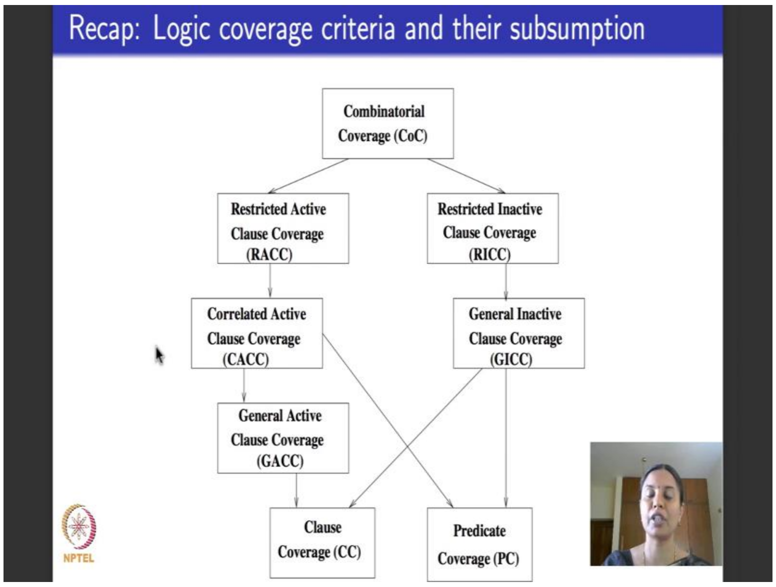
Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 26
Logic Coverage Criteria: Applied to test code

Hello again, welcome to the second lecture of week 6.

(Refer Slide Time: 00:26)

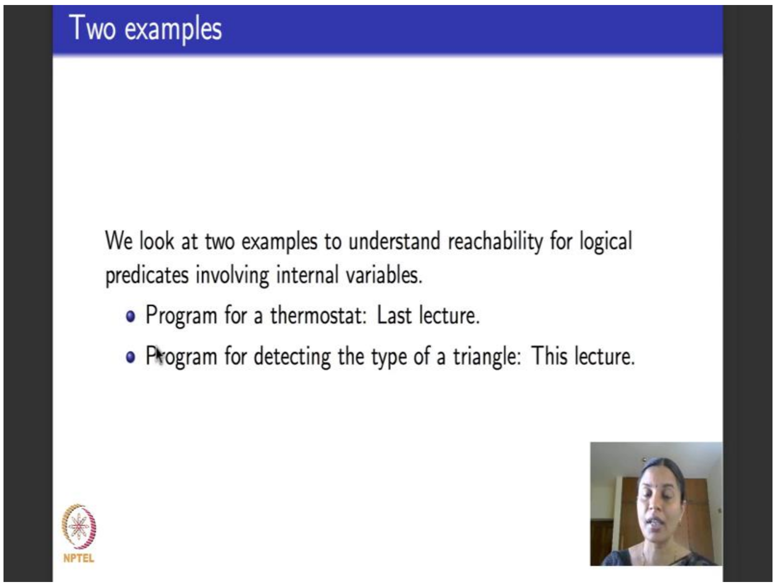


Recap: Logic coverage criteria and their subsumption

$$\begin{tikzcd}
 & \text{Combinatorial Coverage (CoC)} \arrow[dl] \arrow[dr] & \\
\text{Restricted Active Clause Coverage (RACC)} \arrow[d] & & \text{Restricted Inactive Clause Coverage (RICC)} \arrow[d] \\
\text{Correlated Active Clause Coverage (CACC)} \arrow[d] \arrow[ddr] & & \text{General Inactive Clause Coverage (GICC)} \arrow[ddl] \arrow[dd] \\
\text{General Active Clause Coverage (GACC)} \arrow[d] & & \\
\text{Clause Coverage (CC)} & \text{Predicate Coverage (PC)} &
\end{tikzcd}$$

So, what are we going to do today? In this lecture, we are going to look at logic coverage criteria again to test code. So, if you remember last time I told you that logic coverage criteria when it comes to testing code is very important.

(Refer Slide Time: 00:29)



Two examples

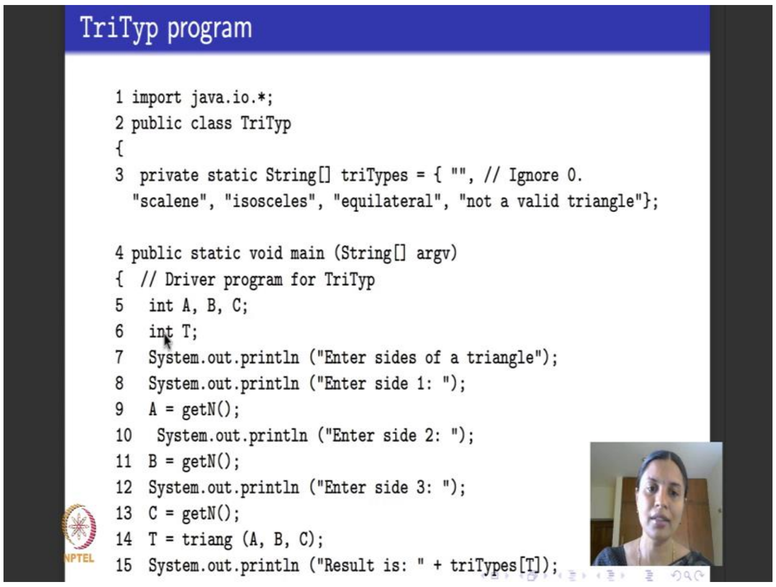
We look at two examples to understand reachability for logical predicates involving internal variables.

- Program for a thermostat: Last lecture.
- Program for detecting the type of a triangle: This lecture.

And to be able to understand the problems related to reachability and controllability well, I will walk you through 2 examples that illustrate the use of logical coverage criteria over source code. In the last lecture, we did the example of thermostat. There was one large predicate with the 4 clauses in that example. We showed how to solve for that predicate, that predicate had the clause with an internal variable. How to reach that predicate how to solve for it, and how to write test cases for predicate coverage. We also saw how to write test cases for code related active clause coverage for that predicate.

In today's lecture, we will take another program which is a very popular program that you will find in a few other text books in software and testing to test for decision coverage or logic coverage.

(Refer Slide Time: 01:27)



TriTyp program

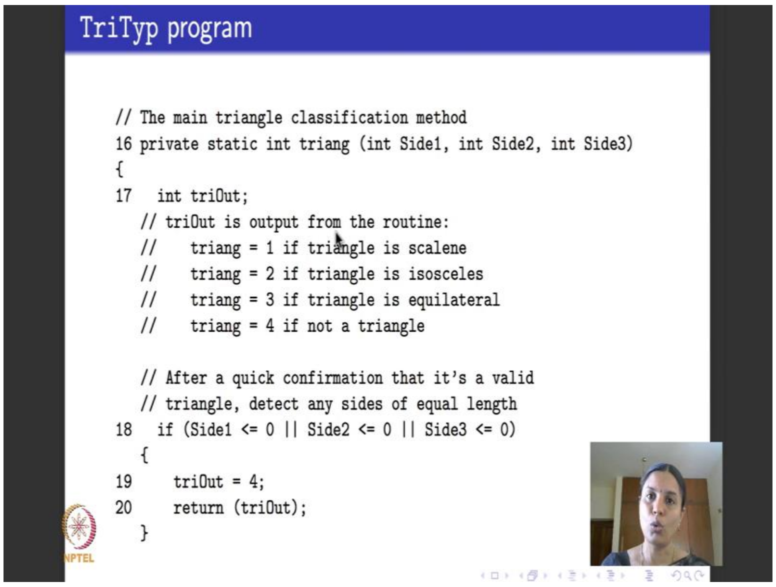
```
import java.io.*;
public class TriTyp
{
 private static String[] triTypes = { "", // Ignore 0.
 "scalene", "isosceles", "equilateral", "not a valid triangle"};

public static void main (String[] argv)
{  // Driver program for TriTyp
  int A, B, C;
  int T;
  System.out.println ("Enter sides of a triangle");
  System.out.println ("Enter side 1: ");
  A = getN();
   System.out.println ("Enter side 2: ");
  B = getN();
  System.out.println ("Enter side 3: ");
  C = getN();
  T = triang (A, B, C);
  System.out.println ("Result is: " + triTypes[T]);
```

And I will tell you how to apply the predicate coverage criteria that we have learnt to be able to do testing for that program. That program is about detecting the types of the triangle. As again this is a program that I have taken from the textbook first edition of Paul Ammann and Jeff Offutt, the authors wrote this program. In other books you might find the control flow graph corresponding to a program and this program basically is called TriTyp, in short for type of a triangle and it detects these 3 types of triangle or it says that it is not a valid triangle.

So, in the triangle could be scalene triangle in which case all the 3 sides are not are of different length in a scalene triangle. In an isosceles triangle 2 of the 3 sides have the same length. In an equilateral triangle all the 3 sides have a same length. Lets say one of the sides that is entered is a negative number then this program will output say in that the triangle that you are trying to give us input in 3 sides is not a valid triangle.

Here is the program. So, what it does is that it has 3 integer variables A, B and C represented to be taken as inputs for the 3 sides of a triangle and then it has an integer T we will see what this does. T is for giving the output T returns the output which is a type of triangle. So, the 3 sides of a triangle that taken as input here. So, the first side is taken as input in A, the second side is taken as a input in B, the third side is taken as a input in C and then this main program calls this method called triang with inputs A, B and C as the 3 sides. This triang runs its code. I have given you the code for triang in the next few slides and returns a number to be stored in T and then the system outputs saying the result is T. So, T is a number that represents one of these 3 types of a triangle or T could be a number that says that it is not a valid triangle. So, here is the main triangle classification method.

(Refer Slide Time: 03:27)



TriTyp program

```
// The main triangle classification method
16 private static int triang (int Side1, int Side2, int Side3)
{
17   int triOut;
   // triOut is output from the routine:
   //    triang = 1 if triangle is scalene
   //    triang = 2 if triangle is isosceles
   //    triang = 3 if triangle is equilateral
   //    triang = 4 if not a triangle

   // After a quick confirmation that it's a valid
   // triangle, detect any sides of equal length
18   if (Side1 <= 0 || Side2 <= 0 || Side3 <= 0)
   {
19     triOut = 4;
20     return (triOut);
   }
```

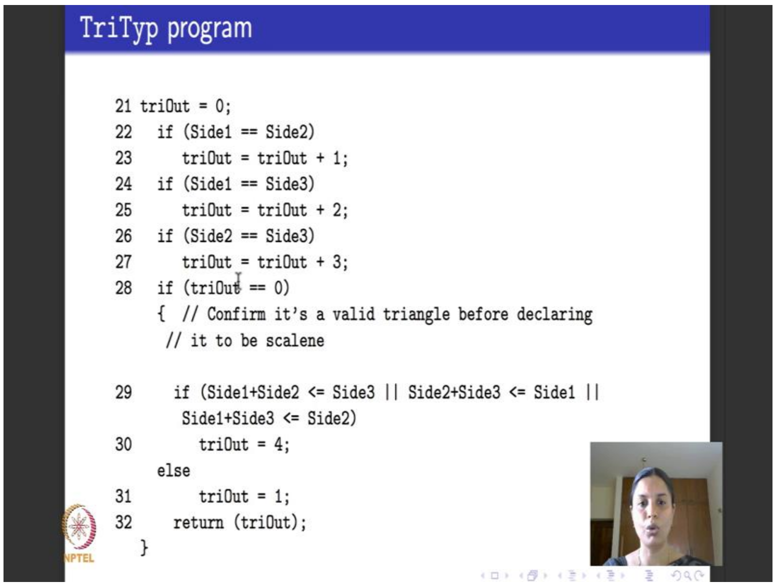
This one, triang. As I told you code will be presented over several slides please read it as a contiguous fragment of code and from end to end the code may not be executable. We are concentrating only on the fragment as its relevant for us to be able to understand test case design.

So, this is the second slide containing a continuation of the same code. It has triang method. So, here is what it outputs? It has an internally variable within this method called triOut; triOut is the output from that method. So, if what does it output? It outputs one if the triangle is a scalene triangle, it outputs the number 2 if the triangle is isosceles, it outputs the number 3 if the triangle is equilateral, it outputs 4 if 3 sides constituted that were entered do not constitute a valid triangle. So, what we first do here in this method is we first check if the triangle is valid or not. Why will it not be a valid triangle ? It will not be a valid triangle if one of the sides is a negative length. So, that is what this if checks for. It says if side one is less than or equal to 0 or side 2 is less than or equal to 0 or side 3 is less than or equal to 0 then you set triOut to be 4. What us triOut to be 4 mean ? If you go above and see triOut 4 mean that it is not a triangle.

So, you set triOut to be 4 and return triOut. So, here basically after taking the 3 sides as inputs in A, B and C, they get passed to the triang method A, B and C. triang method takes them as inputs side1, side 2 and side 3. First thing that it checks is if any of the sides is less than or equal to 0 that is this if statement in line number 18. If any of the 3 sides is less than or equal to 0, it out sets triOut to 4 which we interpret in the code to mean as it is not a valid triangle, returns it and comes out now it. Now after this its ruled out the fact that it is not a valid invalid triangle.

From now on it is working with the valid triangle it just has to find its type. So, what are the 3 types that the code can output? The 3 types that the code can output are whether it is a scalene triangle, isosceles triangle or an equilateral triangle. If it is a scalene triangle it outputs one, if its isosceles it outputs 2, if its equilateral it outputs 3. This is how the code is written.

(Refer Slide Time: 05:50)



TriTyp program

```
21 triOut = 0;
22   if (Side1 == Side2)
23      triOut = triOut + 1;
24   if (Side1 == Side3)
25      triOut = triOut + 2;
26   if (Side2 == Side3)
27      triOut = triOut + 3;
28   if (triOut == 0)
     {  // Confirm it's a valid triangle before declaring
      // it to be scalene

29     if (Side1+Side2 <= Side3 || Side2+Side3 <= Side1 ||
        Side1+Side3 <= Side2)
30        triOut = 4;
     else
31        triOut = 1;
32     return (triOut);
   }
```

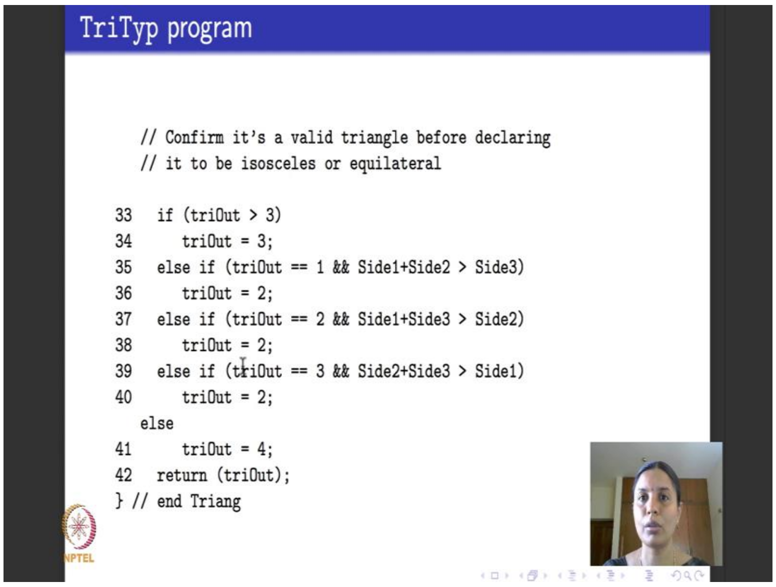
So, here is the rest of the code which determines which of the 3 types the triangle is and outputs the corresponding number. So, another thing that I would like to tell you is that the style in which this particular code is written is slightly long drawn and complicated. You could directly may be write a simpler code that will directly detect the type of the triangle to be one of the 3 types. By no means is this code shortest code, shortest length code and the simplest code to detect the type of a triangle. Its written in a slightly roundabout way with lots of predicates labeling various statements just to illustrate the richness of the logic coverage. Remember we are using this example to understand logic coverage criteria better.

So, this code is written in a slightly long way. It started with lot of the predicates one after the other so that we could use it to understand logic coverage better, but it is a same thing, still correctly computes which is the type of the triangle. So, if it is not 4, which means if it is not a valid triangle, the code continuous here. It sets triOut to be 0 and then what it does it takes one side after the other and compares to see if its equal. So, let us say if side one is equal to side 2 it increment triOut makes it one, if side one is equal to side 3 it increments triOut makes it 2, if side 2 is equal to side 3 it increments triOut and adds 3.

So, by then it would have correctly computed if all 3 sides are equal, then the all 3 if conditions would have passed and the value of the variable triOut would be 3 correctly. If triOut is 0 then what it does here is it checks again. So, it says if side one plus side 2 is less than or equal to side 3, side 2 plus side 3 is less than or equal to side one, or side one plus side 3 is less than or equal 2 sides 2, there is a problem right? It cannot be the case because this violates the property of a triangle. So, it out, but saying it is not a valid triangle else it says yes it is a scalene triangle. So, this comment says that basically confirms that it is a valid triangle before declaring it to be scalene if triOut is 0 and it returns once it computes.

Now in this part it is trying to compute wether trying to figure out whether triangle is isosceles or scalene.

(Refer Slide Time: 08:02)

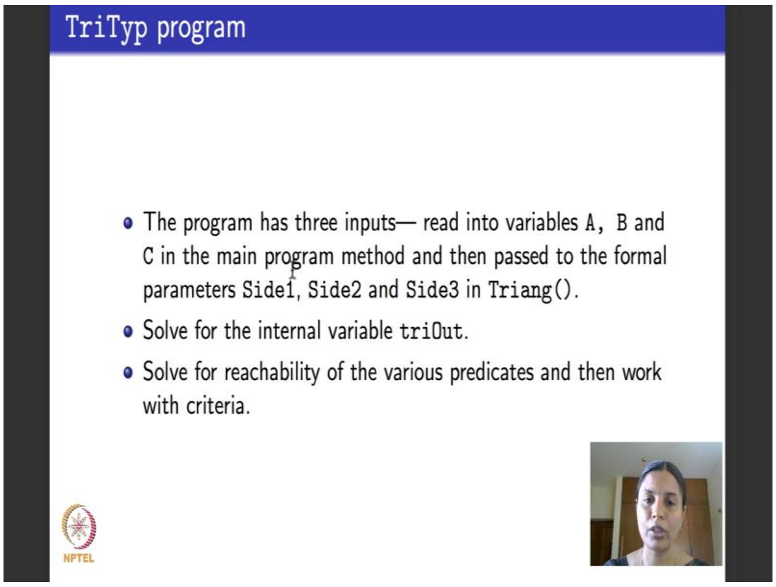


TriTyp program

```
   // Confirm it's a valid triangle before declaring
   // it to be isosceles or equilateral

   if (triOut > 3)
      triOut = 3;
   else if (triOut == 1 && Side1+Side2 > Side3)
      triOut = 2;
   else if (triOut == 2 && Side1+Side3 > Side2)
      triOut = 2;
   else if (triOut == 3 && Side2+Side3 > Side1)
      triOut = 2;
   else
      triOut = 4;
   return (triOut);
} // end Triang
```

So, let say triOut value turns out to be greater than 3. It says if is equal to 3 then you directly set it to be 3 an output. Otherwise now you check if its equilateral or not which means you check if sum of 2 sides is greater than one side. These are the 3 ways in which sum of the 2 of the sides could be greater than the third side those are the 3 conditions here and the corresponding values of triOut are given here. In all the cases it sets triOut to be 2 because that is what it has to output to if the triangle is scalene and it returns that. Otherwise it says if any of these conditions are violated then it says is not a valid triangle and returns it. So, that brings us to an end of the code.

(Refer Slide Time: 08:51)

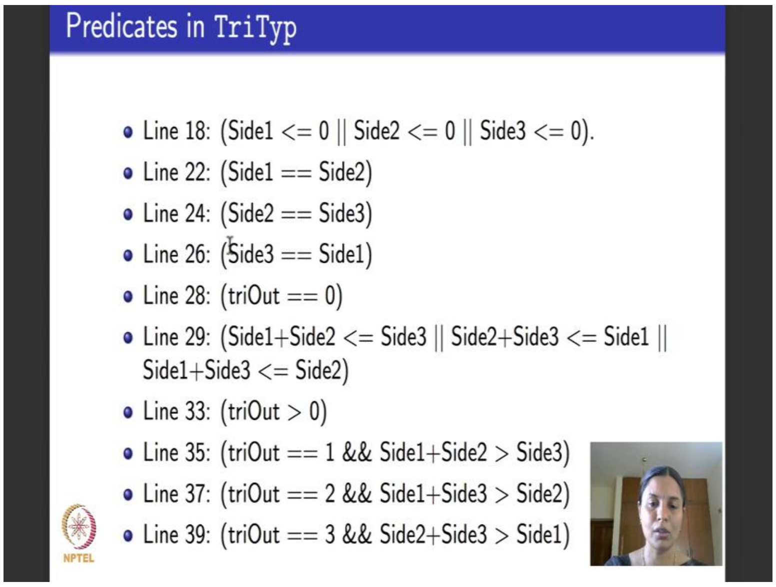


TriTyp program

- The program has three inputs— read into variables A, B and C in the main program method and then passed to the formal parameters Side1, Side2 and Side3 in Triang().
- Solve for the internal variable triOut.
- Solve for reachability of the various predicates and then work with criteria.

So, in summary what is this program have a program has 3 input variables a B and C in the main program method. The 3 inputs are passed as formal parameters to the triangle method as side one, side 2 and side 3. Triangle method has an internal variable called triOut which is set to the value T to be returned back to the main method. So all these predicates, if you see there are so many predicates this triang method: the first predicate that you encounter is here at line number 18 which is this. and then we have 3 more predicates here are at lines 22, 24, 26 and 28, fourth one each of these 4 predicates have exactly one clause each. In line number 29 there is one more predicate with 3 clauses and here again in lines 33, 35, 37 and 39, there are 4 more predicates.

(Refer Slide Time: 09:47)



Predicates in TriTyp

- Line 18: (Side1 <= 0 || Side2 <= 0 || Side3 <= 0).
- Line 22: (Side1 == Side2)
- Line 24: (Side2 == Side3)
- Line 26: (Side3 == Side1)
- Line 28: (triOut == 0)
- Line 29: (Side1+Side2 <= Side3 || Side2+Side3 <= Side1 || Side1+Side3 <= Side2)
- Line 33: (triOut > 0)
- Line 35: (triOut == 1 && Side1+Side2 > Side3)
- Line 37: (triOut == 2 && Side1+Side3 > Side2)
- Line 39: (triOut == 3 && Side2+Side3 > Side1)

So, here is where we have listed all the predicates as they come in the code. At line 18 there was this predicate we checked whether the triangle was valid or not and then this was the part where it starts building towards checking if the triangle is isosceles or equilateral and if it versus scalene triangle it outputs this. Now it does to check if it is a valid triangle in line 29 and then these are the parts where it checks whether it is a isosceles or equilateral triangle.

So, this line what I have done is we have gone through the code here, pulled out all the predicates from the code with their numbers I just listed it here. So, if you are, you find it a little fast you could move back and forth between the code and this and check if the predicate and a corresponding to each line has been currently listed in this code. After this what we want to do?

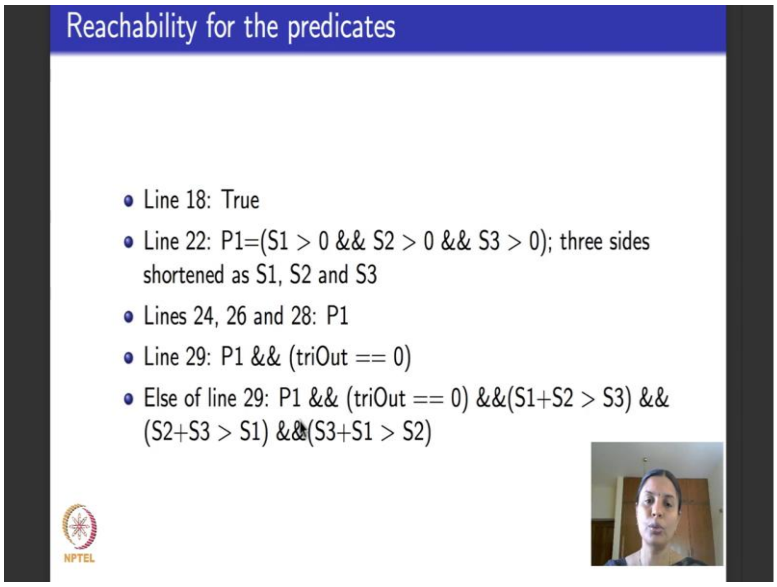
So, this is the set of full predicates that we want to work with. So, you can be ambitious and you can say that I will attempt each of the logic coverage criteria that we learnt for each of the predicates here. You could do that. If you try to do that you will realize that for these predicates to the predicates in line 22, 24, 26, 28 and the one line 33, how many clauses do each of these predicates have? They have only one clause right because they have no Boolean operators in between them and each of these predicates, the whole predicate is just one clause.

So, if you remember for predicates with one clause just predicate coverage is enough because you really cannot do anything much. For other kinds of coverage criteria you need more than one clause of the predicate. So, for each of these predicates you could directly write test cases that will test the predicate to be true once and false once. In fact, please try and do that as a small exercise for these predicates, the predicates in lines 22, 24 and 26, because you just have to give 3 values of sides such that here. For 22 side, one is the same as side 2 and for 24, side 2 is the same as side 3 and for 26, side 3 is the same as side one and then you are done. But for predicates in lines 28 and 33 you have to be able to give a value of triOut. Where does triOut come ? Let us go back and look at the code. triOut is an internal variable that belongs to this method triang which is the main triangle classification method.

So, these predicates in lines 28 and 33 which have only one clause which deals with triOut deal with an internal variable. But what are main inputs to the program? Three sides of a triangle. So, to be able to test this predicate let say triOut is equal to 0, I need to be able to give values for the 3 sides such that triOut is equal to 0 becomes true when the predicate should be true and I should be able to give values for the 3 sides, side one, side 2 and side 3 such that triOut is a value not equal to 0 for this predicate to be false.

Similarly, here for line 33, I should give values for inputs sides one, 2 and 3 such that triOut becomes any value greater than 0 which means it could take 1, 2, 3 or 4, itcould even be an invalid triangle. Yo make this predicates false I must make triOut to be precisely equal to 0 which means I have to be able to give values in terms of inputs which are sides 1, 2 and 3.

(Refer Slide Time: 13:18)



Reachability for the predicates

- Line 18: True
- Line 22: P1=(S1 > 0 && S2 > 0 && S3 > 0); three sides shortened as S1, S2 and S3
- Lines 24, 26 and 28: P1
- Line 29: P1 && (triOut == 0)
- Else of line 29: P1 && (triOut == 0) &&(S1+S2 > S3) && (S2+S3 > S1) &&(S3+S1 > S2)

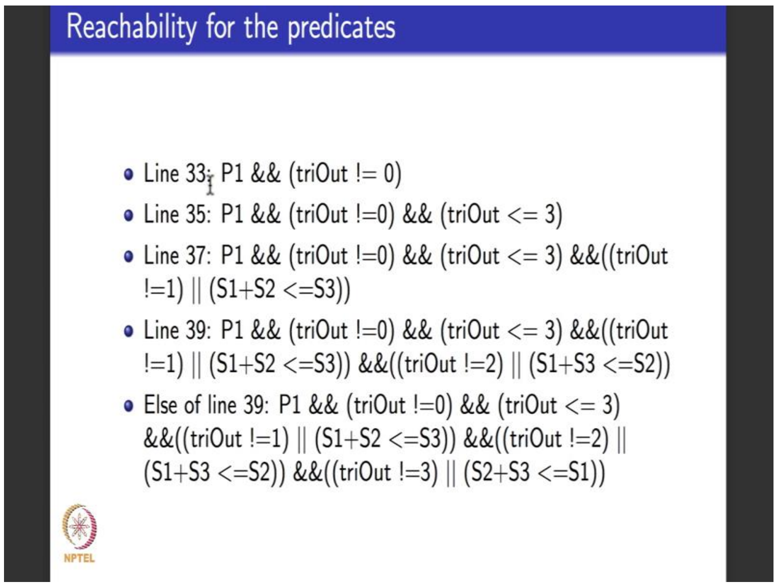
So, what we have to first do is we have to be able to find out reachability of the various predicates. So, from these slides onwards, so, this is the exhaustive list of all the predicates in the program. From these slides onwards I have illustrated this subsequent concept reachability, solving for the internal variable, computing coverage criteria. I have illustrated it only for a subset of the predicates, mainly because it is a repeat exercise if I want to be able to do it for all the predicates and for you to be able to learn it will be good if you can work it out for one or 2 predicates on your own. In particular I have not done for these predicates, in lines 35, 37 and 39.

So, feel free to work them out as a small exercise that you could do for yourself. So, what is the reachability of predicate in line 18? The predicate in line 18, if you go back to the program, happens to be the first predicate. So, it will always be reached because that is right the first statement in the method. So, the reachability for that is true means it will always be reached there is nothing else you need to do.

Now let us look at the next predicate. Predicate in line 22. As I told you to be able to reach here you have to be able to give any values side one, side 2, side 3 such that the predicate in line 18 skipped this if statement does not come and does not exist the method from here. If you do not want the method to exit from here you should be able to give values for side one, side 2 and side 3 such that this if condition is false. Once that becomes false you come here and you start executing the other predicates. That is what is written here. It says in line 22 for we to be able to reach I must give side one to be greater than 0, I have abbreviated as side one S I D E one as S1 just to make it fit neatly into a slide. Similarly side 2 is abbreviated as S2, side 3 is also abbreviated as S3 and this whole thing we have gone ahead and called it as P 1 this is a new notation that I am introducing now so, that I could write P 1 here for reachability of predicates in lines 24, 26 and 28.

So, lines 24, 26 and 28 each of these predicates can be reached provided the values of all the 3 sides of the triangle is greater than 0 that is what it says. Now for line 29, I should get values of all the 3 sides greater than 0 and in addition the internal variable triOut must be equal to 0. Now line 29 was an if part with an else part which I dint give a line number for. Let us go back to the program. if you see this if part has this else part which I dint give a line number for. So, the reachability for that is the negation of the reachability for line predicate in line one.

(Refer Slide Time: 15:58)



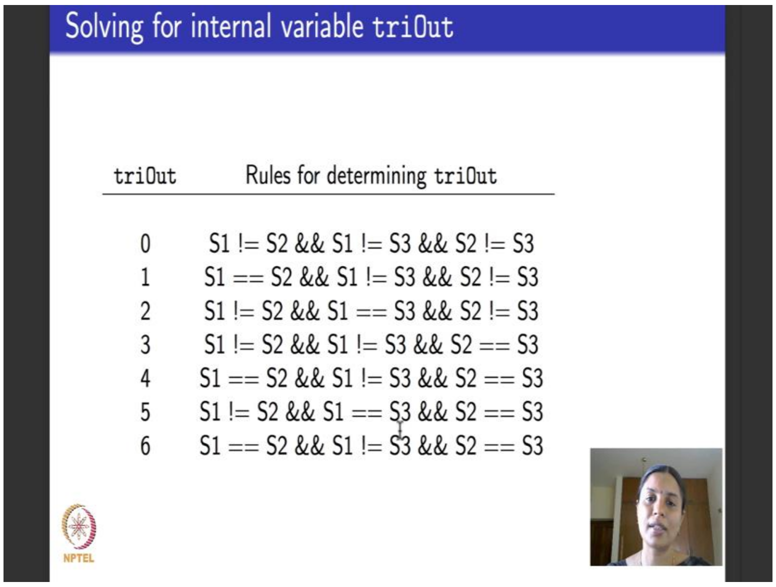
Reachability for the predicates

- Line 33: P1 && (triOut != 0)
- Line 35: P1 && (triOut !=0) && (triOut <= 3)
- Line 37: P1 && (triOut !=0) && (triOut <= 3) &&((triOut !=1) || (S1+S2 <=S3))
- Line 39: P1 && (triOut !=0) && (triOut <= 3) &&((triOut !=1) || (S1+S2 <=S3)) &&((triOut !=2) || (S1+S3 <=S2))
- Else of line 39: P1 && (triOut !=0) && (triOut <= 3) &&((triOut !=1) || (S1+S2 <=S3)) &&((triOut !=2) || (S1+S3 <=S2)) &&((triOut !=3) || (S2+S3 <=S1))

So, which means that it is this and these things are true. So, now, I continue and list reachability conditions for each of the other predicates. In each of these conditions I must be able to skip pass line number 18 which is P 1 and then write appropriate conditions. If you see, all this appropriate conditions deal with an internal variable triOut and there is a condition on triOut. Condition this one says triOut should not be equal to 0, this one says triOut should be less than or equal to 3, this one says triOut should not be equal to 3, this one says triOut should not be equal to 1 and so on.

So, we need to be able to solve each of these predicates for the internal variable triOut. In particular we need to be able to give values for the 3 sides A, B, C or 1, 2, 3 such that here, in line 33, triOut turns out to be not equal to 0. In line 37 triOut turns out to be less than or equal to 3 and so on.

(Refer Slide Time: 16:53)

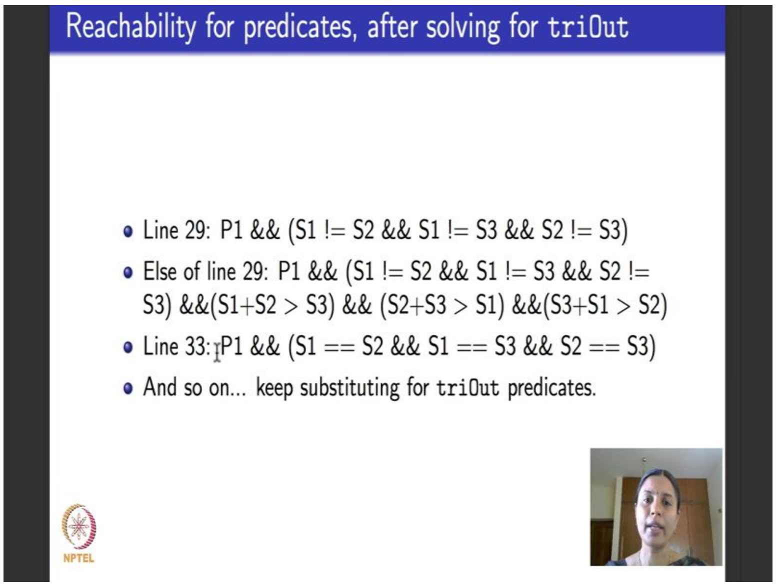


Solving for internal variable triOut

triOut	Rules for determining triOut
0	S1 != S2 && S1 != S3 && S2 != S3
1	S1 == S2 && S1 != S3 && S2 != S3
2	S1 != S2 && S1 == S3 && S2 != S3
3	S1 != S2 && S1 != S3 && S2 == S3
4	S1 == S2 && S1 != S3 && S2 == S3
5	S1 != S2 && S1 == S3 && S2 == S3
6	S1 == S2 && S1 != S3 && S2 == S3

So, how do I solve? This is the table that tells you how to solve for the 3 predicates. So, if you go back look at the program and see when will the variable triOut be, take the values 0. It will take the value 0 if all the 3 sides are not equal to each other. So, that is what is this condition say triOut will be 0 if side one is not equal to side 2 and side one is not equal to side 3 and side 2 is not equal to side.

(Refer Slide Time: 17:34)



Reachability for predicates, after solving for triOut

- Line 29: P1 && (S1 != S2 && S1 != S3 && S2 != S3)
- Else of line 29: P1 && (S1 != S2 && S1 != S3 && S2 != S3) &&(S1+S2 > S3) && (S2+S3 > S1) &&(S3+S1 > S2)
- Line 33: P1 && (S1 == S2 && S1 == S3 && S2 == S3)
- And so on... keep substituting for triOut predicates.

So, similarly triOut will be one if one side a one set of sides are equal. Let us say side one is equal to side 2 and the other 2 are different similarly for 2, 3, 4, 5 and 6 right. These are the values that triOut will take. Now what I do is I go back and take this predicate in line number 29 it says P 1 and triOut is equal to 0. So, for triOut 0 to be 0 this is the condition that I have written out, this is the condition for reachability and controllability for triOut to be 0.

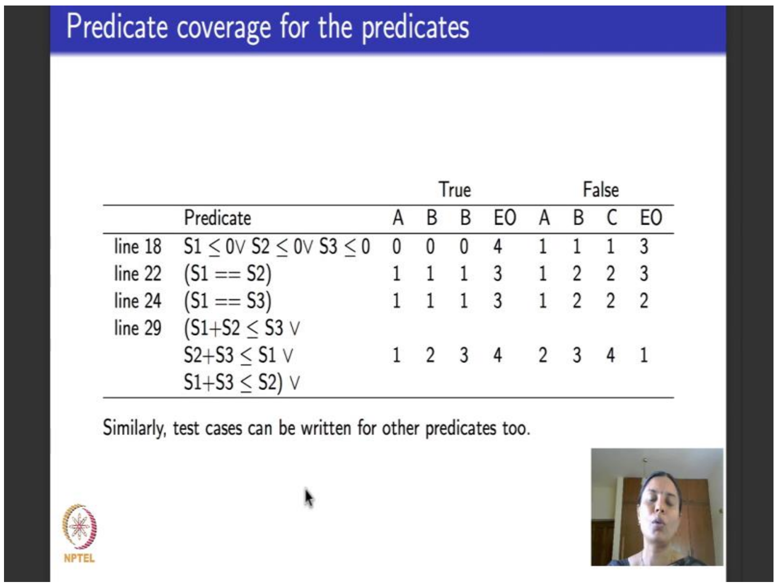
So, I will take that and substitute it back here. Now what I have achieved? I have achieved in rewriting predicate that comes in line number 29 purely in terms of inputs to the program. The predicate P 1 was in terms of the 3 sides and the fact the triOut was 0, I have removed that and rewritten it as another predicate that exactly will map to triOut being 0.

So, this is the predicate that I take and test for predicate coverage or clause coverage or correlated active clause coverage or whatever coverage criteria that I want to apply, whatever logic coverage criteria that I want to apply this is the predicate that I will test it for. What I have achieved in this predicate? what I have achieved is that I have achieved in completely eliminating the internal variable triOut from this program. I have rewritten the predicate to be purely in terms of inputs and outputs.

So, whatever coverage criteria I want to be able to do this in predicate now it becomes easier for me because I can directly give values to the 3 sides such that each of this clauses have true false values to achieve my desired coverage criteria. Similarly the second item here, for the else statement of line 29, I have gone ahead and replaced the value of triOut to be 0 with this rule in the table. So, I have directly gone ahead and substituted, that gives me the long looking predicate and I had go on doing this for each of this predicates right, each of the predicates of in this side line 39, 37, 39 else 39 all of them have specific values for triOut.

So, I go on removing those triOut not equal to 0, triOut less than or equal to 3 and replace it with appropriate values and finish. I have not given them, but you can complete that by substituting it. Now what I have done the set of predicates that I have here, when I complete that set fully is all the predicates in the program with no internal variable in it. The predicates directly talk about the conditions of 3 sides and what are the values that each of this conditions will represent. I am ready to apply all my logic coverage criteria on these predicates.

(Refer Slide Time: 20:02)



Predicate coverage for the predicates

		True				False			
	Predicate	A	B	B	EO	A	B	C	EO
line 18	$S1 \leq 0 \vee S2 \leq 0 \vee S3 \leq 0$	0	0	0	4	1	1	1	3
line 22	$(S1 == S2)$	1	1	1	3	1	2	2	3
line 24	$(S1 == S3)$	1	1	1	3	1	2	2	2
line 29	$(S1+S2 \leq S3 \vee$ $S2+S3 \leq S1 \vee$ $S1+S3 \leq S2) \vee$	1	2	3	4	2	3	4	1

Similarly, test cases can be written for other predicates too.

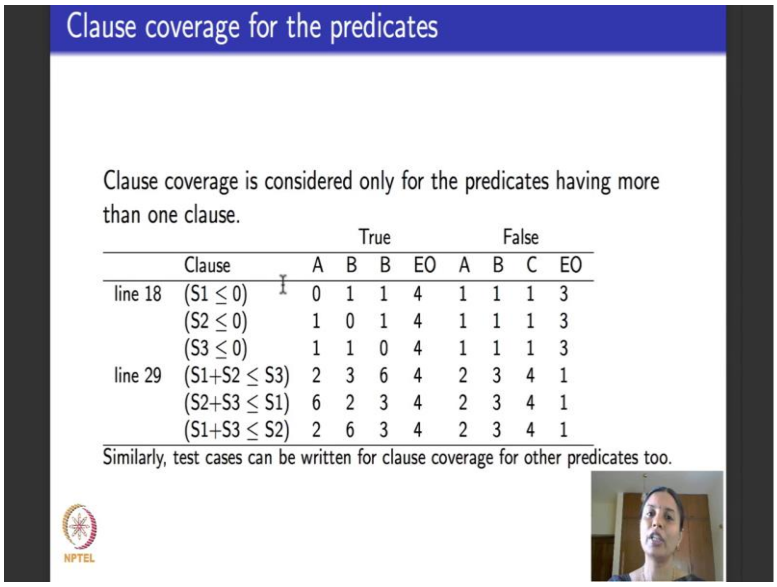
Now, so, what are the various logic coverage criteria that I have to apply? So, for example, I have taken predicate in line 18 which was like this and this is the table that tests for predicate coverage for each of these predicates. I have given you again only a subset of the predicates, you can write test cases for other predicates also.

How do you read this table? This first column here gives you the line number in which the predicate comes in the program. The second column actually lists what the predicate is and these represent test case values for predicate coverage. The first set of values here test when the predicate is true the second set of values here test when the predicate is false. So, you give inputs A, B and C, I am sorry this is small error here, read this B that I am pointing to C and then what does EO represent? EO represents expected output. So, it says if the inputs to the 3 sides of a triangle A, B and C are these 3 then the expected output should be 3 and if the expected output is 3, then predicate coverage is tested for being false.

So, similarly predicate coverage is tested for being true. All the 3 triangles have been put 0, the expected output is 4 which it means it is a not a valid triangle. Similarly for predicate in line 22 which S 1 is equal to S 2, I have given some value here and the output here should be correctly, if you see all the 3 values here are the same, so output is an equilateral triangle. Here the output is false soit is only 2 sides are equal.

Similarly for here it is predicate is true again predicate is false correct because side one is different from side 3. Now in line 29 we had this predicate S 1 plus S 2 is less than or equal to S 3 or S 2 plus S 3 is less than or equal to S 1 or S 1 plus S 3 is less than or equal to S 2 and these are the test case values that will test for the predicate being true. This is the expected output and for the predicate being false these are the test case values and this is the expected output. So, here for some of the predicates in my triangle program, I have written test cases for achieving predicate coverage for both true and false.

(Refer Slide Time: 22:16)



Clause coverage for the predicates

Clause coverage is considered only for the predicates having more than one clause.

		True				False			
	Clause	A	B	B	EO	A	B	C	EO
line 18	$(S1 \leq 0)$	0	1	1	4	1	1	1	3
	$(S2 \leq 0)$	1	0	1	4	1	1	1	3
	$(S3 \leq 0)$	1	1	0	4	1	1	1	3
line 29	$(S1+S2 \leq S3)$	2	3	6	4	2	3	4	1
	$(S2+S3 \leq S1)$	6	2	3	4	2	3	4	1
	$(S1+S3 \leq S2)$	2	6	3	4	2	3	4	1

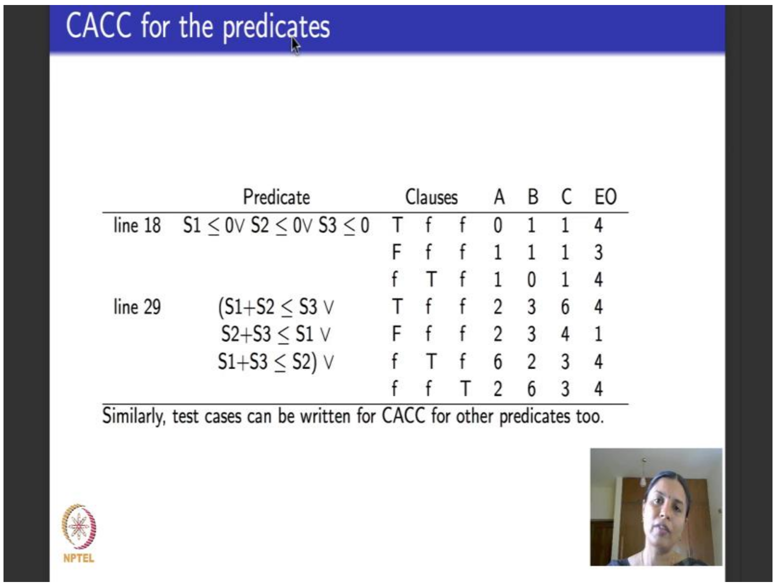
Similarly, test cases can be written for clause coverage for other predicates too.

So, similarly we can do clause coverage. We dint do clause coverage for the thermostat example in the last lecture. That predicate also had 4 clauses there. Just for illustrative purposes I have done clause coverage here for some of the predicates. Please remember one point that we discussed little while ago in this lecture. If a predicate has only one clause then that does not make sense to do clause coverage. So, a predicate should have more than one clause to be able to do clause coverage.

So, I have taken 2 example predicates that have more than one clause. The first one is the predicate in line number 18 that had 3 clauses: S 1 less than or equal to 0 first clause, S 2 less than or equal to 0 second clause, S 3 less than or equal to 0 third clause. Again in the same way, read this whole thing. These are 3 sides of the triangle, expected output for the clause coverage to be true this clause to be true. These are 3 sides of the triangle, expected output for clause to be false.

Similarly, for the predicate in line number 29 there are 3 clauses and these are the test cases that will test for each clause to be true in turn and each clause to be false in turn. Similarly for all other predicates that have more than one clause you could go ahead and write clause coverage.

(Refer Slide Time: 23:28)



CACC for the predicates

	Predicate	Clauses			A	B	C	EO
line 18	$S1 \leq 0 \vee S2 \leq 0 \vee S3 \leq 0$	T	f	f	0	1	1	4
		F	f	f	1	1	1	3
		f	T	f	1	0	1	4
line 29	$(S1+S2 \leq S3 \vee$	T	f	f	2	3	6	4
	$S2+S3 \leq S1 \vee$	F	f	f	2	3	4	1
	$S1+S3 \leq S2) \vee$	f	T	f	6	2	3	4
		f	f	T	2	6	3	4

Similarly, test cases can be written for CACC for other predicates too.

Now, I have attempted correlated active clause coverage for the predicates. Here again to be able to do, consider a predicate like this in line 18: S 1 less than or equal to 0 or S 2 less than or equal to 0 or S 3 less than or equal to 0. It has 3 clauses: you have to make each clause the major clause in turn make it determine the predicate. That is making, we are assuming that this is a compute p subscript a and assuming that this is B, compute p b and assuming that this is c compute p c. Then you populate a table that looks like this which makes the first clause true and false second clause true and false then you have to eliminate duplicate rows.

And then finally, you will get this set of true false values. I have skipped all those steps, but I advise you to strongly try and do that it will be a good exercise and revising how to make each clause a major clause and make it determine the predicate and prune the truth table by removing the rows that occur as repetitions. Once you do that you will realize that you will be reduced with just these 3 rows: the first row makes clause a true; a is the major clause; clauses b and c false. Here is the set of test case for that, here is the expected output. Second row makes clause a the major clause this time its false b and b are false. Again here are the inputs, here is the expected outputs.

Similarly, for this exercise these predicate also we have done this. So, hopefully this second example would have helped you to understand how to do various logic coverage criteria for the predicates in the fairly large program this program that detected the type of triangle was a fairly large program, several predicates. Feel free to write to me on the forum, if you want a have any doubts or if you are not able to complete some of the coverage criteria. Try them out and write to me will be happy to answer any questions.

Thank you.