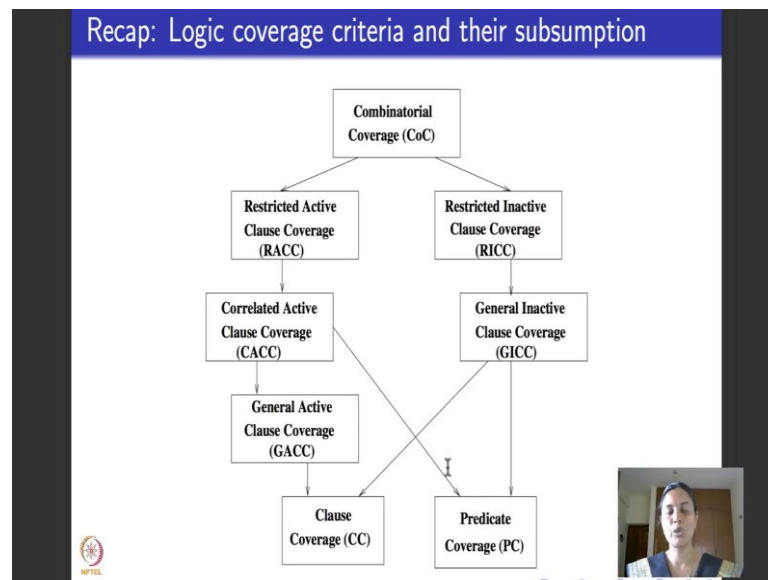


Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 29
Logic Coverage Criteria: Applied to finite state machines


Hello there. This is the last lecture of week 6. We will be done with logic coverage criteria with this lecture. Next week I will begin with the new test case design algorithm. What are we going to do today? We are going to look at how a logic coverage criterion applies to finite state machines.

(Refer Slide Time: 00:29)



So, this is same slide that I had shown you every lecture that we did in logic coverage criteria, meant to be a recap of the various coverage criteria that we saw and their subsumption. In today's example when we do finite state machines, we will see may two or three of these coverage criteria and see how to write test requirements in test cases for those, for specifications that come from finite state machines.

(Refer Slide Time: 00:55)



Finite state machines

- Finite state machines as graphs occur as specification models, design models, in UML state diagrams etc.
- Transitions in finite state machines have guards or triggers, which are basically logical expressions/predicates.
- We will see how to apply logic coverage criteria over such FSMs through an example.

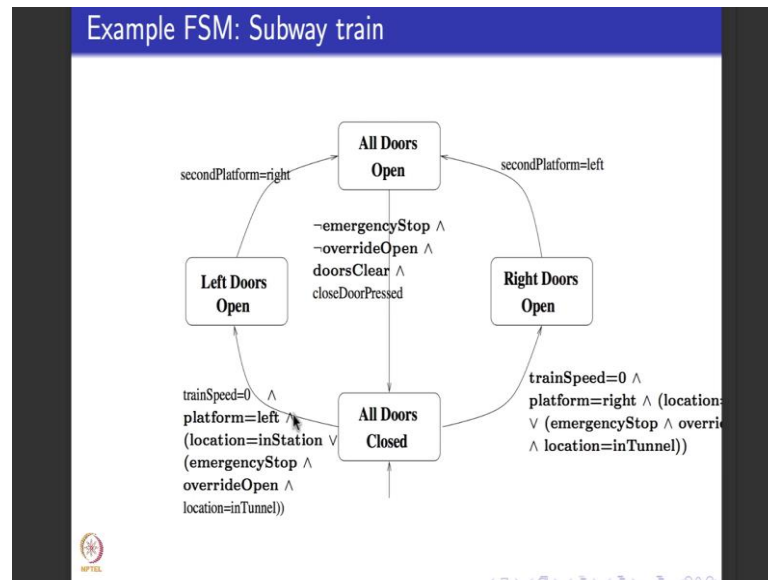
NPTEL

Navigation icons: back, forward, search, etc.

So, we introduce finite state machines when we did graph based testing. They have nodes or vertices which are states and the edges in finite state machines are called transitions. Edges are labelled with actions events and also guards, which are basically conditions that tell you that when an event can be happen, and when it happens the resulting action can happen.

Finite state machines have designated initial states. They may or may not have final states. They are very popular model for specification, for design, they occur as UML state diagrams. Especially in the embedded control software domain finite state machines are routinely used to specify control algorithms.

(Refer Slide Time: 01:39)



So, we will see how to apply logic coverage criteria for finite state machines using an example. So, here is an example of a very small finite state machine. In real life finite state machines that correspond to realistic models hardly are so small, but what people do was what is called modular design. They come up initially a very high level finite state machine that acts specification or is it design model. As and when they get clarity they keep refining or adding details of their finite state machines. So, even it is a small machine here that we are looking at in this example, it has enough details, enough predicates for us to be able to look at how logic coverage criteria applies to this.

So, this is a machine of, let us say, a model of how the doors of a subway train runs, subway train or a metro train. So, these trains keep moving from one station to the other, and the way they move, they have doors, the trains have doors on both sides. And typically for safety reasons, if the platform on a particular station comes on the left side then it is expected the only the doors on the left hand side open. The doors on the right hand side are closed. And if a platform, if particular station comes in such a way that there is platform accessible on both sides, then both sides doors open and when the train is moving, which means it is inside a subway tunnel, then you expect both doors to be closed.

So, here is a state machine of a subway train that has 4 states that talk about what are the status of all doors in the train. So, the initial state is the state which says all the doors of

the train are closed. From that state it could move to one of these two states. It could either move to a state where all the left doors are open or it could move to a state where all the right doors are open. And from either of these states, left doors open or all right doors open, it could move to a state which says all the doors are open. So, each of these moves or transitions are guarded by a whole set of conditions as you can see here.

For example, from the state all doors closed to the state left doors open, if I have to take that transition then all these conditions must be true. The final state machine has access to all these variables. Like in program you see the variables right in the beginning. Finite state machines they are not explicitly declared, but we assume that eventually this finite state machine design or specification is meant to be modeled, implemented using codes. So all these will surface as a real variables that will be in the code corresponding to the finite state machine.

So, there is a variable that talks about the speed of a train, called trains speed. It says that train speed should be 0, which means the train should not be moving. There is a variable which says where is the platform, is the platform on the left hand side or is the platform on the right hand side. So, in this case the guard to transition to left doors open, the platform should be on the left hand side and there is another variable which talks about the location of the station, location of the train. It is says the train is in station which means it is come inside the station, and it is either this or there is an emergency stop.

Somebody is pressed the emergency stop button and because this an emergency stop somebody is trying to open the door by overriding it, while the train is still in the tunnel. But the train is not moving in all the cases. So, there are so many variables and this large guard tells you when the transition from all doors close to left doors open happens. So, it is says train should not be moving, platform should be in the station and the train should, platform should be on the left side and train should be in station. Either this or there was an emergency stop, somebody is trying to override open the doors and the train is still in the tunnel.

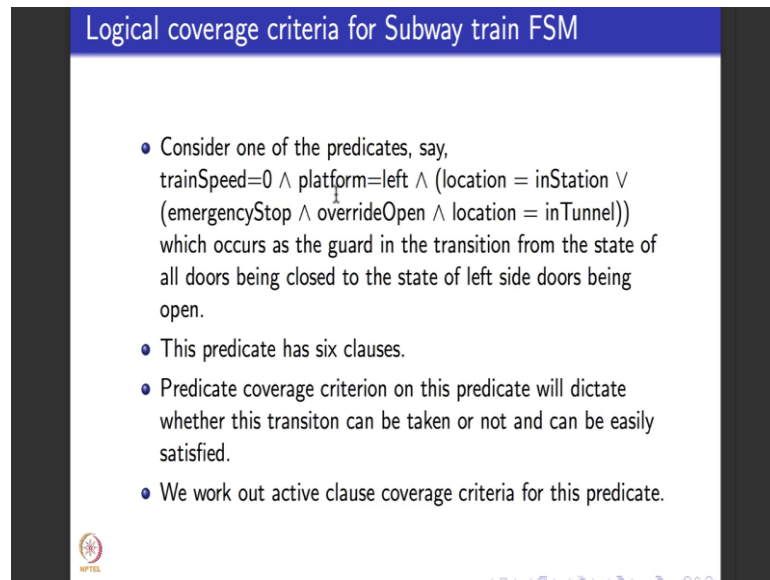
So now, similarly from the state all the doors closed to the state right doors open, when do I go? Again the train should not be in moving, platform should be on the right, location should be in station, I am sorry this is gone out of the slide in little bit emergency stop, override open, location in tunnel.

It is an exact replica of this. The only condition that is changed is the platform is come on the right hand side for this transition. Now it is so happens that the second platform is on the right hand side and there is a platform on the right hand side then apart from the left doors you can open the right doors implicitly, and you go to a state which says all door are open. And similarly if there is a second platform, when the right doors are open, if there is a second platform on the left hand side then you could open the left doors also and go to a state all doors open. And from the state all doors are open to the state all doors closed, how do we go? We go when the doors are clear and somebody presses the close door button and there is no emergency stop and nobody has pressed override open.

So, is it clear how the simple state machine looks like and why I have had drawn at this way? W have drawn at this way because the focus of this lecture is to be able to illustrate the use of logical predicates as they come in finite state machines. So, this state machine even at this level could have actions. We have not really specified all that because I want to be able to focus on logical predicate. So, if you see there are so many predicates that we can consider, five of them here. Just for illustrate purposes let me take this predicate, the one on the bottom left which goes, which lets you go from the state all doors closed to the state left doors open.

We will take this predicate and illustrate the use of logic coverage criteria that we learnt on testing this predicate, indirectly testing the influence of that predicate or guard on the finite state machine.

(Refer Slide Time: 07:39)



Logical coverage criteria for Subway train FSM

- Consider one of the predicates, say,
 $\text{trainSpeed}=0 \wedge \text{platform}=\text{left} \wedge (\text{location} = \text{inStation} \vee (\text{emergencyStop} \wedge \text{overrideOpen} \wedge \text{location} = \text{inTunnel}))$
which occurs as the guard in the transition from the state of all doors being closed to the state of left side doors being open.
- This predicate has six clauses.
- Predicate coverage criterion on this predicate will dictate whether this transition can be taken or not and can be easily satisfied.
- We work out active clause coverage criteria for this predicate.

NPTEL

So, I have just written the predicate that I want to consider. So, this is the predicate, it says train speed 0, platform is on the left, the train is in station or somebody is pressed the emergency stop, doing override open while the train is still in tunnel. This occurs as a guard in the transition from the state of all doors being closed to the state of left doors being open. How many clauses does this predicate have? Let us count, this is one: train speed equal to 0, platform equal to left is a second clause, location in station is the third clause, emergency stop, fourth one, override open fifth clause, location is equal to in tunnel, sixth clause.

Predicate coverage criteria for this predicates is very easy. It will tell you basically whether you can take this transition or not. You can take this transition if this predicate is true which means if the clauses that are connected by an AND are true otherwise you cannot take the transition. So, that is what predicate coverage criteria will say. Now we will work out active clause coverage criteria for this predicate.

(Refer Slide Time: 08:54)

Conditions under which each clause determines the predicate

- For clause $\text{trainSpeed}=0$: $\text{platform}=\text{left} \wedge (\text{location} = \text{inStation} \vee (\text{emergencyStop} \wedge \text{overrideOpen} \wedge \text{location} = \text{inTunnel}))$
- For clause overrideOpen : $\text{trainSpeed} = 0 \wedge \text{platform} = \text{left} \wedge (\neg \text{location} = \text{inStation} \wedge \text{emergencyStop} \wedge \text{location} = \text{inTunnel})$.
- We can similarly compute for all the other clauses.

Page 6 of 11

So, I take this predicate, it has 6 clauses. What is the first step to do to determine active clause coverage criteria? Per clause I have to say call it as a major clause and see when that clause determines the predicate. So, if it is one of the clauses in the predicate, I do what is called determining p subscript a . So, I have worked it out offline. I seriously urge you to take this predicate and work out for each of the 6 clauses when that clause will determine the predicate using the formula that says replace the clause with true in the predicate, replace the clause which is false in the predicate and XOR them. That is the formula if you remember.

So, I have used that formula I have worked it out and for the first clause in the predicate, which is train speed is equal to 0 after simplification, this is the p_a value, that is this is the p train speed is equal to 0 value. And similarly I have also worked out for the clause override open after doing the same formula, substituting override open with true one, substituting override open with false once, and XOR-ing them and simplifying the resulting formula, I will get this simplified predicate.



Similarly, for all the remaining four clauses we can compute if that clause is the major clause, the conditions under which that clause will determine the predicate. I have just done two of them here, the remaining four can be worked out. In fact, for these two also I have given you the final resulting formula, I have not given you the steps of working.

(Refer Slide Time: 10:23)

TR: CACC for the predicate

Major clause	Speed=0	platform =left	inStation	emergency Stop	override Open	in Tunnel
trainSpeed=0	T	t	t	t	t	t
trainSpeed!=0	F	t	t	t	t	t
overrideOpen	t	t	f	t	T	t
¬overrideOpen	t	t	f	t	F	t

We can compute CACC TR for other predicates in a similar fashion.



So, we take the first one which is train speed 0 and the other one that we worked out which is override open, and remember what are these? This is like a formula in conjunctive/ disjunctive normal form that we saw in the last lecture. So, it is very easy to write ACC test requirement for these formulas.

What you do? You make all the other clauses to take value true and the clause that is your major clause, make it true once make it false once. Similarly when our override open is a major clause you make it true once, you make it false once. The rest the clauses are all may true provided the diagonal, along the diagonal, it is so happens that override open is the fourth clause in this predicate. One, two, three, four, fifth clause, sorry it is the fifth clause in this predicate. So, the diagonal corresponding to that will be false the rest of the values will all be true.

So, this is how you write correlated active clause coverage criteria TR for that. I have just given you for the two clauses for which we worked out here and give when that clause determines the predicates. There are four more clauses, after you work them out you can fill up this table and see how it looks like. So this the TR for CACC for the predicate. So, to be able to give test cases for this TR, you have to make train speed value 0, if it ensure that the platform is said to left, in station as a Boolean variable should be made true, emergency stop should be made true, override open should be made true, and in tunnel should be made true. This how a test case look for that will look like.


(Refer Slide Time: 11:51)

One problematic predicate in the FSM

- While writing CACC TR for the predicate inStation, we encounter a unique problem with making the TR feasible.

Major clause	Speed=0	platform =left	inStation	emergency Stop	override Open	in Tunnel
inStation	t	t	T	f	f	f
¬inStation	t	t	F	f	f	f

- The FSM model has only two locations: inStation and inTunnel. They both are false in one TR (second row above), this is not possible.
- This could be an error in the model.



So, now you will realize there is a problem here. If you think a little bit about it, in this example itself we do have a bit of a problem. If you see for train speed 0 to be the major clause and for it to be influencing the predicate, look at the test case that. Each of these values should be made true. Now if you see is it actually feasible to make each of them true in the realistic scenario that the finite state machine would be implemented in, it will not. Why is that so, because if you see it says in station is set to true and in tunnel is also said to true in these both cases; which means that the train is in the station and in the tunnel at the same time. That is not practically possible right, it is either in the station or in the tunnel. So, there could be a possible error or you could interpret it as that the finite state machines that we saw at this level of abstraction, does not give enough details to be able to specify what we need.

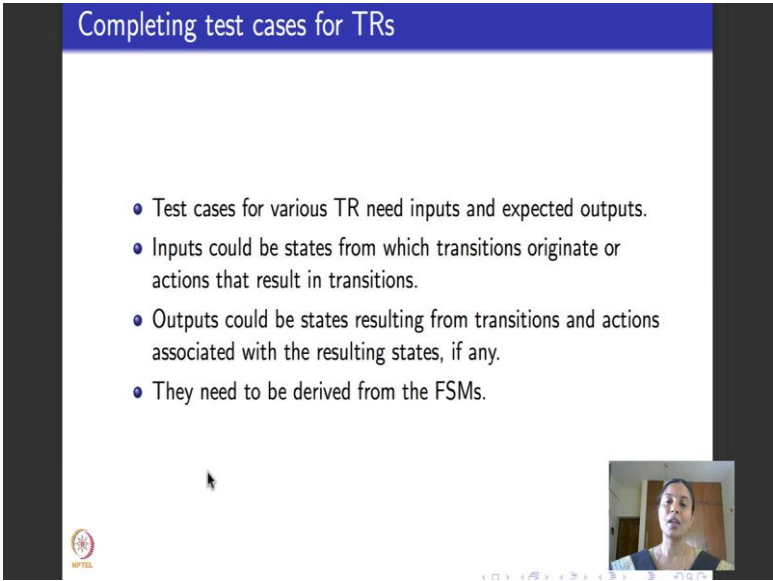
In fact, you could find several other errors. Here is one more. Suppose I try to write CACC TR test requirement for another clause, in station, which is here, which is one of the clauses that come in the predicate. So, here again you will realize that this is the major clause, I make it true once false once, the rest of it takes true false values. If you see what I have done here is it says in station is false here and in station is a false here which means it says the reverse of what the previous thing said. It is saying that the train neither in station nor in tunnel it is just disappeared. That also is wrong right there is something incomplete in the specification. This one said the train is both in the station and in the

tunnel, not clear, this says that the train is neither not in the station not and not in the tunnel. In the second row here if you see both are marked false that is also not possible.

So, it could be interpreted as two things, you could say that in this finite state machine model I have found two errors. There is inconsistency in specifying exactly where will the train be the predicates or guards do not really factor in the consideration that the train can be in either in the station or in tunnel. So, these guards are not clearly specified and that there is an error in the finite state machine.

Or another way to be treated it as is to say that there is incomplete, this is incomplete because the way it is specifies it does not give details about train being in exactly one of the location. So, more details have to be added or that the finite state machine has to be refined it to be able to complete that. Either way whatever it is we have found a possible bug in model and logic coverage criteria are very useful to be able to do this.

(Refer Slide Time: 14:47)



Completing test cases for TRs

- Test cases for various TR need inputs and expected outputs.
- Inputs could be states from which transitions originate or actions that result in transitions.
- Outputs could be states resulting from transitions and actions associated with the resulting states, if any.
- They need to be derived from the FSMs.

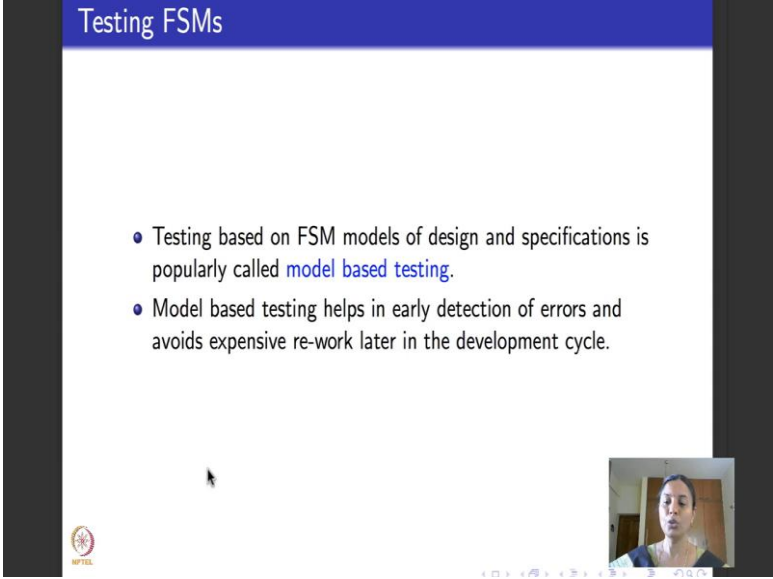
The slide is a presentation slide with a blue header. It contains a bulleted list of four points. In the bottom right corner, there is a small video inset showing a person speaking. At the bottom left, there is a small logo for NPTEL.

So, similarly if you work out for the other remaining clauses, you could find more inconsistencies and more errors. I just did two because in this particular case just these two were good enough to find inconsistencies or potential errors in the finite state machine as a specification. But there are, could be a couple of other issues also. Now when we are looking at an code, like when we saw logic coverage criteria as it applies to code, when we saw the two example the thermostat and the triangle type, we told you

that if a particular logic coverage criteria has internal variables, how to solve it for them how to ensure reachability.

But when we apply logic coverage criteria for finite state machines, as I told you when I introduced to you this example there is no explicit notion of inputs and outputs. So, what is a test case? Test case is directly given in terms of inputs and outputs that the finite state machine looks like, and output could also be given in terms of which is the resulting state. Sometimes they could be things wrong in the state that the Turing machine goes into even though state is not an explicit variable that occurs as a guard. So, it is up to the tester to infer all these information from informal specifications of finite state machines or semi formal specification of finite state machines, and complete the test case to identify a potential error or fault in the specification. They have to be explicitly derived, you cannot expect them to be present and readily available as they are with code.

(Refer Slide Time: 16:15)



Testing FSMs

- Testing based on FSM models of design and specifications is popularly called **model based testing**.
- Model based testing helps in early detection of errors and avoids expensive re-work later in the development cycle.

So, usually testing based on finite state machine is a very popular testing, there are huge number of papers that are in this area. Broadly called as model based testing, it helps in early detection of errors. Why it so, because finite state machines typically a models of specification. Specification and design are done before we write code. So, I have found the potential error writes there before I write code instead of finding it much later after I write code. It helps in early detection of errors if I detect it early rather than detecting it late, I save myself a lot of trouble with rework. So, it is saves lot of cost and lot of time

and you can check out the testing literature for a huge amount of material related to test case design and finite state machines, we have just covered logic base testing.

So, this brings us to an end of logic base testing. There will be an assignment this week that will deal with logic base testing for code, specifications and finite state machines. I will try and upload video next week that will let you solve tell you how to solve that assignment, but feel free to do it before you see that video.

Thank you.