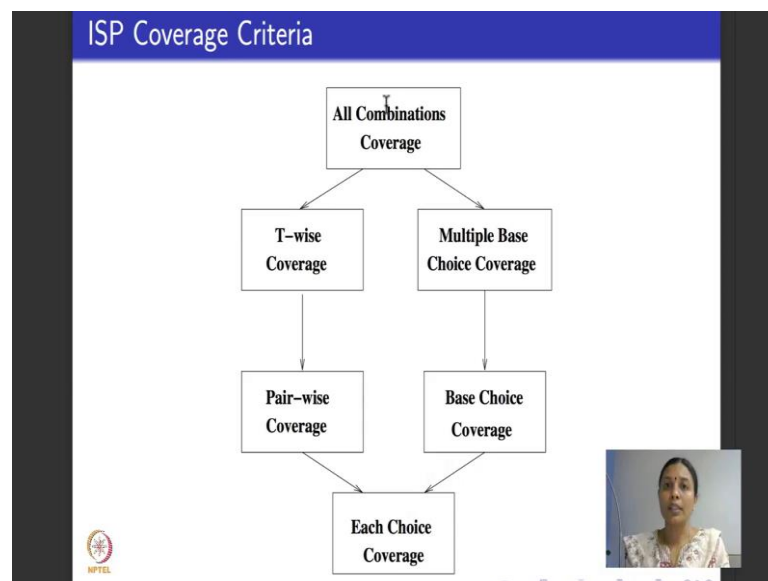


**Software Testing**  
**Prof. Meenakshi D'Souza**  
**Department of Computer Science and Engineering**  
**International Institute of Information Technology, Bangalore**

**Lecture - 34**  
**Input Space Partitioning Coverage Criteria: Example**

Hello again, welcome to the last week lecture of 7. We will be done with input space partitioning this week. So, what did we do till now? I began the week with telling you about functional testing, we saw various popular categories of functional testing, then I moved on to looking at one particular core functional testing technique namely that of input of space partitioning. We saw various of partitioning inputs in the third lecture interface based partitioning in functionality based partitioning. In the last lecture I told you about coverage criteria that you could define based on input space partitioning.

(Refer Slide Time: 00:45)



So, we have 6 different coverage criteria. The exhaustive input space partitioning coverage criteria is what is called all combinations coverage or ACoC write up here, and the weakest is consider each choice of values from each partition of the blocks that is each choice criteria, it is arbitrary consider to be a weak coverage criteria. In between the popular one is pair wise coverage criteria where for every partition of the input space per pair of partitions, you park and then let the others vary- then you park another pair of partitions let the others vary.

This is supposed to be one of the most powerful input space partitioning coverage criteria that is used. An extension of pair wise would be T wise coverage criteria where instead of working with a pair of partitions you work with T partitions at a time for a T that is greater than or equal to 2. The T happens to be all the partitions put together then we saw that it means nothing but all combinations criteria.

The ones on the left hand side where doing coverage arbitrarily without focusing on any particular characteristic or particular partition. We defined what is called base choice partition, which is the partition that would you like to focus on for some reason. And then we defined two coverage criteria based on the base choice partition: one is single base choice coverage also called base choice coverage. The next is multiple base choice coverage, where the number of base choices could be more than 1; you could choose more than 1 base choice coverage.


(Refer Slide Time: 02:36)


TriTyp example

Consider the interface-based ISP TR for the TriType example:

Partition	$b_1$	$b_2$	$b_3$	$b_4$
$q_1$ ="Relation of Side 1 to 0"	$> 0$	$= 0$	$= 1$	$< 0$
$q_2$ ="Relation of Side 2 to 0"	$> 0$	$= 0$	$= 1$	$< 0$
$q_3$ ="Relation of Side 3 to 0"	$> 0$	$= 0$	$= 1$	$< 0$

The above partitioning considers the relation of each side to 0 or 1, covers both valid and invalid inputs.





So, we saw all these 6 coverage criteria in the last lecture. We also took an abstract example and showed you how test cases or TR for each these of coverage criteria will look like. What I will do today is we will take one of the examples that we have done before in particular, we will take that triangle type example and then we will consider the partitions that we had defined in the third lecture for this week based on interface domain partitioning. And I will tell you how each of these coverage criteria can be defined on partitions for triangle type.

So, just to recap, triangle type was a program that took 3 sides of a triangle as input and it was defining the, classifying the triangle; whether it was not a valid triangle or it was a scaly in triangle or an isosceles triangle or an equilateral triangle based on the sides. So, one classic input space partitioning would be to consider what is the relationship of each side to, let us say, a 0 number or a 1 number.

Why is this kind of criteria important? It is important because if the side is less than 1 than we define invalid triangles and all other cases we defined various kinds of valid triangles. So, if you remember last lecture, I mean in the third lecture I had shown you this example for interface base inputs partitioning test requirement for that the TriTyp example. They were 3 criteria the first one said what is the relationship of side 1 to 0: is it greater than 0, is it equal to 0, is it equal to 1 or is it equal to 0. Similarly what is the relationship of side 2 to 0 and what is the relationship of side 3 to 0? Each these 3 cases we considered 4 partitions. So, we take side 1, side 2 and side 3 to be greater than 0, all 3 sides to be equal to 0, all 3 sides to be equal to 1 and all 3 sides to be less than 0.



So, these are 4 partitioning and it covers between them both valid and invalid triangles as we saw.

(Refer Slide Time: 04:12)

Possible test case values

One possible set of values that constitute test cases satisfying the partitions in the previous slide are:

Param	$b_1$	$b_2$	$b_3$	$b_4$
Side1	2	1	0	-1
Side2	2	1	0	-1
Side3	2	1	0	-1



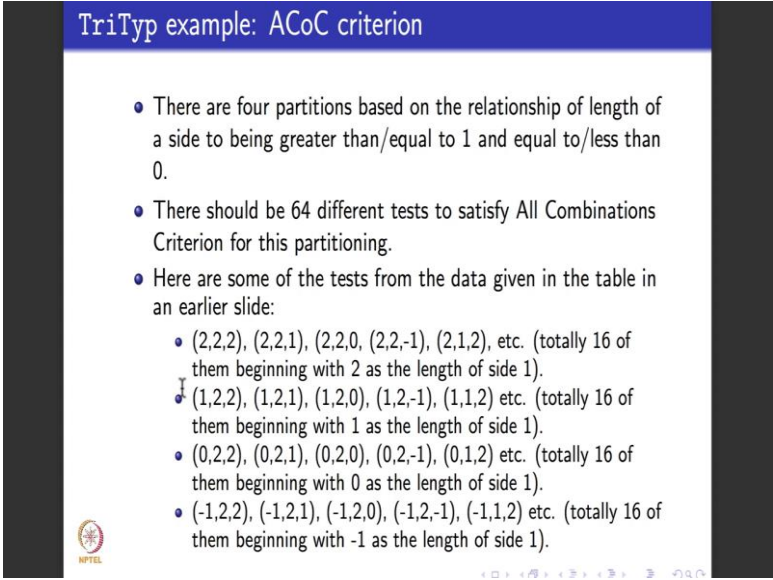
Now, what could be possible test cases for this test requirement? Here is one example value. So, let us say side 1 if you go back it is a choose values of side 1 greater than 1 0

equal to 0 equal to 1 less than 0. Similarly for side 2 and side 3. So, I have chosen value like that I have chosen side 1 to be 2, side 2 to be 1, side 3 to be 0, side 4 to be minus 1.

I could choose different values here, any different value, specially here for b 1 and b 4, but just for simplicity sake I have kept that also as 2 and minus 1. But any other number greater than 0 or any other number less than 0 will be good enough as values for the 3 sides it just as to meet criteria for this four partitions.

Now this is my set of test cases. On these set of test cases I am going to do go ahead and apply all these coverage criteria and see.

(Refer Slide Time: 05:10)



TriTyp example: ACoC criterion

- There are four partitions based on the relationship of length of a side to being greater than/equal to 1 and equal to/less than 0.
- There should be 64 different tests to satisfy All Combinations Criterion for this partitioning.
- Here are some of the tests from the data given in the table in an earlier slide:
  - (2,2,2), (2,2,1), (2,2,0), (2,2,-1), (2,1,2), etc. (totally 16 of them beginning with 2 as the length of side 1).
  - (1,2,2), (1,2,1), (1,2,0), (1,2,-1), (1,1,2) etc. (totally 16 of them beginning with 1 as the length of side 1).
  - (0,2,2), (0,2,1), (0,2,0), (0,2,-1), (0,1,2) etc. (totally 16 of them beginning with 0 as the length of side 1).
  - (-1,2,2), (-1,2,1), (-1,2,0), (-1,2,-1), (-1,1,2) etc. (totally 16 of them beginning with -1 as the length of side 1).

Suppose let me start with all combinations coverage. How many test case values are there, how many different partitions are there? 4 of them. How many inputs are there? 3 of them. So as per our formula how many different all combinations criteria test cases will be there?

There are 4 partitions that are based on relationship of length of a side being greater than or equal to 0 or one. So, what will be then total number of test cases the total number of test cases should be 64 right because for each of that I can choose any of 4 of them, any one of them, any of them keep going right. That is what the formula for all combinations coverage criteria it told us it is difficult for me to enumerate all the 64 test cases, but I have made an attempt to get you started if you want do that on your own. How do we go

about doing it? Keep an ordering, there is no need of ordering it is not enforced by the coverage criteria, but just to get your enumeration of this large number criteria 64 different cases right it helps to keep an ordering. So, what is what is the ordering that I have here if you see I have begun with side 1 being as 2 and now I vary side 2 and side 3. So, the first one says all 3 sides 1, 2 and 3 are all value 2.

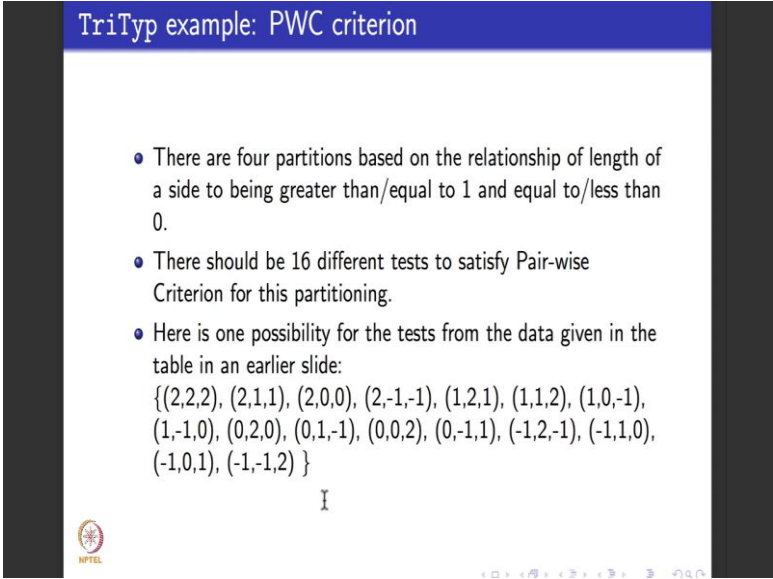
Remember in this listing, I have kept side 1 to be 2. Now, I vary side 2 to side 3 what are the values left out side 2 could be 2, side 3 could be one and side 2 could be 2, side 3 could be 0 and I move on, I say side 2 could be 2 side 3 could be minus 1. So, between these 4 cases what I have covered I have covered the option of side 1, side 2 being both 2 and side 3 ranging over the 4 different values 2, 1, 0 and minus 1. Now I change side 2, I say side 1 is still 2 side 2, let us say becomes one now you vary sides 0 for all these options. So, you start with 2 and now the next one in this list would which I have not written would be 2, 1, 1, the next one after that would be 2, 1, 0. The next one after that would be 2, 1 minus 1 I keep going like this. Four, four, four, each. So, totally I will get 16 of them beginning with side 2 as the length; length 2 as the length of side 1. Similarly let us consider length one as the length of side 1. I do the same thing I park side 1 as length one let us a 2 as length 2 and I vary side 3 first. So, here I make it side 3, 2, side 3, one side 3, 0 side 3, minus 1 and I move on.

I say side 1 is one side 2 is also one and I vary side 3. So, that will be 1, 1, 2 the next one in this list will be 1, 1, 1. The next one would be 1, 1, 0 and so on. So, here again I will get 16 of them beginning with one as a length of side 1. Now I say length of side 1 is 0 and again I do the same enumeration and I will get 16 more. Similarly length of side 1 is minus 1 I do the same enumeration I get 16 more. So, if I basically what does, it parks one number in this table varies across the other numbers parks.

The next number in the table there is at across the number and I will do this exhaustively that is what all combinations coverage criteria suppose to be. So, it results in 64 cases test cases. Please remember one more fact. Here when I talk about test cases I am just giving you the output a full test case you also have to give expected output, I have not given expected output I have just given the input. The complete test case you have to give the expected output also I prove I did not give the expected output I thought it is easier to explain just how the inputs vary based on the partitions.

So, moving on we realize that just for a small example like type of a triangle doing input space partitioning was effective enough this was good enough partitioning, but if you consider all combination criteria you end up with too many test cases. 64 of them is a large number of test cases we would really not want so, many of so let us move on and look at other coverage criteria.

(Refer Slide Time: 09:00)



TriTyp example: PWC criterion

- There are four partitions based on the relationship of length of a side to being greater than/equal to 1 and equal to/less than 0.
- There should be 16 different tests to satisfy Pair-wise Criterion for this partitioning.
- Here is one possibility for the tests from the data given in the table in an earlier slide:  
 $\{(2,2,2), (2,1,1), (2,0,0), (2,-1,-1), (1,2,1), (1,1,2), (1,0,-1), (1,-1,0), (0,2,0), (0,1,-1), (0,0,2), (0,-1,1), (-1,2,-1), (-1,1,0), (-1,0,1), (-1,-1,2)\}$

So, now, I told you after that the most useful and important coverage criteria is pair wise coverage. So, for same triangle type example let us workout the test case inputs for pair wise coverage criteria. How will they look? As I told you as a recap there are 4 partitions based on the relationship of length of the side being greater than or equal to 0 and less than or equal to 0 or 1.

Now for pair wise coverage criteria if you apply the formula that we saw in the last lecture, how many different test cases should be there? There should be 16 different test cases for pair wise coverage criteria and as I showed you in the last example for that abstract partitioning there could be several different ways in which you could meet pair wise coverage. Here I have listed one possibility for tests for achieving pair wise coverage.

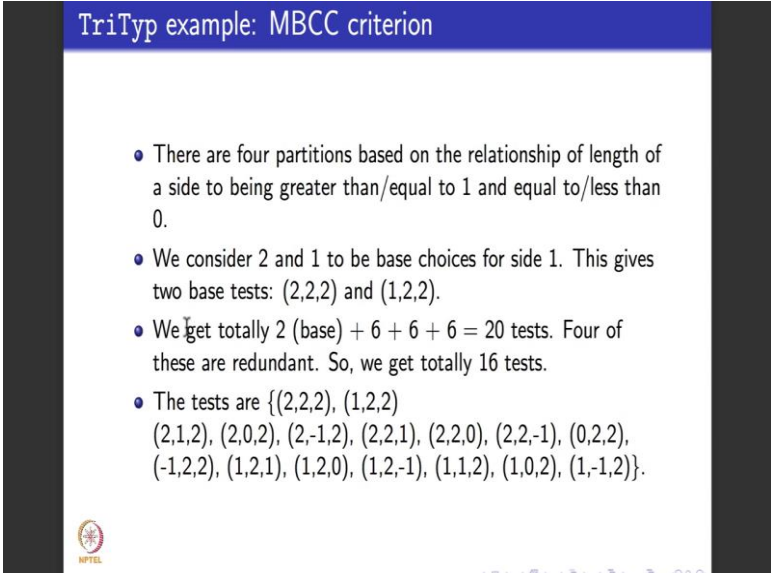
How do you read this? How do you understand it? The first 4 triples 3 triples in the set consider side 2 to be parked at number 2 and then it varies side length of sides one length of sides 2 and 3. So, side 1 is parked at the number 2, here one side 1 is to side 1 is to

here in the second one, side 1 is to here in the third one, side 1 is to here again in the fourth and then I vary sides 2 and 3 length.

So, here I made all 2 of them, side 2 and 3 both two, here I have made both one, here I have made both 0 and here I have made both minus 1 now what do I do at consider side 1 as 1. So, the next 4 in the list of that side 1 if you notice is one here in this one in this one in the eighth one and then I varies sides 2 and 3. I have made side 2 and 3, 2 and 1 here 1 and 2 here 0 and minus 1 here minus 1 and 0 here.

This is just my choice you could do anything it does not matter right. Several different test cases could be written to achieve pair wise coverage criteria. The next 4 that is from the ninth to the twelfth test case here park side 1 as 0 and then varies sides to 2 and 3. Here again the variations of sides 2 and 3 the numbers that you see a just my choice, feel free to substitute to the any other values for sides 2 and 3 that will equally well satisfied pair wise coverage criteria. And the last 13, 14, 15, and 16 side 1 is minus 1 and again sides 2 and 3 are varying as per one choice that I have given you. Here again you could feel free to substitute values for sides 2 and 3 in any way that is satisfy pair wise coverage criteria.

(Refer Slide Time: 11:44)



TriTyp example: MBCC criterion

- There are four partitions based on the relationship of length of a side to being greater than/equal to 1 and equal to/less than 0.
- We consider 2 and 1 to be base choices for side 1. This gives two base tests: (2,2,2) and (1,2,2).
- We get totally  $2 \text{ (base)} + 6 + 6 + 6 = 20$  tests. Four of these are redundant. So, we get totally 16 tests.
- The tests are  $\{(2,2,2), (1,2,2), (2,1,2), (2,0,2), (2,-1,2), (2,2,1), (2,2,0), (2,2,-1), (0,2,2), (-1,2,2), (1,2,1), (1,2,0), (1,2,-1), (1,1,2), (1,0,2), (1,-1,2)\}$ .

So, for the triangle type program for input space partitioning given in this table with these as the values for pair wise coverage criteria you get 60, sorry, you get 16 different test cases which I given here. Now we will move on last time I introduced you multiple

base choice coverage criteria, but I did not really give you an example of what multiple base choice coverage criteria will be. So, I thought I will put that instead of doing those single base coverage criteria for this lecture. So, here is how the example for the MBCC criteria looks like. There are 4 partitions again the that reference table, what is side 3 sides related to 0 and 1. So, what I do I have many choices for my base choice? Let us consider for my multiple base choice coverage criteria the number of base choices to be 2 and I consider 2 and one to be base choices for side 1. This gives us 2 base tests right which will be 2 for side 1 which is 2 here, which is one here, and sides 2 and 3 would be 2 and 2 same.

Now, if you apply the formula for MBCC criteria at totally how many tests will I get? These 2 base tests that come from this above item here and then 6 more each combination. So, totally I will get twenty tests and if you try to enumerate all these twenty tests you will realize that 4 of them are redundant, they come as repetitions. I have removed the repetitions. I have directly given you the succinct reduced setup 16 different test inputs for this that satisfied multiple base choice coverage criteria. The 16 different test inputs are like this, these 2 are the 2 base choice test I have listed them here right up in the front and how are these obtained if you try to study them. What would you do here? Here again you will notice that side 2 is side 1 is 2 here and then I varied one and 2, 0, 2 minus 1, 2, 2, 1, 2, 0, 2, minus 1 now the next part I have put side 2 side 1 as 0 and then I have consider 2, 2 because these are my base choices right. So, I have park all and reduce the rest.

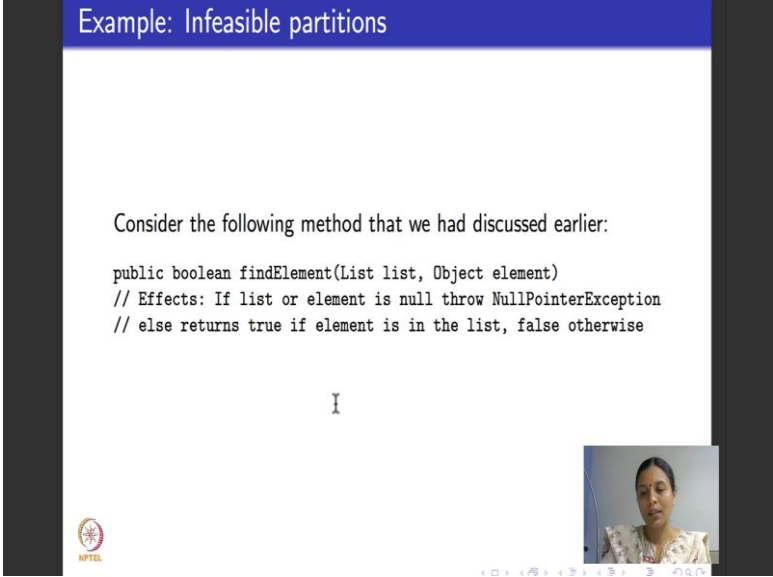
Similarly, here is minus 1, 2, 2. Similarly I do 1, 2, 1, 1, 2, 0, 1, 2, minus 1, 1, 2, 2, 1, 0, 2 and 1, minus 1, 2. So, for 2, 1, 1, I consider all options for 0 and minus 1, I consider only 2 options. So, totally there are 16 different test cases that will satisfy multiple base choice coverage criteria for the triangle type example. I hope this small exercise helps you to appreciate, how to do input space partitioning, and how to exhaustively list these test cases. Now these will be our actual test cases. To make them actual test cases what you have to do is to able up and expected output values. Like for example, if I take this first 3 case what is the triangle that I am looking at 2, 2, 2, all 3 sides are equal. So, I am looking at an equilateral triangle.

Now, what is the try type example suppose to give you as output for equilateral triangle? It was supposed to give a number put that as the expected output as the fourth tuple that



corresponds to the expected output that is how you arrive at a complete test case specification for each of these test cases.

(Refer Slide Time: 14:53)



Example: Infeasible partitions

Consider the following method that we had discussed earlier:

```
public boolean findElement(List list, Object element)
// Effects: If list or element is null throw NullPointerException
// else returns true if element is in the list, false otherwise
```

I

NPTEL

Now, moving on before I wind up I would like to make you understand one simple point. Like you know graph coverage criteria and logical coverage criteria for input space partitioning you can get infeasible test requirements. For example, if you remember let us revisit this find element method that we saw in the third lecture. If you remember I give you this specification of a find element method and because we are doing input space partitioning please remember we do not really need the code.

We just need to know what the method is what its inputs are and what its output are. So, it is enough to just give that much, This find element method that we discussed 2 lectures ago took a list of objects as input and then took an element as also an input and it basically checks if this element is present in the list. If it is present in list it is written true otherwise it is written false it is written exception if the list or the element is null.



(Refer Slide Time: 15:51)

**Infeasible combinations of partitions**

- Some combinations of partitions can be infeasible in the input domain model.
- For e.g., consider the boolean `findElement(list,element)` method with the following partitions:

	Partitions			
Characteristics	1	2	3	4
A: length and contents	one element	> than one, unsorted	> than one, sorted	> than one, all identical
B: match	element not found	element found once	element found more than once	—

- Invalid combinations are (A1,B3) and (A4, B2).



So, for this, for example, let us I have done input space partitioning. I have considered 2 different characteristics for partitioning my input. One characteristic that I have considered is the length and contents of the list. How long is this list comes as input? What are its contents what are the elements the constituents of the list? The second criteria that I have considered is there a match? It is a functionality based in input space criteria which means is the element found in the list it directly deals functionality of the concerned method.

So, with these 2 characteristics one is based on the length of the list and the kind of elements in the list. The second characteristic is based on whether the element is found in the list or not. Here are some example partitions. There are 4 partitions in the first case, 3 partition in the second case. So, the 4 partition in the first case are the length of the list is one, it has only one element, the length of the list is more than 1, the elements of the list are an unsorted order arbitrary order. The third partition says, the length of the list is more than 1, the elements of the list are in sorted order. The fourth partition says the length of the list is more than 1.

All the elements in the list are identical they are replicas of each other this is just one partition in based on this characteristic. For the second characteristic which I have called as match which basically test whether the element is present in the list or not, there are 3

partitions. The first partition says that element is not present in the list second partition it says element is present in the list, but only once.

Third partition says the element is present in the list more than once. This is some partitions that I have come up with. My goal is to be able to test this method based on these partition test cases obtained from these partitions. Now before moving on you will realize that some combinations of partitions are not valid, they are infeasible. For example, if you see the combination a, 1 and b, 3 what is a, 1 a one is this a is here, 1 is here, a, 1 is the list has exactly 1 element. What is b, 3 ? b is here which is about the match 3 says element found more than once. Clearly, a, 1 says- the list has exactly one element it is a length one b, 3 says the element is found more than once. These 2 cannot be coexist because of the list is of length one how can the element be found more than once. So, it is impossible to write a test case for the combination a, 1, b, 3. So, it is an infeasible combination.

Similarly, a, 4 and b, 2 is also an infeasible combination. What does a, 4 say? a and 4 it is says the for length of list is greater than 1 all the elements in the list are identical this list consisting of same elements copied several times. What does b 2 say? It says that the element is found in the list and it is found exactly one. Clearly, if the length of the list is more than 1 and the element copies itself again and again in the list the element cannot be found once; so this is another infeasible coverage criteria.

(Refer Slide Time: 19:13)

**Constraints among partitions**

- **Constraints** are relations between partitions from different characteristics.
- Two kinds of constraints:
  - A block from one characteristic *cannot* be combined with a block from another characteristic.
  - A block from one characteristic *must be* combined with a block from another characteristic.

NPTEL

Video inset showing a presenter.

So, when we design partitions we have to be careful if it is quite natural that we will get infeasible coverage criteria we have to be able to rule them out. So, constraints capture these infeasibility amongst partitions. What are constraints? Constraints are relations between partitions from different characteristics like we saw here right they cannot be a relation when the element list has more than 1 element all identical and you say the element is found only once it cannot be a relation at all. So, there are 2 kinds of constraints one which says that a block from one characteristic cannot be combined with a block from another characteristic. The example that we saw in these slides belongs to the first category. The second category which I have not shown in that example says that a block from characteristic must be combined with the block one characteristic.

If you go back here for the same example for example, I could say when the list has exactly one element then it must be combined with the element not found or with the element found more than, found exactly once. It cannot be combined with an element found more than once.

(Refer Slide Time: 20:29)

ISP coverage criteria and constraints among partitions

- For ACoC, PWC and TWC, the only option is to drop the infeasible combinations from consideration.
- Constraints can be handled better with BCC and MBCC criteria. The base case(s) can be altered to handle infeasible constraints.

So, cannot and must, these 2 are what the constraint say. You must combine some partitions with some partitions to make sense for test cases you should not combine some partitions with certain other partitions to make sensible test cases that are actually feasible. So, what do we do for all combinations coverage criteria, pair wise and T wise coverage criteria, you really cannot do much, you just have to drop the infeasible test

requirements. Why that is so, because, if you remember these coverage criteria do not have any intelligence they just blindly combine the partitions. And if partitions are infeasible there is nothing you can do about it, just drop them out. But if you have base choice coverage criteria or multiple base choice coverage criteria then may be you need not to choose base choices as those partitions that result in too many infeasible test requirement or test cases. So, you have better hold of infeasible test requirements by choosing the choice of your base choice to handle or minimize the infeasible test requirements appropriately. So, while partitioning the input and while writing test cases based on input base partitioning, please remember that sometimes you will get infeasible test requirements and you might have to omit that.

So, next week we will move on to a completely different module and testing called mutation testing and this will be the end of input space partitioning for you.

Thank you.