CSE412

Software Engineering

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

Topic 5

Black Box Testing

Black Box Testing

- Black-box testing, also called behavioral testing, focuses on the functional requirements of the software
- Different techniques
  - Graph-Based Testing Methods
  - Data Coverage Analysis
  - Boundary Value Analysis

Data Coverage Analysis

- Create columns to use in the data coverage analysis

  Create columns that map to source data columns to use in the analysis. You can create columns that use source data as is or in ranges of data values. You can also create columns that map data values across specific groups of values.

- Run the data coverage task and view the results of the analysis on the data coverage page

  You can run the data coverage task and view the results in the data coverage page. You can edit the results view to select and view specific cells in the results.

- Edit the parameters used in the analysis

  You can edit the parameters including the columns and filters applied and view updated analysis results in the data coverage page.

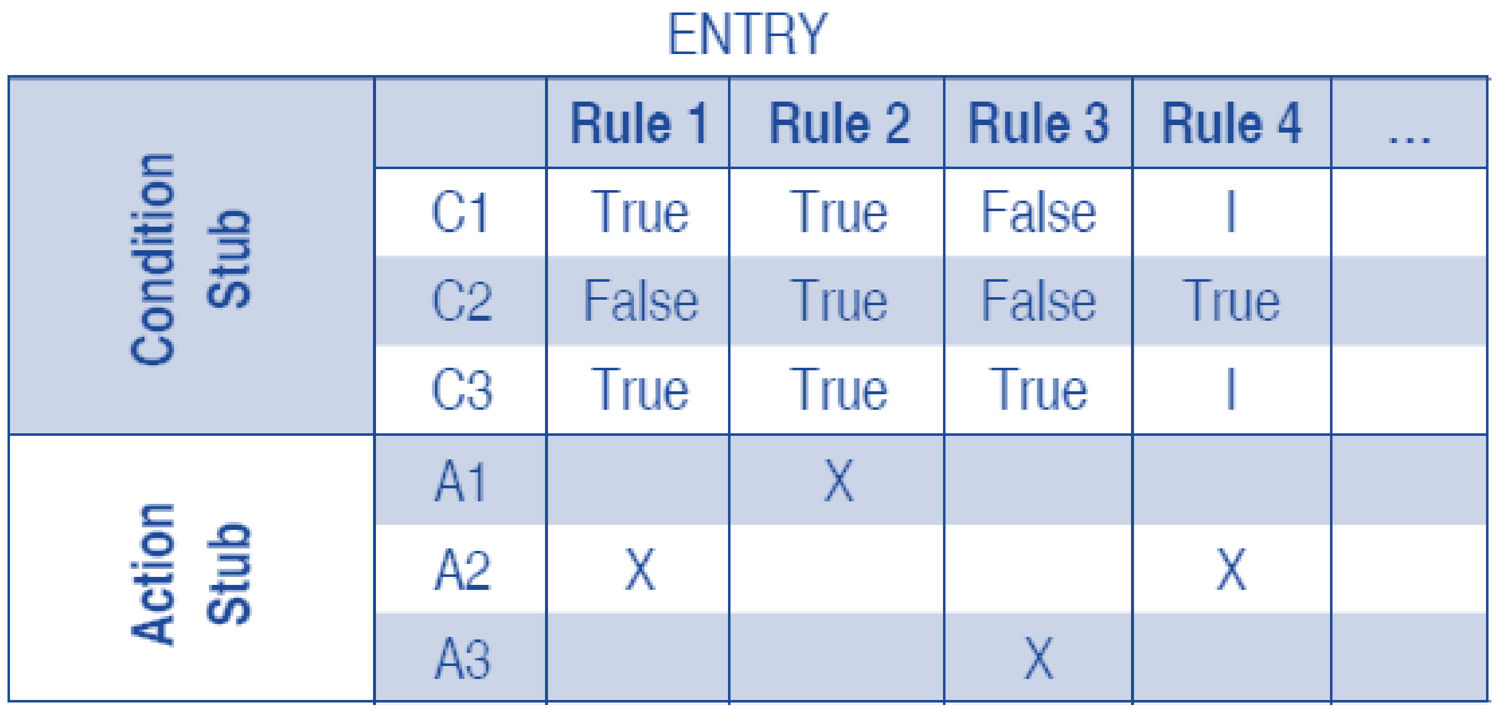
- Update data values in the source data

  You can update data values across cells that you analyze. For example, based on the data coverage analysis results, a cell contains data values below the minimum threshold that you set. You can update data in other cells to create the minimum required data values in the cell.

Decision Table

- Decision table is a useful method to represent the information in a tabular method. It has the specialty to consider complex combinations of input conditions and resulting actions.
- Decision tables obtain their power from logical expressions. Each operand or variable in a logical expression takes on the value, TRUE or FALSE.

Formation Of Decision Table



ENTRY

		Rule 1	Rule 2	Rule 3	Rule 4	…
Condition Stub	C1	True	True	False	I	
	C2	False	True	False	True	
	C3	True	True	True	I	
Action Stub	A1		X			
	A2	X			X	
	A3			X		

Decision table structure

Decision Table Structure

- Condition Stub
  - It is a list of input conditions for which the complex combination is made.
- Action Stub
  - It is a list of resulting actions which will be performed if a combination of input condition is satisfied.
- Condition Entry
  - It is a specific entry in the table corresponding to input conditions mentioned in the condition stub. When we enter TRUE or FALSE for all input conditions for a particular combination, then it is called a Rule.
  - When the condition entry takes only two values—TRUE or FALSE, then it is called Limited Entry Decision Table.

Decision Table Structure

  - When the condition entry takes several values, then it is called Extended Entry Decision Table.
  - In limited entry decision table, condition entry, which has no effect whether it is True or False, is called a Don’t-Care state or immaterial state.
- Action Entry
  - It is the entry in the table for the resulting action to be performed when one rule (which is a combination of input condition) is satisfied. ‘X’ denotes the action entry in the table.

Test Case Design Using Decision Table

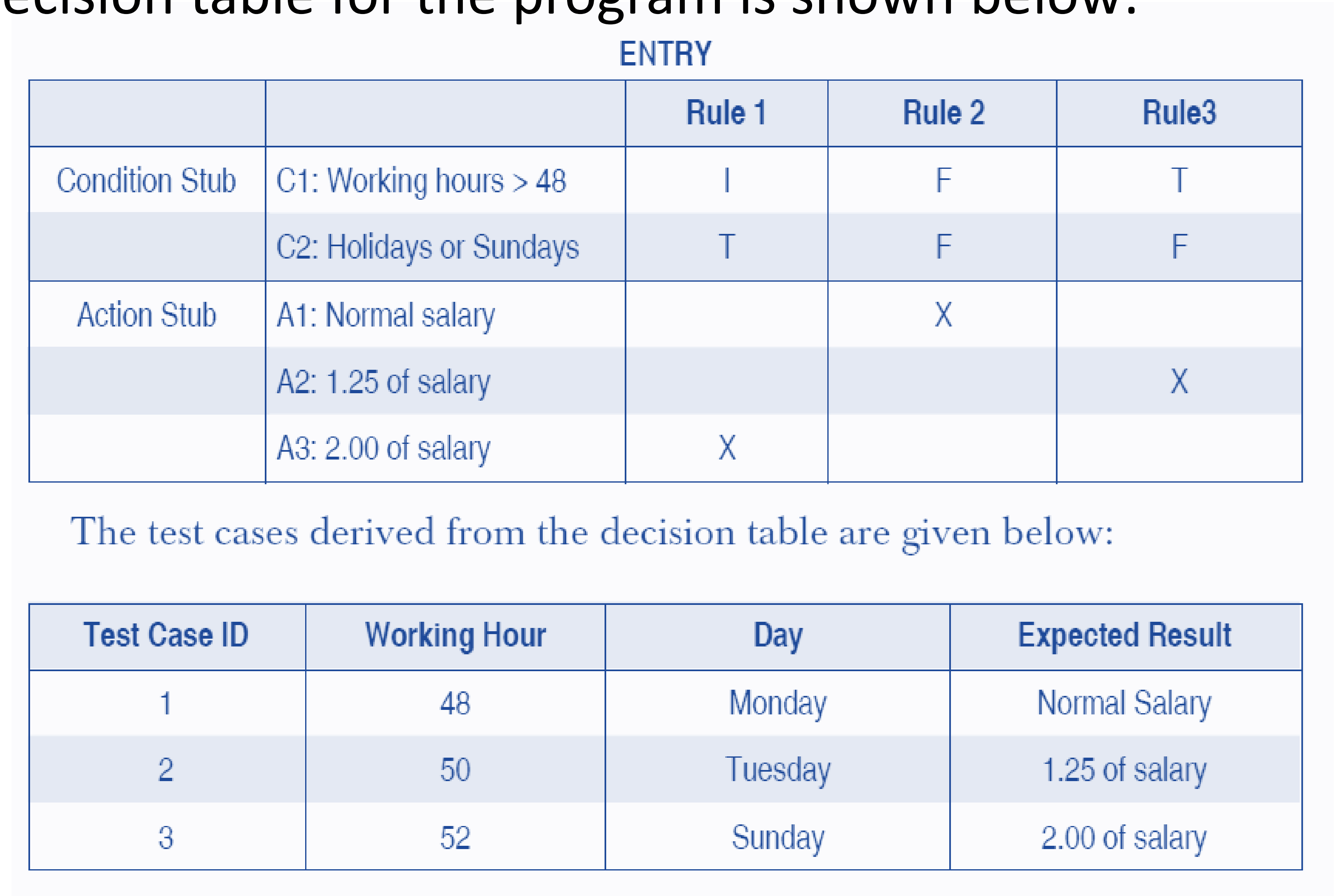
- For designing test cases from a decision table, following interpretations should be done:
  - Interpret condition stubs as the inputs for the test case.
  - Interpret action stubs as the expected output for the test case.
  - Rule, which is the combination of input conditions, becomes the test case itself.
  - If there are **k** rules over n binary conditions, there are at least **k** test cases and at the most $\mathbf{2^n}$ test cases.

Example

A program calculates the total salary of an employee with the conditions that if the working hours are less than or equal to 48, then give normal salary. The hours over 48 on normal working days are calculated at the rate of 1.25 of the salary. However, on holidays or Sundays, the hours are calculated at the rate of 2.00 times of the salary. Design test cases using decision table testing.

Example: Solution

- The decision table for the program is shown below:



ENTRY

		Rule 1	Rule 2	Rule3
Condition Stub	C1: Working hours > 48	I	F	T
	C2: Holidays or Sundays	T	F	F
Action Stub	A1: Normal salary		X	
	A2: 1.25 of salary			X
	A3: 2.00 of salary	X		

The test cases derived from the decision table are given below:

Test Case ID	Working Hour	Day	Expected Result
1	48	Monday	Normal Salary
2	50	Tuesday	1.25 of salary
3	52	Sunday	2.00 of salary

Example 2

A wholesaler has three commodities to sell and has three types of customers. Discount is given as per the following procedure:

(i) For DGS & D orders, 10% discount is given irrespective of the value of the order.

(ii) For orders of more than Rs 50,000, agents get a discount of 15% and the retailer gets a discount of 10%.

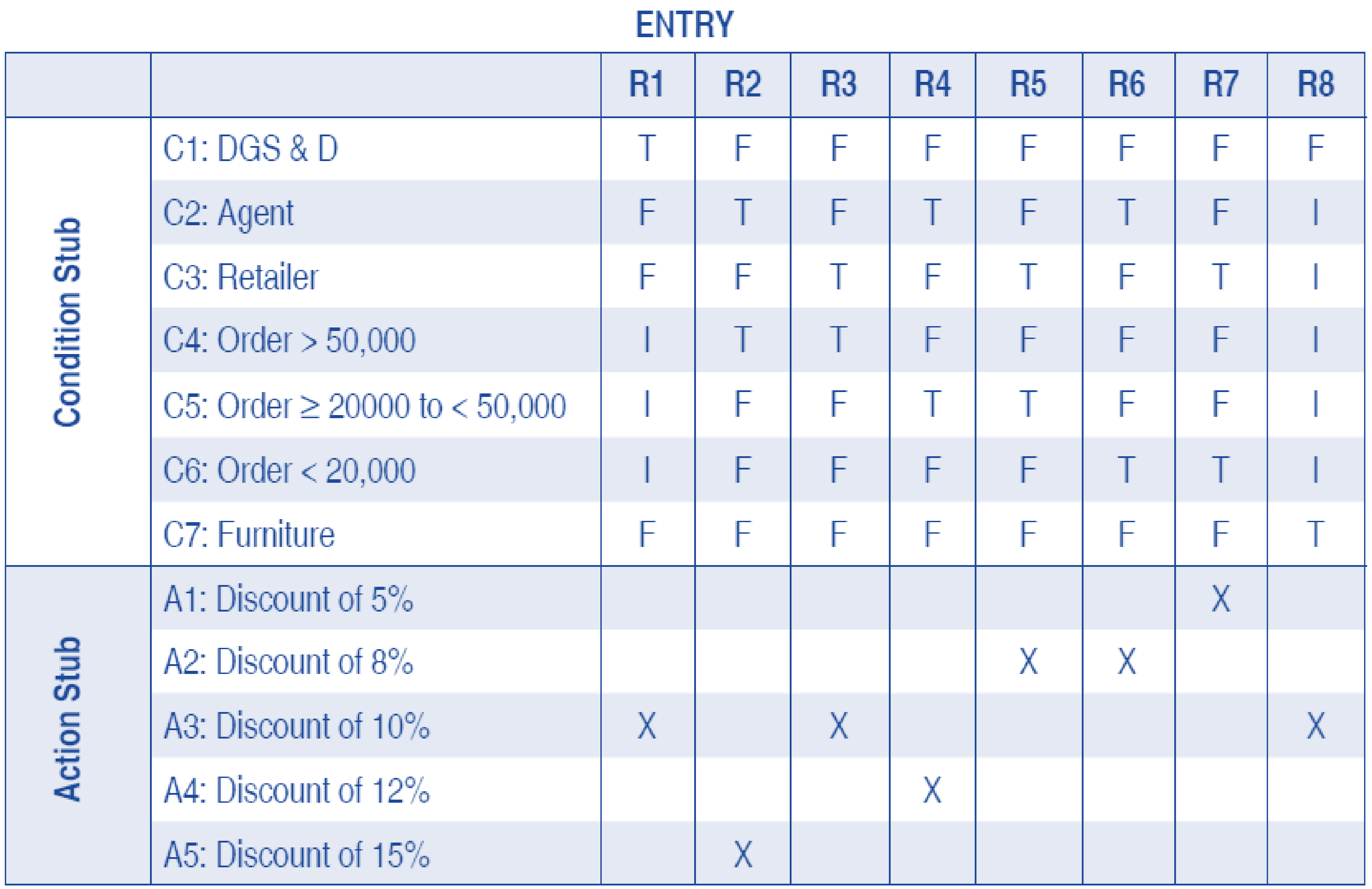
(iii) For orders of Rs 20,000 or more and up to Rs 50,000, agents get 12% and the retailer gets 8% discount.

(iv) For orders of less than Rs 20,000, agents get 8% and the retailer gets 5% discount.

The above rules do not apply to the furniture items wherein a flat rate of 10% discount is admissible to all customers irrespective of the value of the order.

Design test cases for this system using decision table testing.

Example 2: Solution

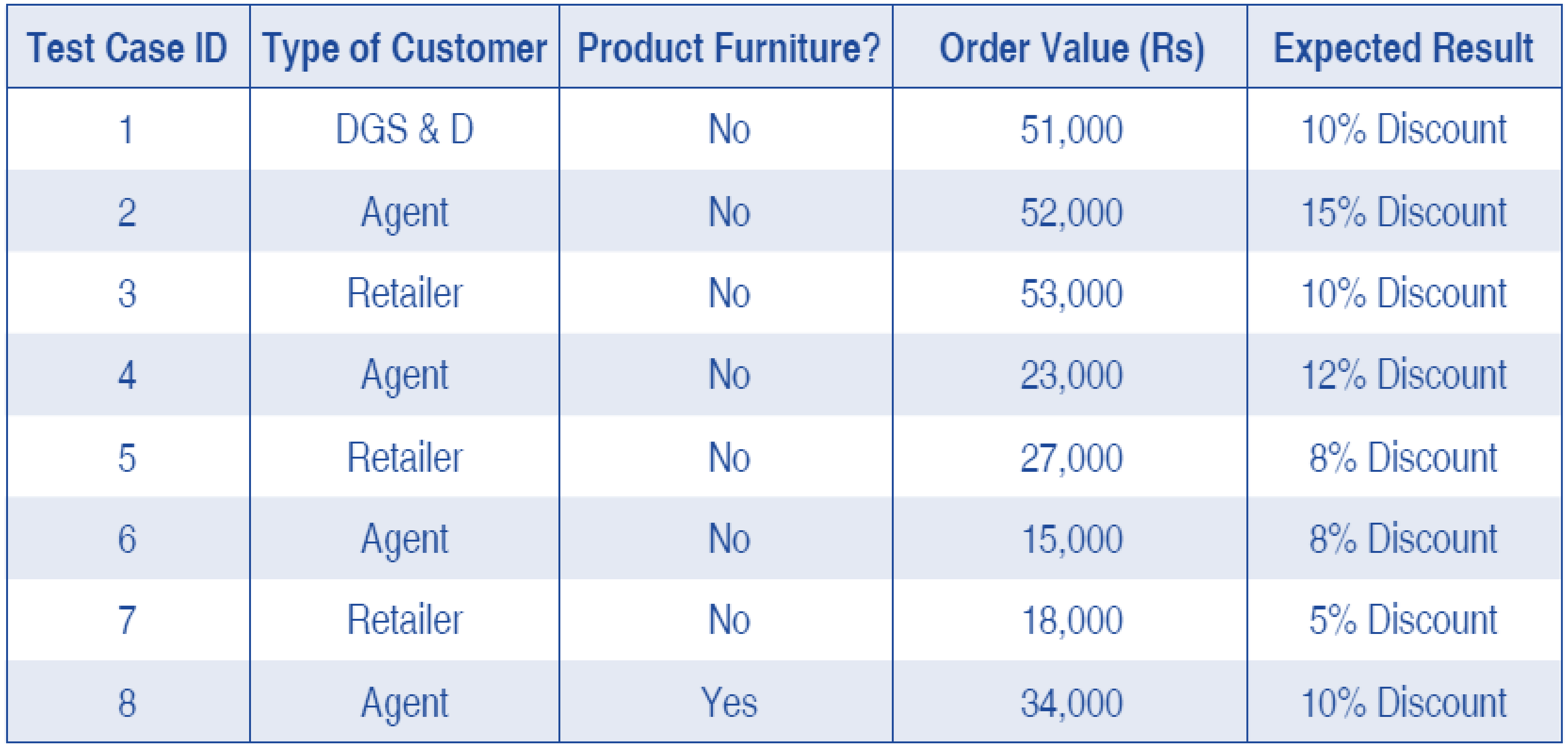


ENTRY

		R1	R2	R3	R4	R5	R6	R7	R8
Condition Stub	C1: DGS & D	T	F	F	F	F	F	F	F
	C2: Agent	F	T	F	T	F	T	F	I
	C3: Retailer	F	F	T	F	T	F	T	I
	C4: Order > 50,000	I	T	T	F	F	F	F	I
	C5: Order ≥ 20000 to < 50,000	I	F	F	T	T	F	F	I
	C6: Order < 20,000	I	F	F	F	F	T	T	I
	C7: Furniture	F	F	F	F	F	F	F	T
Action Stub	A1: Discount of 5%							X	
	A2: Discount of 8%					X	X		
	A3: Discount of 10%	X		X					X
	A4: Discount of 12%				X				
	A5: Discount of 15%		X						

Example 2: Solution

The test cases derived from the decision table are given below:



Test Case ID	Type of Customer	Product Furniture?	Order Value (Rs)	Expected Result
1	DGS & D	No	51,000	10% Discount
2	Agent	No	52,000	15% Discount
3	Retailer	No	53,000	10% Discount
4	Agent	No	23,000	12% Discount
5	Retailer	No	27,000	8% Discount
6	Agent	No	15,000	8% Discount
7	Retailer	No	18,000	5% Discount
8	Agent	Yes	34,000	10% Discount

Example 3

A university is admitting students in a professional course subject to the following conditions:

(a) Marks in Java ≥ 70

(b) Marks in C++ ≥ 60

(c) Marks in OOAD ≥ 60

(d) Total in all three subjects ≥ 220 OR Total in Java and C++ ≥ 150

If the aggregate mark of an eligible candidate is more than 240, he will be eligible for scholarship course, otherwise he will be eligible for normal course. The program reads the marks in the three subjects and generates the following outputs:

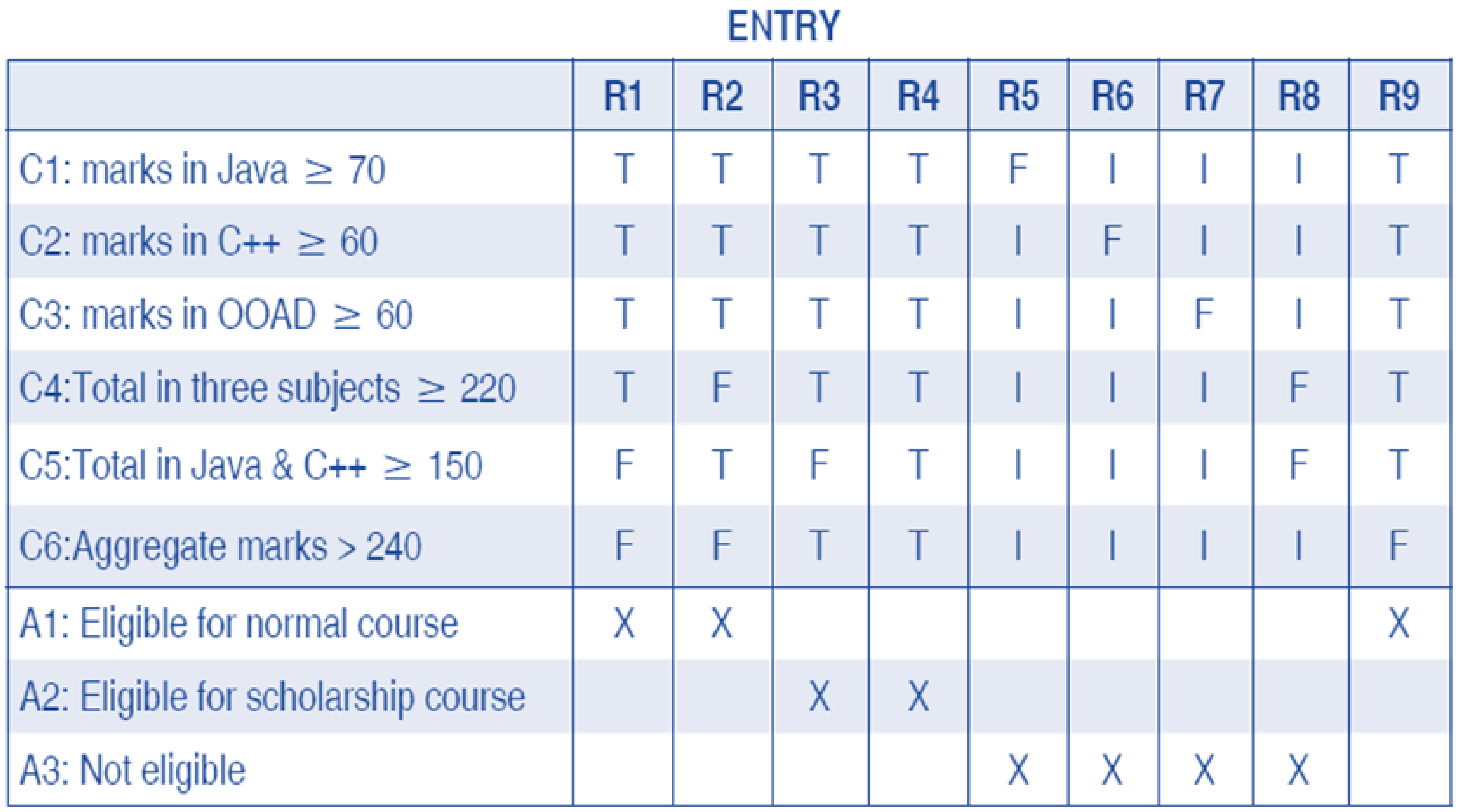
(i) Not eligible

(ii) Eligible for scholarship course

(iii) Eligible for normal course

Design test cases for this program using decision table testing.

Example 3: Solution



ENTRY

	R1	R2	R3	R4	R5	R6	R7	R8	R9
C1: marks in Java $\geq$ 70	T	T	T	T	F	I	I	I	T
C2: marks in C++ $\geq$ 60	T	T	T	T	I	F	I	I	T
C3: marks in OOAD $\geq$ 60	T	T	T	T	I	I	F	I	T
C4:Total in three subjects $\geq$ 220	T	F	T	T	I	I	I	F	T
C5:Total in Java & C++ $\geq$ 150	F	T	F	T	I	I	I	F	T
C6:Aggregate marks > 240	F	F	T	T	I	I	I	I	F
A1: Eligible for normal course	X	X							X
A2: Eligible for scholarship course			X	X					
A3: Not eligible					X	X	X	X	

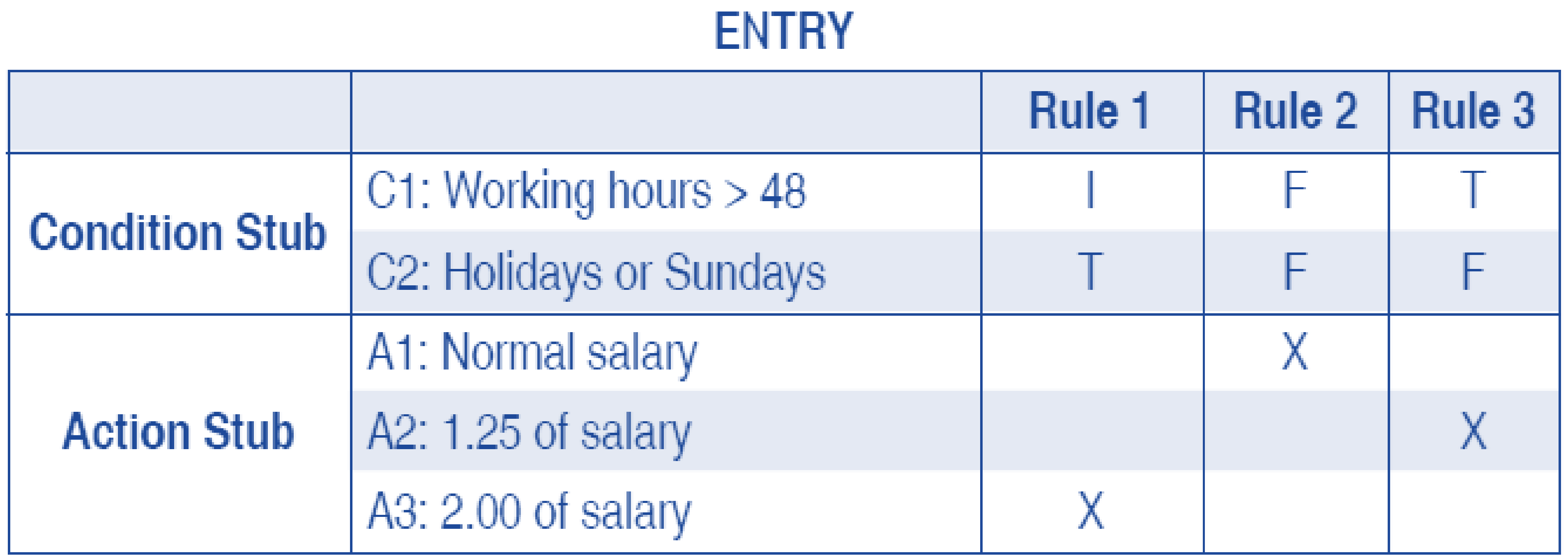
Example 3: Solution

The test cases derived from the decision table are given below:

Test Case ID	Java	C++	OOAD	Aggregate Marks	Expected Output
1	70	75	60	224	Eligible for normal course
2	75	75	70	220	Eligible for normal course
3	75	74	91	242	Eligible for scholarship course
4	76	77	89	242	Eligible for scholarship course
5	68	78	80	226	Not eligible
6	78	45	78	201	Not eligible
7	80	80	50	210	Not eligible
8	70	72	70	212	Not eligible
9	75	75	70	220	Eligible for normal course
10	76	80	85	241	Eligible for scholarship course

Example 4 (Immaterial Cases)

- Consider Example 1 once again whose decision table is shown below with immaterial cases.

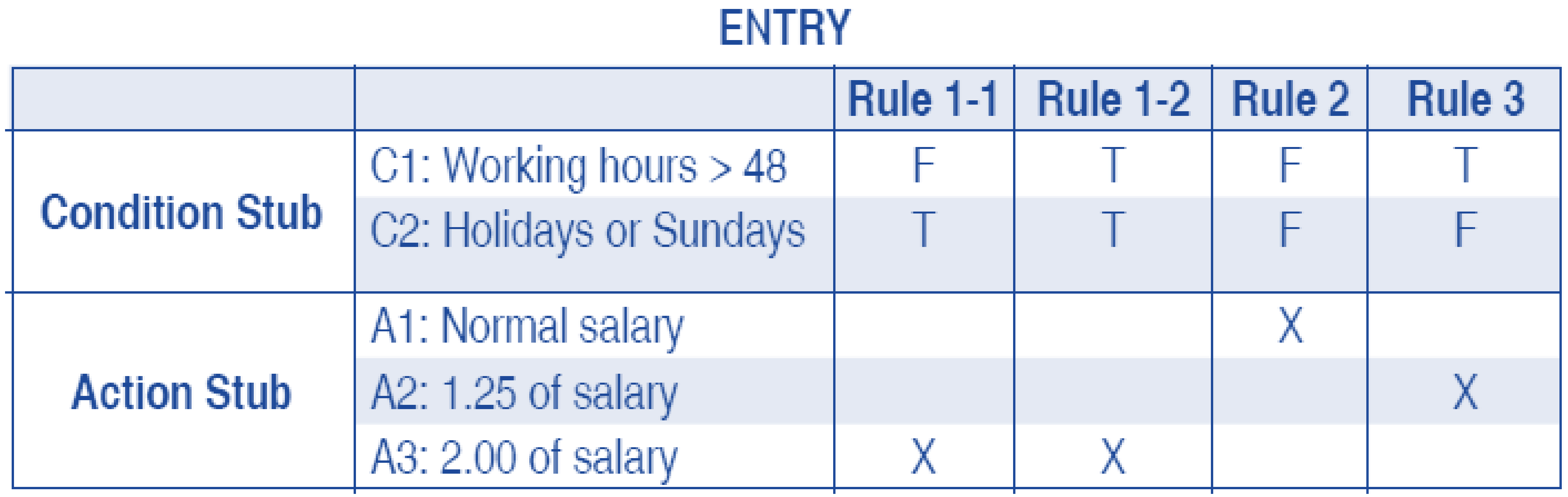


ENTRY

		Rule 1	Rule 2	Rule 3
Condition Stub	C1: Working hours > 48	I	F	T
	C2: Holidays or Sundays	T	F	F
Action Stub	A1: Normal salary		X	
	A2: 1.25 of salary			X
	A3: 2.00 of salary	X		

Example 4: Solution

- The immaterial test case in Rule 1 of the above table can be expanded by taking both T and F values of C1. The expanded decision table is shown below:

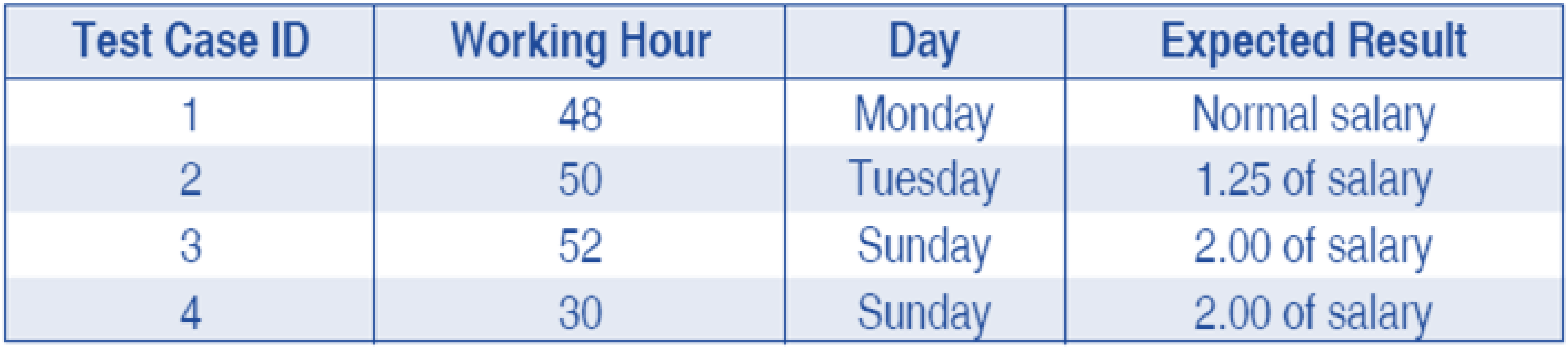


ENTRY

		Rule 1-1	Rule 1-2	Rule 2	Rule 3
Condition Stub	C1: Working hours > 48	F	T	F	T
	C2: Holidays or Sundays	T	T	F	F
Action Stub	A1: Normal salary			X	
	A2: 1.25 of salary				X
	A3: 2.00 of salary	X	X		

Example 4: Solution

The test cases derived from the decision table are given below:



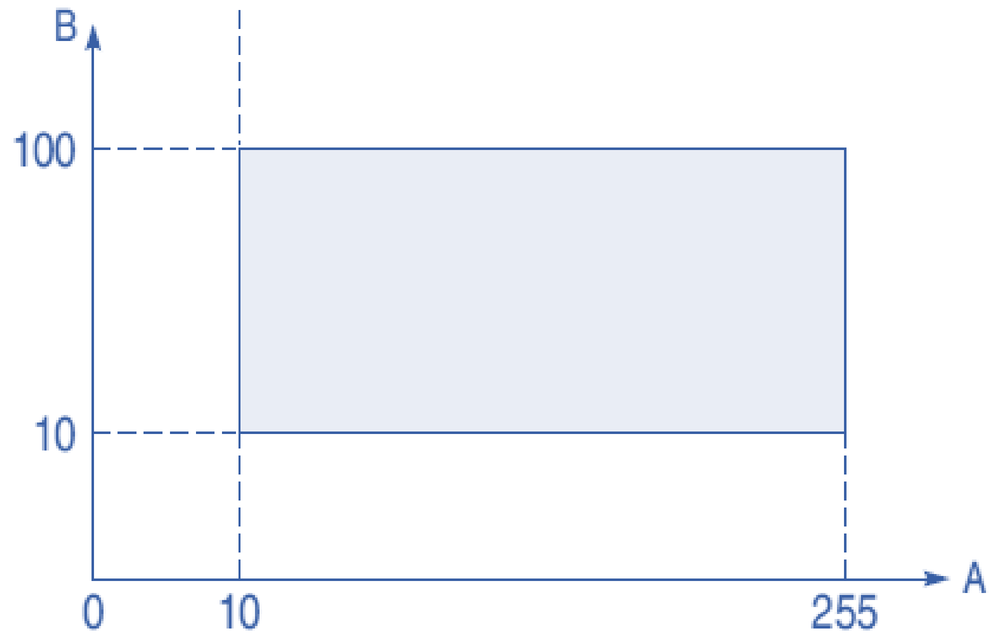
Test Case ID	Working Hour	Day	Expected Result
1	48	Monday	Normal salary
2	50	Tuesday	1.25 of salary
3	52	Sunday	2.00 of salary
4	30	Sunday	2.00 of salary

Boundary Value Analysis (BVA)

- It is considered a technique that uncovers the bugs at the boundary of input values. Here, boundary means the maximum or minimum value taken by the input domain.
- It an effective test case design requires test cases to be designed such that they maximize the probability of finding errors.
- With the experience of testing team, it has been observed that test cases designed with boundary input values have a high chance to find errors. It means that most of the failures crop up due to boundary values.

BVA: Example

If A is an integer between 10 and 255, then boundary checking can be on 10(9,10,11) and on 255(256,255,254). Similarly, B is another integer variable between 10 and 100, then boundary checking can be on 10(9,10,11) and 100(99,100,101)



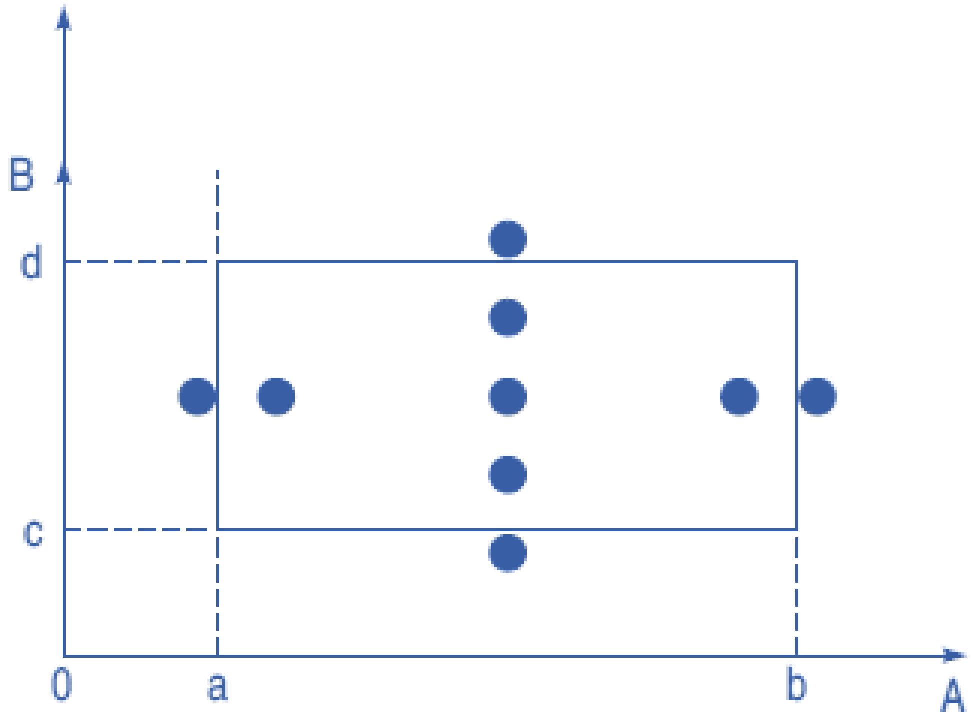
Boundary Value Checking (BVC)

- In this method, the test cases are designed by holding one variable at its extreme value and other variables at their nominal values in the input domain.
- The variable at its extreme value can be selected at:
    - Minimum value (Min)
    - Value just above the minimum value (Min+)
    - Maximum value (Max)
    - Value just below the maximum value (Max−)

BVC: Example

Let us take the example of two variables, A and B. If we consider all the above combinations with nominal values, then following test cases can be designed:

1. A_{nom}, B_{min}
2. A_{nom}, B_{min+}
3. A_{nom}, B_{max}
4. A_{nom}, B_{max-}
5. A_{min}, B_{nom}
6. A_{min+}, B_{nom}
7. A_{max}, B_{nom}
8. A_{max-}, B_{nom}
9. A_{nom}, B_{nom}



Robustness Testing Method

- The idea of BVC can be extended such that boundary values are exceeded as:
  - A value just greater than the Maximum value (Max+)
  - A value just less than Minimum value (Min−)
- When test cases are designed considering the above points in addition to BVC, it is called robustness testing.

Robustness Testing Method: Example

- Let us take the previous example again. Add the following test cases to the list of 9 test cases designed in BVC:

10. A_{max+}, B_{nom}	11. A_{min-}, B_{nom}
12. A_{nom}, B_{max+}	13. A_{nom}, B_{min-}

- It can be generalized that for **n** input variables in a module, **6n + 1** test cases an be designed with robustness testing.

Worst-case Testing Method

- We can again extend the concept of BVC by assuming more than one variable on the boundary. It is called worst-case testing method.
- following test cases to the list of 9 test cases designed in BVC as:

10. A_{min}, B_{min}
11. $A_{min\ +}$, B_{min}
12. A_{min}, B_{min+}
13. $A_{min\ +}$, B_{min+}
14. A_{max}, B_{min}
15. A_{max-}, B_{min}
16. A_{max}, B_{min+}
17. A_{max-}, B_{min+}
18. A_{min}, B_{max}
19. A_{min+}, B_{max}
20. A_{min}, B_{max-}
21. A_{min+}, B_{max-}
22. A_{max}, B_{max}
23. A_{max-}, B_{max}
24. A_{max}, B_{max-}
25. A_{max-}, B_{max-}

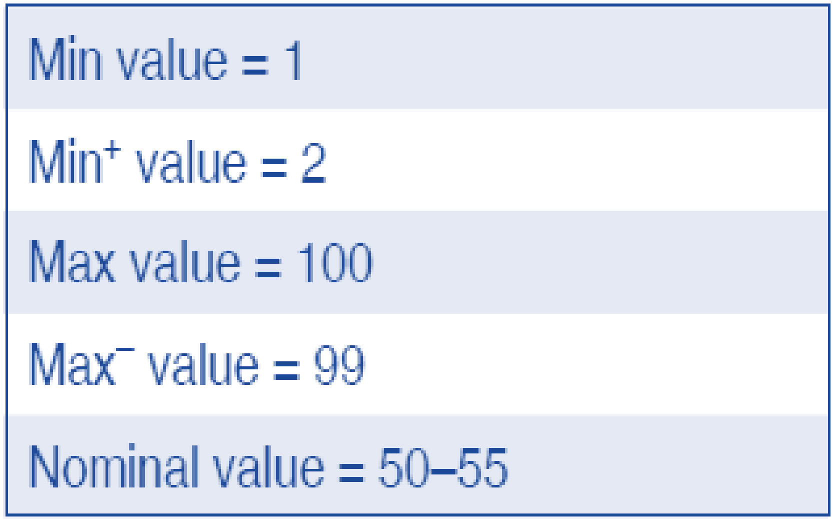
- It can be generalized that for n input variables in a module, 5n test cases can be designed with worst-case testing.

Example 1

A program reads an integer number within the range [1,100] and determines whether it is a prime number or not. Design test cases for this program using BVC, robust testing, and worst-case testing methods.

Example 1 : Solution

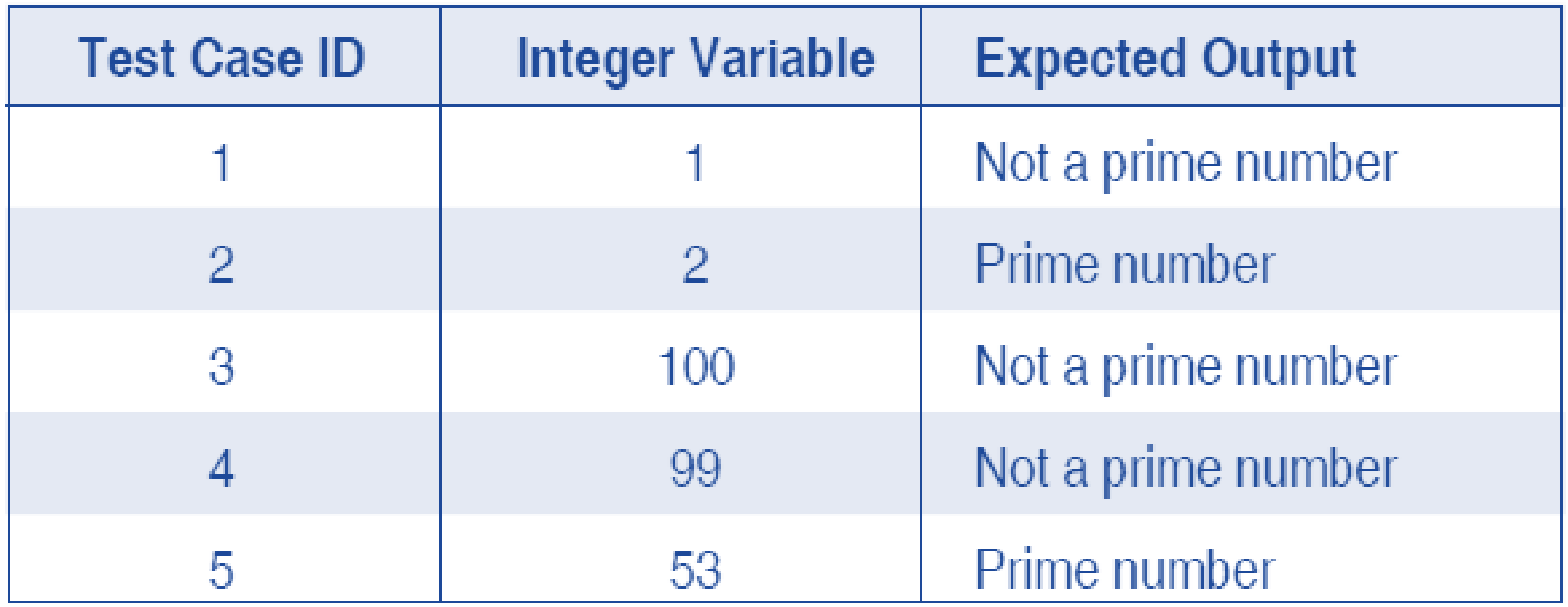
- **Test cases using BVC:** Since there is one variable, the total number of test cases will be 4n + 1 = 5. In our example, the set of minimum and maximum values is shown below:



Min value = 1
Min$^+$ value = 2
Max value = 100
Max$^-$ value = 99
Nominal value = 50–55

Example 1 : Solution

Using these values, test cases can be designed as shown below:



Test Case ID	Integer Variable	Expected Output
1	1	Not a prime number
2	2	Prime number
3	100	Not a prime number
4	99	Not a prime number
5	53	Prime number

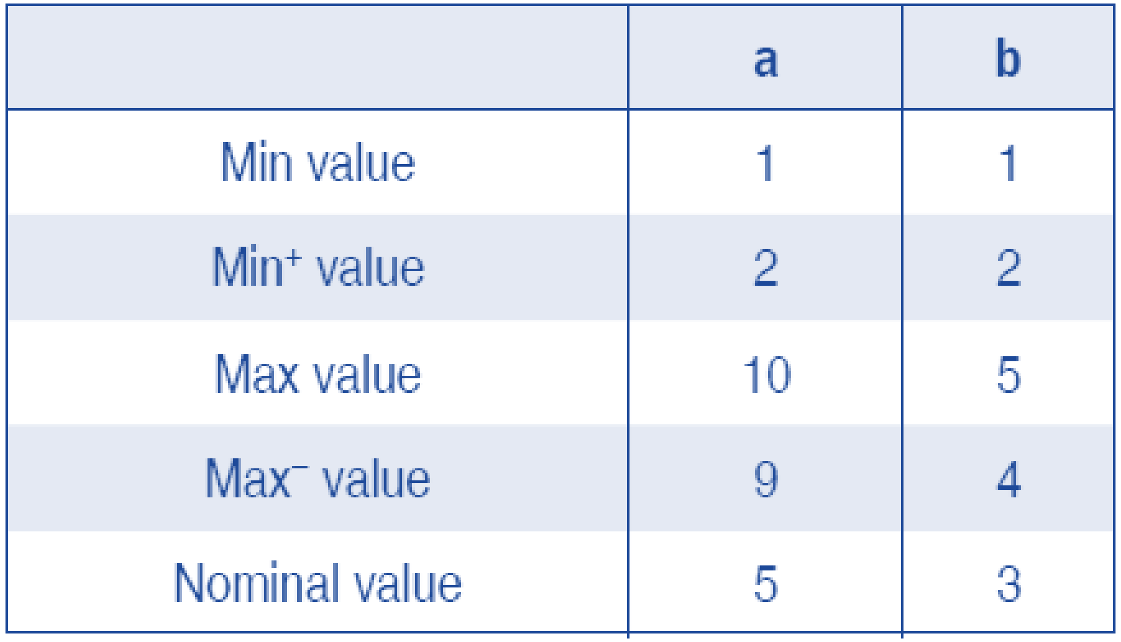
Example 2

A program computes $\mathbf{a}^{\mathbf{b}}$ where a lies in the range [1,10] and b within [1,5]. Design test cases for this program using BVC, robust testing, and worst-case testing methods.

Example 2 : Solution

- **Test cases using BVC:**

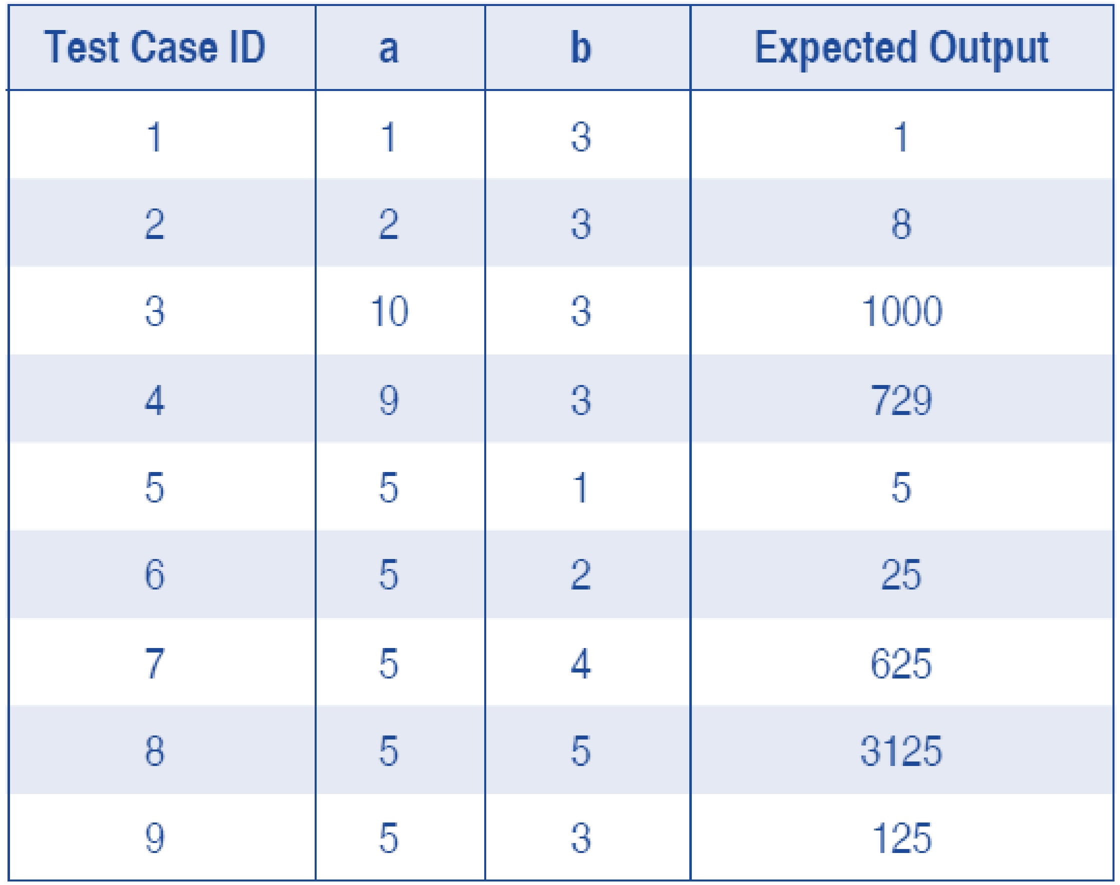
Since there are two variables, a and b, the total number of test cases will be 4n + 1 = 9. The set of boundary values is shown below:



	a	b
Min value	1	1
Min^+ value	2	2
Max value	10	5
Max^- value	9	4
Nominal value	5	3

Example 2 : Solution

Using these values, test cases can be designed as shown below:



Test Case ID	a	b	Expected Output
1	1	3	1
2	2	3	8
3	10	3	1000
4	9	3	729
5	5	1	5
6	5	2	25
7	5	4	625
8	5	5	3125
9	5	3	125

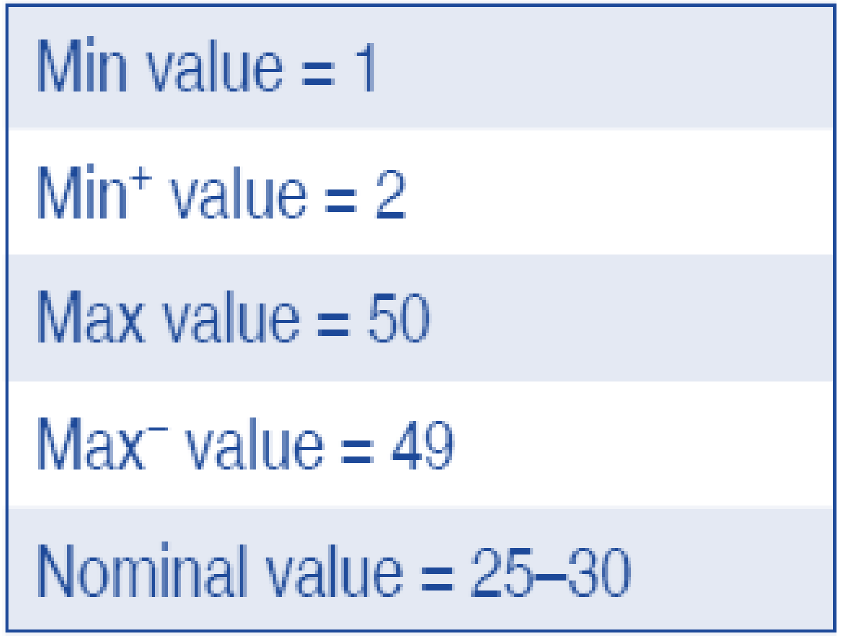
Example 3

A program reads three numbers, A, B, and C, within the range [1, 50] and prints the largest number. Design test cases for this program using BVC, robust testing, and worst-case testing methods.

Example 3 : Solution

- **Test cases using BVC:**

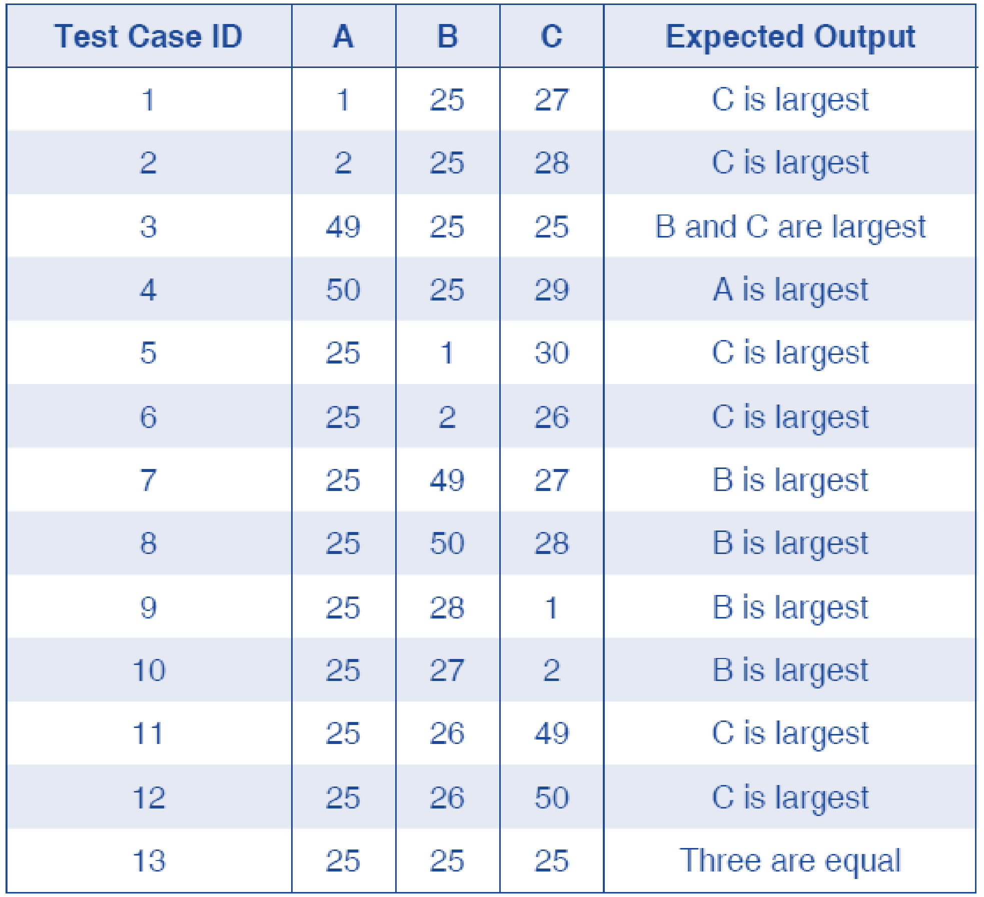
Since there is one variable, the total number of test cases will be 4n + 1 = 5. In our example, the set of minimum and maximum values is shown below:



Min value = 1
Min$^+$ value = 2
Max value = 50
Max$^-$ value = 49
Nominal value = 25–30

Example 3 : Solution

Using these values, test cases can be designed as shown below:



Test Case ID	A	B	C	Expected Output
1	1	25	27	C is largest
2	2	25	28	C is largest
3	49	25	25	B and C are largest
4	50	25	29	A is largest
5	25	1	30	C is largest
6	25	2	26	C is largest
7	25	49	27	B is largest
8	25	50	28	B is largest
9	25	28	1	B is largest
10	25	27	2	B is largest
11	25	26	49	C is largest
12	25	26	50	C is largest
13	25	25	25	Three are equal