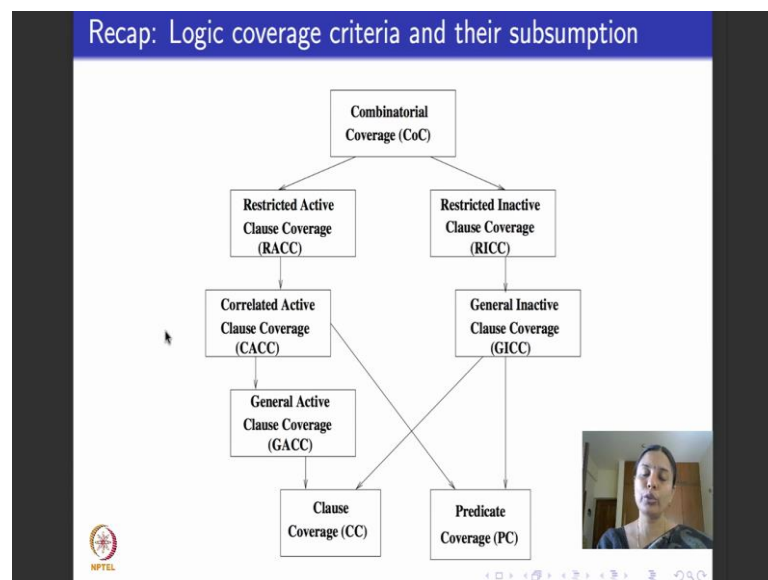


Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 25
Logic Coverage Criteria: Applied to test code

Welcome to week 6, this is the first lecture of week 6. We will continue with logic coverage criteria the whole of this week. Last week we introduced logic coverage criteria, I introduced you to the basics of logic. So, all the coverage criteria, the active clause inactive clause coverage criteria elementary ones, like predicate, clause coverage and then I showed you what was this subsumption relations.

(Refer Slide Time: 00:32)



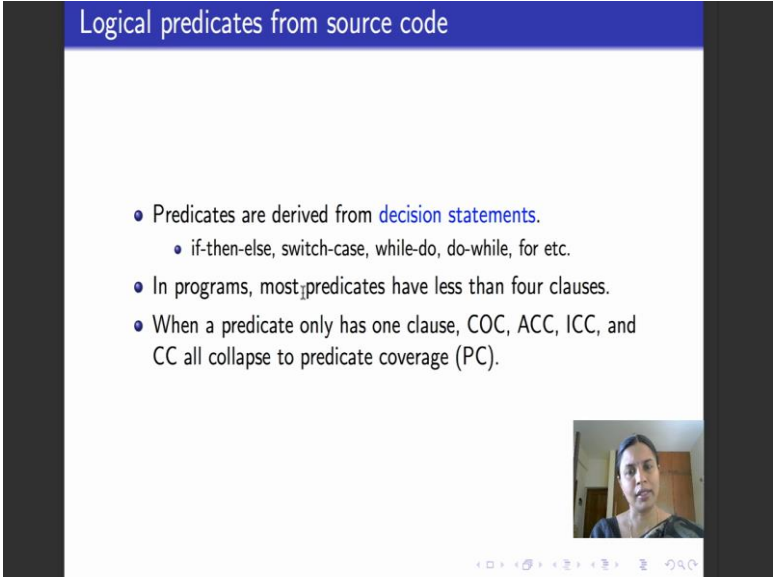
So, this slide is a recap from last week where we saw all these coverage criteria. To begin with predicate coverage, clause coverage then we saw three categories in active clause coverage, two categories in inactive clause coverage. Then we saw this for the sake of completeness, all combinations coverage which is not going to be useful practically at all. What we will see today is, take source code, see how these coverage criteria that we saw can be used to test for source code.

So, in today's lecture I will show you one example of testing using logic coverage criteria for source code. In the next lecture I will show you another example of testing using logic coverage criteria for source code. I decided to do two examples of source

code unlike graphs because logic coverage criteria is quite predominant and popularly used. So, it needs some amount of practice to be able to understand how to put the coverage criteria that we learnt to practical use.

We will do 2 examples of source code this lecture and next lecture after that I will tell you how to look at logic coverage criteria for things like design constraints exactly like we saw for graph coverage criteria, we will look at pre conditions examples and see how logic coverage criteria applies to them. Finally, we will take specifications as modeled as finite state machines with guards and apply logic coverage criteria to guard so then to see how it works. So, that is the plan for this week.

(Refer Slide Time: 02:06)



Logical predicates from source code

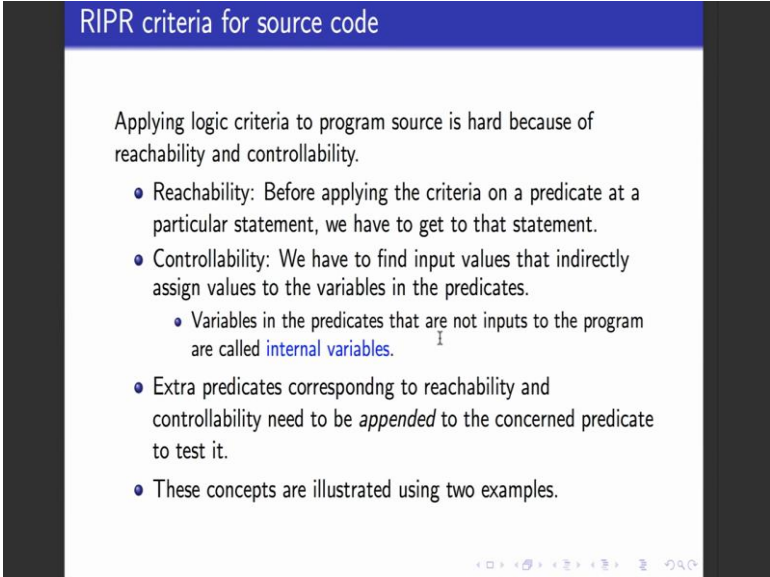
- Predicates are derived from **decision statements**.
 - if-then-else, switch-case, while-do, do-while, for etc.
- In programs, most predicates have less than four clauses.
- When a predicate only has one clause, COC, ACC, ICC, and CC all collapse to predicate coverage (PC).

So, today we begin with source code. When we consider source code where do logical predicates comes from ? They come from every kind of decision statement and the source code which leads to branching in the source code.

For example they could come from if then else statement, it is labeled by predicate every switch case statement is labeled by a predicate statements like while do, do while, do until, repeat until all of them are labeled by predicates, for statements have predicates right. So, programs are started with the decision statements and every decision statement has a predicate in it. So, when I have a predicate corresponding to a decision statement I can apply all the logic coverage criteria that we have learnt to be able to test the program. Typically most programs will have less than four clauses.

In fact, we look at this is the number of clauses in the program and how it matters and then how to apply logical coverage criteria in detail in the third lecture, but typically its consider a good practice or empirical studies in software engineering have shown that most programs have 4 clauses. But typically many programs have predicate where the whole predicate itself is just one clause. It may not may or may not have and ors or other Boolean combinations in which case all the various coverage criteria that we learnt here, all of them basically boil down to only predicate coverage right, which means clause coverage, combinatorial coverage, active clause coverage, inactive clause coverage everything is the same as the predicate coverage because the whole predicate is just one clause.

(Refer Slide Time: 03:40)



RIPR criteria for source code

Applying logic criteria to program source is hard because of reachability and controllability.

- Reachability: Before applying the criteria on a predicate at a particular statement, we have to get to that statement.
- Controllability: We have to find input values that indirectly assign values to the variables in the predicates.
 - Variables in the predicates that are not inputs to the program are called *internal variables*.
- Extra predicates corresponding to reachability and controllability need to be *appended* to the concerned predicate to test it.
- These concepts are illustrated using two examples.

Now, to be able to do logical predicate coverage where do I find predicates? As I told you here, you find them all in decision statements. The same more problem comes again in the problem of RIPR criteria or reachability and controllability. So, when I have a predicate that is deep down in my program that lets say labels in if statement, it could be that the variables involved in the predicate are all internal variables. There could be no inputs, there could be no outputs, but how do we give test cases? We give test cases in terms of inputs and then we say what is the expected output. But let us say predicate has no inputs, it has only internal variables then you have to be able to make sure that a test case in terms of inputs can be re written in such a way that the predicate can be reached

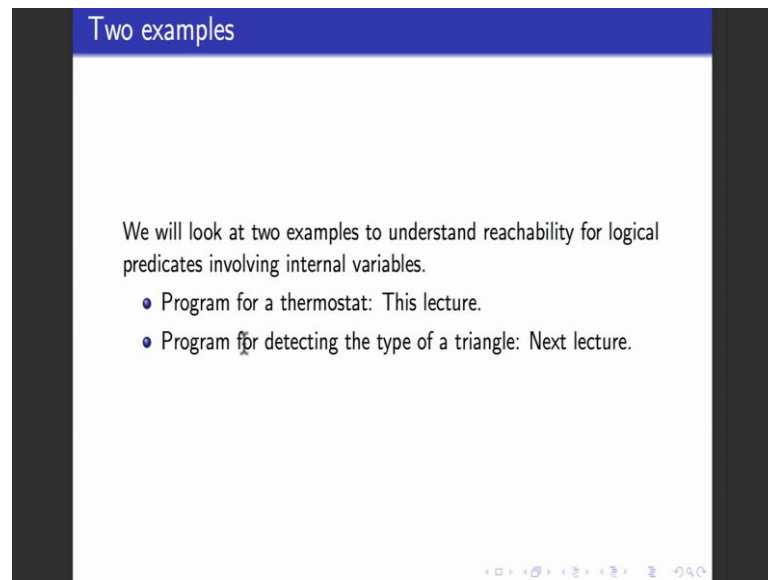
right. You are guaranteed to reach the statement corresponding to the predicate, that is the condition of reachability.

We have to be able to get to that statement meaning, we have to be able to give input values that reach make the program reach the statement by executing all the statements that come prior to that. The next is controllability or what is called infection and propagation. In controllability we have to be able to again give input values in test cases. But in some sense indirectly assign values to the variables that are present in the predicate. Let say there is a particular predicate inside a program that uses all internal variables and I want to test that predicate for predicate coverage. That is make it true once make it false once, but please remember this predicate has all internal variables.

So, to do predicate coverage I should be able to reach this predicate and using input values I should be able to give values to the internal variables such that the predicate is made true once and false once. That is the problem of infection of propagation put together and called as controllability. So, what are internal variables? They are variables in the predicates that do not occur as inputs to the program. So, sometimes to be able to assign values to the internal variables, I need extra predicates where I can add saying that these are the things conditions that have to be satisfied to achieve reachability and controllability and it is the same as adding the conditions as predicate clauses to the original predicate that needs to be tested, because one is to test the predicate I have to be able to reach the predicate and control the predicate. So, I add these as extra conditions.

So, we will see all this through the example of thermostat in today's lecture and then we will see one more example in the next lecture. What is reachability, what is controllability what do we mean by appending extra predicates to the original predicate that corresponds to reachability and controllability, we will see it through we using examples.

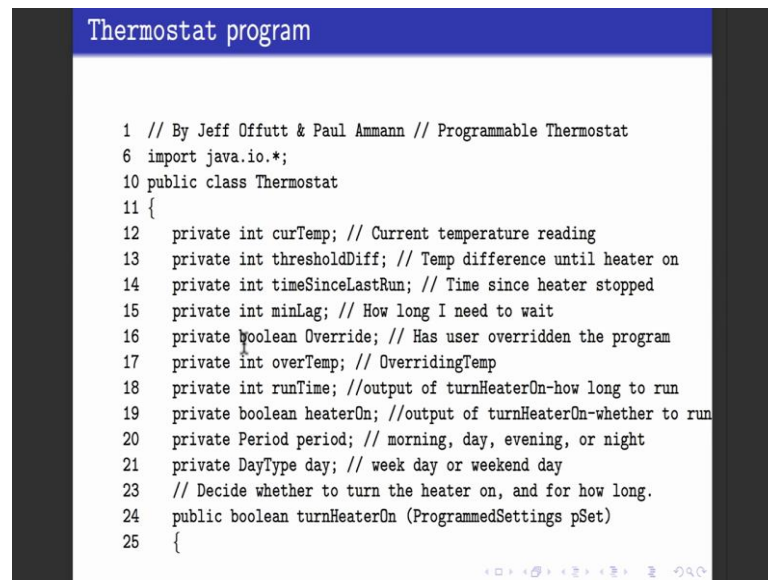
(Refer Slide Time: 06:25)



So, as I told you logic coverage criteria is probably one of the most important testing criteria, very powerful used across board in all kinds of software so it is important to get that correct. So, I will do source code for logic coverage criteria and illustrate the problems of reachability and controllability through two examples. The first example we will be doing is a program for a thermostat.

What is the thermostat? A thermostat is a device that controls the heating of a particular entity. So, there is a small piece of java program that is written for a thermostat. I will show you the program we look at the predicates in the program and see how to apply logic coverage criteria. In the next lecture for this week I will take you through another program which is an elementary high school program which basically takes the inputs as three sides of a triangle and tells you what that what is the type of the triangle? Is this an equilateral triangle isosceles triangle and so, on is a very good old program that not only in this book in this lecture you will find it in several lectures in testing. So, it is a nice thing, a nostalgic thing to be able to go through that program once again. So, what we will do today is we will begin with the thermostat program.

(Refer Slide Time: 07:31)

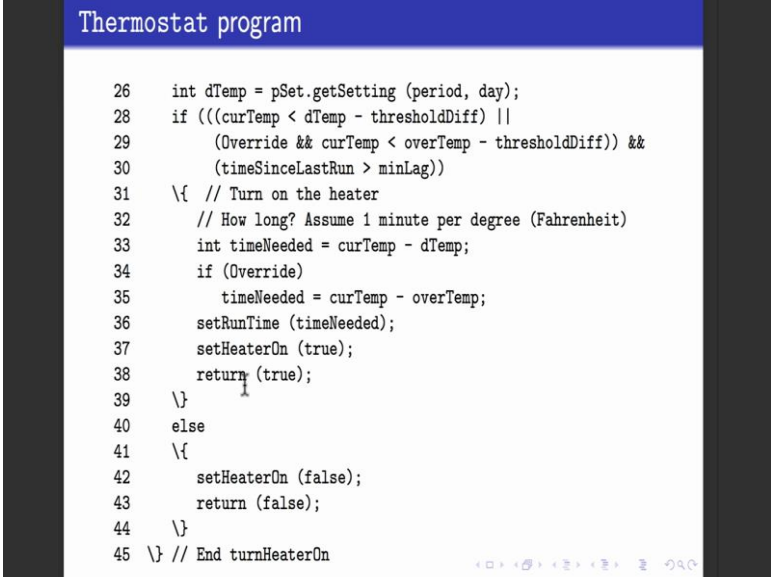


So, here is the code for the thermostat program that is taken from the textbook that we are following the book on Introduction to Software Testing. I have taken the code as it is the authors of the code or the authors of the book as its acknowledged here. So, there is a class called thermostat. What do these lines twelve to twenty one give you they give you all the variables that this program handles. So, there is an integer variable called current temp here right hand side the comment tells you what it represents, it represents the reading of the current temperature it could be the temperature of a particular room right. And then next variable is called threshold difference abbreviated as threshold diff, it tells you what is a temperature difference until the heater gets switched on and then the third integer variables says time since last run, which means what was the time since the heater stopped last time.

So, what is the thermostat trying to do? Maybe its controlling a room in a building in a very cold region it is trying to keep the room heated in a particular ambient temperature. So, this program tells you when to switch it on, when to switch it off and what will be the basic functionality of a thermostat. So, the fourth variable is a Boolean variable called override; override means the user switches off the automatic control in the thermostat and he is trying to configure the temperature setting himself. Then there is one more integer variable called overriding temperature which means when thermostat is in the over written mode what is the temperature that it is been configured for. And there is another integer variable called runtime which is, when to run the heater on, when to turn

the heater on and how long to run it and then there is something called period which tells you whether its morning day evening or night. Maybe it implicitly influences a temperature in which the thermostat will be running and then there is another type enumerated data type called day, which is of the type weekday or weekend day. So, here is the main code corresponding to the thermostat. So, the method that is called is called turn heater on which takes program settings input as ptest to the program. So, there is a pre programmed setting for the thermostat that is turned, that is used to be able to turn the heater on.

(Refer Slide Time: 09:43)



```
Thermostat program

26  int dTemp = pSet.getSetting (period, day);
28  if (((curTemp < dTemp - thresholdDiff) ||
29      (Override && curTemp < overTemp - thresholdDiff)) &&
30      (timeSinceLastRun > minLag))
31  \{ // Turn on the heater
32      // How long? Assume 1 minute per degree (Fahrenheit)
33      int timeNeeded = curTemp - dTemp;
34      if (Override)
35          timeNeeded = curTemp - overTemp;
36      setRunTime (timeNeeded);
37      setHeaterOn (true);
38      return (true);
39  \}
40  else
41  \{
42      setHeaterOn (false);
43      return (false);
44  \}
45  \} // End turnHeaterOn
```

So, here is the code corresponding to the thermostat. Please remember I had in all these slides when I show a piece of code I showed across several slides. So, that it becomes visible, but you should, when you read it read it as a piece of code that is meant for continuous execution. As I told you when we did graphs it might be the case that the code as it is may not be taken in readily executable, it could be a fragment of a code.

But we will focus on the things that we need in the code. So, I may not give you the complete code and the code will run through several slides. So, here is the second slide where the code continues, this is the code corresponding to this method that begins in line in twenty four. So, what is this say? This method has another internal integer variable called dTemp which basically tells you some, it ask you to input the steeing, which is the period which is day through this right pSet, and then it has this code. So, it

says there is an; if statement which has several conditions here. So, it says if current temperature is less than d temperature minus threshold difference or read this thing as or override is true and current temperature is less than override temperature minus threshold difference and time since last run is greater than minimum lag.

So, what it says is basically current temperature is, the room is become cold right that is what the first one says. Second one says that the override is on Boolean variable overhead is on which means users trying to make over and run the thermostat and the temperature in the override mode is still minus the threshold, current temperature is still less than that. So, the room is again cold even in the override mode and heat has been off for some time. So, what do I have to do? I have to turn on the heater right, when I say I have to turn on the heater, we also have to specify for how long to turn on and assume that the heater heats at the rate of per degree of heat, per degree Fahrenheit of heating, it takes one minutes to heat, right. So, using this gradation in the temperature of increase in the temperature of heater is specify how long to keep the heater on? If you keep it on for too long maybe the room will become too hot. So, you need to be able to set it. So, that the room temperatures just optimal and comfortable.

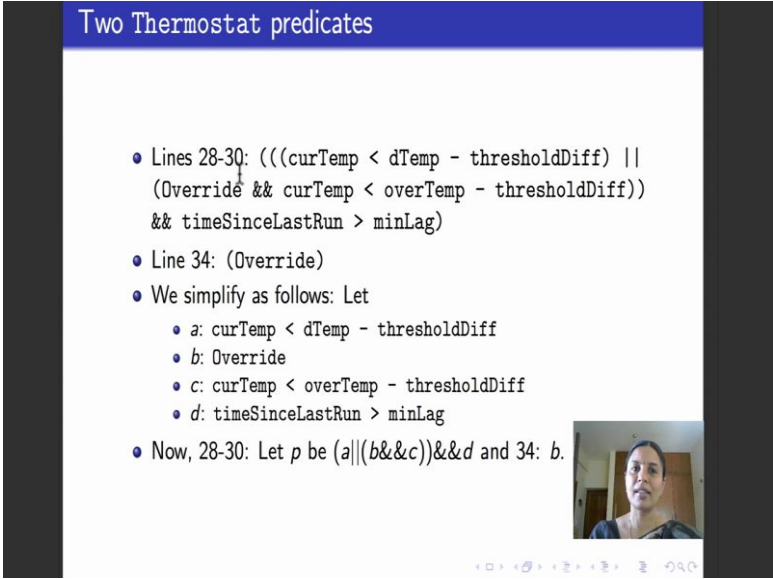
So, this is the code that does that. So, it first computes the time needed to heat it the turn the heater on which is the difference between the temperature in the room and the desired temperature dTemp. And then it says if override is on hen the diff time needed to turn the heater on is the difference between the current temperature. And the overridden temperature and then it says set run time to be time needed, set heater on to be true and then you return true. This is the method that returns a Boolean variable right. Otherwise you said set heater on is false then you return false.

So, is it clear please, what the simple code does? Tts the code corresponding to a thermostat, tries to keep the temperature in a room in a building into optimal setting it uses all these variables, which basically tell you what is the current temperature for how long is the heater been on or off. And whether the heater is been overridden, the override mode being taken by user in which case what do I do. And this is and what sort of it day, it is morning day evening night or is it a week day, weekend maybe weekdays where all the people in the house occupants in the house are away. So, you do not have to turn the heater on maybe in weekends you have to keep it on for longer all these things matter. Then it uses one Boolean method called turn heater on which basically decides whether

to turn the heater on or not. So, that Boolean method has an internal variable dTemp and then it has if temperature, decides how long to turn on the heater on and does this a setting and turns sets the heater on variable to be true. Otherwise it keeps the heater off and returns false.

So, now our goal is to be able to do this program, test this program by applying the logic coverage criteria that we have learnt. So, the first thing to look for when we apply logic coverage criteria is; what are the predicates in the statement the program and where are they? So, you go back and look at the code in this program. The first part of the program is just declarations, no predicates here, obvious. Second part of the program has this long statement here, this one if statement which is this between lines 28 twenty nine 30 and then there is one more if statement here, at line 34 if override.

(Refer Slide Time: 14:12)



Two Thermostat predicates

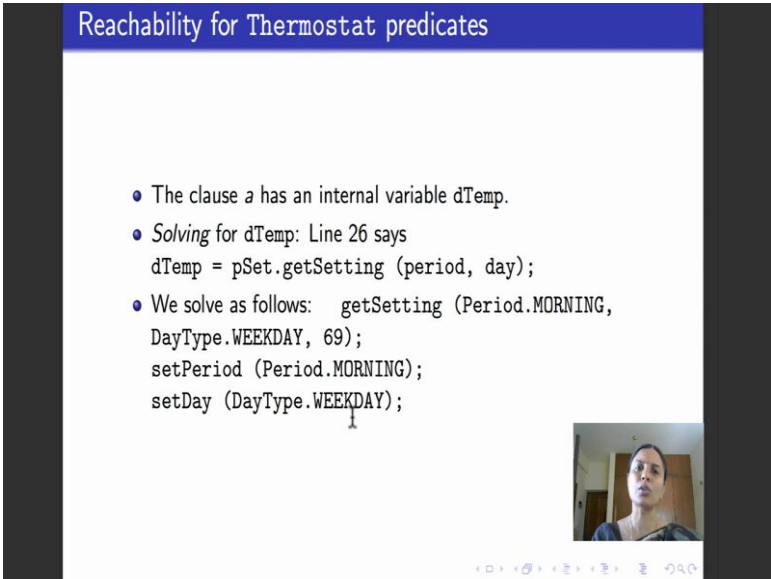
- Lines 28-30: `((curTemp < dTemp - thresholdDiff) || (Override && curTemp < overTemp - thresholdDiff)) && timeSinceLastRun > minLag)`
- Line 34: `(Override)`
- We simplify as follows: Let
 - a: `curTemp < dTemp - thresholdDiff`
 - b: `Override`
 - c: `curTemp < overTemp - thresholdDiff`
 - d: `timeSinceLastRun > minLag`
- Now, 28-30: Let p be $(a || (b \& \& c)) \& \& d$ and 34: b .

So, what I have done in the next slide is that I tell told you what the 2 thermostat predicates are. So, there is a predicates in line 28 to 30. I have just copied the predicate that occurs as label of the if statement here and there is one more predicate called override at line 34. So, this is a pretty long thing right and may be difficult to read. So, let us simplify it we call a as this: current temperatures less than d temperature minus threshold difference. We call b as override, we call c as this clause current temperature is less than override temperature minus threshold difference and we called d as time since last run greater than minLag. So, there are 4 clauses: a which is the first one, b override,

c which is the third one and d which is the fourth one. Now the predicates looks readable in the notation that we have used to understanding it.

What is a predicate? Basically it is a or b and c and d right where a is this first part b is this override c is this third part and d is the fourth part, And then next predicate override is just one clause predicate b. So, this predicate first predicate p which occurs at lines 28 to 30 is interesting for us to test because it has got 4 classes a, b c and d and it is called combination of ors and ands.

(Refer Slide Time: 15:25)



Reachability for Thermostat predicates

- The clause *a* has an internal variable *dTemp*.
- Solving for *dTemp*: Line 26 says
`dTemp = pSet.getSetting (period, day);`
- We solve as follows: `getSetting (Period.MORNING,
DayType.WEEKDAY, 69);
setPeriod (Period.MORNING);
setDay (DayType.WEEKDAY);`

So, let us go ahead and test it for that predicate right. So, now, if you look at this predicate, this predicate has one internal variable called *dTemp*. We will go back to the code for a minute. Remember here *dTemp* is a internal variable that was local to the method turn heater on. The rest of the variables in these predicates are all inputs and outputs. So, as I told you, we have to be able to do reachability first which means we have to be able to solve and make sure that a predicate occurring in these lines is actually reached. Which means what ? In this case because it is a smaller program, it is easy to do solving here means just getting appropriate values for *dTemp* give a value for period and day for *dTemp* and let it assign, let it get assigned to that value and then you will be able to reach this predicate. There are no other conditions. Clause *a* as I told you has an internal variable *dTemp*.

So, I have to be able to solve for dTemp. So, I just give some values. So, this is the line that it is given in the program and look at it. So, it says this pSet is get setting period. So, I give it some value as a period is morning, day as a weekday. So, that is it I have assigned value to dTemp.

(Refer Slide Time: 16:39)

Predicate coverage: p being true

- a: $\text{curTemp} < \text{dTemp} - \text{thresholdDiff}$: true
- b: `Override`: true
- c: $\text{curTemp} < \text{overTemp} - \text{thresholdDiff}$: true
- d: $\text{timeSinceLastRun} > (\text{minLag})$: true

Now, what I do I take back and these are the 4 clauses a, b, c and d. Please look at the title of the slide it says I am testing for predicate coverage of this clause for the value of the predicate p being true. So, what sort of a predicate is that? That is a predicate that looks like this; it has got 2 ands and an or so, one simple way of making the whole predicate true is to make all the 4 clauses true which means I make a, b, c and d all of them true. That is what I have written here. I say a which is current temperature less than desired temperature minus threshold difference is true b is true, c is true and d is true. So, to make all these things true what should I give I should give values for current temperature such that current temperature values indeed less than desired temperature minus threshold difference.

Override is a Boolean variable. I make it true directly. Similarly to make c true, I should be able to make current temperature true and make the overridden temperature minus threshold difference to be a value greater than the current temperature and so on.

(Refer Slide Time: 17:42)

Predicate coverage: p being true: Test cases

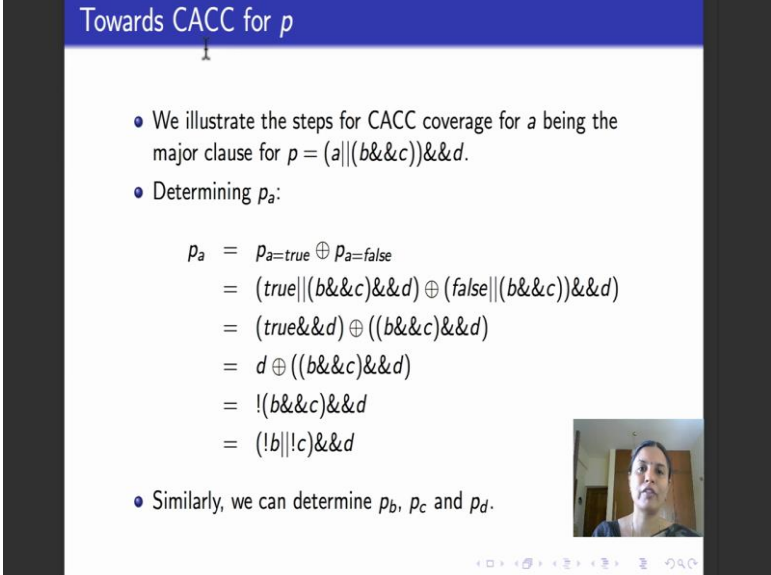
- `thermo = new Thermostat(); // Needed object`
- `settings = new ProgrammedSettings(); // Needed object`
- `settings.setSetting (Period.MORNING, DayType.WEEKDAY, 69); // dTemp`
- `thermo.setPeriod (Period.MORNING); // dTemp`
- `thermo.setDay (DayType.WEEKDAY); // dTemp`
- `thermo.setCurrentTemp (63); // clause a`
- `thermo.setThresholdDiff (5); // clause a`
- `thermo.setOverride (true); // clause b`
- `thermo.setOverTemp (70); // clause c`
- `thermo.setMinLag (10); // clause d`
- `thermo.setTimeSinceLastRun (12); // clause d`
- `assertTrue (thermo.turnHeaterOn (settings)); // Run test`

So, here are a set of values that I gave for the predicate being true. So, I need this object thermostat, I need the object program settings and then I pass values to dTemp by saying period is morning day type is weekday and then I set let say current temperature to be 63, threshold difference to be 5, set some value. Is it clear that current temperature my this a will become true right. Because current temperature will be less than d temperature minus threshold difference why because this is 69 no, and then override has to be true. Now I am working on clause c. I have set overridden temperature to be 70. So, that lag n makes c to be true because current temperature will be less than overridden temperature minus threshold difference, 65 is less than, 63 is less than 65 and then to make clause d true, I set minLag and time since last run such that the difference is, time since last run is greater than minLag.

So, time since last run is twelve minLag is 10. So, it is greater than 10. So, fine, right? So this slide, what is it contain? It contains the final set of test cases that will test which predicate, this predicate p occurring at lines 28 to 33 for predicate coverage being true. So, similarly for predicate coverage being false, I have to be able to give test cases that will make a, b, c and d true or false appropriately. So, for the true for this predicate to be false, what could I do? I should make d false because it is and. Once I make d false, a, b and could e anything because anything anded with false becomes false or I could make this whole thing false along the d being true or false. So, like added here I decide what I want to do in terms of making each of the clauses true or false to make the predicate b

false and then I give values such that they are met. I have not given them to you in my slides, but I hope it is clear how to do it.

(Refer Slide Time: 19:47)



Towards CACC for p

- We illustrate the steps for CACC coverage for a being the major clause for $p = (a || (b \& c)) \& d$.
- Determining p_a :

$$\begin{aligned}
 p_a &= p_{a=true} \oplus p_{a=false} \\
 &= (true || (b \& c)) \& d \oplus (false || (b \& c)) \& d \\
 &= (true \& d) \oplus ((b \& c) \& d) \\
 &= d \oplus ((b \& c) \& d) \\
 &= !(b \& c) \& d \\
 &= (!b || !c) \& d
 \end{aligned}$$
- Similarly, we can determine p_b , p_c and p_d .

Now, for the same predicate in lines 28 to 33 I want to be able to let say I tempt correlated active clause coverage which was another coverage criteria that we saw. You could attempt any other coverage criteria, you could do clause coverage, you could do generalized active clause coverage you could do inactive clause coverage, but I have put the CACC is an example to tell you how correlated active clause coverage is done for p . So, if you remember the lecture from last week, to be able to do correlated active clause coverage, we have to take each clause to be a major clause and first make the major clause determine the predicate. Once the major clause determines the predicate, I write one set of test cases that make the predicate true when the major clause is true and one set of test cases that make the predicate false when the major clause is false or true.

So, this predicate has 4 clauses a , b , c and d . Each of these 4 clauses can take turns to be the major clause and each of them can determine p . So, I will worked out for when a will determine p . If you remember from the last weeks lecture when a determines p will be called p_a . So, p_a was obtainable by using this formula. So, you make a true and xor it with making a false in the same predicate. So, I have reworked this solution for p_a . So, p_a true is what? Substitute a to be true in this predicate. So, you will get true or-ed with b and c and d xor with false or-ed with b and c and d . Now you simplify this, true or-ed

with b and c is nothing but true. So, that b true and d false or with b and c is nothing, but b and c. So, that be b and c and d, I go on simplifying it further then I get this final value.

As I told you in case you are not clear about how to go from these fourth line to fifth line and fifth line to sixth line, why are they equivalent ? You could stop at fourth line or fifth line, they are equivalent. So, in terms of obtaining p_a assuming that you would stopped at fourth line or fifth line that is also good enough. You need not know how to simplify this to be able to get the last line. Even if it is stopped here it is good enough. So, same way we can determine p_b , p_c and p_d . I have not worked it for you, but in the same way you could do that. In fact, you must try and do it as a little exercise. So, now, what I have done is here is how the test requirement for CACC will look like.

(Refer Slide Time: 22:04)

CACC for p

TR for CACC for p for each clause to be major clause:

	a	b	c	d
p_a :	T	t	f	t
	F	t	f	t
p_b :	f	T	t	t
	f	F	t	t
p_c :	f	t	T	t
	f	t	F	t
p_d :	t	t	t	T
	t	t	t	F

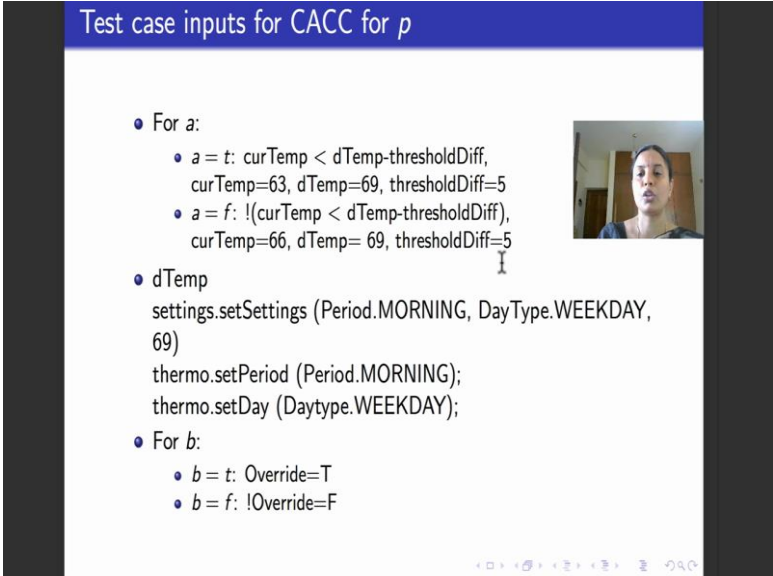
Rows 3 and 5 are duplicates, six tests are needed to achieve CACC TR for p .

So, how do you read this? There are 4 clauses a , b , c and d and this p_a is when a is the major clause, p_b is when b is the major clause, p_c is when c is the major clause, p_d is when d is the major clause. When a is the major clause this capital T and capital f when the column corresponding to a indicate that a is made true once, a is made false once. The rest of the values for b , c and d indicated by small t and small f for true and false tell you the values that the clauses b , c and d take for a to determine p . Similarly for p_b when b determines p , b is made true once false once. The other clauses a , c and d which are minor clauses when b is the major clause take values such that predicate becomes true once and false once. Similarly for c , when c is the major clause the true false values

corresponding to c are written in capital letters the rest of the clauses take values such that c determines p and finally for d , right.

So, now if this is how I write and fill up the test cases. Now I examine the table once again to see how many repetitions are there in the table. So, if you see row number three false true, a is false, b is true c is true, d is true is the same as row number 5, a is false here, b is true, c is true d is true. Please do not worry about capital T true being capital T here in for b and for capital T here for c . It just still means true. So, these are basically duplicates. So, I do not have to repeat. So, if we ignore the duplicates I basically need 6 test cases to be able to satisfy correlated active clause coverage criteria for p . What are the 6 test cases? The rows that do not come as duplicates, these, all other rows. So, what I do? I take one row at a time and like I did for predicate coverage I give values to all the variables that makes each clause true or false in turn right. So, for a to become true, here is the set of values.

(Refer Slide Time: 24:16)



The slide is titled "Test case inputs for CACC for p ". It contains a list of test case inputs for variables a , $dTemp$, and b . A small video inset of a person is visible on the right side of the slide.


- For a :
 - $a = t$: $curTemp < dTemp - thresholdDiff$,
 $curTemp=63$, $dTemp=69$, $thresholdDiff=5$
 - $a = f$: $!(curTemp < dTemp - thresholdDiff)$,
 $curTemp=66$, $dTemp=69$, $thresholdDiff=5$
- $dTemp$
`settings.setSettings (Period.MORNING, DayType.WEEKDAY, 69)`
`thermo.setPeriod (Period.MORNING);`
`thermo.setDay (Daytype.WEEKDAY);`
- For b :
 - $b = t$: $Override=T$
 - $b = f$: $!Override=F$

So, a is this for a to become true current temperature could be 63, desired temperature could be 69, threshold difference could be 5. For a to become false, current temperature could be 66, desired temperature could be 69 and threshold difference will be 5. In this case current temperature will not be less than desired temperature minus threshold difference. And $dTemp$ we saw, you have to pass that value by setting values for morn for a period and for dtype. b is very simple because it is just is Boolean variable override.

(Refer Slide Time: 24:53)

Test case inputs for CACC for p , contd.

- For c :
 - $c = t$: $\text{curTemp} < \text{overTemp} - \text{thresholdDiff}$,
 $\text{curTemp}=63$, $\text{dTemp}=72$, $\text{thresholdDiff}=5$
 - $c = f$: $\neg(\text{curTemp} < \text{overTemp} - \text{thresholdDiff})$,
 $\text{curTemp}=66$, $\text{dTemp}=67$, $\text{thresholdDiff}=5$
- For d :
 - $d = t$: $\text{timeSinceLastRun} > \text{minLag}$
 $\text{timeSinceLastRun}=12$, $\text{minLag}=10$
 - $d = f$: $\neg(\text{timeSinceLastRun} > \text{minLag})$
 $\text{timeSinceLastRun}=8$, $\text{minLag}=10$




Similarly, for c , c is this clause I give values for current temperature desired temperature and threshold difference to make c true once false once. Similarly for d . So, once I have this I go back to this table. For every row in the table a true, b true, c false, d true. here is the test case.

(Refer Slide Time: 25:12)

Test cases for CACC, #1

$\text{dTemp} = 69$ (period = MORNING, daytype = WEEKDAY)

- 1. T t f t
`thermo.setCurrentTemp (63);`
`thermo.setThresholdDiff (5);`
`thermo.setOverride (true);`
`thermo.setOverTemp (67); // c is false`
`thermo.setMinLag (10);`
`thermo.setTimeSinceLastRun (12);`
- 2. F t f t
`thermo.setCurrentTemp (66); // a is false`
`thermo.setThresholdDiff (5);`
`thermo.setOverride (true);`
`thermo.setOverTemp (67); // c is false`
`thermo.setMinLag (10);`
`thermo.setTimeSinceLastRun (12);`




Read this as a true, b true, c false, d true in order: a, b, c, d. Here is the test case, this is the complete test case for the first row in this test requirement for CACC.

(Refer Slide Time: 25:30)

Test cases for CACC, #1

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

- 3. f T t t
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72); // to make c true
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
- 4. F f T t thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (false); // b is false
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);




Similarly, I give 6 set of test cases right, this is a second one, this is the third one, this is the fourth one where a is false, b is false, c is true, d is true.

(Refer Slide Time: 25:38)

Test cases for CACC, #1

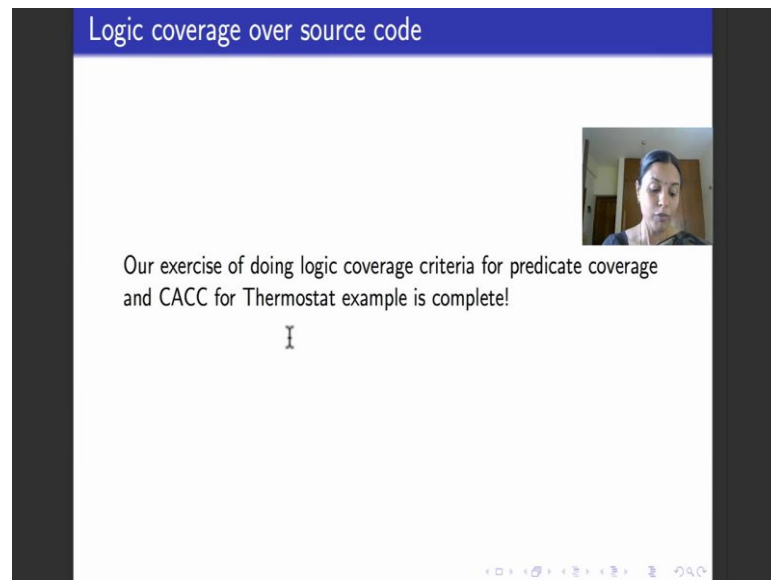
dTemp = 69 (period = MORNING, daytype = WEEKDAY)

- 5. t t t t T
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
- 6. t t t F
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (8); // d is false



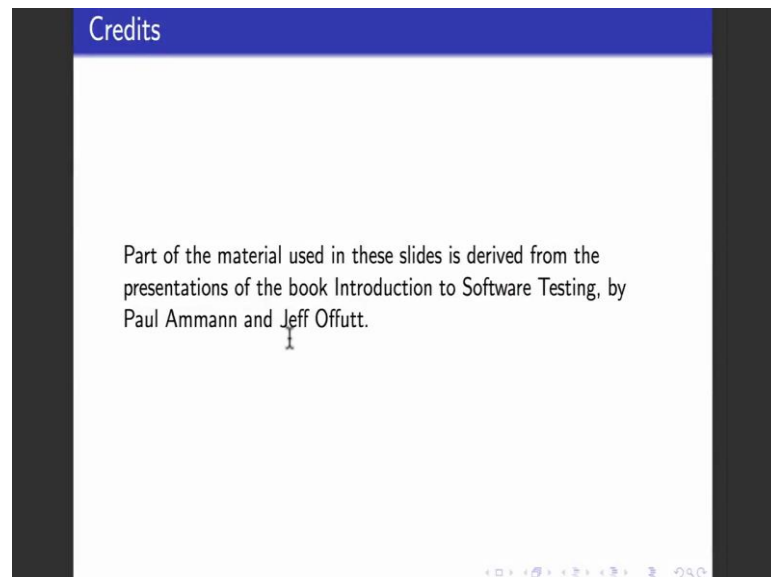
Similarly, this is the fifth one. All 4 are true, this is the sixth one, a, b and c are true and d is false. So, these are the final test case values that will help us to test correlated active clause coverage criteria for the predicate p in lines 28 and 30. So, similarly you could do clause coverage for that predicate, generalized active coverage criteria for the predicate, RACC for the predicate, all other conditions that you want.

(Refer Slide Time: 26:06)



Hopefully this exercise of this example would have helped you to understand how to apply logic coverage criteria for source code and how to write test cases end to end. What we will do in the next lecture is I will take you through another examples slightly longer and more complicated than this.

(Refer Slide Time: 26:32)



So, that you understand reachability and controllability well and tell you how to apply logical coverage criteria for that example.

Thank you.