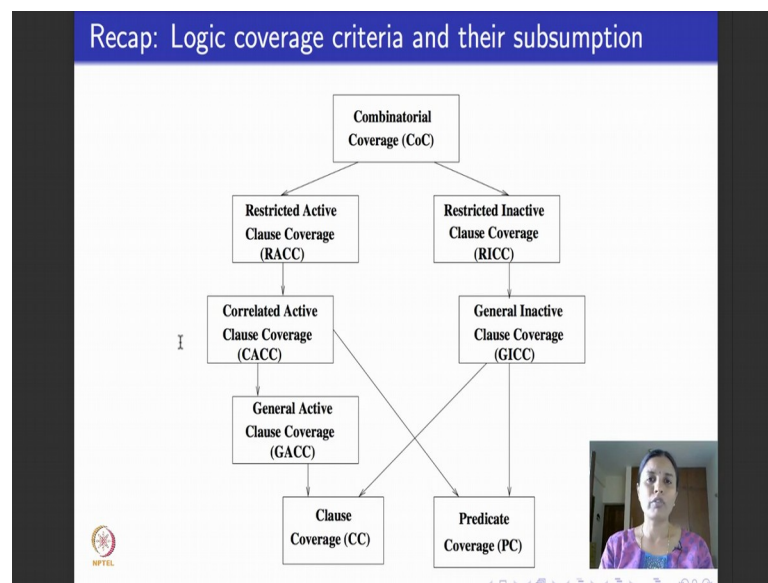


Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 24
Logic Coverage Criteria: Making clauses determine predicate

Hello again. We are in the last lecture of week 5, we are in the middle of doing logic coverage criteria based testing. I defined what the various logic coverage criteria are and showed you the subsumption relations. Today what we are going to see is suppose you have to take a real program and you use this logic coverage criteria to actually correctly write your test requirements and subsequently define test values; how would you go about doing it?

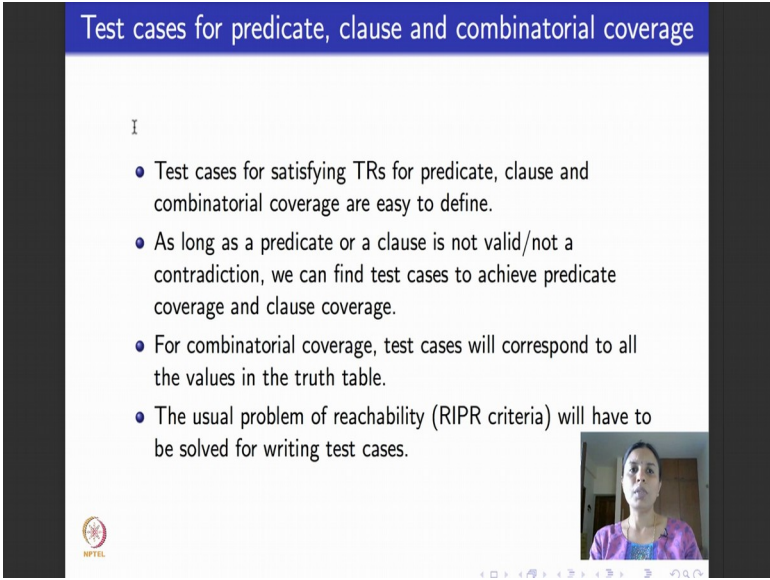
(Refer Slide Time: 00:41)



So, just to recap, what we saw till now, here is all the logic coverage criteria that we saw. We began with predicate and clause coverage, the most elementary form of logic coverage. Predicate coverage says test for the predicate to be true once and false once, clause coverage says test each clause to be true and false once. Then we saw all combinations coverage, right on top in terms of subsumption. It says test for the entire truth table of all the clauses in the predicate. We say that is too expensive because it reads to an exponential number of test cases. So, what can we do in between?

The in-between coverage criteria are broadly classified as active coverage criteria on the left hand side, 3 kinds and inactive coverage criteria on the right hand side, 2 kinds. Both these cases, the premise first choose one clause at a time in the predicate, call it the major clause. In active coverage criteria, major clause determines the predicate, the minor clauses take values as the major clause determines the predicate. In inactive coverage criteria the minor clauses take values such that the major clause does not determine the predicate. So, this is a summary slide that contains all the logic coverage criteria that we saw and how they are all related to each other, what subsumes what and what does not subsume what?

(Refer Slide Time: 02:06)



Test cases for predicate, clause and combinatorial coverage

I

- Test cases for satisfying TRs for predicate, clause and combinatorial coverage are easy to define.
- As long as a predicate or a clause is not valid/not a contradiction, we can find test cases to achieve predicate coverage and clause coverage.
- For combinatorial coverage, test cases will correspond to all the values in the truth table.
- The usual problem of reachability (RIPR criteria) will have to be solved for writing test cases.

NPTEL

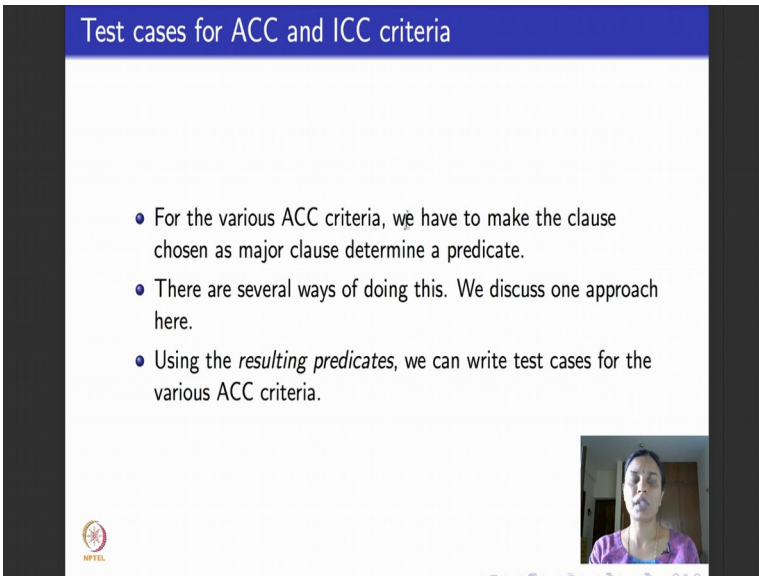
A small video inset in the bottom right corner shows a woman with dark hair, wearing a purple patterned top, speaking.

The focus of today's lecture would be to design test cases. So, we take one at a time and then see how to design test cases. We saw during the logic coverage criteria lecture that we did as a third lecture of week 5, these, how to design test cases for clause and predicate coverage and for all combinations coverage. They are reasonably easy because as long as a predicate is not always true, which means a predicate is not valid or as long as the predicate is not always false, which means the predicate is never valid which means it is a contradiction. So, it is always possible to get test cases that will do coverage criteria for both predicate coverage and clause coverage. You have to identify a set of values that will make the predicate or the clause true once and another set of values that will make the predicate or the clause false once.

For all combinations coverage it is again easy. Write out the entire truth table and then write test cases for every row in the truth table that will make each of the clauses true and false in turn. The only problem that we might have to solve is suppose there is a predicate that comes deep down in a program as a part of an if statement which is way down in a program in the sense that it comes, let us say, after a few hundred lines in the program. It so happens that the variables that you encounter in the if statement, none of those variables deal with input values directly. So, I should be able to give inputs to the program first of all to reach that if statement and exercise my desired coverage criteria for the predicate in that if statement.

So, you remember right in the first week we saw this thing of observability and controllability and then I told you about RIPR criteria. So, we will have to do reachability and then we will have to do infection to be able to achieve our coverage requirement and in turn propagation. So, what we will do in the next week, the first module that we will see we look at logic coverage of source code. Take a piece of program look at all the predicates in the program and see how to apply the coverage criteria that we learnt. At that point we will deal with this last item here of reachability or RIPR criteria in detail which basically means that if a predicate in a program deals with internal variables that are not inputs, how to give input values that will make program reach and achieve the desired coverage criteria on that predicate.

(Refer Slide Time: 04:50)



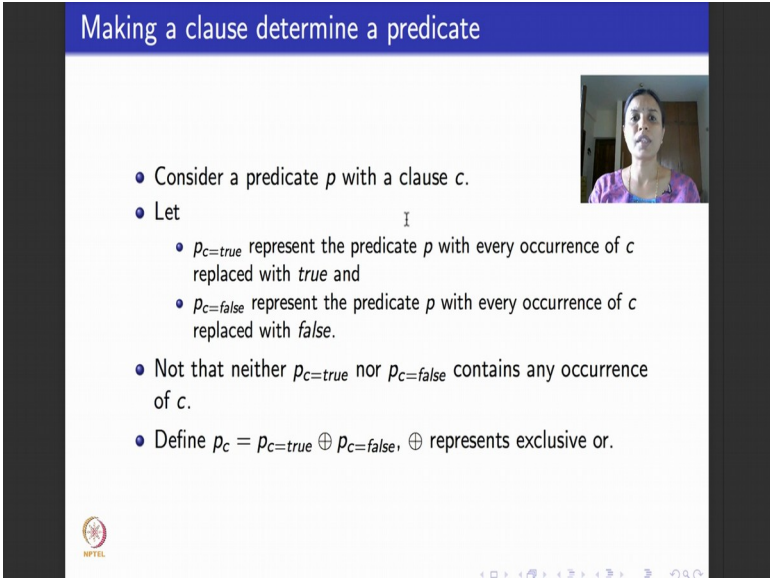
Test cases for ACC and ICC criteria

- For the various ACC criteria, we have to make the clause chosen as major clause determine a predicate.
- There are several ways of doing this. We discuss one approach here.
- Using the *resulting predicates*, we can write test cases for the various ACC criteria.

NPTEL

So, now what about designing test cases for the rest of these coverage criteria? For the 3 active clause coverage criteria and the 2 inactive clause coverage criteria. The first thing to do and remember is that for active clause coverage criteria. We chose each clause will take turn as a major clause and the minor clauses take values such as the major clause determines the predicate. Now, what we will do is how to make a major clause that is chosen in any point in time determine the predicate. Now major clause determining a predicate means the predicate should take true or false values as and when the major clause takes true or false values. So, we will see how to do that by using examples and then based on that how to choose TRs and test cases that satisfy the TRs.

(Refer Slide Time: 05:30)



Making a clause determine a predicate

- Consider a predicate p with a clause c .
- Let
 - $p_{c=true}$ represent the predicate p with every occurrence of c replaced with *true* and
 - $p_{c=false}$ represent the predicate p with every occurrence of c replaced with *false*.
- Not that neither $p_{c=true}$ nor $p_{c=false}$ contains any occurrence of c .
- Define $p_c = p_{c=true} \oplus p_{c=false}$, \oplus represents exclusive or.

NPTEL

So, what I will do is I will walk you through several examples that tell you how to make a major clause determine a predicate. In all these examples we do a generic definition based approach of deciding when a major clause determines a predicate. So, here is what we do to make a major clause determine a predicate. So, you consider predicate p and let us say there is a clause c in the predicate p . I want c to be the major clause which means to be able to do active clause coverage criteria, I want c to be able to determine p and when I do inactive clause coverage criteria, I want c to be able to not determine p . For the purposes of this lecture, I will focus on active clause coverage criteria and tell you how to make a particular chosen clause which is the major clause determine a predicate p . So, I fix a predicate p and I pick up a clause c in that predicate, call it the major clause. The idea is when will c determine p ? For small predicates it will be easy.

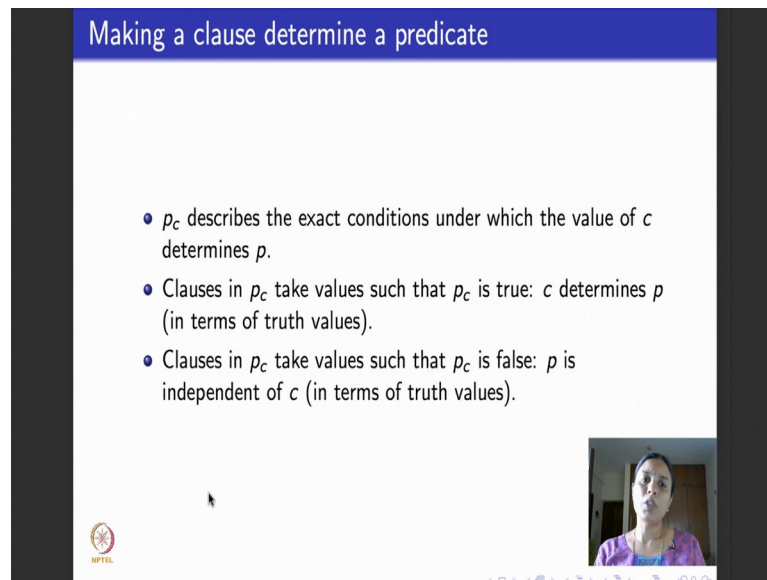
But for large predicates it may not be easy to know when exactly c will determine p . So, we use a default definition based approach which is like a set of steps that you need to do to make c determine p . So, here are the steps. So, what we will do first is we will take the predicate p , take every occurrence of c in p . By the way without loss of generality, I will assume that in the all predicates that we see particular clause occurs exactly once. If it occurs more than once the second occurrence is usually redundant and we remove the second occurrence. So, what I do is I take the predicate p , consider the place where the clause c occurs in the predicate p and replace c with true. So, call that resulting formula or resulting predicate by this notation, p subscript c is equal to true.

Now, I do the second thing that I do is I take the predicate p , consider the clause c replace c with value false wherever c occurs and simplify the logical formula corresponding to the predicate call that p c false. So, is it clear what P_c true and P_c false are? P_c true means take the predicate p , consider an occurrence of c in p replace that occurrence with true and simplify the formula. P_c false says take the predicate p , consider the occurrence of c in the predicate p , replace c with false.

So, once I replace c with true or false to get these 2 predicates P_c true and P_c false, please remember that p has no more occurrences of c . c is completely eliminated from the predicate p . Now I define the following formula P_c . I take this for atomic, this formula P_c true and XOR it with the formula P_c false. Read this notation that looks like a lens of a this thing as XOR. You remember what XOR as a logical operator is? XOR says a XOR b says that it will be true exactly when one of a or b in is true right. So, or says it will be true when one of them is true, it could be the case that both of them will also be true to make an or true. XOR is exclusively or exclusive or means a particular a XOR with b is true if and only if one of a or b is true.

So, I take p , fix a clause c , my major clause, take copy of p replace c with true simplify the predicate keep it on one side that is P_c true. Then I take the same predicate p replace the clause c with false, simplify it, keep it on one side, that is P_c false. Now I XOR these 2 formulas call it P_c .

(Refer Slide Time: 09:37)



Making a clause determine a predicate

- p_c describes the exact conditions under which the value of c determines p .
- Clauses in p_c take values such that p_c is true: c determines p (in terms of truth values).
- Clauses in p_c take values such that p_c is false: p is independent of c (in terms of truth values).

NPTEL

What do we do with P_c ? It turns out that P_c describes the exact conditions under which the value of c determines p . Why is that so? That is so because if you see the clauses that occur in P_c first of all c is not there they take values such that when p is P_c is true c will determine p in terms of its truth values and the clauses in P_c take values such that when P_c is false p .

So, the truth or the falsity is independent of truth or falsity of c . Why does this happen? Instead of doing a proof, I will show you several examples of how to take a predicate and a clause how to determine P_c and understand why P_c will represent the conditions under which the clause c determines P .



(Refer Slide Time: 10:26)

Making a clause determine a predicate: Examples

- Consider $p = a \vee b$.
- Then,

$$\begin{aligned} p_a &= p_{a=true} \oplus p_{a=false} & (1) \\ &= (true \vee b) \oplus (false \vee b) & (2) \\ &= true \oplus b & (3) \\ &= \neg b & (4) \end{aligned}$$

- For major clause a to determine p , the only minor clause b must be false.
- Symmetrically, $p_b = \neg a$.



So, here is the first formula. So, the predicate that I want to illustrate by using an example is this predicate. I have taken the predicate p which is $a \vee b$. How many clauses are there in the p , in the predicate p ? There are 2 clauses: one clause is a one clause is b . Now I want to be able to compute p_a which means I fix a to be my major clause and I want a to determine p . So, as per my formula, what should I do? I take this p and first replace every occurrence of a in the predicate p by true and then I replace every occurrence of a in the predicate p by false and XOR them. That is what I have written in the first line here. This is as per our definition. p_a is the same as p with a replaced with true, XOR-ed with p with a replaced with false right. What is p with a replaced with true? p in this example is $a \vee b$. So, you take this predicate $a \vee b$, instead of a write true, that is what I have written in the second line here.

So, this is $a \vee b$ which is the predicate p with a replaced with true and p with a replaced false is false or b with a replaced with false and then my goal is to XOR them. Now what is the semantics of $true \vee b$? $True \vee b$ should basically be the same as true right because true is always true. So, b does not really influence. So, $true \vee b$ simplifies to be true logically and $false \vee b$ simplifies to be b logically.

So, you can look up the basic rules of inference of proposition and logic and these would be some elementary rules. So, this is what is called the generalization, this is what is

called absorption. So, $\text{true} \vee b$ simplifies to true, $\text{false} \vee b$ simplifies to b . Now again one more rule. I have to XOR true with b which is the same as $\neg b$ right because it says what either exclusively true will be true $\vee b$ will be true right. So, it should be the case that not b is true or in other words b should be false.

So, what I have done here? I have taken the predicate p , I want a to be the major clause in p and I want to compute $p \wedge a$. $p \wedge a$ says, $p \wedge a$ is true whenever a determines p . So, I did use the formula that we had and simply substituted simplified, and got $p \wedge a$ to be $\neg b$ right. Now if we see how do I read this, as I read this as follows: for the major clause a to determine the predicate p the only minor clause b ; b is the only other clause here must be false.

Why is that so? Even without working this out if you independently see this formula here I want a to determine p which means what I want p to be true or false exactly when a is true or false right. Suppose b was true, then, the whole predicate will become true independent of the value of a . So, b has to be false for a to influence when p becomes true and when p becomes false. That is exactly the conclusion that we have derived by using this formula and substituting. We have derived that $p \wedge a$ is the same as $\neg b$ or, in other words, for major clause a to determine the predicate p , $\neg b$ should be true or b should be false. Symmetrically suppose b was a major clause, for b to determine p if I replace the same derivation, in the same derivation a with b , I will get $\neg a$, which means what for b to determine p , a should be false.

(Refer Slide Time: 14:15)



Making a clause determine a predicate: Examples

- Consider $p = a \wedge b$.
- Then,

$$\begin{aligned} p_a &= p_{a=true} \oplus p_{a=false} & (5) \\ &= (true \wedge b) \oplus (false \wedge b) & (6) \\ &= b \oplus false & (7) \\ &= b & (8) \end{aligned}$$

I

- For major clause a to determine p , the only minor clause b must be true.
- Symmetrically, $p_b = a$.



So, we look at a couple of more examples. The second example that we consider will be the predicate p which is now $a \wedge b$ instead of $a \vee b$ which we saw in the last slide. I repeat the exercise, now I want a as my major clause and I want to know when a will determine p . So, I use the formula that we learnt, I say p_a is the same as p with a substituted with true XOR-ed with p with a substituted with false. So, I take p which is $a \wedge b$ here I substitute a to be true. So, I get $true \wedge b$, XOR it with a substituted with false which gives me $false \wedge b$. $true \wedge b$, if I use the rules of logic, maps to b and $false \wedge b$, if I again use the rules of logic, maps to false. Why is this so, because $true \wedge b$ is more general and is a restrictive operator.

So, it will be true exactly when b is true and false and b will be false all the time because it is anded with false right. So, b XORed with false as per rules of propositional logic is equivalent to b . So, what have we concluded? If a is a major clause, for a to determine p , sorry a to determine the predicate p , p_a is the same as b which means for major clause a to determine p the minor clause b must be true. So, if you put it back and intuitively look at it, suppose b was true. Suppose b was true then a will completely determine p no because if a is true, p will be true and a is false, p will be false. That is exactly what we have concluded here and symmetrically if I consider b to be the major clause and repeat this exercise for b I will get $p_b = a$ which is the conditions under which b determines p to be equivalent to a .

(Refer Slide Time: 16:06)


Making a clause determine a predicate: Examples

- Consider $p = a \equiv b$.
- Then,

$$p_a = p_{a=true} \oplus p_{a=false} \quad (9)$$
$$= (true \equiv b) \oplus (false \equiv b) \quad (10)$$
$$= b \oplus \neg b \quad (11)$$
$$= true \quad (12)$$

- For any value of b , a determines p without regard to the value for b .

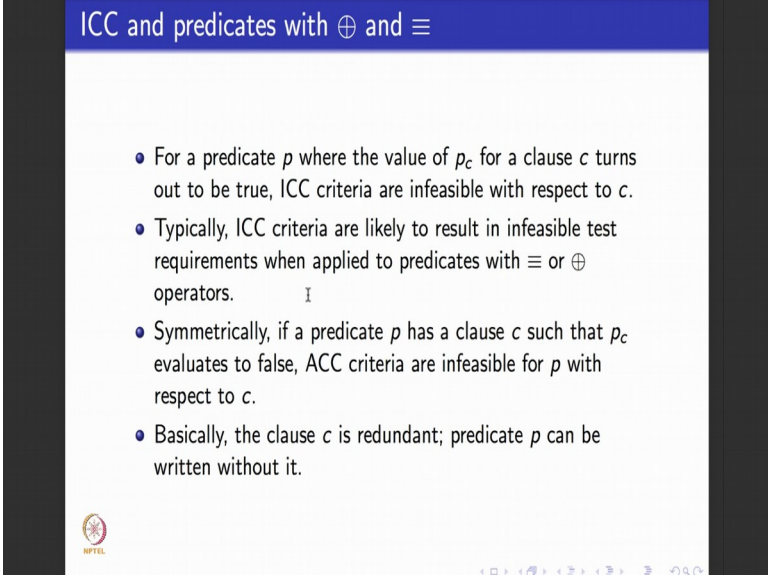
I



So, one more example before we move on. Here I am considering the predicate p to be a is equivalent to b which means $a \equiv b$. I again want a to be the major clause and compute p_a which says when a determines p . So, I use the same formula, take the predicate p , replace the occurrence of a with true XOR it with replace the occurrence of a with false. True equivalent to b turns out to be b , false equivalent to b turns out to be $\neg b$. b XOR with $\neg b$ means what always $b \vee \neg b$ is true which is like a valid formula or a tautology. So, this simplifies to be true.

So, here what has happened? When we try to make a the major clause and make a determine the predicate p , the answer that we get is true. How do you read an answer like true? You read it as no matter what happens, for any value of b if I choose a to be the major clause a can never determine p . So you can come to several conclusions as we saw in these examples. For a predicate like this, I want a to determine p then b must be false. For a predicate which looks like p is equal to a and b if I want a to determine p , then b must be true. For a predicate which looks like a is equivalent to b , I try to make a determine p , but I end up concluding that a can never determine p .

(Refer Slide Time: 17:37)



ICC and predicates with \oplus and \equiv

- For a predicate p where the value of p_c for a clause c turns out to be true, ICC criteria are infeasible with respect to c .
- Typically, ICC criteria are likely to result in infeasible test requirements when applied to predicates with \equiv or \oplus operators.
- Symmetrically, if a predicate p has a clause c such that p_c evaluates to false, ACC criteria are infeasible for p with respect to c .
- Basically, the clause c is redundant; predicate p can be written without it.

NPTEL

So, what, how do I understand this thing of never being able to determine the predicate in terms of testing? In terms of testing, for a predicate p where the value of p_c for a clause turns out to be true, that is what happened here right in this example for the predicate p is equal to a equivalent to b , the value of p_a turns out to be true. In that case it means that a clause cannot determine a predicate right, cannot determine a predicate because the minor clauses cannot take any other values.

So, which means what? Inactive clause coverage criteria is typically infeasible because inactive clause coverage criteria if you remember from the previous lecture, what does it say? It says that minor clauses take values such that major clause does not determine p . But in this case when I end up computing p_a for a to be a major clause and I get the answer true, I conclude that no matter what happens the clause that I have chosen as the major clause cannot determine.

So, I cannot basically do inactive clause coverage criteria. Typically it is known that when I have predicates with XOR or equivalence, ICC criteria turn out to be infeasible because I will end up with these funny results like true and all that. Symmetrically if you think of it a little bit, if I try to compute similar thing p_a and then suppose I end up with false, then I can conclude that active clause coverage criteria will not be feasible. That is what I have written here in the third point. I say if a predicate p has a clause c such that p_c evaluates to false then active coverage criteria will be infeasible for p with respect to c .

It basically means the clause is redundant you might as well remove it from the predicate p.


(Refer Slide Time: 19:32)

Redundant clause in a predicate: Example

- Consider the predicate $p = a \wedge b \vee a \wedge \neg b$. This is just $p = a$.
- Computing p_b , we get

$$\begin{aligned}
 p_b &= p_{b=true} \oplus p_{b=false} & (13) \\
 &= (a \wedge true \vee a \wedge \neg true) \oplus (a \wedge false \vee a \wedge \neg false) & (14) \\
 &= (a \vee false) \oplus (false \vee a) & (15) \\
 &= a \oplus a & (16) \\
 &= false & (17)
 \end{aligned}$$

- It is impossible for b to determine p .



So, I will show an example here. When do I say the clause is redundant? Purposefully in this example I have considered a predicate that looks like this. So, what is the predicate read as? It reads as a and b or with a and not b right. So, now, what is this basically? If you use the rules of logic, propositional logic a little bit and try to simplify it b or not b are neutral, they neutralize each other they will result to be true and a you just have 2 copies of a which are redundant copies of a. So, p the predicate p is basically just a, but for some reason it occurs in complicated form like this. So, if the predicate p is just a, but b is occurring is a sort of a useless form here, I want to be able to conclude that b cannot determine p.

So, how do I do that? I compute p_b and I have to end up with false. That is what this derivation illustrates. What does this derivation illustrate? It says p_b is you do the same formula, replace b with true XOR it with b replaced with false. So, I take the predicate p which is this I replace every occurrence of b with true which is what I have done here and here I replace every occurrence of b with false which is what I have done here and simplify this formula. So, if you simplify this formula $a \wedge true \vee a \wedge \neg true$, you will get a or false and here if you simplify this formula, you will get false or a a or false is the same as a false or a is the same as a a XOR-ed with a is basically false. So, it is

impossible for b to determine p. So, it is as good as saying this b here which occurs in the predicate p is fairly useless, is redundant you might as well remove it.

(Refer Slide Time: 21:26)

Clauses determining a predicate: More examples

- Consider $p = a \wedge (b \vee c)$.
- Then,

$$p_a = p_{a=true} \oplus p_{a=false} \quad (18)$$

$$= (true \wedge (b \vee c)) \oplus (false \wedge (b \vee c)) \quad (19)$$

$$= (b \vee c) \oplus false \quad (20)$$

$$= b \vee c \quad (21)$$

- a determines p when $b \vee c$ is true.
- Three choices make $b \vee c$ true:
 $(b = c = true), (b = true, c = false), (b = false, c = true)$.
- For CACC: Pick one pair when a is true and another when a is false.
- For RACC: Choose the same pair for both values of a.

So, we will go back and do one more example to understand how a clause determines a predicate because it will be useful for you. I will also give you in the assignment for this week, I will give you a set of exercises where you should work out for at least one more formula, how a particular clause determines a predicate.

So, here is another example. The predicate that I have taken this time is p which reads as $a \wedge b \vee c$. Now I want to determine p a which is basically I have chosen a to be the major clause and I want to determine conditions under which a determines p. So, I use the formula p a is p a true XORed with p a false. So, I replace a with true in this predicate p, I get this. I replace a with false in the predicate p, I get this part. I now use the rules of logic to simplify it. True ANDed with $b \vee c$ is the same as $b \vee c$, false ANDed with $b \vee c$ is the same as false, $b \vee c$ XORed with false is the same as $b \vee c$.

So, now what it says is that for a to determine p, $b \vee c$ must be true. That is not too surprising right because if you see p is a ANDed with $b \vee c$ right. So, unless this whole thing is true a cannot influence p. So, $b \vee c$ must be true. $b \vee c$ must be true, you apply the condition for $b \vee c$ as another formula and find out when it will be true. You use the semantics for or. If you use the semantics for \vee , $b \vee c$ will be true in 3 possible cases:

both b and c are true, which is listed here, b is true c is false or b is false c is true. So, there are 3 choices to make $b \vee c$ true. Now suppose I have to do correlated active clause coverage for a then what can I do ? I will pick one pair when a is true and another pair when a is false. So, I can pick a to be true and let us say I pick this, b true, c true. In which case the predicate p will evaluate to true and now a is false I pick some other pair let us say I pick the pair b false, c true.

So, in which case what will happen? This whole thing will be true and a is false. So, p will turn out to be false. So, a will correctly determine p. Now for restricted active clause coverage criteria what do I have to do ? I take value of a to be true and false again, a is the major clause and I choose the same pair because that is what RACC says right. The minor clauses all take the same value.

So, I could take for example, a to be true and b and c to be true, in which case the whole predicate, if I substitute it back will evaluate it to true. Or I could take a to be false and b and c to be true in which case the whole predicate when substituted back will evaluate to false. So, is it clear? Please, how to do p a, p b and p c. So, to be able to be completely satisfied GACC, RACC, CACC and other criteria for this predicate what I have worked out on this slide is only for a. So, you have to repeat a similar exercise for b being the major clause, similar exercise for c being the major clause and then write all conditions for GACC, CACC and RACC. .


(Refer Slide Time: 25:00)


Clauses determining a predicate: More examples


- Consider $p = a \wedge (b \vee c)$.
- Then,

$$\begin{aligned}
 p_b &= p_{b=true} \oplus p_{b=false} & (22) \\
 &= (a \wedge (true \vee c)) \oplus (a \wedge (false \vee c)) & (23) \\
 &= (a \wedge true) \oplus (a \wedge c) & (24) \\
 &= a \oplus (a \wedge c) & (25) \\
 &= a \vee \neg c & (26)
 \end{aligned}$$

- Symmetrically, p_c is $a \vee \neg b$.





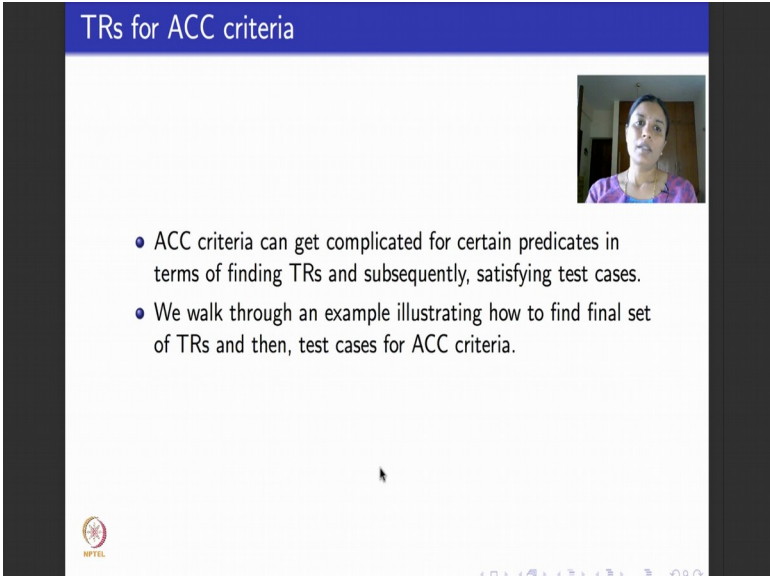


So here for the same predicate, in this slide, I have worked out what happens when b is the major clause. When b is the major clause I am interested in computing $p \wedge b$. So, I take $p \wedge b$ true XOR with $p \wedge b$ false take this predicate p replace b with true I get this part take this same predicate replace b with false I get the part here. Simplify these logical formulae. You might have to remember few rules of propositional logic to be able to do this simplification. So, $a \wedge \text{true}$ is a ; $a \wedge \text{false}$ is same as false, $\text{true} \vee c$ is same as true, $\text{false} \vee c$ is same as c .

So, $a \wedge \text{true}$ is a , $a \wedge \text{false}$ is false, I cannot simplify it further, I keep it like this. Formula $a \wedge b$ XOR with $a \wedge c$ simplified to $a \wedge (b \oplus c)$, the last 2 parts from this line 25 equation 25 to equation 26 may not be very obvious. One simple way to convince yourself that 25 and 26 are the same would be to write the truth table for 25 and write the truth table for 26 and check that the truth tables are the same. That is the easy way to understand why this simplifies to this. Even otherwise if you do not know this simplification it is to leave it like this. $a \wedge (b \oplus c)$ and say this is the condition under which b determines p right because that also be equally handled in terms of writing test cases.

So, similarly because b and c come with this disjunction inside the bracket I can do a similar exercise for P_c and it will be symmetric and P_c will turn out to be $a \vee b$ OR-ed with not b .

(Refer Slide Time: 26:42)



The slide is titled "TRs for ACC criteria" in a blue header. It features a video feed of a woman in the top right corner. The main content consists of two bullet points:

- ACC criteria can get complicated for certain predicates in terms of finding TRs and subsequently, satisfying test cases.
- We walk through an example illustrating how to find final set of TRs and then, test cases for ACC criteria.

At the bottom left, there is a small NPTEL logo. At the bottom right, there are navigation icons for a presentation slide.

Now, how to write test requirements for active clause coverage criteria? So, sometimes active clause coverage criteria can get little complicated for certain predicates in terms of finding TRs and subsequently finding satisfying test cases. So, I will walk you through one final example and see how to actually do the end to end TR generation and test case writing for active clause coverage.

So, this time I have taken similar predicate, but slightly different. If you go back, the predicate we had here was $a \wedge b \vee c$.

(Refer Slide Time: 27:20)

TRs for ACC criteria: Example

- Consider the predicate $p = (a \vee b) \wedge c$.
- We first give the truth table for p .

	a	b	c	$(a \vee b) \wedge c$
1	T	T	T	T
2	T	T	F	F
3	T	F	T	T
4	T	F	F	F
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Clauses for p_a , p_b and p_c are given below.

p_a	$\neg b \wedge c$
p_b	$\neg a \wedge c$
p_c	$a \vee b$

Now, what I have taken is $a \vee b \wedge c$. You could do a similar exercise for that predicate also it does not matter, but just to illustrate I have taken another predicate. What was my goal now? My goal is to be able to, for each of these clauses, 3 clauses taking turns to be the major clause to be able to find test requirements for GACC, CACC and RACC. So, there are 3 clauses in this predicate a , b and c . So, what I do is I first write out the truth table of the predicate. It helps to write out the truth table because you can understand it little easily.

So, I have written out the truth table here. Truth table has eight rows, you can work it out for yourself as a small exercise and I have done using this kind of working out that we did for this formula I have calculated p_a , p_b and p_c . I have not shown you the working please try out as a small exercise, but if you use the formula p_a is p_a replaced with true

XOR-ed with a replaced with false and then use the rules of logic to simplify it you should get something like this.



(Refer Slide Time: 28:33)

TRs for GACC criteria: Example

- TR for GACC: Each major clause be true and false, minor clauses be such that major clause determines the predicate.
- TR for GACC: $\{(a = \text{true} \wedge p_a, a = \text{false} \wedge p_a), (b = \text{true} \wedge p_b, b = \text{false} \wedge p_b), (c = \text{true} \wedge p_c, c = \text{false} \wedge p_c)\}$.
- The following table gives true/false values for TR for GACC:

	a	b	c	p
$a = \text{true} \wedge p_a$	T	F	T	T
$a = \text{false} \wedge p_a$	F	F	T	F
$b = \text{true} \wedge p_b$	F	T	T	T
$a = \text{false} \wedge p_b$	F	F	T	F
$c = \text{true} \wedge p_c$	T	F	T	T
$c = \text{false} \wedge p_c$	F	T	F	F

- First and fifth rows are identical, second and fourth rows are identical. Only four tests are needed to satisfy GACC.

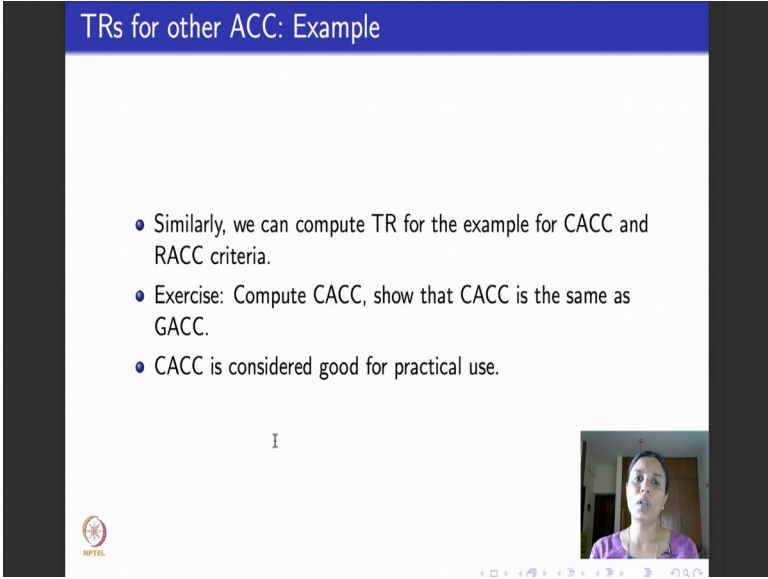
Similarly, for p b, similarly for p c. Now what I am going to do I want to be able to do first do test requirements let us say for generalized active clause coverage criteria. So, let us recollect the definition of GACC. GACC definition says each major clause will be true and false and minor clauses will be such that the major clause determines the predicate. So, now, what will be the TR for GACC ? I am writing what I wrote in English here in terms of true false values. So, the first pair here is when a is the major clause, a is true once AND-ed with a determining p which is p a; a is false AND-ed with a determining p which is again p a. Now the second pair here is b's turn to be the major clause b is the major clause.

So, b is made true once AND-ed with p b which says when b determines p b is made false once, again AND-ed with p b. The third pair here is c's turn to be the major clause: c is true once, false once. For c to determine p, I specify it using p c. So, here is a table that gives you the true false values for the test requirements for GACC. So, a is the major clause first 2 rows I am sorry there is a typo here it should be b is the major clause for the third and fourth row c is the major clause for the fifth and sixth row right. So, a is true once if you see concentrate on the ones in bold in the table a is true once false, once b is true once false once, c is true once false once. The rest of the values take val rest of the

clauses, b and c in this case and a and c in this case and a and b in this case take values such that the predicate becomes true once false once, true once false once, true once false once. So, this is how I do GACC for the predicate p.

Now, if you look at this table you will see that there are repetitions like for example, the first row is the same as the fifth row; a is true, b is false, c is true, a is true, b is false, c is true because the assignments are the same, the predicate evaluates to be true. Similarly in the table, the second and the fourth rows are the same. So, I do not have to repeat them. So, I keep only one copy for each of these. So, how many tests totally are needed? Four test cases are needed one for row 1, one for row 2, one for row 3 and one for row 6 which is not a repeat of any of the earlier rows. So, totally my test requirement for GACC for this example predicate p for each clause a, b and c taking turns to be the major clause can be achieved by the set of four TRs which are basically these rows in the table.

(Refer Slide Time: 31:32)



TRs for other ACC: Example

- Similarly, we can compute TR for the example for CACC and RACC criteria.
- Exercise: Compute CACC, show that CACC is the same as GACC.
- CACC is considered good for practical use.

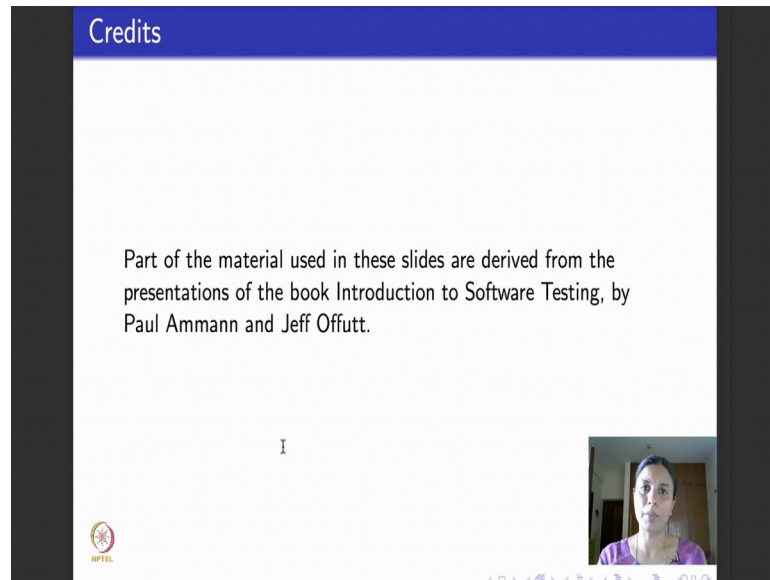
I

NPTEL

Similarly, you can work out TRs for CACC and for RACC just as a small exercise because it may not be very obvious to do. I urge you all to please work out the test requirements for CACC and for RACC and feel free to get back in touch with me in the forum if you have any doubts about working out. You will realize after working out this particular example correlated active clause coverage criteria is the same as general active clause coverage criteria. Typically for several predicates CACC is considered to be very

good to use it is practically useful and quite useful thing when compared to other predicates.

(Refer Slide Time: 32:15)



So, this will be the last lecture of week 5. What I will do next week? The first 2 modules we will continue with logic coverage criteria, but we will see how it is practically useful. In the next lecture we will take a piece of program and see how to apply logical coverage criteria that we learnt to test that program. And in the lecture after that, we will take a specification as a finite state machine and see how to apply logical coverage criteria that we learnt to be able to test that specification, write test cases for that specification.

Thank you.