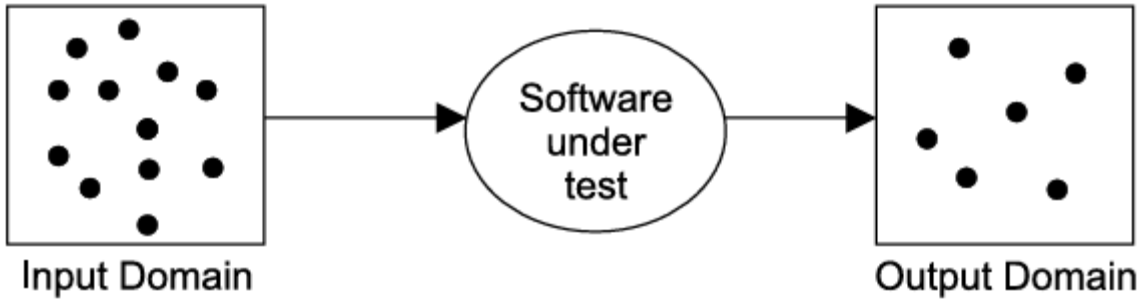
BlackBox Testig

1. Introduction to Functional Testing

- Functional testing is a **black-box testing** technique. This means the tester has no knowledge of the internal code or design of the system. The focus is solely on what the system is supposed to do, based on the requirements and specifications.
- **Key Characteristics:**
- **Focus on functionality:** Does the system meet the specified business and user requirements?
- **External perspective:** It's performed from the user's point of view.
- **Driven by requirements:** Test cases are derived directly from the functional requirements document.

1. Introduction to Functional Testing

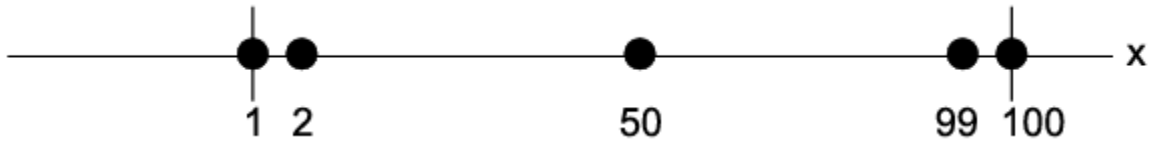
- Functional testing techniques attempt to design those test cases which have a higher probability of making a software fail.
- These techniques also attempt to test every possible functionality of the software.
- Many activities are performed in real life with only black box knowledge like driving a car, using a cell phone, operating a computer
- These techniques can be used at all levels of software testing like unit, integration, system and acceptance testing.



2.1 BOUNDARY VALUE ANALYSIS

- Here, we concentrate on input values and design test cases with input values that are on or close to boundary values
- Experience has shown that such test cases have a higher probability of detecting a fault in the software.

(i) Minimum value
(ii) Just above minimum value
(iii) Maximum value
(iv) Just below maximum value
(v) Nominal (Average) value



2.1 BOUNDARY VALUE ANALYSIS

- The number of inputs selected by this technique is 4n + 1 where ‘n’ is the number of inputs.

Table 2.1. Test cases for the ‘Square’ program

Test Case	Input x	Expected output
1.	1	1
2.	2	4
3.	50	2500
4.	99	9801
5.	100	10000

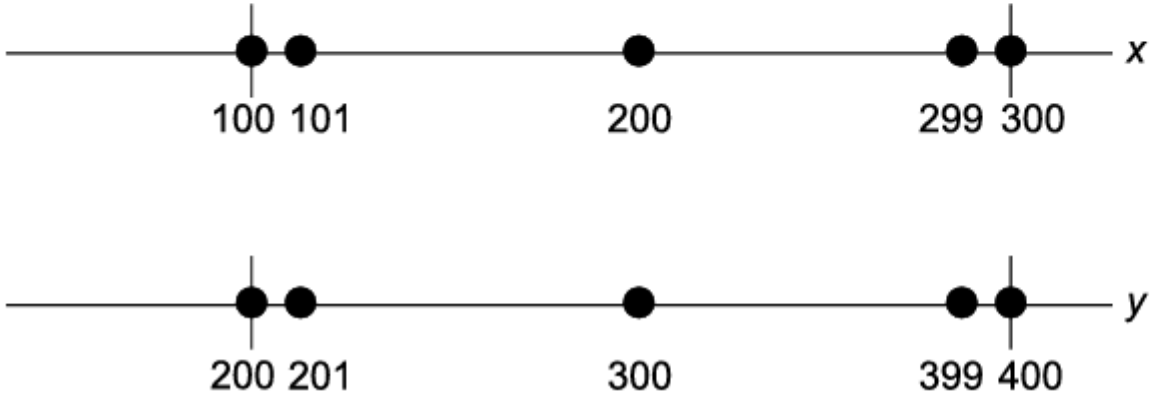
2.1 BOUNDARY VALUE ANALYSIS

Consider a program 'Addition' with two input values x and y and it gives the addition of x and y as an output. The range of both input values are given as:

$$100 \leq x \leq 300$$

$$200 \leq y \leq 400$$

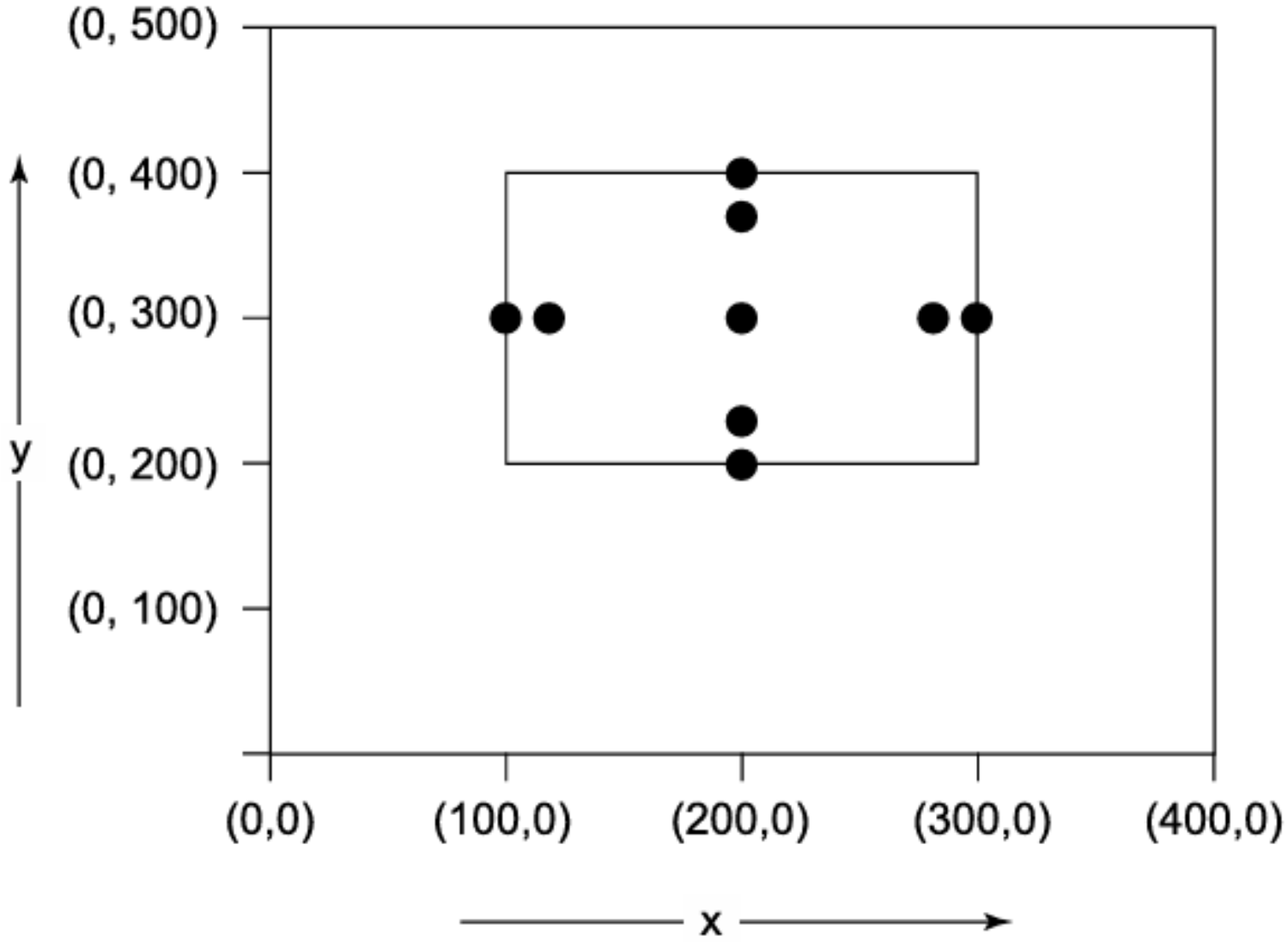
The selected values for *x* and *y* are given in Figure 2.3.



2.1 BOUNDARY VALUE ANALYSIS

- We also consider ‘single fault’ assumption theory of reliability which says that failures are rarely the result of the simultaneous occurrence of two (or more) faults.
- Normally, one fault is responsible for one failure. With this theory in mind, we select one input value on
  - boundary (minimum)
  - just above boundary (minimum +)
  - just below boundary (maximum –)
  - on boundary (maximum)
  - nominal (average)

2.1 BOUNDARY VALUE ANALYSIS



2.1 BOUNDARY VALUE ANALYSIS

Table 2.2. Test cases for the program 'Addition'

Test Case	x	y	Expected Output
1.	100	300	400
2.	101	300	401
3.	200	300	500
4.	299	300	599
5.	300	300	600
6.	200	200	400
7.	200	201	401
8.	200	300	500
9.	200	399	599
10.	200	400	600

2.1 BOUNDARY VALUE ANALYSIS

- Consider a program for the determination of the largest amongst three numbers. Its input is a triple of positive integers (say x,y and z) and values are from interval [1, 300].

Table 2.3. Boundary value test cases to find the largest among three numbers

Test Case	x	y	z	Expected output
1.	1	150	150	150
2.	2	150	150	150
3.	150	150	150	150
4.	299	150	150	299
5.	300	150	150	300
6.	150	1	150	150
7.	150	2	150	150
8.	150	299	150	299
9.	150	300	150	300
10.	150	150	1	150
11.	150	150	2	150
12.	150	150	299	299
13.	150	150	300	300

2.1 BOUNDARY VALUE ANALYSIS

Example 2.2: Consider a program for the determination of division of a student based on the marks in three subjects. Its input is a triple of positive integers (say mark1, mark2, and mark3) and values are from interval [0, 100].
The division is calculated according to the following rules:

Marks Obtained (Average)	Division
75 – 100	First Division with distinction
60 – 74	First division
50 – 59	Second division
40 – 49	Third division
0 – 39	Fail

Total marks obtained are the average of marks obtained in the three subjects i.e.

$$\text{Average} = (\text{mark1} + \text{mark 2} + \text{mark3}) / 3$$

The program output may have one of the following words:
[Fail, Third Division, Second Division, First Division, First Division with Distinction]
Design the boundary value test cases.

2.1 BOUNDARY VALUE ANALYSIS

Table 2.4. Boundary value test cases for the program determining the division of a student

Test Case	mark1	mark2	mark3	Expected Output
1.	0	50	50	Fail
2.	1	50	50	Fail
3.	50	50	50	Second Division
4.	99	50	50	First Division
5.	100	50	50	First Division

Test Case	mark1	mark2	mark3	Expected Output
6.	50	0	50	Fail
7.	50	1	50	Fail
8.	50	99	50	First Division
9.	50	100	50	First Division
10.	50	50	0	Fail
11.	50	50	1	Fail
12.	50	50	99	First Division
13.	50	50	100	First Division

2.1.1 ROBUSTNESS TESTING

- This is the extension of boundary value analysis.
- Here, we also select invalid values and see the responses of the program.
- Hence, two additional states are added i.e. just below minimum value (minimum value–) and just above maximum value (maximum value +).
- Thus, the total test cases in robustness testing are 6n + 1, where ‘n’ is the number of input values.

(i) Minimum value
(ii) Just above minimum value
(iii) Just below minimum value
(iv) Just above maximum value
(v) Just below maximum value
(vi) Maximum value
(vii) Nominal (Average) value

2.1.1 ROBUSTNESS TESTING

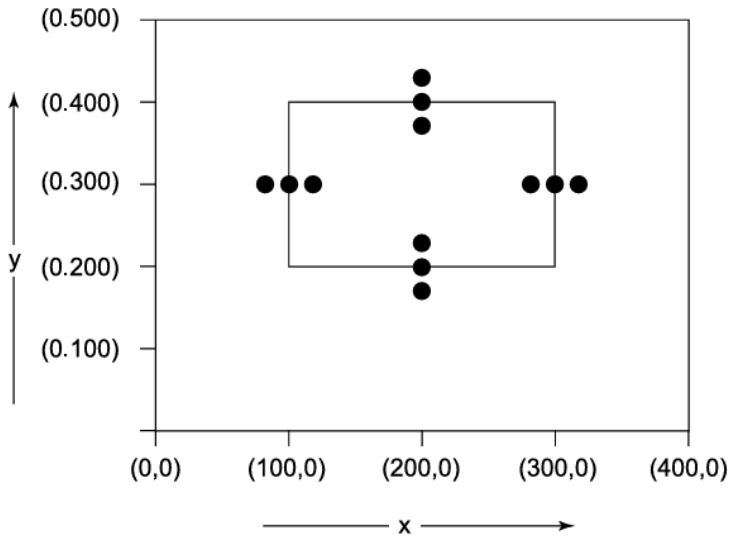


Table 2.7. Robustness test cases for two input values x and y

Test Case	x	y	Expected Output
1.	99	300	Invalid Input
2.	100	300	400
3.	101	300	401
4.	200	300	500
5.	299	300	599
6.	300	300	600
7.	301	300	Invalid Input
8.	200	199	Invalid Input
9.	200	200	400
10.	200	201	401
11.	200	399	599
12.	200	400	600
13.	200	401	Invalid Input

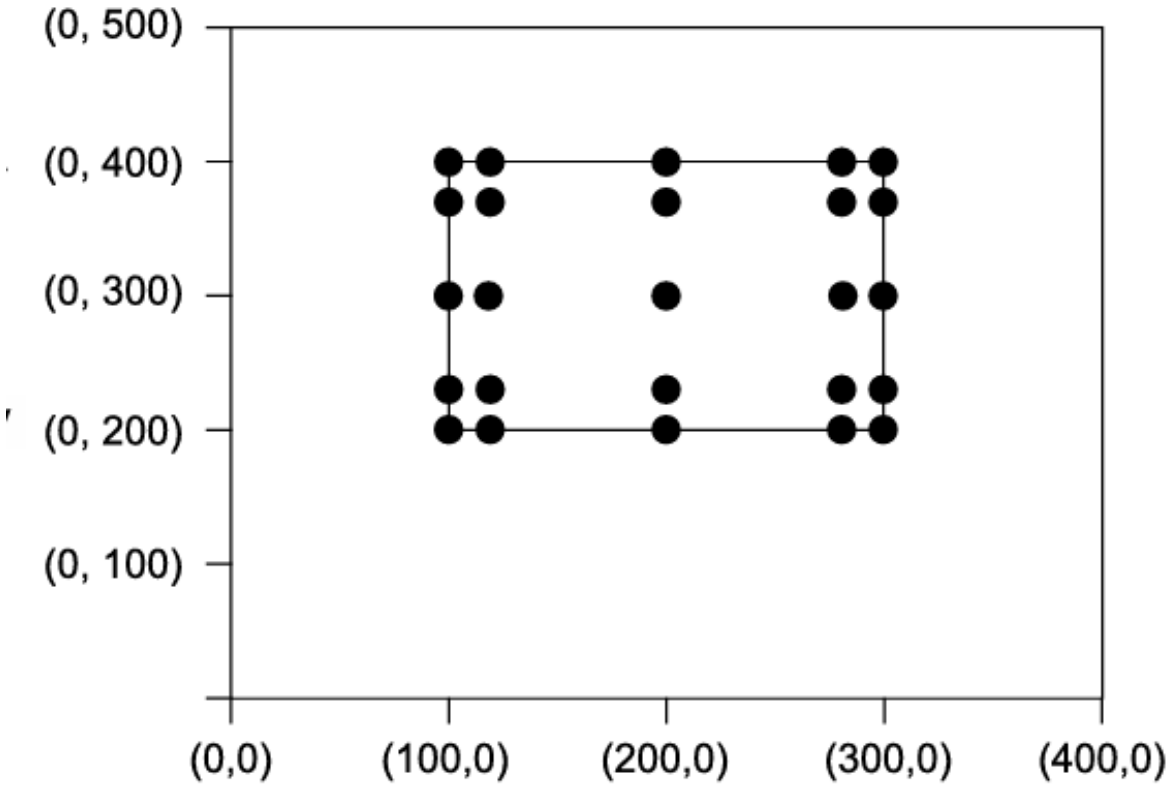
2.1.2 Worst-Case Testing

- This is a special form of boundary value analysis where we don't consider the 'single fault' assumption theory of reliability.
- Now, failures are also due to occurrence of more than one fault simultaneously.
- The restriction of one input value at any of the above mentioned values and other input values must be at nominal is not valid in worst-case testing.
- This will increase the number of test cases from 4n + 1 test cases to 5^n test cases, where 'n' is the number of input values.

(i) Minimum value
(ii) Just above minimum value
(iii) Just below maximum value
(iv) Maximum value
(v) Nominal (Average) value

2.1.2 Worst-Case Testing

- The program ‘Addition’ will have 5^2 = 25 test cases and these test cases are given in Table 2.8



2.1.2 Worst-Case Testing

- The program ‘Addition’ will have 5^2 = 25 test cases and these test cases are given in Table 2.8

Table 2.8. Worst test cases for the program ‘Addition’

Test Case	x	y	Expec
1.	100	200	300
2.	100	201	301
3.	100	300	400
4.	100	399	499
5.	100	400	500
6.	101	200	301
7.	101	201	302
8.	101	300	401
9.	101	399	500
10.	101	400	501
11.	200	200	400
12.	200	201	401
13.	200	300	500
14.	200	399	599

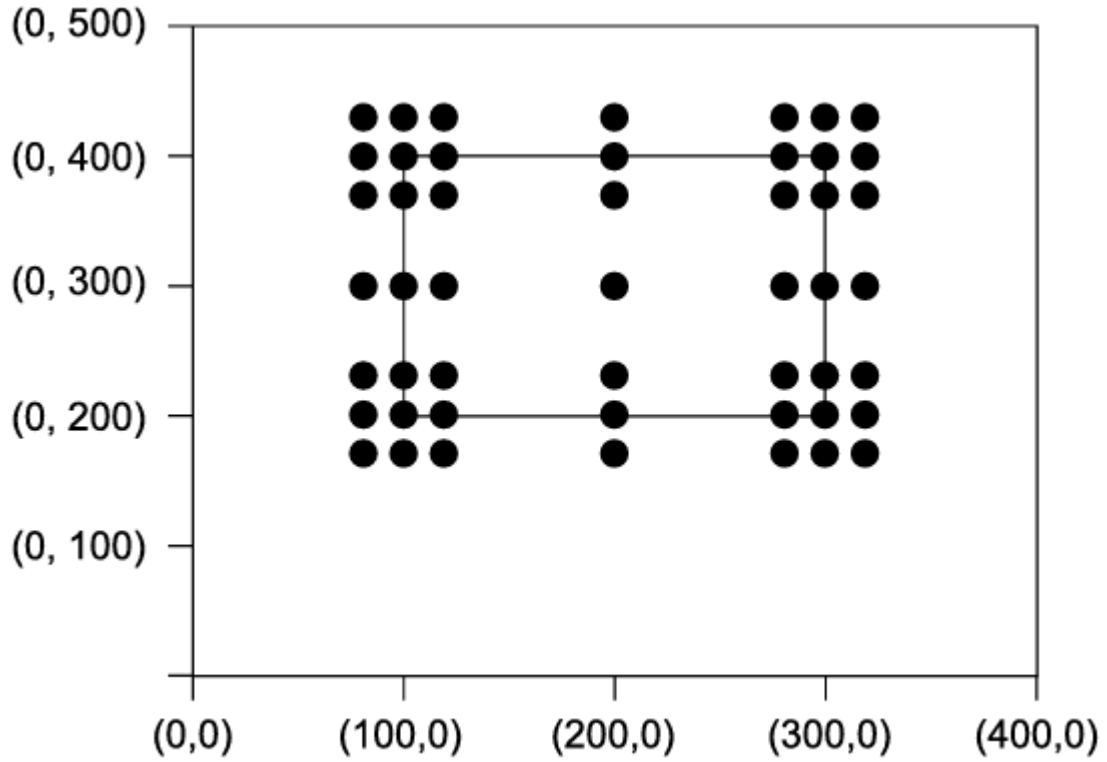
Test Case	x	y
15.	200	4
16.	299	2
17.	299	2
18.	299	3
19.	299	3
20.	299	4
21.	300	2
22.	300	2
23.	300	3
24.	300	3
25.	300	4

2.1.3 Robust Worst-Case Testing

- In robustness testing, we add two more states i.e. just below minimum value (minimum value–) and just above maximum value (maximum value+). We also give invalid inputs and observe the behaviour of the program.
- There are seven states (minimum -, minimum, minimum +, nominal, maximum –, maximum, maximum +)
- total of 7^n test cases will be generated

2.1.3 Robust Worst-Case Testing

- The inputs for the program ‘Addition’ are graphically shown in Figure 2.8. The program ‘Addition’ will have 7^2 = 49 test cases and these test cases are shown in Table 2.9



2.1.3 Robust Worst-Case Testing

Test Case	x	y	Expected Ou
1.	99	199	Invalid input
2.	99	200	Invalid input
3.	99	201	Invalid input
4.	99	300	Invalid input
5.	99	399	Invalid input
6.	99	400	Invalid input
7.	99	401	Invalid input
8.	100	199	Invalid input
9.	100	200	300
10.	100	201	301
11.	100	300	400
12.	100	399	499
13.	100	400	500
14.	100	401	Invalid input
15.	101	199	Invalid input
16.	101	200	301
17.	101	201	302
18.	101	300	401
19.	101	399	500
20.	101	400	501
21.	101	401	Invalid input
22.	200	199	Invalid input
23.	200	200	400
24.	200	201	401
25.	200	300	500
26.	200	399	599
27.	200	400	600
28.	200	401	Invali
29.	299	199	Invali
30.	299	200	499
31.	299	201	500
32.	299	300	599
33.	299	399	698
34.	299	400	699
35.	299	401	Invali
36.	300	199	Invali
37.	300	200	500
38.	300	201	501
39.	300	300	600
40.	300	399	699
41.	300	400	700
42.	300	401	Invali
43.	301	199	Invali
44.	301	200	Invali
45.	301	201	Invali
46.	301	300	Invali
47.	301	399	Invali
48.	301	400	Invali
49.	301	401	Invali

2.1.4 Applicability

- Boundary value analysis is a simple technique and may prove to be effective when used correctly.
- Here, input values should be independent which restricts its applicability in many programs.
- This technique does not make sense for Boolean variables where input values are TRUE and FALSE only, and no choice is available for nominal values, just above boundary values, just below boundary values, etc.
- This technique can significantly reduce the number of test cases and is suited to programs in which input values are within ranges or within sets.
- This is equally applicable at the unit, integration, system and acceptance test levels. All we want is input values where boundaries can be identified from the requirements.

2. Equivalence Partitioning

This technique divides the input data of a program into logical "equivalence classes" or partitions. The idea is that if a test case works for one value in a partition, it will work for all values in that same partition. This helps reduce the total number of test cases.

- **Valid Equivalence Classes:** Inputs that are considered valid and accepted by the system.
- **Invalid Equivalence Classes:** Inputs that are considered invalid and should be rejected.

Example: A textbox for age (18-60)

- **Valid Class:** Numbers between 18 and 60 (e.g., 25, 40).
- **Invalid Class 1:** Numbers less than 18 (e.g., 17, 0).
- **Invalid Class 2:** Numbers greater than 60 (e.g., 61, 100).
- **Invalid Class 3:** Non-numeric values (e.g., "abc", "@").

2.2.1 Creation of Equivalence Classes

- The entire input domain can be divided into at least two equivalence classes: one containing all valid inputs and the other containing all invalid inputs.
- Each equivalence class can further be sub-divided into equivalence classes on which the program is required to behave differently.
- The input domain equivalence classes for the program ‘Square’ which takes ‘x’ as an input (range 1-100) and prints the square of ‘x’ (seen in Figure 2.2) are given as:

(i) $I_1 = \{ 1 \leq x \leq 100 \}$ (Valid input range from 1 to 100)
(ii) $I_2 = \{ x < 1 \}$ (Any invalid input where x is less than 1)
(iii) $I_3 = \{ x > 100 \}$ (Any invalid input where x is greater than 100)

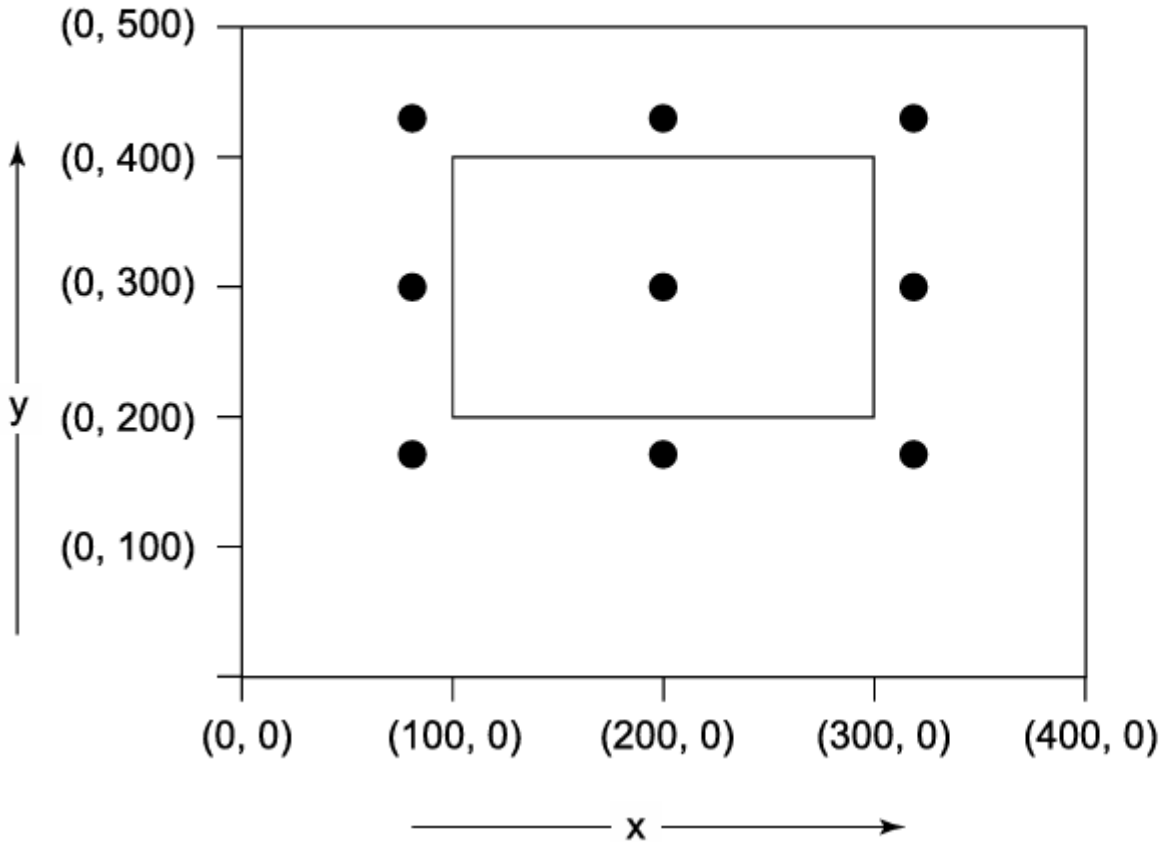
2.2.1 Creation of Equivalence Classes

The following equivalence classes can be generated for program 'Addition' for input domain:

(i) $I_1 = \{ 100 \le x \le 300 \text{ and } 200 \le y \le 400 \}$ (Both x and y are valid values)
(ii) $I_2 = \{ 100 \le x \le 300 \text{ and } y < 200 \}$ (x is valid and y is invalid)
(iii) $I_3 = \{ 100 \le x \le 300 \text{ and } y > 400 \}$ (x is valid and y is invalid)
(iv) $I_4 = \{ x < 100 \text{ and } 200 \le y \le 400 \}$ (x is invalid and y is valid)
(v) $I_5 = \{ x > 300 \text{ and } 200 \le y \le 400 \}$ (x is invalid and y is valid)
(vi) $I_6 = \{ x < 100 \text{ and } y < 200 \}$ (Both inputs are invalid)
(vii) $I_7 = \{ x < 100 \text{ and } y > 400 \}$ (Both inputs are invalid)
(viii) $I_8 = \{ x > 300 \text{ and } y < 200 \}$ (Both inputs are invalid)
(ix) $I_9 = \{ x > 300 \text{ and } y > 400 \}$ (Both inputs are invalid)

2.2.1 Creation of Equivalence Classes

The graphical representation of inputs is shown in Figure 2.9 and the test cases are given in Table 2.19.



2.2.1 Creation of Equivalence Classes

Table 2.19. Test cases for the program 'Addition'

Test Case	x	y	Expected Output
I_1	200	300	500
I_2	200	199	Invalid input
I^3	200	401	Invalid input
I_4	99	300	Invalid input
I_5	301	300	Invalid input
I_6	99	199	Invalid input
I_7	99	401	Invalid input
I_8	301	199	Invalid input
I_9	301	401	Invalid input

2.2.1 Creation of Equivalence Classes

- From output domain

$O_1 =$ {square of the input number 'x'}

$O_2 =$ {Invalid input)

Table 2.20. Test cases for program 'Square' based on output domain

Test Case	Input x	Expected Output
O_1	50	2500
O_2	0	Invalid Input

$O_1 =$ { Addition of two input numbers x and y }

$O_2 =$ {Invalid Input}

Table 2.21. Test cases for program 'Addition' based on output domain

Test Case	x	y	Expected Output
O_1	200	300	500
O_2	99	300	Invalid Input

2.2.2 Applicability

- It is applicable at unit, integration, system and acceptance test levels.

Example 2.9: Consider the program for determination of the largest amongst three numbers specified in example 2.1. Identify the equivalence class test cases for output and input domain.

Solution: Output domain equivalence classes are:

O_1= {<x, y, z > : Largest amongst three numbers x, y, z }

O_2= {<x, y, z > : Input values(s) is /are out of range with sides x, y, z }

The test cases are given in Table 2.22.

Table 2.22. Output domain test cases to find the largest among three numbers

Test Case	x	y	z	Expected Output
O_1	150	140	110	150
O_2	301	50	50	Input values are out of range

I_1 = { $1 \leq x \leq 300$ and $1 \leq y \leq 300$ and $1 \leq z \leq 300$ } (All inputs are valid)

I_2 = { $x < 1$ and $1 \leq y \leq 300$ and $1 \leq z \leq 300$ } (x is invalid , y is valid and z is valid)

I_3 = { $1 \leq x \leq 300$ and $y < 1$ and $1 \leq z \leq 300$ } (x is valid, y is invalid and z is valid)

I_4 = { $1 \leq x \leq 300$ and $1 \leq y \leq 300$ and $z < 1$ } (x is valid, y is valid and z is invalid)

I_5 = { $x > 300$ and $1 \leq y \leq 300$ and $1 \leq z \leq 300$ } (x is invalid, y is valid and z is valid)

I_6 = { $1 \leq x \leq 300$ and $y > 300$ and $1 \leq z \leq 300$ } (x is valid, y is invalid and z is valid)

I_7 = { $1 \leq x \leq 300$ and $1 \leq y \leq 300$ and $z > 300$ } (x is valid, y is valid and z is invalid)

I_8 = { $x < 1$ and $y < 1$ and $1 \leq z \leq 300$ } (x is invalid, y is invalid and z is valid)

I_9 = { $1 \leq x \leq 300$ and $y < 1$ and $z < 1$ } (x is valid, y is invalid and z is invalid)

I_{10} = { $x < 1$ and $1 \leq y \leq 300$ and $z < 1$ } (x is invalid, y is valid and z is invalid)

I_{11} = { $x > 300$ and $y > 300$ and $1 \leq z \leq 300$ } (x is invalid, y is invalid and z is valid)

I_{12} = { $1 \leq x \leq 300$ and $y > 300$ and $z > 300$ }(x is valid, y is invalid and z is invalid)

I_{13} = { $x > 300$ and $1 \leq y \leq 300$ and $z > 300$ } (x is invalid, y is valid and z is invalid)

I_{14} = { $x < 1$ and $y > 300$ and $1 \leq z \leq 300$ } (x is invalid, y is invalid and z is valid)

I_{15} = { $x > 300$ and $y < 1$ and $1 \leq z \leq 300$ } (x is invalid, y is invalid and z is valid)

I_{16} = {$1 \leq x \leq 300$ and $y < 1$ and $z > 300$ } (x is valid, y is invalid and z is invalid)

2.2.2 Applicability

I_{17} = { $1 \leq x \leq 300$ and $y > 300$ and $z < 1$ } (x is valid, y is invalid and z is invalid)

I_{18} = { $x < 1$ and $1 \leq y \leq 300$ and $z > 300$ } (x is invalid, y is valid and z is invalid)

I_{19} = { $x > 300$ and $1 \leq y \leq 300$ and $z < 1$ } (x is invalid, y is valid and z is invalid)

I_{20} = { $x < 1$ and $y < 1$ and $z < 1$ } (All inputs are invalid)

I_{21} = { $x > 300$. and $y > 300$ and $z > 300$ } (All inputs are invalid)

I_{22} = { $x < 1$ and $y < 1$ and $z > 300$ } (All inputs are invalid)

I_{23} = { $x < 1$ and $y > 300$ and $z < 1$ } (All inputs are invalid)

I_{24} = { $x > 300$ and $y < 1$ and $z < 1$ } (All inputs are invalid)

I_{25} = { $x > 300$ and $y > 300$ and $z < 1$ } (All inputs are invalid)

I_{26} = { $x > 300$ and $y < 1$ and $z > 300$ } (All inputs are invalid)

I_{27} = { $x < 1$ and $y > 300$ and $z > 300$ } (All inputs are invalid)

2.3 DECISION TABLE BASED TESTING

Decision table testing is used for systems with complex business logic that can be represented as conditions and actions. It provides a systematic way to identify all possible combinations of conditions and the resulting actions.

Structure of a Decision Table:

- **Conditions:** The inputs or states that influence the outcome.
- **Rules:** The columns representing all possible combinations of conditions.
- **Actions:** The outputs or actions that result from each rule.

2.3 DECISION TABLE BASED TESTING

Condition

Action

Table 2.30. Decision table

Stubs	Entries
c_1 c_2 c_3	
a_1 a_2 a_3 a_4	

Four Portions

1. Condition Stubs
2. Condition Entries
3. Action Stubs
4. Action Entries

2.3.2Limited Entry and Extended Entry Decision Tables

- The decision tables which use only binary conditions are known as limited entry decision tables.
- The decision tables which use multiple conditions where a condition may have many possibilities instead of only ‘true’ and ‘false’ are known as extended entry decision tables

Table 2.31. Typical structure of a decision table

Stubs	R_1	R_2	R_3	R_4
c_1	F	T	T	T
c_2	-	F	T	T
c_3	-	-	F	T
a_1	X	X		X
a_2			X	
a_3	X			

2.3.3 ‘DoNot Care’ Conditions and Rule Count

Table 2.32. Decision table for triangle problem

Condition	c_1: $a < b + c$?	F	T	T	T	T	T	T	T	T	T	T
	c_2: $b < c + a$?	-	F	T	T	T	T	T	T	T	T	T
	c_3: $c < a + b$?	-	-	F	T	T	T	T	T	T	T	T
	c_4: $a^2 = b^2 + c^2$?	-	-	-	T	T	T	T	F	F	F	F
	c_5: $a^2 > b^2 + c^2$?	-	-	-	T	T	F	F	T	T	F	F
	c_6: $a_2 < b_2 + c_2$?	-	-	-	T	F	T	F	T	F	T	F
	Rule Count	32	16	8	1	1	1	1	1	1	1	1
Action	a_1 : Invalid triangle	X	X	X								
	a_2 : Right angled triangle							X				
	a_3 : Obtuse angled triangle									X		
	a_4 : Acute angled triangle										X	
	a_5 : Impossible				X	X	X		X			X

2.3.3 ‘DoNot Care’ Conditions and Rule Count Impossible Conditions

Table 2.32. Decision table for triangle problem

Condition	c_1: $a < b + c$?	F	T	T	T	T	T	T	T	T	T	T
	c_2: $b < c + a$?	-	F	T	T	T	T	T	T	T	T	T
	c_3: $c < a + b$?	-	-	F	T	T	T	T	T	T	T	T
	c_4: $a^2 = b^2 + c^2$?	-	-	-	T	T	T	T	F	F	F	F
	c_5: $a^2 > b^2 + c^2$?	-	-	-	T	T	F	F	T	T	F	F
	c_6: $a_2 < b_2 + c_2$?	-	-	-	T	F	T	F	T	F	T	F
	Rule Count	32	16	8	1	1	1	1	1	1	1	1
Action	a_1 : Invalid triangle	X	X	X								
	a_2 : Right angled triangle							X				
	a_3 : Obtuse angled triangle									X		
	a_4 : Acute angled triangle										X	
	a_5 : Impossible				X	X	X		X			X

2.3.5 Applicability

- Decision tables are popular in circumstances where an output is dependent on many conditions and a large number of decisions are required to be taken.
- They may also incorporate complex business rules and use them to design test cases.
- Every column of the decision table generates a test case.
- As the size of the program increases, handling of decision tables becomes difficult and cumbersome.
- In practice, they can be applied easily at unit level only. System testing and integration testing may not find its effective applications.

2.3.5 Applicability

Table 2.33. Decision table

c_1: x > = 1?	F	T	T	T	T	T	T	T	T	T	T	T	T	T
c_2: x <= 300?	-	F	T	T	T	T	T	T	T	T	T	T	T	T
c_3: y > = 1?	-	-	F	T	T	T	T	T	T	T	T	T	T	T
c_4: y <= 300?	-	-	-	F	T	T	T	T	T	T	T	T	T	T
c_5: z > = 1?	-	-	-	-	F	T	T	T	T	T	T	T	T	T
c_6: z <= 300?	-	-	-	-	-	F	T	T	T	T	T	T	T	T
c_7: x>y?	-	-	-	-	-	-	T	T	T	T	F	F	F	F
c_8: y>z?	-	-	-	-	-	-	T	T	F	F	T	T	F	F
c_9: z>x?	-	-	-	-	-	-	T	F	T	F	T	F	T	F
Rule Count	256	128	64	32	16	8	1	1	1	1	1	1	1	1
a_1 : Invalid input	X	X	X	X	X	X								
a_2 : x is largest								X		X				
a_3 : y is largest											X	X		
a_4 : z is largest									X				X	
a_5 : Impossible							X							X

2.3.5 Applicability

Table 2.34. Test cases of the given problem

Test Case	x	y	z	Expected Output
1.	0	50	50	Invalid marks
2.	301	50	50	Invalid marks
3.	50	0	50	Invalid marks
4.	50	301	50	Invalid marks
5.	50	50	0	Invalid marks
6.	50	50	301	Invalid marks
7.	?	?	?	Impossible
8.	150	130	110	150
9.	150	130	170	170
10.	150	130	140	150
11.	110	150	140	150
12.	140	150	120	150
13.	120	140	150	150
14.	?	?	?	Impossible

: mark1 > = 0 ?	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
: mark1 < = 100 ?	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
: mark2 > = 0 ?	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
: mark2 < = 100 ?	-	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
: mark3 > = 0 ?	-	-	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
: mark3 < = 100?	-	-	-	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
: 0 ≤ avg ≤ 39 ?	-	-	-	-	-	T	T	F	T	T	T	F	F	F	F	F	F	F	F	F	F	F
: 40 ≤ avg ≤ 49 ?	-	-	-	-	-	-	T	F	F	F	F	T	T	T	T	F	F	F	F	F	F	F
: 50 ≤ avg ≤ 59 ?	-	-	-	-	-	-	-	T	F	F	F	T	T	F	F	T	T	T	F	F	F	F
: 60 ≤ avg ≤ 74 ?	-	-	-	-	-	-	-	-	T	F	F	-	F	T	F	T	F	F	T	T	F	F
: avg ≥ 75 ?	-	-	-	-	-	-	-	-	-	T	F	-	-	F	F	-	T	F	T	F	T	F
ule Count	1024	512	256	128	64	32	8	4	2	1	1	4	2	1	1	2	1	1	1	1	1	1
: Invalid marks	X	X	X	X	X	X																
: First division with stinction																					X	
: First division																				X		
: Second division																		X				
: Third division															X							
: Fail											X											
: Impossible							X	X	X	X		X	X	X		X	X		X			X

2.3.5 Applicability

Table 2.36. Test cases of the given problem

Test Case	mark1	mark2	mark3	Expected Output
1.	-1	50	50	Invalid marks
2.	101	50	50	Invalid marks
3.	50	-1	50	Invalid marks
4.	50	101	50	Invalid marks
5.	50	50	-1	Invalid marks
6.	50	50	101	Invalid marks
7.	?	?	?	Impossible
8.	?	?	?	Impossible
9.	?	?	?	Impossible
10.	?	?	?	Impossible
11.	25	25	25	Fail
12.	?	?	?	Impossible
13.	?	?	?	Impossible
14.	?	?	?	Impossible
15.	45	45	45	Third division
16.	?	?	?	Impossible
17.	?	?	?	Impossible
18.	55	55	55	Second division
19.	?	?	?	Impossible
20.	65	65	65	First division
21.	80	80	80	First division with distinction
22.	?	?	?	Impossible

2.3.5 Applicability

$I_1 = \{ A1 : 0 \leq mark1 \leq 100 \}$

$I_2 = \{ A2 : mark1 < 0 \}$

$I_3 = \{ A3 : mark1 > 100 \}$

$I_4 = \{ B1 : 0 \leq mark2 \leq 100 \}$

$I_5 = \{B2 : mark2 < 0 \}$

$I_6 = \{ B3 : mark2 > 100 \}$

$I_7 = \{ C1 : 0 \leq mark3 \leq 100 \}$

$I_8 = \{ C2 : mark3 < 0 \}$

$I_9 = \{ C3 : mark3 > 100 \}$

$I_{10} = \{ D1 : 0 \leq avg \leq 39 \}$

$I_{11} = \{ D2 : 40 \leq avg \leq 49 \}$

$I_{12} = \{ D3 : 50 \leq avg \leq 59 \}$

$I_{13} = \{ D4 : 60 \leq avg \leq 74\}$

$I_{14} = \{ D5 : avg \geq 75 \}$

$I_1 = \{ A1$
$I_2 = \{ A2$
$I_3 = \{ A3$
$I_4 = \{ B1$

2.3.5 Applicability

Table 2.37. Extended entry decision table

Conditions	1	2	3	4	5	6	7	8	9	10	11
c_1 : mark1 in	A1	A1	A1	A1	A1	A1	A1	A1	A1	A2	A3
c_2 : mark 2 in	B1	B1	B1	B1	B1	B1	B1	B2	B3	-	-
c_3 : mark3 in	C1	C1	C1	C1	C1	C2	C3	-	-	-	-
c_4 : avg in	D1	D2	D3	D4	D5	-	-	-	-	-	-
Rule Count	1	1	1	1	1	5	5	15	15	45	45
a_1: Invalid Marks						X	X	X	X	X	X
a_2: First Division with Distinction					X						
a_3: First Division				X							
a_4: Second Division			X								
a_5: Third Division		X									
a_6: Fail	X										

2.3.5 Applicability

Table 2.38. Test cases of the given problem

Test Case	mark1	mark2	mark3	Expected Output
1.	25	25	25	Fail
2.	45	45	45	Third Division
3.	55	55	55	Second Division
4.	65	65	65	First Division
5.	80	80	80	First Division with Distinction
6.	50	50	-	Invalid marks
7.	50	50	101	Invalid marks
8.	50	-	50	Invalid marks
9.	50	101	50	Invalid marks
10.	-	50	50	Invalid marks
11.	101	50	50	Invalid marks

Decision table optimization

- 1. Rule Consolidation (The "Don't Care" Method)
  - The most common optimization is merging rules that result in the same action regardless of a specific condition’s value.
  - We represent these with a dash - or an "X," signifying a "Don't Care" entry.
  - How it works: If Rule 1 and Rule 2 result in "Action A," and the only difference between them is that Condition 3 is "True" in one and "False" in the other, Condition 3 is irrelevant for that specific outcome.
  - Result: You can merge two columns into one, effectively halving the complexity of that specific logic branch.
- 2. Identify Mutually Exclusive ConditionsIn many systems, certain conditions cannot physically or logically occur at the same time.
  - Identifying these allows you to prune "impossible" columns immediately.
  - Example: If you have conditions for Is_Under_18 and Is_Senior_Citizen, they cannot both be "True."Optimization:
  - Remove all columns where both are "True." This drastically reduces the total count before you even start writing test cases.

Decision table optimization example

- Let’s use an **Auto Insurance Premium** logic. It’s a classic example because it involves risk factors that often overlap, making it a prime candidate for "collapsing."
- **The Requirements**
- **Age < 25:** Young drivers are higher risk.
- **Accident History:** Any accident in the last 3 years.
- **High-Performance Car:** Sports cars increase the premium.
- **Business Rules:**
- If the driver is young **AND** has an accident history, **Reject** the policy (regardless of car type).
- If the driver is 25+ with no accidents and a standard car, **Standard Rate**.
- Any other combination of "High Risk" factors (Young, Accident, or Sports Car) results in a **Surcharge**.

Decision table optimization example

- Let’s use an **Auto Insurance Premium** logic. It’s a classic example because it involves risk factors that often overlap, making it a prime candidate for "collapsing."
- **The Requirements**
- **Age < 25:** Young drivers are higher risk.
- **Accident History:** Any accident in the last 3 years.
- **High-Performance Car:** Sports cars increase the premium.
- **Business Rules:**
- If the driver is young **AND** has an accident history, **Reject** the policy (regardless of car type).
- If the driver is 25+ with no accidents and a standard car, **Standard Rate**.
- Any other combination of "High Risk" factors (Young, Accident, or Sports Car) results in a **Surcharge**.

Decision table optimization example

Conditions	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
C1: Age < 25?	Y	Y	Y	Y	N	N	N	N
C2: Accident History?	Y	Y	N	N	Y	Y	N	N
C3: Sports Car?	Y	N	Y	N	Y	N	Y	N
Actions								
A1: Standard Rate								X
A2: Surcharge			X	X	X	X	X	
A3: Reject	X	X						

Decision table optimization example

- To optimize, we look for rules that have the same outcome where one condition doesn't actually change that outcome.
- **Look at R1 and R2:** Both result in "Reject." The only difference is the Sports Car (C3). This means if you are young and have accidents, we don't care what car you drive.
- **Look at R5 and R6:** Both result in "Surcharge." The only difference is the Sports Car (C3). If you have accidents and are over 25, you get a surcharge regardless of the car.
- **Look at R3 and R4:** Both result in "Surcharge." However, we have to be careful here—if we collapse these, we might lose the distinction of the car type. Wait, actually, if you are young and have *no* accidents, you get a surcharge regardless of the car (one for being young, or one for the car).

Decision table optimization example

Conditions	Rule A	Rule B	Rule C	Rule D	Rule E
C1: Age < 25?	Y	Y	N	N	N
C2: Accident History?	Y	N	Y	N	N
C3: Sports Car?	—	—	—	Y	N
Actions					
A1: Standard Rate					X
A2: Surcharge		X	X	X	
A3: Reject	X				