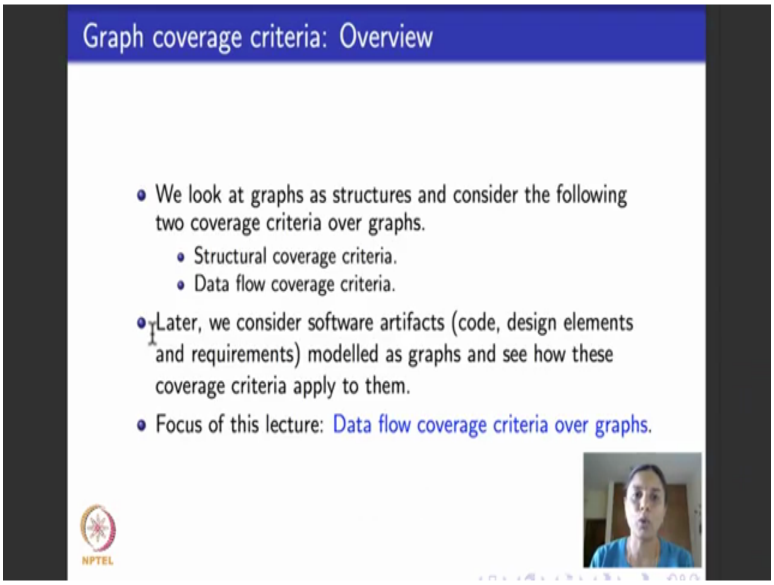
Software Testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture – 12
Algorithms: Data Flow Graph Coverage Criteria

Hello everyone. Welcome to the next module. The focus of this module is to continue with data flow, if you remember in the last video, we had defined what is data flow in a graph; what was definition of a data; what was use of data and how to track a data from it is definition to use? And we saw an example of how this is used.

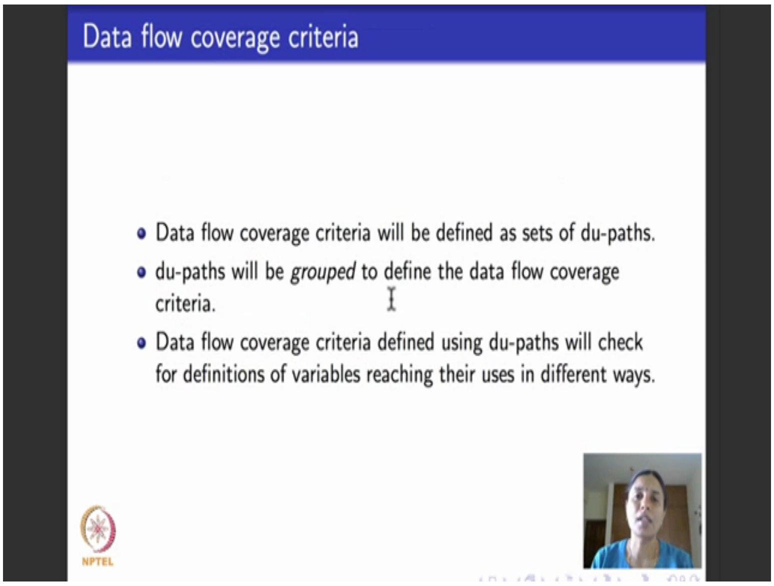
So, what we will see today is definitions of criteria that are based on data flow data defs and data uses. And then also tell you briefly about how to define a work on algorithms that will help us to achieve this data flow coverage criteria. Data flow coverage criteria algorithms is a very vast area early papers came out in the early 80s and this still active research going on. I am not be able to cover all the algorithms that deal with data flow coverage criteria what I am point to you at the end of this lecture would be some links to good reference material you may you could read out more to get to know about algorithms related to data flow.

(Refer Slide Time: 01:12)



So, just to recap what we did till now we are at the module where graphs are models that we use to model software artifacts, we saw structural coverage criteria over graphs then we looked at basic algorithms over graphs algorithms for defining test requirements and test paths for (Refer Time: 01:28) structural coverage criteria. Then I moved on to defining data flow and graphs.

(Refer Slide Time: 01:37)

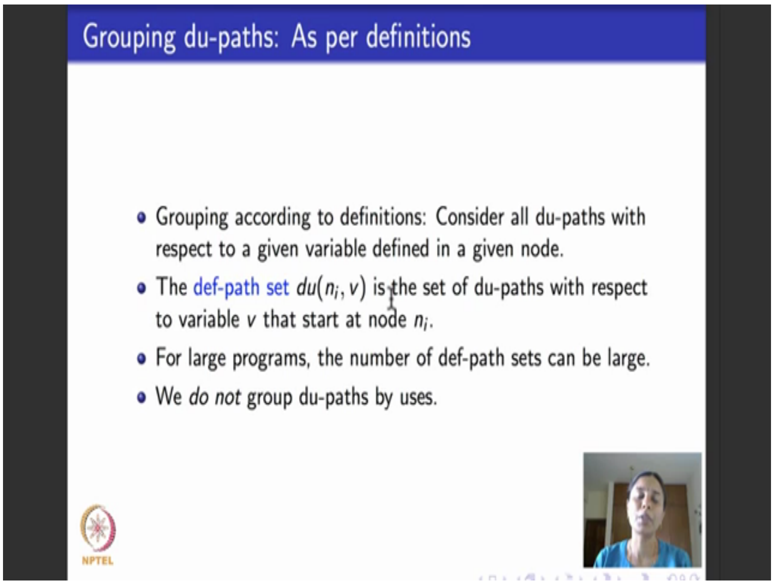


Today, what we will be seeing in the data flow coverage criteria. How is data flow coverage criteria defined? Data flow coverage criteria is basically defined as a set of d u paths d for definition u for use. What is the d u paths, if you remember from the last lecture d u path is a path corresponding to a variable that is given as a parameter to a path that begins at a definition of a variable and goes all the way till the use of the variable, intermediate in this path from it is definition to it is use.

We insist that the variable does not get defined once again. So, the path is definition clear for this variable right. So, what is d u path it is a path from a definition of a variable to the use of a variable. Such that a every intermediate node on the path there is no further definition of the variables.

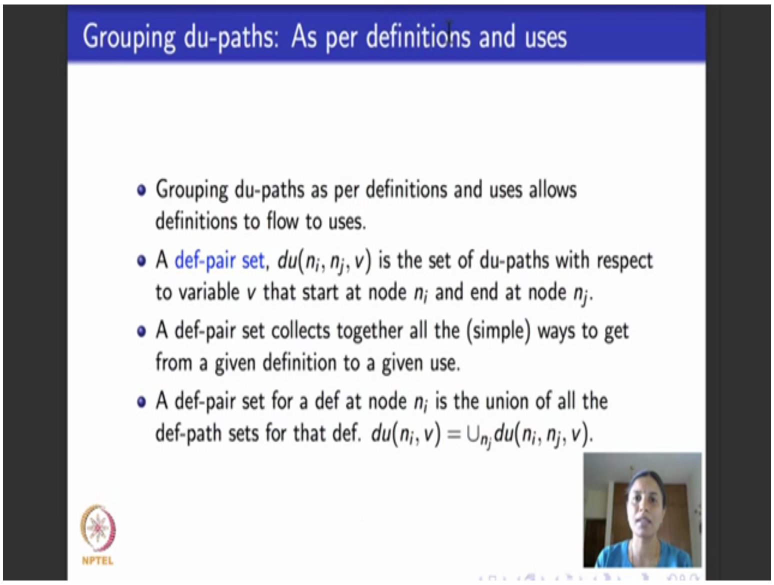
What we will do is we will group various kinds of d u paths to be able to define data flow criteria. What these criteria will basically check is they will basically check how a definition of a variable reaches it is use.

(Refer Slide Time: 02:38)



However, were going to group d u paths? We are going to group d u paths in 2 different ways the first grouping that we will discuss is as per their definitions, the next grouping that we will discuss is as per their definition and use.

(Refer Slide Time: 02:50)



So, how are we going to group d u paths as per definition? So, when we group d u path according to a definition we consider d u path with respect to a given variable defined in an given node, and we define it like this. So, v is the variable that whose definition and use we are tracking it is. So, happens that v u v is designed at this vertex ni in the graph.

So, what is a def path set called d u of the variable v at the node ni it is the set of all d u paths with respect to the variable v that start at the node n i.

What is the d u def path set a def path set at a node ni for a variable v is the set of all d u paths that begin at that node ni for that variable v, please note that the number of such def paths for a large program fairly reasonable size program can be very large, but we still have to find them and work on them to be able to define data flow coverage criteria? It is also work noting at the stage that while we group or def d u path with respect to a place where it begins to get defined, we do not really group it with reference to it is uses. So, turns out that in literature grouping it with reference to it is uses is not considered to be very essential and it is not useful for testing.

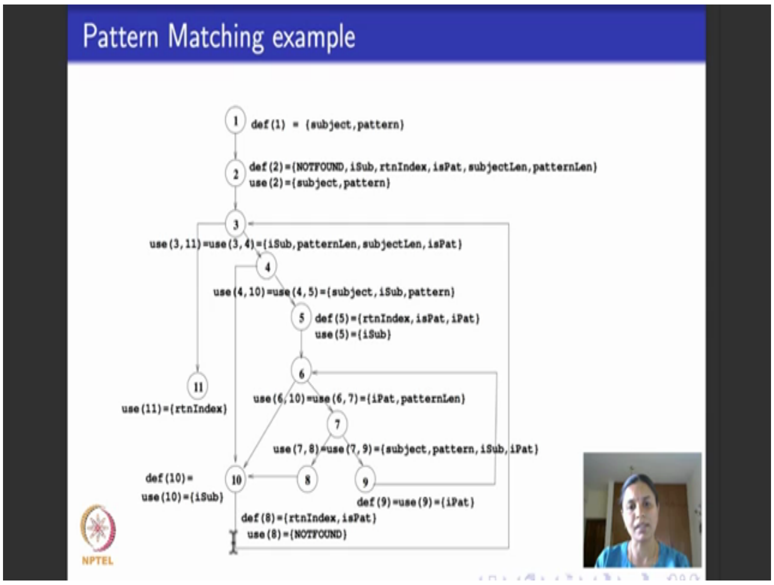
The other kind of grouping that we will do as per definition and use. So, such a grouping is what is called a def pair set. So, we have as usual fixed a variable v. The definition of that variables begins at ni and it is use happens with reference to this node n j right. So, what is a def pair set? Def pair set is a set to d u paths for a variable v that is fixed which begin at node ni and n at node n j right. So, a def pair set collects together always to get from a definition of a variable which is at node ni to it is use which is at node n j.

Please remember when we talk about d u paths we had interested in the last lecture that such path be simple. So, d u paths are always simple. So, def pair set is a collection of such simple paths. A def pair set could also be defined to begin at node ni, in which case it is the union of all the def path sets for that definition, that is if you say a def path set begins at node ni for a variable v. Then we consider all the different uses at which it can end and take the union and then we say that is a def pair set.

So, just to recap what are we going to do we are going to define data flow coverage criteria, by grouping together d u paths. We group together d u paths in 2 different ways the first grouping is as per the definition. We say or a variable v, we collect all d u paths that begin at node ni, that is the def from variable v is that node n i. So, that is called a def path set.

The next grouping is as per definition and use. So, for a big given fixed variable v, we consider all the beginning at a variable ni, and all the ending or used at node n j that is called a def pair set is a collection of pairs ni n j, such that v is defined that ni and used at n j.

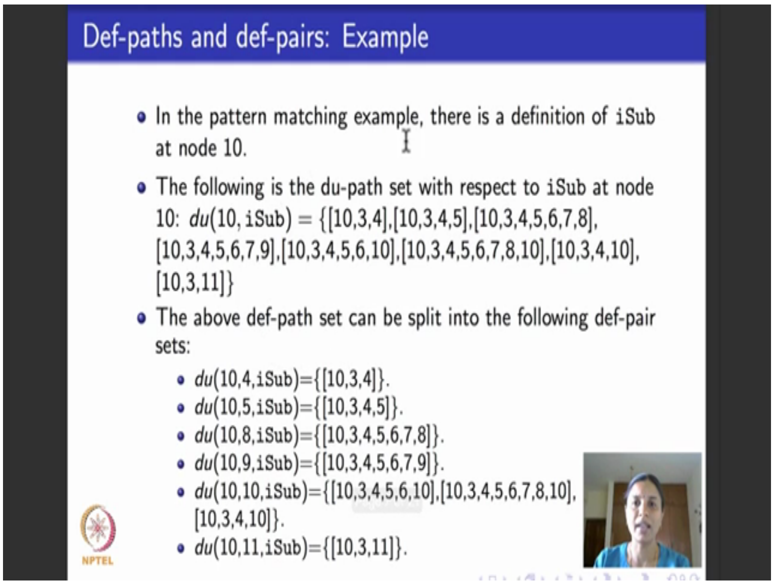
(Refer Slide Time: 06:15)



So, you remember last time we had looked at the small pattern matching example. It searches for a pattern occurrence in the subject actually I would like to mention here that there was a small error in the control flow graph that I had shown you last time. This edge from vertex 10 was drawn to meet at vertex 4 in the control flow graph that I had shown you last time. That is slightly different from the way it occurrence in the code. It should actually be like this this edge at vertex 10 should actually go and meet at vertex 3, this is where the while loop begins.

So, this is the corrected cfg please consider the cfg to understand the example. So, if you remember the grid first drawn the cfg, I had given you labels with all the statements and I taken the same cfg and annotated the graphs with definitions and uses. So, that is what this graph is I have put the same graph with this small correction done. So, what I will do is we will understand how the various d u paths look like.

(Refer Slide Time: 07:19)



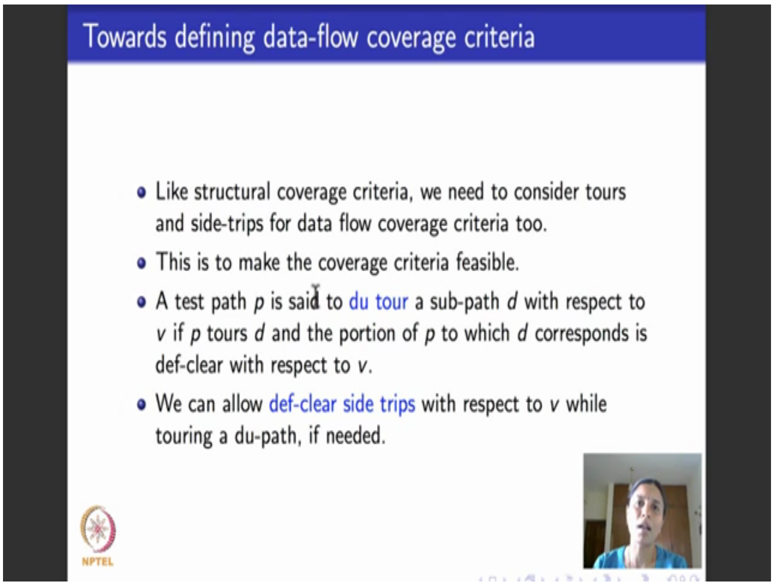
For simplicity I have taken this variable isub that is the variable that I have fixed there is a definition of this variable isub at node 10, if you see here isub definition at 10 variable isub is defined.

What is isub if you remember in the code it is an index that runs over the subject right and tries to pattern match every character in the subject with the corresponding character in the pattern right. So, here is a definition of isub. So, I can group this definition of isub in 2 different ways, at node 10. I can group it with reference to all it is uses at node 10 or I can group it with reference to it is definition at node 10, and use at each of the other nodes where it is used. If you see it is used in this particular node, it is used in this particular edge it is used in this particular node it is also used in this particular edge from 7, 8 to 7, 9. So, there are several places right

So, if I try to trace the def path sets and the d u path sets. So, here are the d u path sets for isub at node ten. So, from 10 I can go back through this edge to 3 and then to 4 right that is what this trace is here and then from 10 to 3 to 4 to 5. So, 10 to 3 to 4 to 5 and then I can do 4 to 5 5 to 6 7 to 8 or 7 to 9 both of them have isub in them that is what these 2 sets paths do 3 and 3, 4, 5, 6, 7, 8, 10, 3, 4, 5, 6, 7, 9 and so on.

So, basically what I am saying here is that you fix this variable isub. You begin it is definition at 10, and look at all the places where it is used it is used in this edge it is used in this edge it is used in this edge. So, trace path trace paths, in the graph that begin it is definition at 10 and end in one of it is uses take all their union that is the set. Alternatively, I can say I begin it is definition at 10 and consider it is use at 4, within which case I get only this path when I say I begin it is definition at 10 and consider it is use only at 5. I get this path beginning at 10 ending at 5. Similarly, from beginning at 10 ending at 8 is this path beginning at 10 ending at 9 is this path and so on right. So, if I take the union of all these things which begins at 10 and end at various vertices I get this set right. So, this is how I define def path set and def pair set for a variable at a particular node and in the control flow graph.

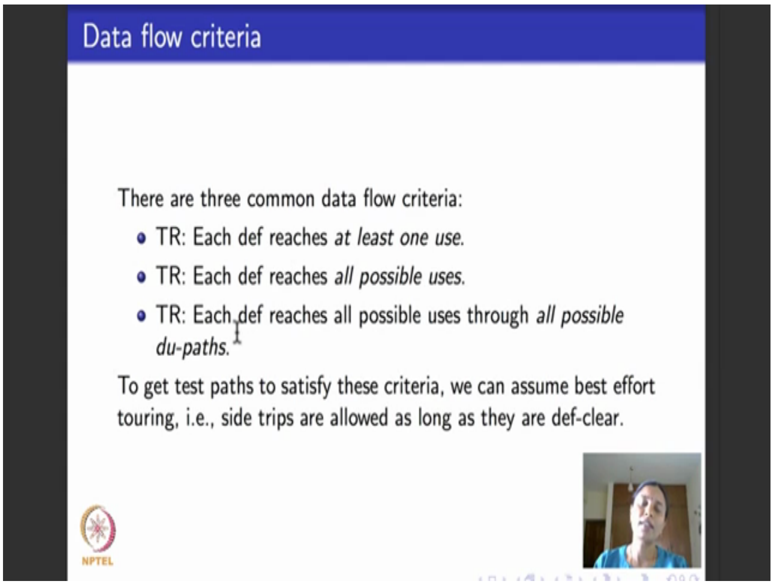
(Refer Slide Time: 10:04)



Now, we are ready to define data flow coverage criteria. So, like structural coverage criteria if you remember while you define structuring coverage criteria I had told you that sometimes structural coverage criteria can get to be infeasible because sometimes certain kinds of loops might insist that a path be traversed at least ones. So, we do test path to trace coverage criteria by taking side trips and detour. That is what we will do here we say a test path p is said to d u tour sub path d as long as it towards the sub path d, and this sub path that it towards is definition free for that variable.

Remember the only thing that we insist is the defines once it defined at a particular node, it reaches it is use and at an other node it could be through a direct test path or it could be through a test path with the side trip or a detour, irrespect to a whether I take the side trip or a detour I insist that that side trip or detour also be definition clip, that is what this says and it says you can freely use side trips and detours, whenever you want to get test paths to satisfy coverage criteria this is exactly like we did for structural coverage criteria.

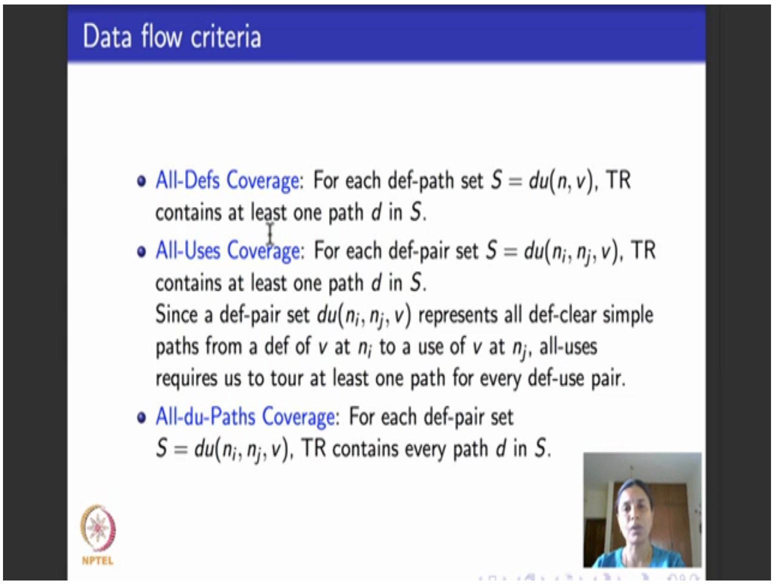
(Refer Slide Time: 11:17)



Now, what are the 3 kinds of data flow coverage criteria that we are going to look at. These are the 3 t rs were the 3 different data flow coverage criteria that we are going to look at, the first coverage criteria says that each definition should reach at least one use. The second coverage criteria says that each definition should reach all possible uses. The third coverage criteria says that each definition reaches all possible uses not only does it reach all possible uses; it reaches all possible uses using all possible different paths that you can trace out in the graph.

So, as I told you these suggest the test requirements. Whenever you need to get test paths to satisfy these test requirements, you can assume what is called best effort touring. Best effort touring basically says that feel free to allow side trips and detours if you want to make these test requirements feasible.

(Refer Slide Time: 12:15)



So, here are the definitions of the coverage criteria. What is all defs coverage say it says that for each definition of a variable v at a node n the test requirement contains at least one path that reaches a use right. All uses coverage says that for a variable v that is defined at a node ni, I reached one path which basically reaches all the uses for every pair of def at ni and use at n j.

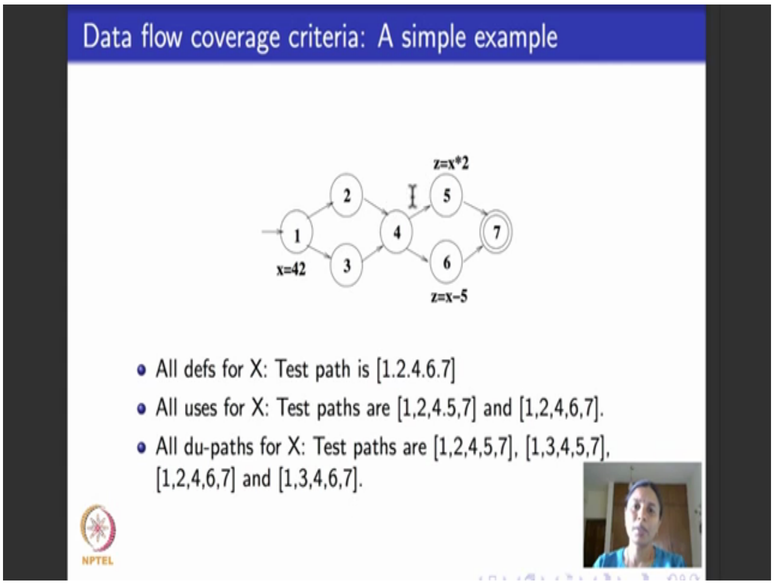
So, to repeat what is all uses coverage fix a variable v, variable v is defined at ni and used at n j. And this is every possible use of the variable v. So, n j where is over different uses of the variable v and (Refer Time: 13:03) test requirement says you cover every possible path, that we takes the variable v, v from it is definition at ni to it is use at n j.

The third ones is all d u path coverage it says for each def pair set ni n j v t r contains every path d in s. If you find these definitions cumbersome the easiest way to understand data flow criteria is to look at the previous slide. So, it says there is one criteria with says every def reaches at least one use that is all defs coverage. I do not worry about covering all the uses, but I want to cover every definition.

The second one all uses coverage says every definition reaches all possible uses. So, that is why it is called all uses coverage. The third definition all d u path coverage says therefore, every definition and every possible use of it take different paths every possible path that goes from a definition to use right is it clear. So, basically 3 different elementary data flow criteria cover every definition make sure it reaches at least one use. The seconds is cover every definition make sure it reaches all it is uses.

The thirds is cover every definition make sure it reaches all it is uses every possible different paths to get to the uses.

(Refer Slide Time: 14:31)



So, you remember this small example that we have seen the last lecture. There were only 2 variables in this small graph x and z, x will defined here and used a 5 and 6, z was defined at nodes 5 and 6. So, suppose I have to cover all defs criteria for x and the test path that I would take is this 1, 2, 4, I could either take 6, 7 or 5, 7. Basically what I want to say is that x is defined at node 1 take a test path that covers this definition and one use it could cover either this use at 5 or it could cover this use at 6. The example test path that we have given covers the definition at one and they use at 6.

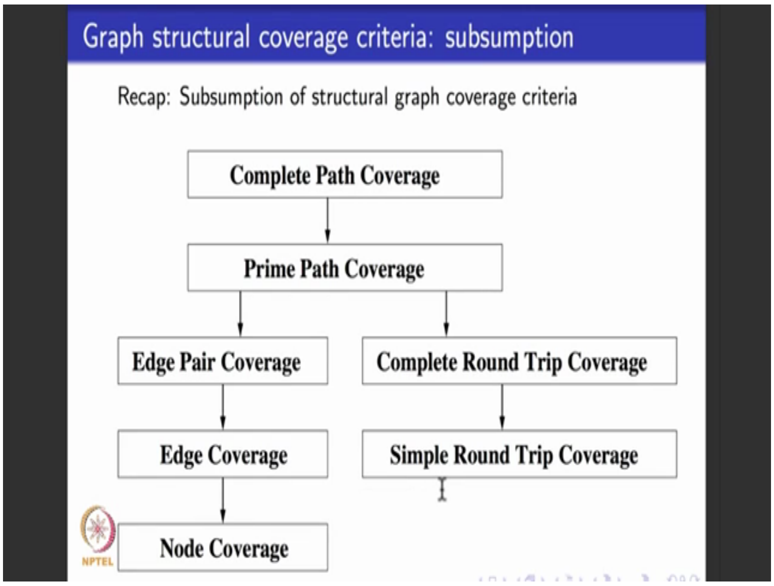
The second criteria all uses for x basically says x is defined here, and used in 2 different places in the graph at node 5 and at node 6. So, the 2 different test path that I should take are 1, 2, 4, 5, 7 which covers the use at 5 and 1, 2, 4, 6, 7 which covers the use at 6. Of course, I could take these 2 test paths by going through node 3 that is also the same that we equally well meet the test requirement. In this particular case I have just taken the 2 test path that go through node 2.

The third condition says from the definition of x at node 1 to which uses a node 5 and 6. You not only cover this definition to all it is uses you consider all the paths that take you from this definition to this use. So, if you see there are 4 different possible paths. I can do this 1, 2, 4, 5, 7 or I can do 1, 3, 4, 5, 7, that will cover the definition of x at node 1 2 is used at node 5 right 2 different ways.

Similarly, if I consider the definition of x at node 1, and this used as node 6 there are 2 different paths again. 1, 2, 4, 6, 7 and 1, 3, 4, 6, 7 that is what I have listed here right. Is it clear. So, just to repeat all def says go from every definition to at least one use, or uses says go from every definition to every use, all d u path says go from every definition to every use taking every possible different paths to do this. These are the 3 elementary graph coverage criteria that we will see.

Now, I want to spend some time trying to make you understand how these coverage criteria are related to each other.

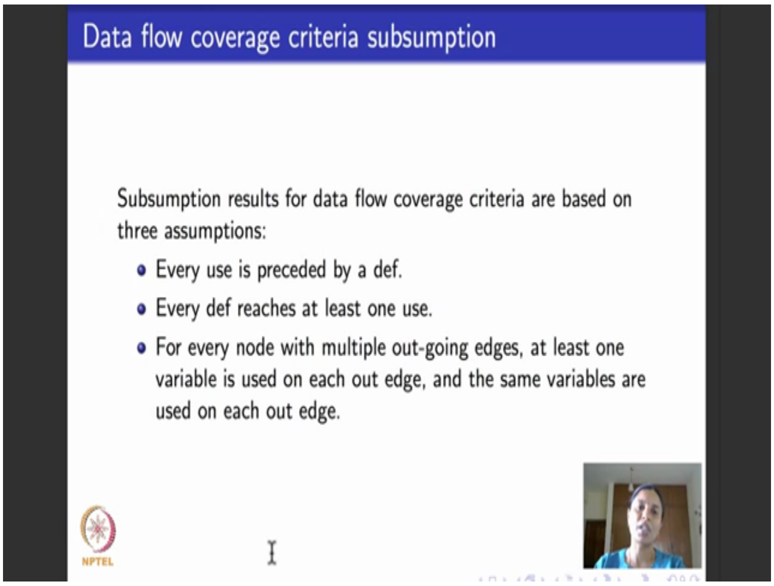
(Refer Slide Time: 17:08)



So, if you remember when we looked at structural coverage criteria we had seen all these coverage criteria and this was the picture that gave how each coverage criterion subsumes the other right. Node coverage edge coverage subsumes node coverage edge pair coverage subsumes edge coverage prime path coverage subsumes all 3 and so on.

Now, we want to be able to add data flow criteria to this picture and understand how the 3 data flow criteria that we define subsume each other and how are they related to the structural coverage criteria.

(Refer Slide Time: 17:43)

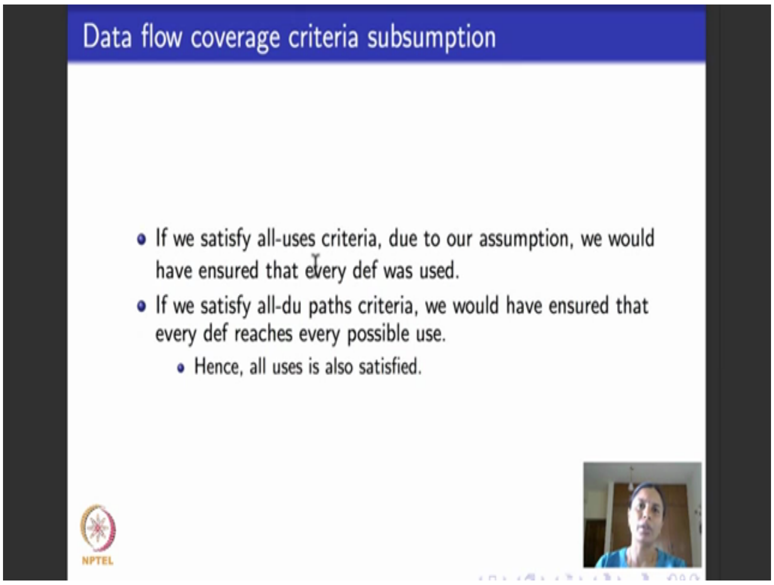


So, as far as data flow coverage criteria is concerned before we look at subsumption I would like to make some elementary assumptions. These are not strong assumptions they are basically true for every program that we expect to be compiled and working fine.

So, the assumptions that we are making are every use is preceded by a definition, otherwise you will you will understand right if there is a use that for a variable that is not defined then compiler will throw an error right (Refer Time: 18:09) the variable is not declared. So, it is not a big restricting assumption this is true of all programs that are syntactically sound and compliable. The second assumption that we make is that every definition reaches at least one use. This may not be found by a compiler, but elementary static program analysis tools will be able to find this it, basically says that there are no variables that I define and never used in the program. So, every variable that is defined used at some place in the program.

The third assumption says that when I have a node in the graph which has branches for multiple outgoing edges, at least one variable should be used on each of the out edge and we assume that the same variables are used on each out edge. What it says is that even if different variables are use to consider the set by considering the union of all the variables.

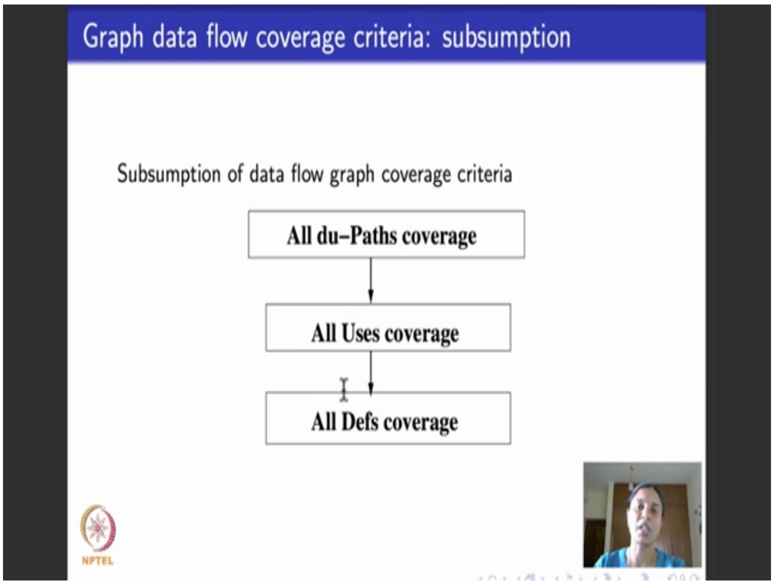
(Refer Slide Time: 18:59)



So, under these assumptions we are ready to look at data flow coverage criteria subsumption what are the subsumption thing. So, there are 3 criteria if you remember what are the 3 criteria all defs coverage all uses coverage all d u paths coverage. Because all uses says from every def you go to every possible use it is a reasonably straightforward to see that all uses coverage subsume or definitions coverage.

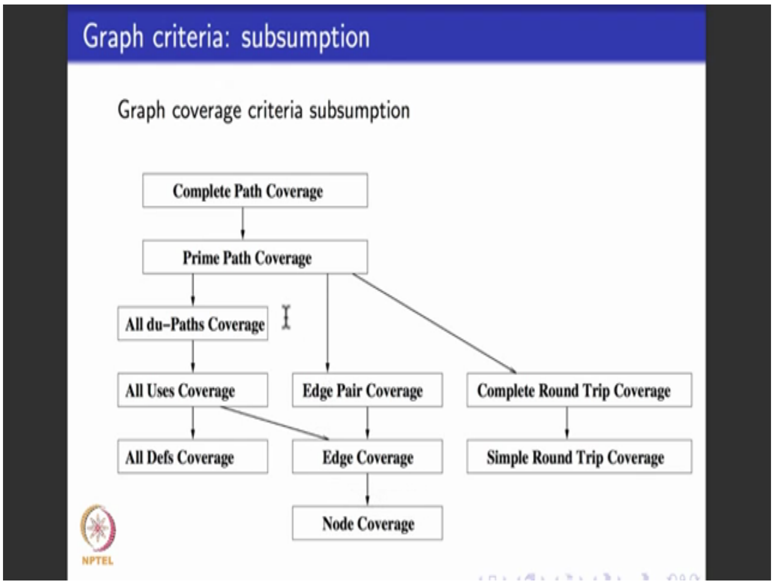
Intern all d u paths coverage says you not only reach every def to every possible use you figure on all possible ways of going from the def to use right. So, it by default as subsumes all uses coverage. So, that is what is given here it says, that if we satisfy all uses criteria by our definition we would have ensured that every definition was used. So, it subsumes all defs criteria again if we satisfy all d u paths criteria, we would have ensure that every definition reaches every possible use. So, it subsumes all uses.

(Refer Slide Time: 20:00)



So, the picture for data flow coverage criteria subsumption looks somewhat like this. All d u path coverage subsumes all uses coverage which in terms of subsumes all defs coverage. So, this is exclusively for data flow coverage criteria. This diagram is exclusively for structural coverage criteria, but both deal with graphs. So, I would be able to want to reach a stage such that I can relate one to the other. That is what we are going to do. Our goal is to understand such a picture.

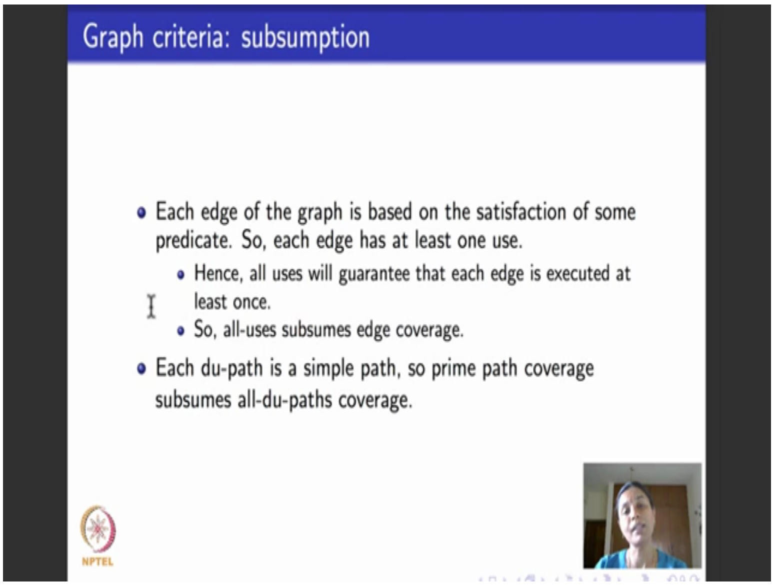
(Refer Slide Time: 20:31)



So, if you see this is 2 of the subsumption criteria figures merged into one, on this left hand side that I am tracing out here up through my mouse this path, which begins at complete path coverage prime path coverage goes on to this edge pair edge node and these round trip coverage criteria was purely related to structural coverage criteria. These 3 where the data flow coverage criteria subsumption that we saw now.

What have I done? Now I have put this extra arrow here and then this extra arrow here. So, I have come up with 2 extra subsumption relation. First one says the prime path subsume all d u paths, the second one says all uses subsume edge coverage. These 2 are the only 2 new subsumption relations that I have put the rest were all explained earlier. Why do these 2 additional new ones hold? They hold because of the following reason.

(Refer Slide Time: 21:33)

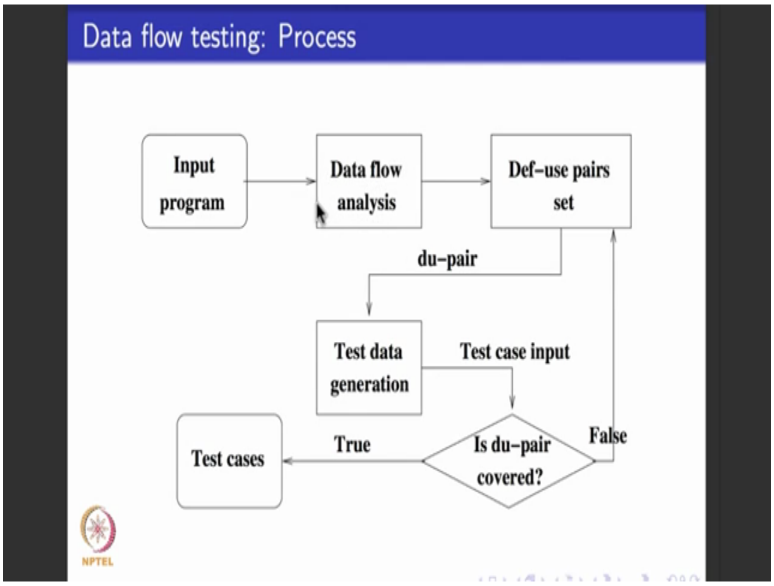


You remember each edge in the graph is based on the satisfaction of some predicate that was one of the assumptions that we meet.

So, each edge has at least one use and because of this when I cover all uses I definitely cover all edges. So, only use this subsumes edge coverage. (Refer Time: 21:53) because every d u path is a simply path prime path coverage subsumes all d u path because prime paths and simple paths that are not sub paths of any other paths. So, prime path coverage subsumes all d u paths. This is of course, a small point to be noted here is that this this subsumption is with reference to only feasible test criteria, but for now we can safely assume that this holds for most of the programs that we will look at.

Now that we have seen data flow coverage criteria and how the subsumption works and this is the overall picture for graph coverage criteria. I would like to spend some time looking or discussing algorithms for data flow coverage criteria.

(Refer Slide Time: 22:35)

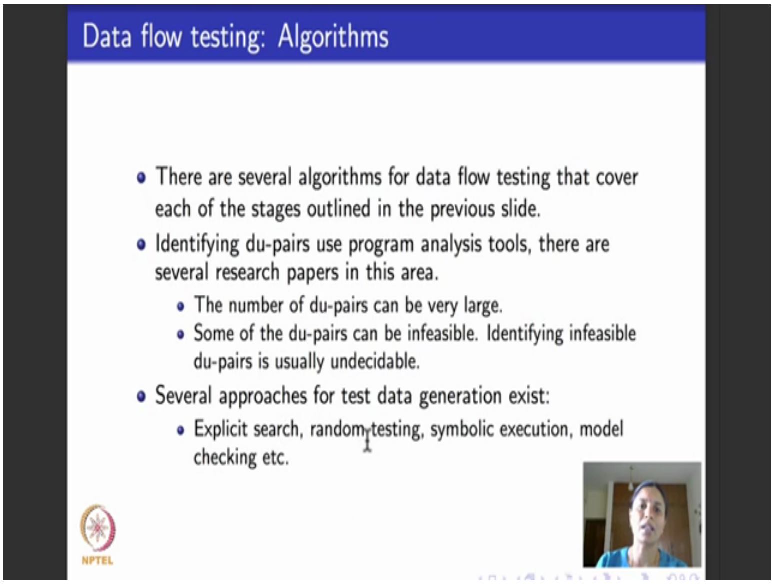


So, when I do data flow coverage this is the overall flow chart of the process for data flow coverage. So, I begin with my input program, I do some kind of a data flow analysis on that program. For this you could use any readymade static program analysis tool available or you could write your own elementary data flow analysis tool. And then using some kind of data flow analysis I pull out the def use pair sets.

Once I have a d u pair my goal is to be able to define some coverage criteria and generate test data for that coverage criteria. If I finish achieving my coverage criteria I am ready with my test cases, otherwise maybe the coverage criteria that I defined was infeasible. I go back pull out another d u pair and attempt all over again to work with a new coverage criteria that would again be feasible for infeasible.

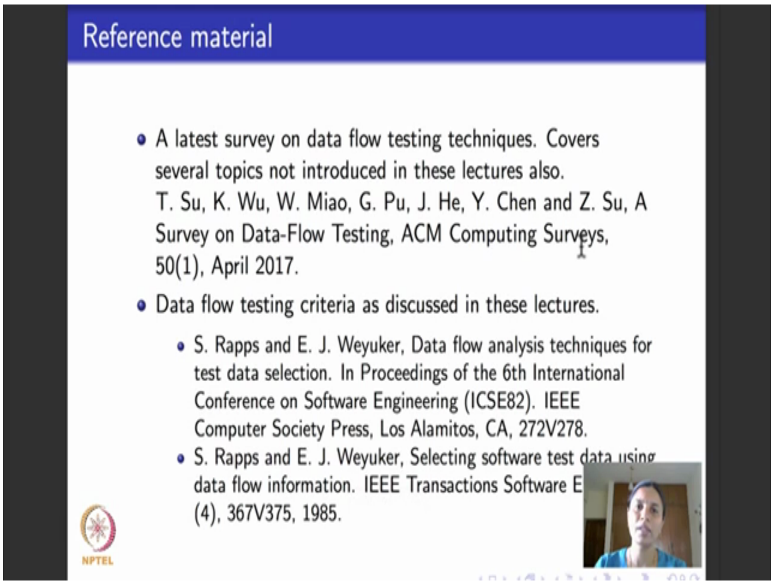
So, this part data flow analysis is not in the scope of this course. So, I will leave you to read this on your own. Feel free to pick up any basic books related to the first few chapters of compilers or use some elementary static program analysis tools to be able to do this. We will try to do a brief discussion on what are the algorithms for this part.

(Refer Slide Time: 23:51)



It, So, happens that for test data generation there are. So, many different algorithms you can do algorithms based on explicit search based on random search based on symbolic execution based on model checking several different techniques and about 4 decades of research has gone into algorithms in these areas. I will not spend time looking at these various algorithms, because we would like to move on looking at other testing test case definition terminologies.

(Refer Slide Time: 24:18)



But what I would like you to point out in this good survey that is come out with the recently dated April 2017 available as one of the ACM computing surveys that is an exhaustive survey on data flow technique testing techniques and you will be able to find reference to papers that use all these approaches for test data generation in this survey.

Most of the data flow testing criteria that we discussed in this paper has been borrowed by these 2 papers. So, weyuker and her student rapps. So, these 2 are very classical papers for data flow testing techniques. As you see they are data early 80s and quite exhaustively refer to the material that I have presented is basically derived from these 2 papers and the textbook on software testing by (Refer Time: 25:08). The next module we will model graph source for (Refer Time: 25:11) as graphs and we will see how the various structure coverage criteria, that we saw till now can be used to test code and then we look at design requirements and by next week I hope to finish the module on graph based testing.

Thank you.