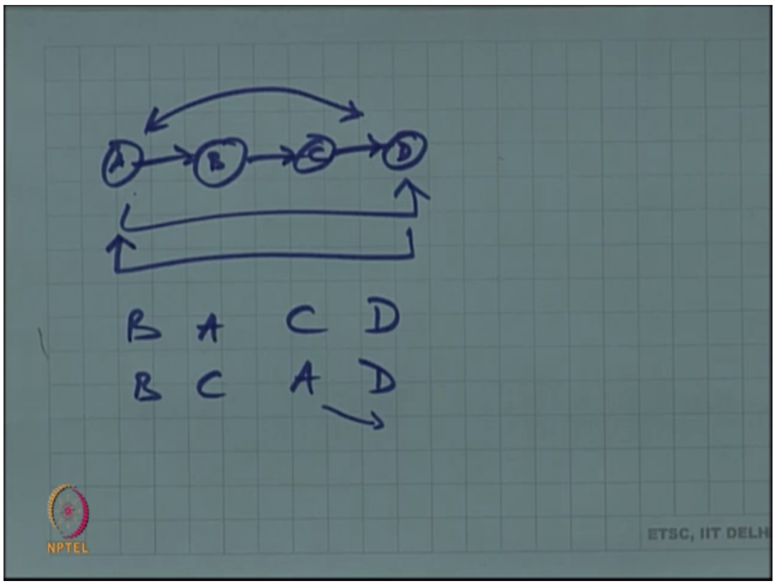
Ordering Heuristic

- Sifting effect can be used to evaluate different ordering heuristics
  - swap neighbours when it is favourable
    - carry out series of swaps
  - any pair of variables can swap places
    - through a series of neigbour swaps
- Heuristic:
  - first find best place for x1
  - then find best place for x2...

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

So, the question then is how do you come up with an algorithm that recognises this. It could actually be used to evaluate many different ordering heuristics. You consider neighbours and you swap them when it is favourable right. And even though we argued in terms of interchanging only neighbouring nodes in an ordering, in fact, you could generalize that to interchanging any pair of nodes. How do you generalize this to any pair of nodes? You achieve that by doing a series of sifts I am going to the other.

(Refer Slide Time: 22:06)



So, I have if I actually want to interchange these two, then I over a series of sifting transformations, I should be able to interchange. Take A all the way there, and take D all the way here like. First interchange A, B right, yeah then.

Student: A to D.

Yeah. So, first move A to the right place, and then I can move D to the right place through a series or if I wanted to I could use this basic mechanism. Of course, every swap like that leads to some change in the ROBDD. So, the effect being local just means that managing it is fine algorithmically, but of course, the small changes over large number of sifts will lead to major changes in the ROBDD, but such observations could lead to a heuristic that could work in the following way.

The problem of course is that the number of permutations is exponential. But if you want it to be reasonably small in terms of number of transformations that you try out the algorithm for example could be the following. You first find the best place for variable x 1. How do you find you keep moving x 1 to every one of those keep the rest of the orderings the same and just try to move f x 1 by swapping it with the next with the next the next and so on. In the process, you would have tried out n different transformations each of which is easy to evaluate our evaluation is just whether it is leading to a smaller structure or not. And then after I have found the right place for x 1, I could then freeze the spot for x 1, then try to do it for x 2, do it for x 3 and so on.

Student: (Refer Time: 24:22) n square.

(Refer Time: 24:22) n square which is better than exponential. This is not the optimal procedure, this may not give us the best ordering, but it is just a way of doing more comprehensive.

Student: For every search we are computing the ROBDD again and then comparing.

We are not computing ROBDD that is the whole point. We are what are we doing anyway as a result of the swap, how do I find out that swap is favourable or not. I am saying that what is there at the next level and below they do not matter, they do not change at all. So, what is changing is whether some redundant node was found or not that is the only thing. If it was found then it means that this is the better structure than the previous one.

Student: But do we need to really do a swap to find that out?

This is a conceptually swap, this is just a comparison occurs yeah.

Student: That is what.

Yeah, amount of work that you are doing is small, but it is not a revaluation of the ROBDD which is an expensive process. We do not want to do that absolutely yeah, yeah.

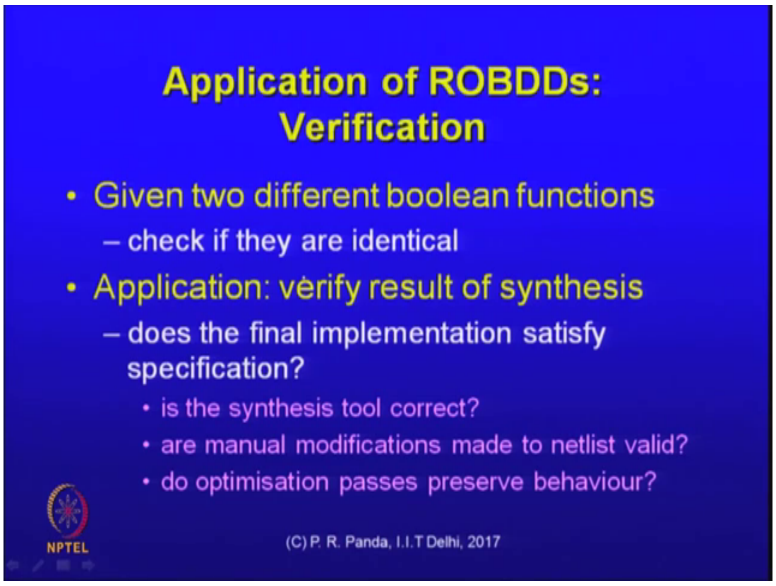
Student: Only the nodes that we are going to swap now.

Yeah, we are just checking.

Student: (Refer Time: 25:31) to the graph need to be checked.

Right,, but the thing is here it occurs only once, but this might actually occur in several places at that level b might occur in several places, and c might occur in even more places right, but. So, in every one of those situations where it occurs you want to examine the effect of the sift ok.

(Refer Slide Time: 25:53)



Application of ROBDDs: Verification

- Given two different boolean functions
  - check if they are identical
- Application: verify result of synthesis
  - does the final implementation satisfy specification?
    - is the synthesis tool correct?
    - are manual modifications made to netlist valid?
    - do optimisation passes preserve behaviour?

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

Let us look at some applications so far what we have focused on is the representation itself. The data structure itself is interesting enough that it is worth the spending that much time to understand what happens and there is a strong dependence on the ordering. So, it is important to do something about it. You cannot do anything about it in the worst case, but often you can do something about it at least simple transformations can be tried out to see if obviously, better solutions exist. It is an optimal structure for a given ordering, but if the ordering is different, then you get essentially something very different right.

So, what are the applications? First there is a verification related application. If you are given two Boolean functions and the objective is to check if they are identical or not, why is this kind of a problem relevant, when will this come up in a design flow it could be that you are trying out some optimization. It is a great way of validating a tool itself that is trying to do some transformation in some complex optimization, and it still needs to make sure that the function that is there is equivalent to whatever was the specification. So, applications are there, but the question is does that final implementation satisfy the specification. It could be two different implementations you are comparing, it could also be that something was specified first and you have an implementation which is an optimized net list perhaps. And you can ask this question was the optimization ok. First of all is the from a tool point of view was the tool ok. It could also be that sometimes we make manual modifications to a net list to improve timing to or whatever. And we could do this kind of a check to verify that we have not destroyed anything in the process of manually changing something.

How do I check? So, I have two functions now I need to check them for equivalence. How can I use ROBDD for this? But actually if two functions are equivalent, then what can you say about ROBDD structures given one some ordering a, b, c, d right.

Student: Somewhat in that they must be the same no not fix ordering will be same, but ah

Right, but if the functions are equivalent then it does not matter what ordering you choose, for that ordering you should get an identical ROBDD. Why is it, I did not do we are saying that there it is unique right the canonicity property means that does not matter if the functions are superficially different with respect to the set of implicants that we use how where we came from. But if the functions are equivalent and the ordering is fixed, we choose an ordering right which does matter you choose some ordering of the variables you should end up with the same ROBDD structure.

Student: Will the same ordering we should what.

Right, but this problem is independent of the ordering if I choose any ordering right, I just want to know that functions are equivalent. If they are equivalent for one ordering they are equivalent for all orderings.

Student: Well, then first of all we need to.

That structures will change, but both the structures will change in the same way if I change the ordering.

Student: So, the two ROBDDs that we are comparing both of them have the same ordering that ordering can be any of the given ones, but for those.

See this is part of our solution, this is not given. Functions are given, question is these two functions are given up to us now I am just saying the hint is that and ROBDD can be used to establish equivalence, I am not saying that the ROBDD is given to us, we have to construct the ROBDD.

Student: Then that is then we use some definite ordering and

Right

Student: Just like ROBDDs and that in the starting itself we did that thing like to compare every node and its child, and then it if it comes in it then both of them are equivalent.

Right. So, what us the strategy? So, given two functions I take one function and I construct it is ROBDD, what do I do with the other one

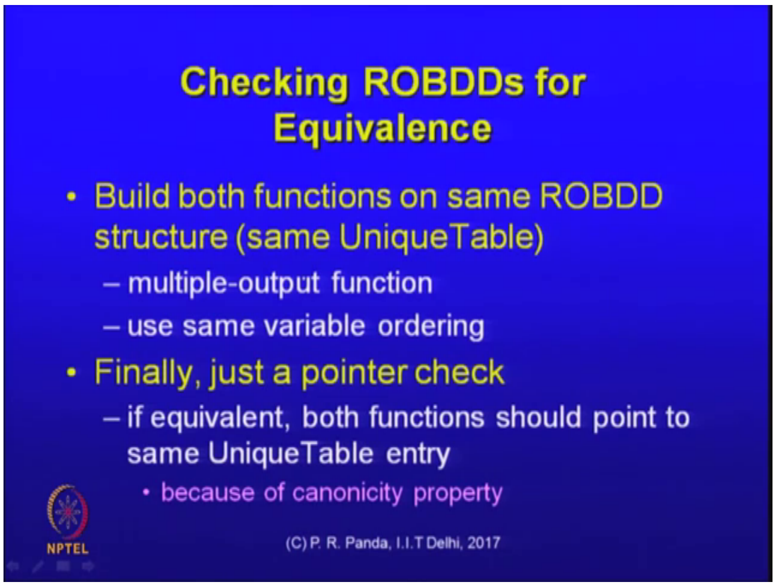
Student: The same ordering I construct the ROBDD for that one two.

Ok, then?

Student: Then we just have to two graphs, and we need to compare the flow from one source node till the end.

Yes, remember what we pointed out that this is a graph isomorphism test which is an expensive test graph isomorphism I do not necessarily want to do, because it is an expensive test, but there is a there would be a simpler way to do that.

(Refer Slide Time: 30:43)



Checking ROBDDs for Equivalence

- Build both functions on same ROBDD structure (same UniqueTable)
  - multiple-output function
  - use same variable ordering
- Finally, just a pointer check
  - if equivalent, both functions should point to same UniqueTable entry
    - because of canonicity property

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

What you do is you just create one ROBDD use that same table to construct the ROBDD for the second function. In that table all the nodes are already there that you needed for the first one. In fact, as you build the second ROBDD you will find that.

Student: (Refer Time: 31:00).

Every function that you need in fact, those nodes are already there, but the second time when you do the traversal in fact, you will find all those nodes.

Student: All nodes are same.

over there.

Student: You do not add any numerical, you do not add any (Refer Time: 31:11).

Yes.

Student: You do not add (Refer Time: 31:11).

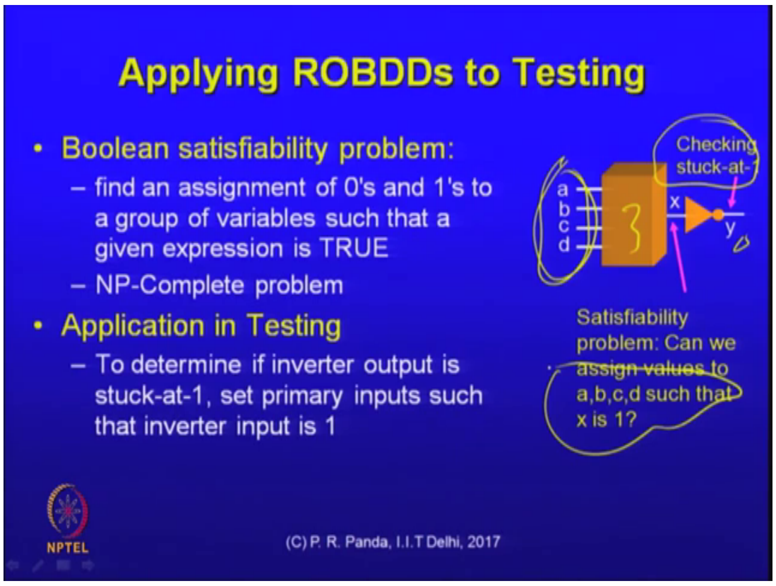
So, at the end equivalence check means what?

Student: Nothing you added.

Means you end up at the same root, ultimately it is just a pointer check are you at the same index or not of the hash table. If you are then every node is associated with one function. So, if you end up at the same node as the root of the first function, then they are equivalent.

Use same variable ordering, but it does not matter what the ordering is you choose whatever the ordering is. For that ordering you expect the graphs to be isomorphic, but isomorphism can be checked implicitly by just constructing the ROBDD, you do not have to use a graph isomorphism algorithm which is in general yeah a difficult problem ok. If course, this process works because of the canonicity property, two functions are equivalent then their associated ROBDDs are identical in structure that is why this process works ok.

(Refer Slide Time: 32:17)



Applying ROBDDs to Testing

- Boolean satisfiability problem:
  - find an assignment of 0's and 1's to a group of variables such that a given expression is TRUE
  - NP-Complete problem
- Application in Testing
  - To determine if inverter output is stuck-at-1, set primary inputs such that inverter input is 1

Checking stuck-at-1
a
b
c
d
3
x
y
0
Satisfiability problem: Can we ~~assign values to~~ a,b,c,d such that x is 1?

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

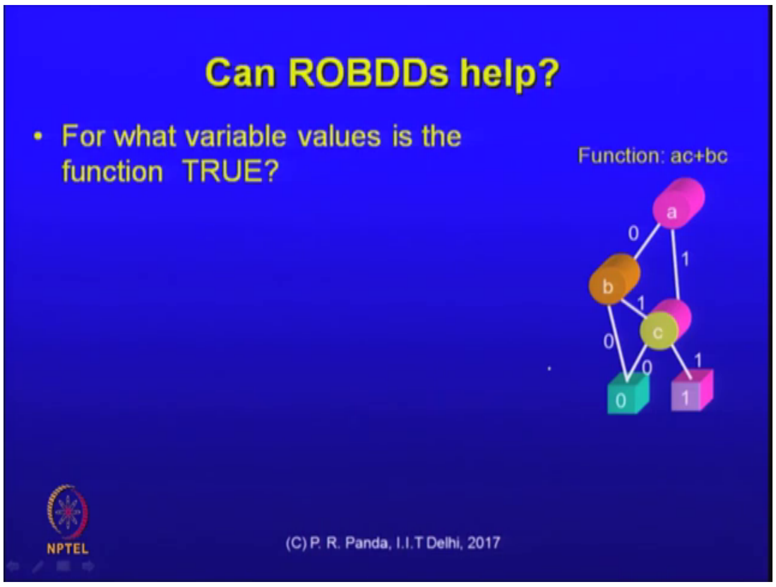
Let us look at a different application. Testing is the process where the chip is already manufactured. And you want to run some quick checks to figure out whether there were any manufacturing defects, What kind of defects, the stuck at 1, stuck at 0, these are common faults that may occur during manufacturing because of randomness in the processes. Stuck at 1 means what essentially there is a short to BDD that is what happens.

We want to do the testing in the following way I want to find whether y is stuck at 1; for that how do I test these are my inputs, I do not have direct access to this functions what are their inside, I am able to access y somehow, but I am able to set a, b, c, d to appropriate values, so that my expectation for example, can be that it is a zero that I am expecting on the y. And I measure a one at y that points to a stuck at one fault perhaps at y.

This is an application of the Boolean satisfiability problem that satisfiability problem is given a function given a Boolean function find an assignment of zeros and ones to the input variables such that that expression is true. For a general expression, it is not obvious what should the inputs be all these a, b, c, what should d for making x whatever, the output is 1.

The application for testing here is to determine in the if that inverter output is stuck at 1, which means that you set the primary inputs to those values such that that inverter input is 1, input as 1 means the output is expected to be 0; and if it is stuck at 1, then you can detect it. So, the problem is can be assigned values to a, b, c, d such that x is 1, the output of that function is 1 ok.

(Refer Slide Time: 34:34)



Can ROBDDs help?

- For what variable values is the function TRUE?

Function: ac+bc

NPTEL

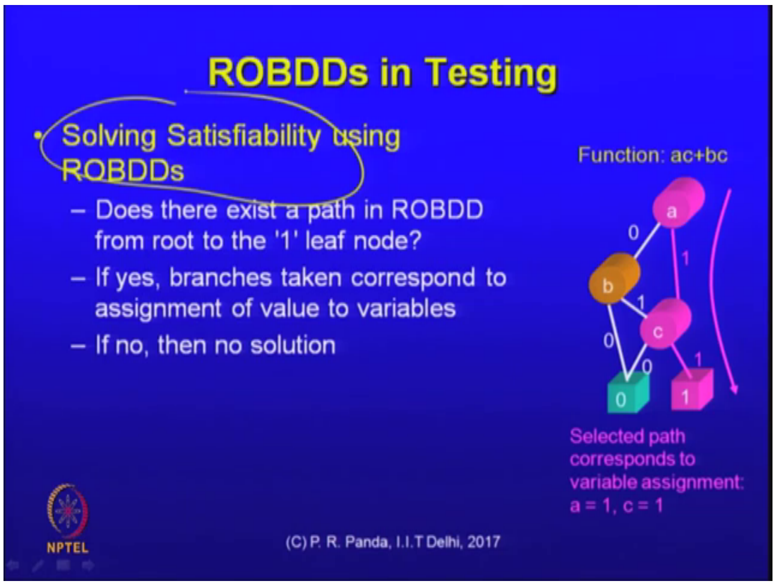
(C) P. R. Panda, I.I.T Delhi, 2017

Can I use ROBDDs for this?

Student: (Refer Time: 34:43) we start from the one node and traverse all the way up.

All the way up to the if there is a path from the root to 1 then in fact you have an assignment of variables yeah that is all that you need essentially is there a path. If there is a path then in fact those branches that you take along the path give you explicitly the values of those variables which will ultimately lead to that function being one.

(Refer Slide Time: 35:35)



ROBDDs in Testing

- Solving Satisfiability using ROBDDs
  - Does there exist a path in ROBDD from root to the '1' leaf node?
  - If yes, branches taken correspond to assignment of value to variables
  - If no, then no solution

Function: ac+bc
a
b
c
0
1
0
1
0
0
1
0
1
Selected path corresponds to variable assignment: a = 1, c = 1

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

This of course is not a magical solution to the Boolean satisfiability problem, which is understood to be a difficult problem in fact that is the standard NP complete problem in terms of which all the other NP complete problems are defined. We show that it is equivalent to Boolean satisfiability, and therefore that is also NP. But this seems like it is an easy thing finding the path from the root to the one node is that difficult, how complex can it be. Answering this question of is there a path from the root to 1?

Student: If that is.

That is fine, that is what we are using to answer this question, that is all I need. It does not matter what else is going on in that BDD, if there is one path, then I have answered my question.

Student: We just need one ok.

Yeah, does a path exists. So, there may be more than one set of variables assignments which will lead me true to truth value that is that is all right. I just need one part. The question was does there exist variable or give me a set of variable assignments which will result in that function being true.

Student: So, that is.

So, I am just looking for one path.

Student:.1.

That is an easy computation, but so what is wrong, I just said that it is satisfiability is a difficult problem.

Student: (Refer Time: 36:45).

In fact, if you want to explicitly find that path, you could actually make those edges run in both direction, maintain both the source and the destination. So, in fact, you start from the one, and go all the way up right. What is the length of that path?

Student: Fixed length can be at most n.

At most n, which is number of variables, no more than that which is a great linear solution to the Boolean satisfiability problem

Student: No, this is not Boolean satisfiability problem, this is one.

This is I am answering the Boolean satisfiability problem. Problem was this is the function and given that function.

Student: (Refer Time: 37:31).

What if I meant of variables to zeros and ones will result in that function being true. It is no different. You know we are saying this is a way to solve satisfiability using ROBDDs. And in fact, we just showed that it is a trivial linear time traversal through the structure.

Student: If it is not a time so that is we can check for the n is better than zero either of one has in the process only.

[FL]

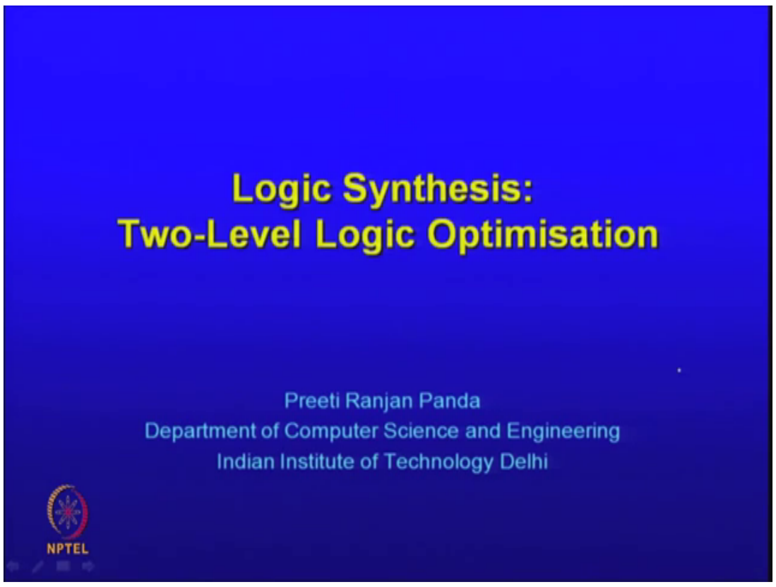
Student: 1 less than.

Yeah, as part of the traversal, this is an annotation right whether it is a 0 1, 0 or a 1, I can annotate it on that edge itself. Anyways since I have reached the parent, assuming there is a way to reach the parent from the child from the parent I know whether it is the left or the right child there are only two children. Well, you have forgotten what we just started off the discussion with it was in the previous class discussion is that there is nothing wrong with this claim that traversing this is a linear structure answering this is a linear time problem.

Of course, is construction of the data structure is an exponential time I am struck algorithm that you have not got away from. Once you have the structure then if you have repeated questions or you could use that same structure and efficiently answer that question, all the questions could be answered on that same function.

If you have other questions you could also answer them, if you have other questions on a subset of that function which translates to a sub tree or a sub graph in that same larger BDD, then also you can answer it efficiently, but this does not get away from the original difficulty of the problem right ok.

(Refer Slide Time: 39:37)



Logic Synthesis:
Two-Level Logic Optimisation

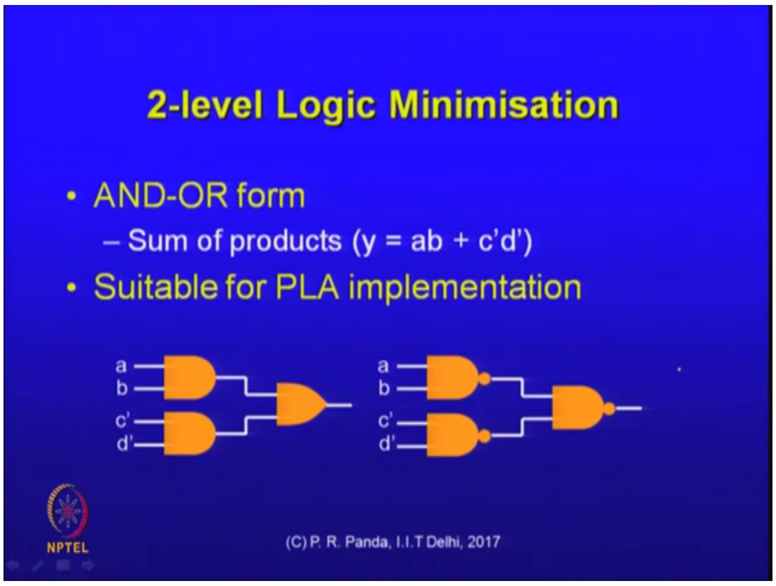
Preeti Ranjan Panda
Department of Computer Science and Engineering
Indian Institute of Technology Delhi

NPTEL

So, that was about the representation. It is a very important data structure and representation that is at the core of several algorithms verification strategies use it all the time synthesis strategies also use it all the time. We only define the structure and did some very elementary analysis, but you could look at operations if you want to perform what would be a good way of performing.

So, far we did not do too many operation, we only talked about the data structure. We asked some questions, but of course, for us to use it for example I have one function for which I have one BDD, I have a different function for which I have a different BDD if you want to do and of the two functions what is involved if you want to do all of the two functions you want to compose then what is the step that is involved in the BDD. So, there are other things that of course, you would like to do once you have a representation of the functions.

(Refer Slide Time: 41:00)



2-level Logic Minimisation

- AND-OR form
  - Sum of products (y = ab + c'd')
- Suitable for PLA implementation

NPTEL

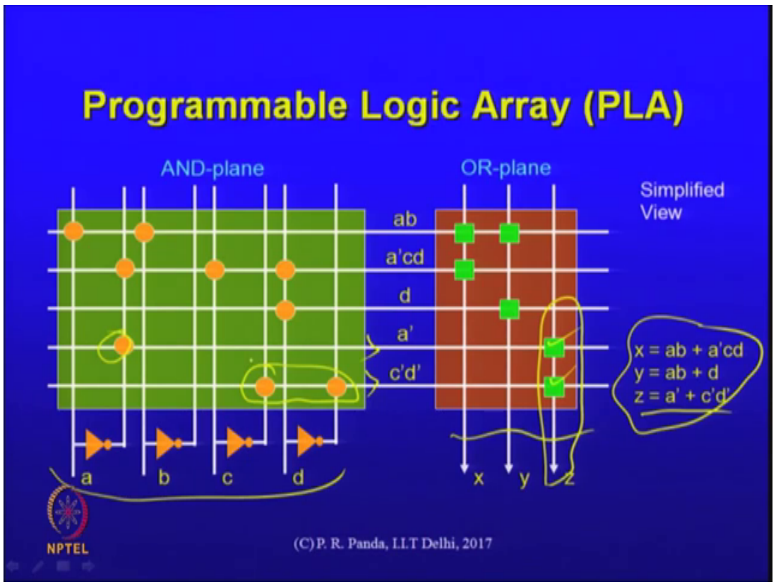
(C) P. R. Panda, I.I.T Delhi, 2017

But let us move on in this discussion to logic optimization following that representation discussion. Some of this has already been discussed in that FSM encoding context, but we will move from there to actually how the logic optimization is performed. So, here too we can think of a two-level logic minimization or a multi level logic minimization. Two-level here meant that it was in AND-OR structure like this. This is a two-level of course, you know that any two-level function like this or any function can be written in the NAND-NAND form the two-levels.

This is suitable for a PLA kind of implementation, but as it turns out the theory that is involved in two-level logic minimization applies irrespective of whether the target is PLA or even if it is multi level logic. When we get to multi level logic optimization, we will see that in fact two-level logic minimization can be called as a subroutine in some of the more complex things that we do in the multi level logic optimization.

So, that is why everything that we are doing in two-level is actually applicable also in the multi-level. So, in that sense it is not PLA specific, it is just that sometimes when you express a more complex functions in terms of let us say product of different two-level expressions, then you go into that two-level expression you can apply all of these what we are looking at here. So, it is applicability is more general than just the PLA structures ok.

(Refer Slide Time: 42:35)



Programmable Logic Array (PLA)

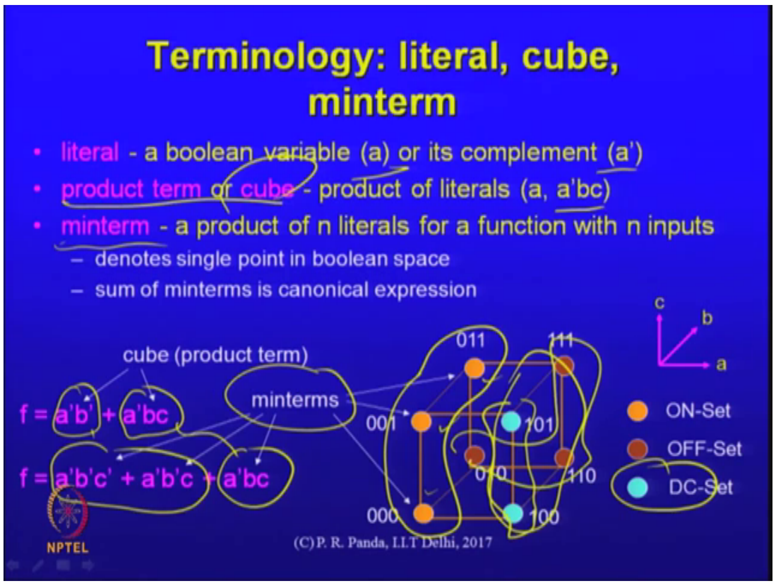
x = ab + a'cd
y = ab + d
z = a' + c'd'

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

Just to recap that structure looked like this, there was an AND-plane, there was an OR-plane, there were all these columns corresponding to an input variable and its complement. And on the output in general you have some things in the case of the FSM we had the d inputs of the flip flops that correspond to another set of outputs, but in general there are some inputs and there are some outputs.

And if you have some function maybe multiple outputs, you know that I owe you can compose it out of essentially connections made in the AND-plane followed by connections made in the OR-plane. So, for z, if I have this as a prime plus c prime d prime then that column for z would consist of two connections here corresponding to the two product terms. The product terms themselves come from the AND-plane of the PLA. So, a prime as this, and c prime d prime is this ok, so that is the architecture.

(Refer Slide Time: 43:46)



Terminology: literal, cube, minterm

- literal - a boolean variable (a) or its complement (a')
- product term or cube - product of literals (a, a'bc)
- minterm - a product of n literals for a function with n inputs
  - denotes single point in boolean space
  - sum of minterms is canonical expression

Let us just quickly go through the terminology we have already introduced most of it literal refers to a Boolean variable or its complement. Both of them are treated as one literal right. A product term or a cube is a product of literals, just product this is not a more complex expression with some and so on it is just product. Why it is called a cube you should be able to see this essentially it translates to a structure in the n-dimensional cube structure that you have for the entire function.

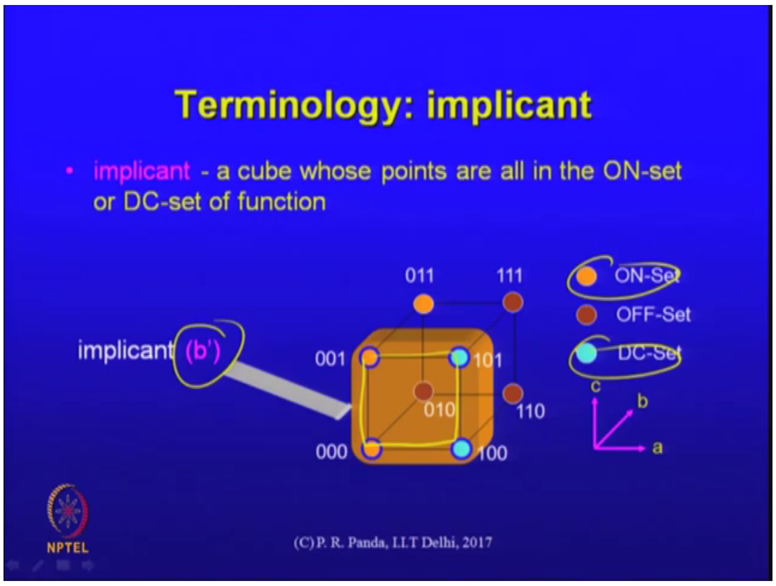
One of these product terms would correspond to a cube of some dimension. If it is two dimension, then it is like perhaps these four points; and if it is three dimensions then it is cube of eight points and so on. So, all of these are cubes, so that is why cube is used here not suggesting that it is only a three-dimensional cube, but it could be any number of dimension.

Minterm we have seen this is a product of all the n literals for a function with n inputs right. And of course, a function can be broken up in terms of decomposed in terms of a set of minterms from a visualization point of view these min terms correspond to vertices in the cube for the function. So, this is a product term that is a cube, that is a cube, and a function like this can be decomposed into a set of.

Student: Minterms.

Minterms right. So, this becomes this, but that one expands to two min terms perhaps. The function may also have some do not care points and that is what we are illustrating here that there are a set of on vertices, there is an on set there is a do not care set which is this and there is an offset which are this.

(Refer Slide Time: 45:51)

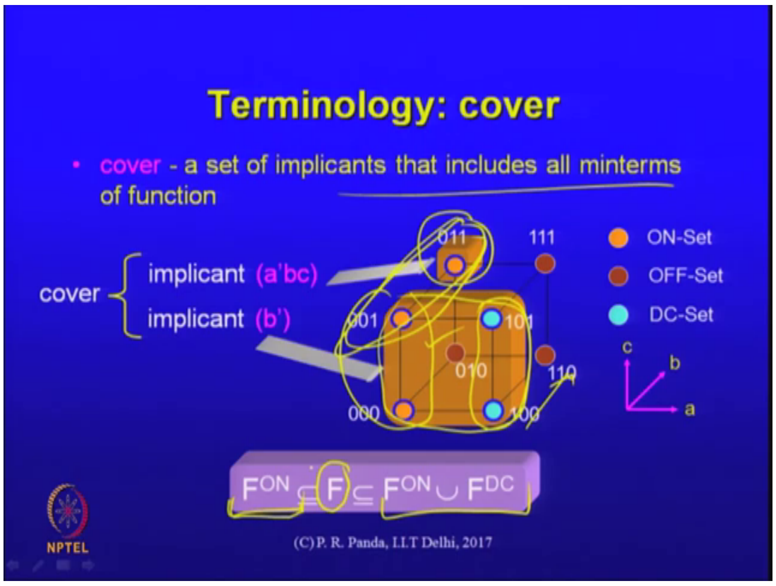


Terminology: implicant

- implicant - a cube whose points are all in the ON-set or DC-set of function

Implicant is a cube whose points are all either in the ON-set or a do not care set of that function ok. So, if you have a b prime then that is an implicant because the four points corresponding to b prime are all either in the ON-set or they are in the do not care set, which means that if that implicant is one then that function is one that is the meaning of implicant right. It implies that the function is true if the implicant is true, so that is an implicant.

(Refer Slide Time: 46:31)



Terminology: cover

- cover - a set of implicants that includes all minterms of function

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

Let us define it cover, yeah.

Student: Is it normal or is it always to assume that impliacnt the do not cares points value will always be one.

No, it means that (Refer Time: 46:46)

We have a choice do not can means we have a choice

Student: Right.

It is just that it does not hurt to have it as one. Some of the do not cares we need not include, but some of the do not cares we can include if it is convenient for us if it helps us minimize it, it is to that extent. So, far it is only a possibility we are saying that it is an implicant in the sense that if b prime is 1, then that function is 1.

Cover is a set of implicants that includes all the minterms of the function. So, this what we have here is a cover, what is highlighted in orange, because that brown node there is not part of that orange implicant. This is the b prime implicant, b prime because this is the b dimension n ok. So, b prime is this one phase and the phase behind it is b. This itself is also in the ON-set. So, that also constitutes an implicant. There are other implicants in this picture. What are they?

Student: (Refer Time: 48:08).

So, this is a function this is a clearly defined function right ON-set is given, OFF-set is given do not care set is given, so that function is specified for us. What are examples of implicant? I have pointed out two examples of implicants. What other implicants are there?

Student: If we choose 001 and 000 to come as one implicant.

This is an example of an implicant yeah. So, there are many other implicant that is also an implicant in fact, whatever this.

Student: Should not found that I mean.

No, it is just is to have it as an implicant. We are saying we do not care, but that means, that I certainly under so the cover then is a set of implicants that includes all the

Student: Minterms.

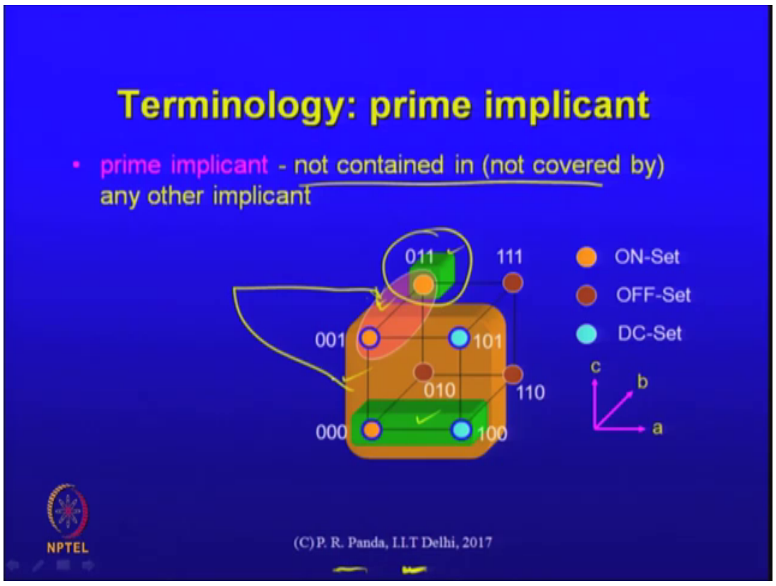
Minterms ok. So, there may be different ways of forming the cover. This is just one example. What other example is there of a different set of implicants that covers this function.

Student: 001, 000 as 1 and then.

This, this one could be one and in fact, that one those two implicants also between them cover the function. These are optional either do not care ones are optional. So, point is I should not include anything from the offset, but I should include everything from the onset yeah, but the other ones I choose are not used depending on my convenience.

So, I expect that the cover would be between the ON-set that is my F ON, and the union of the ON-set. And they do not care set which is all the points that is the maximum number of minterms, I can maximum set of minterms, I can include this is the minimum set of minterms, I can include, but as long as it is between them we call it a cover ok.

(Refer Slide Time: 50:09)



Terminology: prime implicant

- prime implicant - not contained in (not covered by) any other implicant

Let us move on with some other definitions. In fact, this all should be a recap of things that you have done long ago. Prime implicant, the prime implicant refers to an implicant that is not contained in any other implicant yeah, so in this example, I have pointed out some implicants alright, so all of these regions that are highlighted correspond to an implicant these are not the entire set of implicants, but these are some set of implicants. So, this is an implicant, this is an implicant, this is an implicant and that square itself is an implicant. Which of these are prime implicants?

Student: Maybe 011 is a prime implicant.

011 is a prime implicant, prime implicant is defined as it is not fully contained in any other implicant, so is that green one a prime implicant this one consisting of just that one node. The way to answer is, ask if that node or whatever that implicant is it is completely contained in any other implicant, Is it contained or not?

Student: Sir 001 and 011 will.

This one is containing this, completely containing all it is internal minterms, so that is actually not a prime implicant. How about this one that oval?

Student: That one is not completely.

It is not contained in any other implicant, so that is fine. How about this one, the green one?

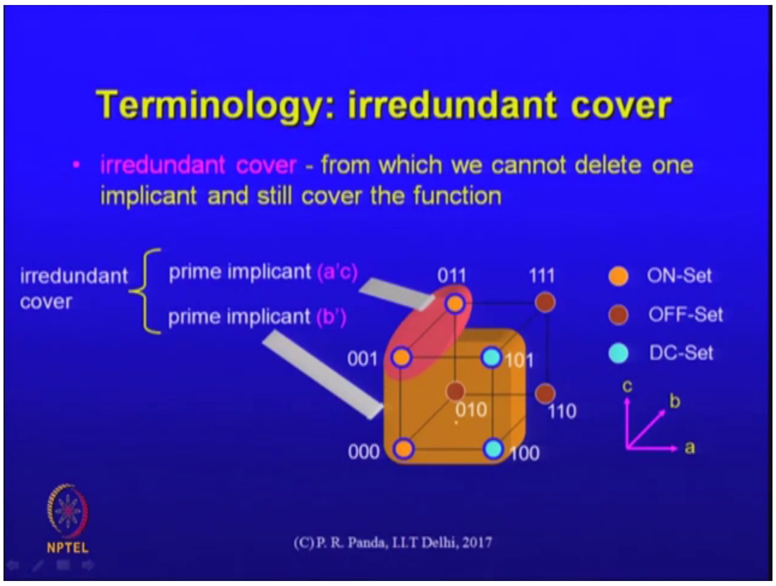
Student: No sir.

It is not, because it is completely contained. How about the orange square?

Student: That is also prime implicants.

Right. So, these two are prime implicants, this one and this these are the two primary considered.

(Refer Slide Time: 52:18)



Terminology: irredundant cover

- irredundant cover - from which we cannot delete one implicant and still cover the function

Continue with the definitions, and irredundant cover there is a reason why we are defining these terms, and irredundant as it is name suggests is one from which you cannot delete an implicant any of those constituent implicants, and still manage to cover the function.

Student: Minterms.

Yeah all the minterms which is covered the function, this should be clear, why we are moving in this direction?

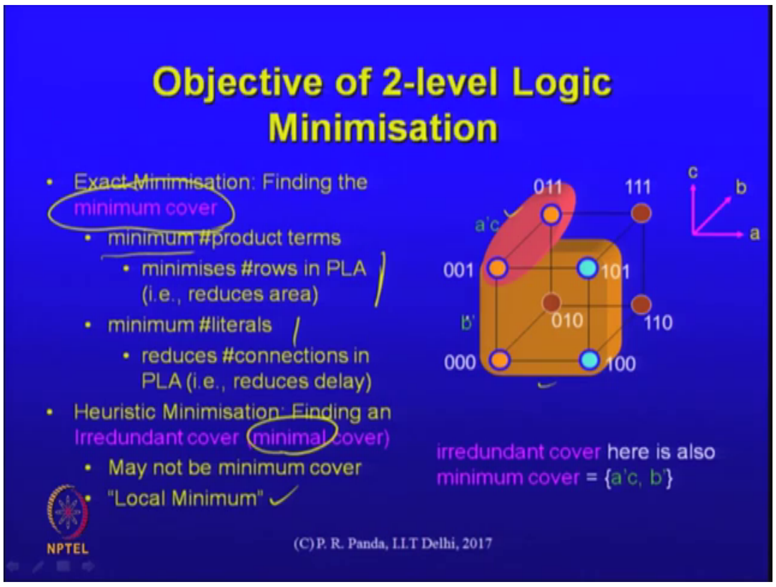
Student: (Refer Time: 52:10).

As opposed to irredundant, if it is redundant cover, it means that there is some implicant that is already covered by some other set of implicants, then why should I keep that cover there, I can get rid of that is where we are heading. So, irredundant is one from which we can delete any. In this example, here we said that this was a prime implicant, this was a prime implicant between those two all the minterms are covered right, so that would be an irredundant cover of that function. There are other implicants there right, but all of them need not be part of an irredundant cover right.

Student: Sir can we say conversion regard, all prime implicants found the irredundant or (Refer Time: 53:36).

All the prime implicants need not be taken, we will come to that yeah. But you could argue that all the constituents of the irredundant cover must be prime implicants ok.

(Refer Slide Time: 53:44)



Objective of 2-level Logic Minimisation

- ~~Exact Minimisation~~: Finding the minimum cover
  - ~~minimum~~ #product terms
    - minimises #rows in PLA (i.e., reduces area)
  - minimum #literals
    - reduces #connections in PLA (i.e., reduces delay)
- Heuristic Minimisation: ~~Finding~~ an Irredundant cover (minimal cover)
  - May not be minimum cover
  - "Local Minimum" ✓

irredundant cover here is also minimum cover = {a'c, b'}

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

NPTEL

So, with those terms being defined let us get to the objective of logic minimization. In an exact minimization, what we want to do is find the minimum cover of course, we are looking for a cover, because that is what our result will be we want to optimize the function, and that optimized function has to contain all the onset points it may contain some of the do not care points if it is useful to do so. And therefore we are looking for a cover, what kind of cover? we somehow have to look for a minimum cover, where minimality is defined in some ways that is actually dependent on the target architecture.

If it is area then we know that the way you would optimize for a multi level logic, may be different from the way you would optimize for two-level logic.

So, for two-level logic minimum cover could be from an area point of view defined as, one that minimises the number of product terms, because that translates to the number of rows in the PLA and that to minimises the area of the PLA. So, it could be just the number of product terms that you are minimizing which is in the picture like this, the number of cubes that you have in the function that is what you are trying to optimize.

It need not be done that way, it could also be that you optimize for timing for instance, and there may be the literal minimization could be alternatively thought of as the cost function. So, this actually they had nothing to do with literals which means that you have two product terms, and it did not matter how many literals are there, if I just want to minimise the number of product terms that is actually how the area function is determined from the PLA that we already saw. Having more literals is better or a smaller number of literals is better?

Student: Smaller smaller.

Smaller must be better that is how we have actually understood. Even area minimization if it is just literal counting, in actually multi level logic it will be back to counting literals for the area too. But here literals are not the first order as far as the area is concerned, but they do affect, they may affect timing in what way; more literals means more connections, if you look at the path, if there are more literals in that path then there are more connections, it increases the load that you are driving ultimately. We will not look at this day, it is easier to maybe argue in terms of, sort of terms and area, but equivalently you can formulate all of these as delay minimizations also in (Refer Time: 56:44).

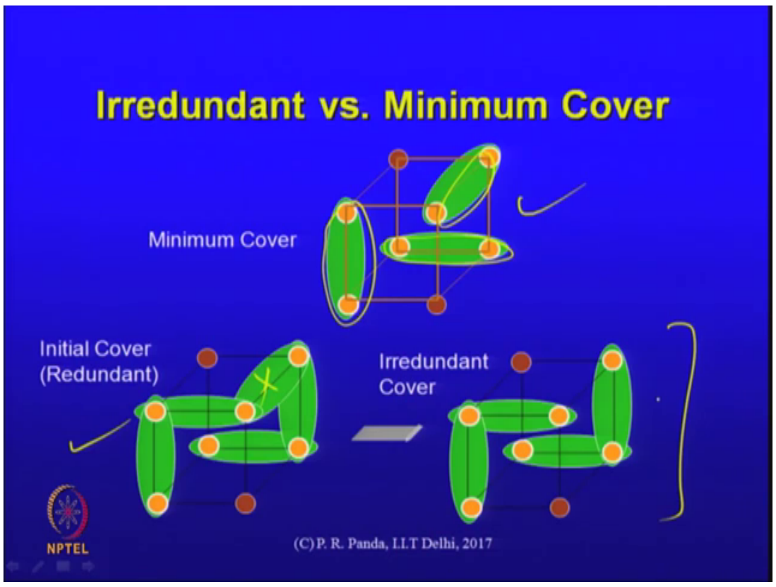
Exact minimization though is a difficult problem finding the minimum cover, so alternatively what an algorithm can try to do in a heuristic minimization is to find an irredundant cover, minimal is distinguished from minimum in this branch of computer science, so minimal means you are not guaranteed that it is the optimum.

Student: Minimum.

Right, minimum is the optimal one. The minimal is just that one from which you cannot trivially drop something and still improve it. Irredundant was defined in that way, irredundant does not guarantee optimality, it just means that you have a set of implicants in there, but you cannot drop any one of them; obviously, and still continue to cover the function ok. So, it may not be minimum, but it could be kind of a local minimum in which you cannot find trivially and; obviously, better solution.

So, in this example here that cover which was irredundant, in fact is also the minimum cover, you cannot improve upon that you can certainly not drop either of these terms from that cover and therefore, it is irredundant. Irredundant though need not be the same as minimum we will see examples later on.

(Refer Slide Time: 58:16)


Irredundant vs. Minimum Cover
Minimum Cover
Initial Cover (Redundant)
Irredundant Cover
NPTEL
(C) P. R. Panda, I.I.T Delhi, 2017

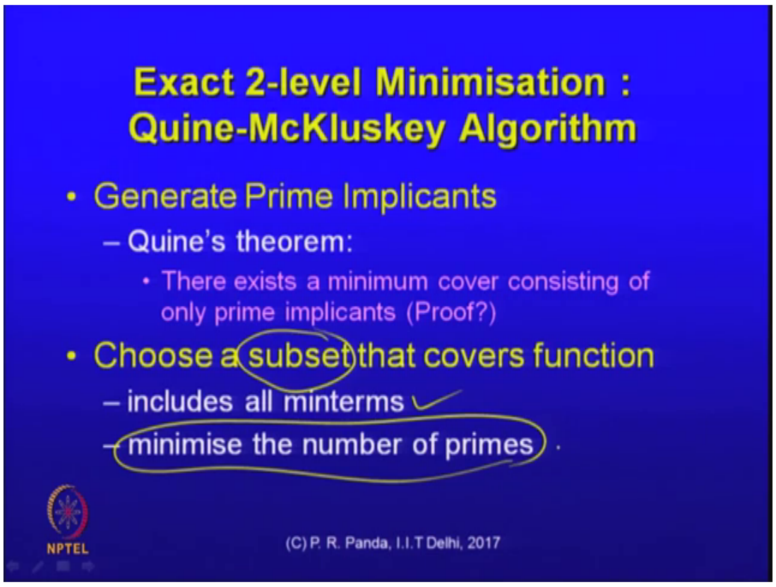
So, let us look at an example here, in which I have this cube of 8 vertices in which two of these brown ones are in the offset; the other six are in the onset. So, the minimum cover is this, like this, this, and this, 3 terms and that is the minimum cover of this. You cannot obviously, grow any of those regions to contain save 4 minterms, all of these would have to minterms each, that is the best we can do. And three is the best we can do, I cannot drop any one of them and still continue to cover all the 6 minterms that is minimum.

However the input might not have been given to me in that form, it is this is the this is the optimal solution, but maybe that initial cover that was given to me was this, it is just some Boolean function that is given or maybe just the entire set of minterms as given or something like that. I could using my logic optimization strategies arrive at a cover like this, this is an irredundant cover.

The original one was redundant why, because one of these could have been dropped. In fact, other terms in that same cover are including one of those implicants, so that implicant could have been dropped, but if I drop that I arrive at this structure, this is an example of an irredundant cover, not optimal, this is not minimal because that is actually the minimum, minimal one would have the three points. But it is just the order in which we did this transformation, let us say we were just trying to drop terms right which were redundant.

And if I decided to drop this one, and I arrived at this that is an example of an irredundant cover, this is it is acceptable as a local minimum we are not guaranteeing anything, but our algorithms in fact are looking for examples of irredundant covers.

(Refer Slide Time: 60:24)



Exact 2-level Minimisation : Quine-McKluskey Algorithm

- Generate Prime Implicants
  - Quine's theorem:
    - There exists a minimum cover consisting of only prime implicants (Proof?)
- Choose a subset that covers function
  - includes all minterms
  - minimise the number of primes

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

Go back a few years, an example of the two-level minimization is this Quine-McKluskey algorithm, which is same as the tabular method for minimizing logic that you should already be familiar with. This is an example of an exact minimization, the objective there was to actually try and find out the minimum cover that is optimal, why it did not work sometimes is that it takes too much time, since you are doing it by hand. You would sometimes not discover that there are certain elements that can be dropped, but the objective indeed is that you try and get the minimum cover.

How did it work, you first generate all the prime implicants, all the prime implicants. And we had a table in which we were checking off things, but the rows of those tables were at the prime implicants. Why prime implicants this relies on first the Quine's theorem that there exists a minimum cover that consists of only prime implicants, you do not have to look for other implicants. What is the proof? In fact, we just said the same thing in a different way, a couple of slides ago, what it means, what this statement means is that a minimum cover will not have an implicant that is non-prime.

Visually can you see that from the diagram itself, what is an implicant that is non-prime, this was an implicant that is non-prime. We are saying that a minimum cover would have only prime implicants, can you argue from this picture that this one should not be there in a minimum cover.

Student: Yes, because the definition of minimum is minimum literals or minimum product terms. So, we are (Refer Time: 62:31).

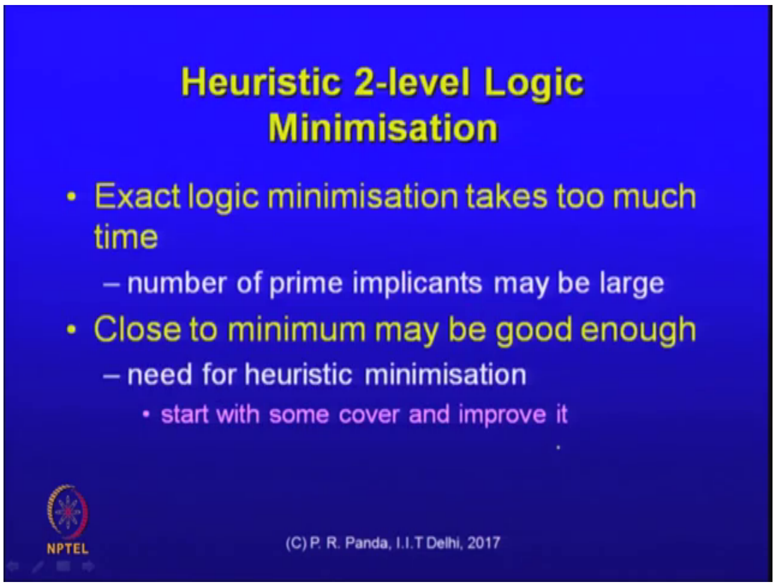
So, they are not literals. So, let us restrict our self to product terms, this is to we are two-level logic, so we will restrict ourselves to product terms. If this term was there a non-prime implicant was there as one of the terms, somebody gave me the set of implicants, and I am now trying to optimize the logic, some of them could be non-prime implicants. If it was there, you could argue that you can replace it by a prime implicant that contains that that fully contains that non-prime implicant right that that is the whole point of defining the prime implicants right.

So, this one must not be there in the minimum cover, because I can just replace it by whatever prime implicant covers it, at the very least it has the same number of terms, by itself it does not reduce the number of terms, but we are just saying that there is a solution with all prime implicates, that is all that we are saying. We are not saying that the solution is better; it is actually not better you are replacing one by the other it does not necessarily reduce the number of terms, it may reduce the number of literals actually; but it does not reduce the number of terms.

And, if my two-level logic the goal is to reduce the number of terms then I am not necessarily improving the quality, but I am saying that I am not losing there would be at least one solution, out of all the optimal solutions in which everything is a prime implicant, and that is why in that algorithm we are looking for prime implicants only.

But all the prime implicants need not be covered, they need not be covered. We can see that from that example earlier, in fact there are five prime implicants in this picture, but all of them need not be part of a minimum cover, some of them could be dropped actually. So, you choose a subset out of those prime implicants that covers the function that subset of course should, include all the minterms is everything in the onset, it should minimise the number of primes this is the difficult part of it. minimise the number of primes because that is what is the objective function, I want to minimise the number of product terms ok.

(Refer Slide Time: 64:46)



Heuristic 2-level Logic Minimisation

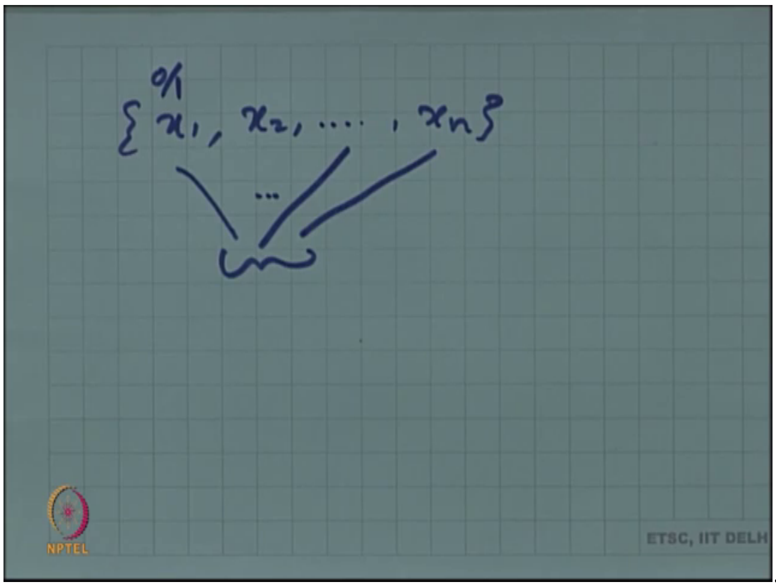
- Exact logic minimisation takes too much time
  - number of prime implicants may be large
- Close to minimum may be good enough
  - need for heuristic minimisation
    - start with some cover and improve it

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

That is the algorithm, but actually that is an exact logic minimization problem statement we are actually trying to find the minimum cover with this, but there are some couple of issues with this, first of all the problem is the number of prime implicants can be very large, cannot get away from it there are functions where the number of prime implicants is exponentially large, the x or is such a function if you look at that function, you will find that the that many implicants are there, you cannot do anything about it, so in the general case this could be very large.

But minimizing the number of primes given a set of primes is also not obvious that is also difficult, because essentially the number of primes means what given a set of n elements, you want to choose m out of them right, but I do not know which m, so I have to actually check out all the subsets right, This also is an exponential number where the number of subsets given a set with n elements, how many subsets are there?

(Refer Slide Time: 66:01)



.

If I have x 1, x 2, you can see the association with this problem. Each one of those is a prime implicant. And my optimal function is some subset of these right, all of those need not be there some can be dropped, but the point is I want to know which ones are to be dropped and I cannot do it exhaustively, if I try to do it exhaustively, how many subsets I need to look at?

Student: Sir, some sort of a problem that assume, one is in the subset that is n, n minus 1, n minus 2.

Right.

Student: Slightly.

This is an exponential number, how many subsets are there of out of n, each of them is either there or not there right, so the number of possibilities is 2 to the power n. So, that is an issue first of all there are a large number of prime implicants. Secondly, the number of subsets also cannot be easily exhaustively tried out. So, that exact logic minimization takes too much time, if it is small enough that is fine, and that is why we were using the tabular method for getting the exact minimum, but for a relatively small number of variables. The number is larger then, it we (Refer Time: 67:32) it difficulties.

So, close to minimum may be good enough even if you cannot get the optimal one, you would have a need for a heuristic minimization, where the strategy is just this. You start with some cover some cover is there it depends on how that input function is represented, but maybe it is a set of implicants some set of implicants that is given maybe it is a an a set of minterms that is given that is also a set of implicants. So, it is there in some form, so that is the default cover. And you just try to improve the quality of that cover by some operations that we will look at, but the heuristic logic minimization essentially uses an overall strategy like this.

What those operations are we have to study, but the overall you can think of it just as a loop in which you try to transform it in some way. It is structure that set of implicants that you are going to choose in some way or the other.

Student: Sir, we will also if going in detail of that paper you gave us in this part.

Which one?

Student: Multi-level logic optimisation must add must add.

No that was for that discussion that was a state encoding discussion only.

Student: It is not related there because that is basically a multi-level logic optimisation problem.

It is see what was the assumption there that you are preparing the input for a multi-level logic optimization tool to work later on that was not doing the optimization. What is the point of state encoding, finally you have a truth table truth table is the beginning for this logic optimization.

So, what we did there was not multi-level logic optimization, what we did there was just anticipating what the logic optimization tool would do later on, and this is that later on, but what would be happening now so that is relevant, but it was relevant for that problem this discussion is almost entirely from the Damocles text book this come.