

**Software Testing**  
**Prof. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 11**  
**MC/DC Testing**

Welcome to **this session**, we will now look at MC/DC testing. So far we have been looking at white ( ) box testing and the different types of coverage **based** testing techniques. So, let us look at white box testing.

(Refer Slide Time: 00:36)

**Recap: Condition Testing**

- **Simple or (basic) Condition Testing:**
  - Each atomic condition made to have both T and F values
  - Example: **if (a>10 && b<50)**
  - **The following would achieve basic condition coverage**
  - **a=15, b=30**
  - **a=5, b=60**
- Does basic condition coverage subsume decision coverage?

In the last session, we were looking at condition testing and if you remember that a simple condition testing, each atomic condition is made to have both true and false values. So, we saw that there are several types of condition testing, one is called as the basic condition testing that is the simplest testing, and here a decision statement like this the conditional expression consists of several atomic conditions, for example, a greater than 10, b less than 50, etcetera and this testing; the basic condition testing.

This requires us to have test cases that will have these expressions atomic expressions evaluating to true and false individually, for example, let us say if we have a is equal to 15 and b is equal to 30. So, a is equal to 15 this will set the first atomic condition to be true and the second one 30 less than 50 will also be true. So, this will become true and the second one is a greater than **a=5**. So, this will

be false and b is 60 less than 50 will be false.

So, does this achieve basic condition coverage, certainly because in the two test cases, a is taking the values true and false, sorry, the first expression is taking the value true and false. In the second test case, the second expression with the two test cases is also taking the values true and false. So, we achieve basic condition coverage, do we also achieve decision coverage? Yes, with this two test cases we also achieve decision coverage because the first test case would evaluate this decision to true and the second test case will evaluate this to false. These two will achieve both basic condition coverage and decision coverage.

But basic condition coverage may not always achieve decision coverage, for example, if we had a greater than the first test case is 15 and 60 and the second test case is 5 and 30. So, let me just have two test cases, alternate test cases 15 and 60, a is 15 and b is 60 and the second test case, a is 5 and b is 30.

In this case, both the test cases will make the decision to evaluate to false and therefore, decision coverage will not be achieved because the complete branch expression is not getting true and false value, the basic condition coverage does not ensure decision coverage. So, what can we tell about the subsumption relationship? Will basic condition coverage subsume decision coverage? The answer is no because the basic condition coverage need not ensure decision coverage.

(Refer Slide Time: 05:11)

### Recap: Condition Testing

- Decision (also called branch) testing is a weak testing criterion.
- Condition testing:
  - Basic condition coverage
  - Basic condition/decision coverage
  - Multiple condition coverage
  - Modified condition and decision coverage

So, we had so far seen that decision testing is a weak testing criterion and there are several types of condition testing. So, decision testing if we are doing that is we are having entire conditional expression in a program; the test cases just evaluate them to true and false. We are not doing a very thorough testing, we need to do condition testing to achieve a more thorough testing and the condition testing are of many types. We just looked at the basic condition coverage.


In basic condition coverage, the atomic conditions in a conditional expression achieve true and false values; all the atomic conditions in the conditional expression are made to assume true and false values by execution of the test cases. But there are other conditional testing techniques. Let us look at that and the conditional testing we saw that the basic conditional testing is not really a stronger testing than decision testing because the basic condition testing does not ensure decision testing.

Now, let us look at the next condition based testing called as basic condition with decision coverage. So, the main idea with basic condition with decision coverage testing is that it ensures basic condition testing and also has test cases to ensure decision testing, decision coverage.

(Refer Slide Time: 07:09)

### Condition Testing

- **Condition/decision coverage:**
  - Each atomic condition made to assume both T and F values...
  - Decisions are also made to get T and F values...
  - **Example: if ( $a > 50$  or  $b < 30$ )**
    - $A=70, b=50$  and  $a=30, b=20$  achieves basic condition coverage
    - Need to include  $a=20, b=50$  for condition/decision coverage



Now, let us look at another example and we had already seen that condition decision coverage not only ensures that the test cases set the atomic condition evaluation true and false. So, all atomic conditions evaluate to true and false sometimes during the test case executions and also the complete decision complete expression also evaluates to true and false during the test case

execution.


Now, let us take these example, a greater than 50 or b less than 30. Now, if we have a is 70 and b is 50 and a is 30 b is 20 it achieves basic condition coverage because the first one sets it true and this one is false and the second one this is false and this is true. So, the first atomic condition takes true value in the first test case and false value in the second test case, where as the second conditional expression takes false value in the first test case and true value in the second test case.

In both these it evaluates to true, both these expressions both these test cases set the branch outcome to true, the decision evaluates to true for both the test cases and therefore, we need to include another test case to achieve the decision coverage that is, a is 20, we sets it to false and b is 50. So, false are false. This becomes false and these two set it to true. So, decision coverage is achieved. For condition decision coverage we might have to augment the test cases achieve condition coverage

(Refer Slide Time: 09:47)

### Condition Testing

- **Multiple condition coverage (MCC):**
  - Atomic conditions made to assume all possible combinations of truth values
  - Example: **if(a>50 or b<30)**
  - Following test suite achieves MCC
    - A=30, b=20
    - A=30, b=40
    - A=60, b=20
    - A=60, b=40

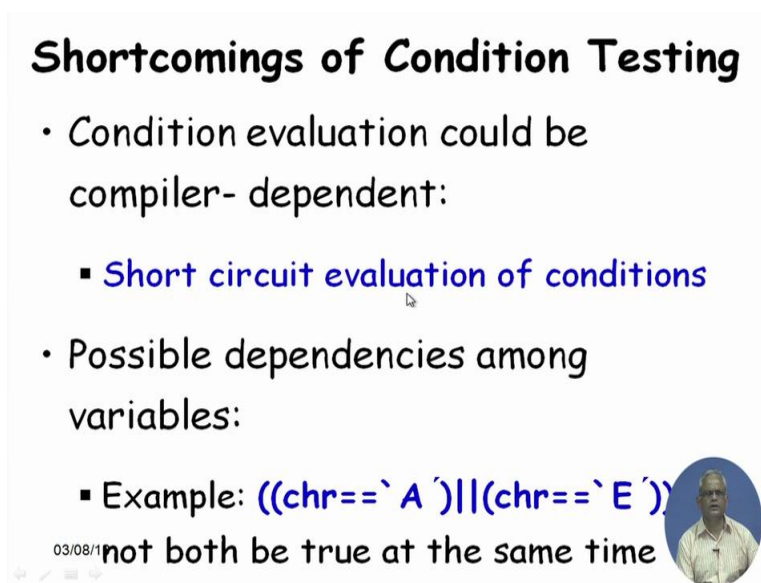


Now, let us look at another condition testing which is the multiple condition coverage. In this testing all the atomic conditions assume true and false values not only that all possible combinations of truth values are assumed by a these test cases all possible combinations so that means, if there are two atomic conditions in the conditional expression, we need four test cases setting this to true and true, true and false, false and true and false and false. If we have three atomic conditions we would

need 8,  $2^3$  which is 8 test cases true, true, true, true, true, false, etcetera.

Now, let us see what is a test suite which will achieve multiple condition coverage for this branch expression this, if statement what will be the test case that will achieve multiple condition coverage. So, we need, a is 30, b is 20, so false and true; a is 30, b is 40, so false and false and then true and false and true and true. So, we need four test cases to achieve multiple condition coverage for this if statement.


(Refer Slide Time: 11:39)



**Shortcomings of Condition Testing**

- Condition evaluation could be compiler- dependent:
  - Short circuit evaluation of conditions
- Possible dependencies among variables:
  - Example: `((chr==`A`)||(chr==`E`))`  
not both be true at the same time

03/08/1



If we think of it, the multiple condition coverage is the strongest of all. The multiple condition coverage ensures both condition coverage and the decision coverage and also the condition decision coverage. So, **that is** are the strongest testing, but let us see what the shortcomings of this condition testing techniques. The first problem is that we looked at a simplistic way of condition expression evaluation, the compiler does not evaluate like that it evaluates what is through what is known as the short circuit evaluation of conditions.

All the conditions are not evaluated as long as the outcome of some condition makes the other condition evaluations redundant other conditions are not evaluated. So, giving various possible values to other conditions will not really matter. We will look at an example of short circuit evaluation and why multiple condition coverage and condition decision coverage, etcetera, the way we looked at is really too simplistic the compiler does not really evaluate it like that and therefore,

the number of test cases required may be less and some of our assumptions **were** actually too simple the compiler does not evaluate like that.

The second problem is dependency among variables for example, if we are checking a character we are reading a character and checking that whether it is A or E we would write this expression character is A or character is E, but then achieving multiple condition coverage for this will not be possible because this can at best one of them can be true we cannot have true, true for this. So, due to dependency among variables certain combinations of conditions may not be possible at all even though our coverage might require some condition values to be set may not be possible to set those values.

(Refer Slide Time: 14:39)

### Short-circuit Evaluation

- `if(a>30 && b<50)...`
- If `a>30` is FALSE compiler need not evaluate `(b<50)`
- Similarly, `if(a>30 || b<50)...`
- If `a>30` is TRUE compiler need not evaluate `(b<50)`

This is just an example of a short circuit evaluation by the compiler and to make things more complicated the short circuit evaluation varies from compiler to compiler, we cannot make assumption that all compilers evaluate a conditional expression like **this** different compilers might do different things.

Now, let us look at some examples that typically the compilers do short circuit evaluation if we have `a > 30` and `b < 50` is a conditional expression, then we know that the first one is non evaluated from left side it evaluates and the first one is evaluated to false then it need not evaluate the second condition. So, whether we give true or false to the second does not really matter because the

compiler does not check it.

Similarly, if the conditional expression is  $a > 30$  or  $b < 50$ , so here also as long as one evaluates to true the other **one** is not checked some compiler may evaluate from the left side and if they found that the first one evaluates to true first atomic condition evaluates to true. The compiler would not check the second expression whether it is true or false. So, our setting true or false in our coverage we will have little meaning with this compiler because it does not use that.

(Refer Slide Time: 16:29)

### Multiple Condition Coverage

- Consider a Boolean expression having  $n$  atomic components:
  - **MCC is strongest**
  - **But, for MCC we require  $2^n$  test cases.**
  - **Therefore practical only if  $n$  (the number of component conditions) is small (two or three).**

Now, let us look at another sort coming of the multiple condition coverage this is a big problem actually the problem is that even though the multiple condition coverage is the strongest, but then the number of test cases required are exponential. So, if we have  $n$  atomic expressions in a branch expression then we need  $2^n$  test cases. So, if  $n$  is 10, we need 1024 test cases.

This is just too many and imagine if we have 20 or 30 atomic conditional expressions atomic conditions appearing in a branch expression and there are many applications where 20 or 30 atomic conditions is quite common for example, in embedded controller applications where various parameters are sensed and depending on certain variables certain actions that values of certain variables certain actions had taken their conditional expressions involving ten fifteen and twenty variables is quite common.



If we have to test those applications and we require multiple condition coverage  $2^{20}$  test cases or  $2^{30}$  test cases for just checking one expression is just too much because there may be dozens of such expressions in a program. So, the multiple condition coverage is meaningful as long as we have only 2 or 3 atomic conditions in a conditional expression. Now, what do we do, how can we achieve the thoroughness of multiple condition coverage with less number of test cases we cannot execute millions of test cases can we have a linear number of test cases in a number of atomic conditions exponential test cases in the number of atomic expressions is not practical.


(Refer Slide Time: 18:59)

**Compound conditions: Exponential complexity**  
 $((a \parallel b) \&\& c) \parallel d) \&\& e$

Test Case	a	b	c	d	e
(1)	T	—	T	—	T
(2)	F	T	T	—	T
(3)	T	—	F	T	T
(4)	F	T	F	T	T
(5)	F	F	—	T	T
(6)	T	—	T	—	F
(7)	F	T	T	—	F
(8)	T	—	F	T	F
(9)	F	T	F	T	F
(10)	F	F	—	T	F
(11)	T	—	F	F	—
(12)	F	T	F	F	—
(13)	F	F	—	F	—

$2^5 = 32$

• Short-circuit evaluation often reduces number of test cases to a more manageable number but not always...



So, with that motivation we will look at the modified condition decision coverage. Now, this is another example, here we have 5 atomic conditions and if we write down here with a short circuit evaluation we have written the short circuit evaluations as dash. So, the compiler will evaluate irrespective of the value of the atomic expression written as dash is a do not care this is becomes do not care.

So, normally we would require  $2^5$  which is 32 test cases for achieving multiple condition coverage for this expression, but with short circuit evaluation, we need only 13 test cases. So, the other 19 test cases are actually redundant even if we write those test cases other 19 test cases, they are as good as nothing.

We need not even have had those test cases and executed that has no effect because of the short



circuit evaluation. This is just an example of the effect of short circuit evaluation. So, short circuit evaluation by the compiler reduces the number of test cases to a lesser number, but not always and we do not know, what is the short circuit evaluation for a specific compiler? We cannot generalize our assumption because it varies from compiler to compiler the specific short circuit techniques they use.

(Refer Slide Time: 20:57)

#### **Modified Condition/Decision Coverage (MC/DC)**

- Motivation: Effectively test **important combinations** of conditions, without exponential blowup to test suite size:
  - "Important" combinations means: Each basic condition shown to independently affect the outcome of each decision
- Requires:
  - For each basic condition *C*, two test cases
  - Values of all *evaluated* conditions except *C* are the same
  - Compound condition as a whole evaluates to *true* for one and *false* for the other

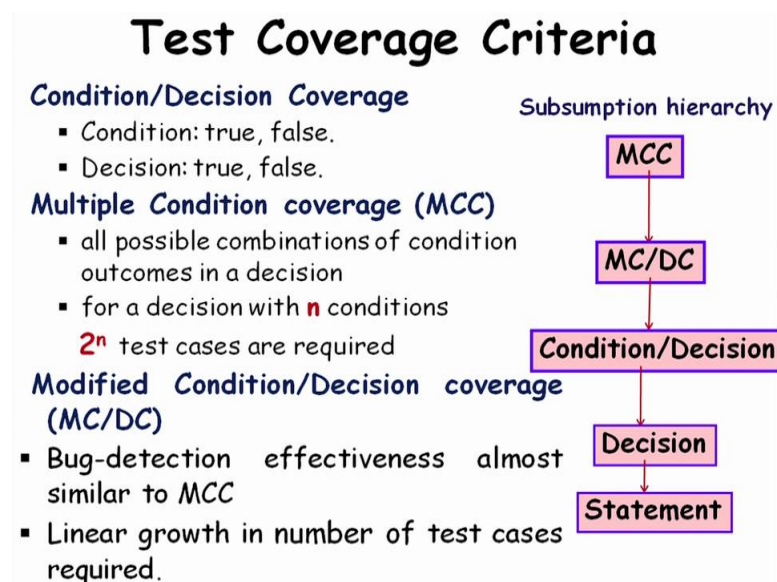
Now, with this motivation that there are many programs where a number of atomic conditions in the conditional expressions is large of the order of 20, 30, etcetera. We cannot use multiple condition coverage, but can we achieve as much fault detection capability as the multiple condition coverage with a reduced number of test cases possibly linear in the number of atomic conditions. So, that is modified condition decision coverage. Now, let us look at the main ideas here it has become an important test technique and many standards especially for safety critical software require MC/DC coverage.

Here the main idea is that we want to test the important combinations of conditions. I do not want to try out all possible combinations of conditions we want to try out the important combinations of condition and what we mean by important is that each basic condition. So, that is each atomic condition it determines the outcome of the branch independently of the other conditions. So, what we mean is that we keep the for any one atomic condition we keep the other conditions constant and just change the atomic condition to true and false and the branch outcome should change to true and false.

So, we need to have such test cases for **every** basic condition. So, to tell it in more concrete terms to achieve MC/DC coverage, we need three, we have three requirements the first is each basic condition should be made to true and false this is just like the condition coverage each basic condition should achieve true and false values and also the values of all other conditions. If we are looking at one atomic condition, the values of all other atomic conditions should be the same when we this basic condition achieves evaluates to true and false, and also we need the compound condition as a whole to evaluate to true and false for this atomic condition.

Now, let us look at this more carefully and also we will look at several examples.

(Refer Slide Time: 24:06)

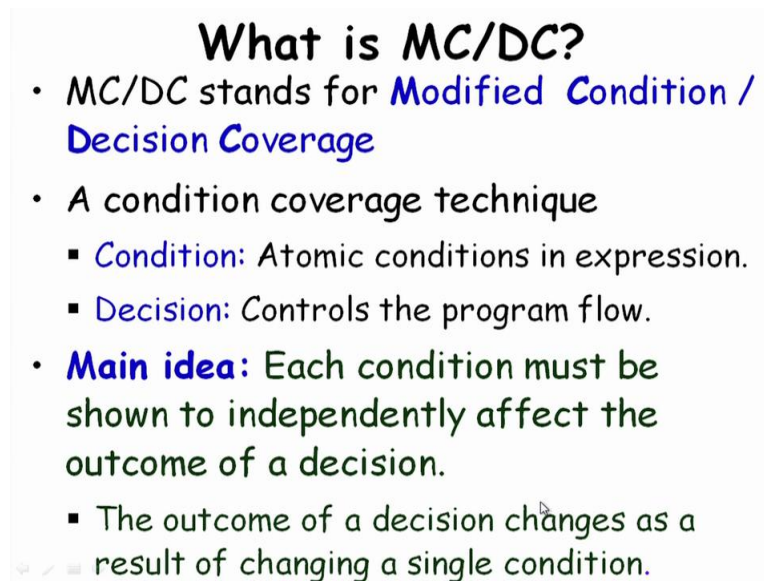


Before that we just want to mention here is that multiple condition decision coverage is a stronger coverage than the condition decision coverage, and we had seen that the condition decision coverage is a stronger testing than the decision coverage and the condition coverage and decision coverage at the branch coverage is a stronger testing than statement coverage, and see here the MC/DC is stronger than condition decision coverage.

So, what we really mean by this is that if we are doing MC/DC testing, we do not have to worry about any other condition testing. We already said that the multiple condition coverage even though it is a strong testing technique, it is a strong testing requirement, but then it is not practical there will be too many test cases exponential in the number of the atomic conditions.

MC/DC stands for Modified Condition Decision Coverage, and Experimentally, it has been seen that the bug detection effectiveness is almost as much as the multiple condition coverage, but the number of test cases required **to achieve** multiple condition, this modified condition decision coverage that is MC/DC is linear in the number of atomic conditions.

(Refer Slide Time: 25:59)



**What is MC/DC?**

- MC/DC stands for **Modified Condition / Decision Coverage**
- A condition coverage technique
  - **Condition:** Atomic conditions in expression.
  - **Decision:** Controls the program flow.
- **Main idea:** Each condition must be shown to independently affect the outcome of a decision.
  - The outcome of a decision changes as a result of changing a single condition.

Let us look at what is MC/DC? It stands for Modified Condition Decision Coverage; it is also a condition coverage technique. The main idea here in MC/DC is that each condition would be shown to independently affect the outcome of a decision. So, the test cases should be such that each atomic condition in the conditional expression would independently affect the outcome of the decision. The decision would be true when **an** atomic condition is true and it will be false when the atomic condition is false. So, we can say that the decision changes when the atomic condition changes from true to false.

The number of test cases required here would obviously, be at least twice as much as the number of atomic conditions and therefore, we can say that it is linear it is not like  $2^n$ , it will be  $2*n$  or near about that. It will be linear in the number of atomic conditions. So, we will stop here and will resume our discussion on MC/DC testing in the next session.

Thank you.