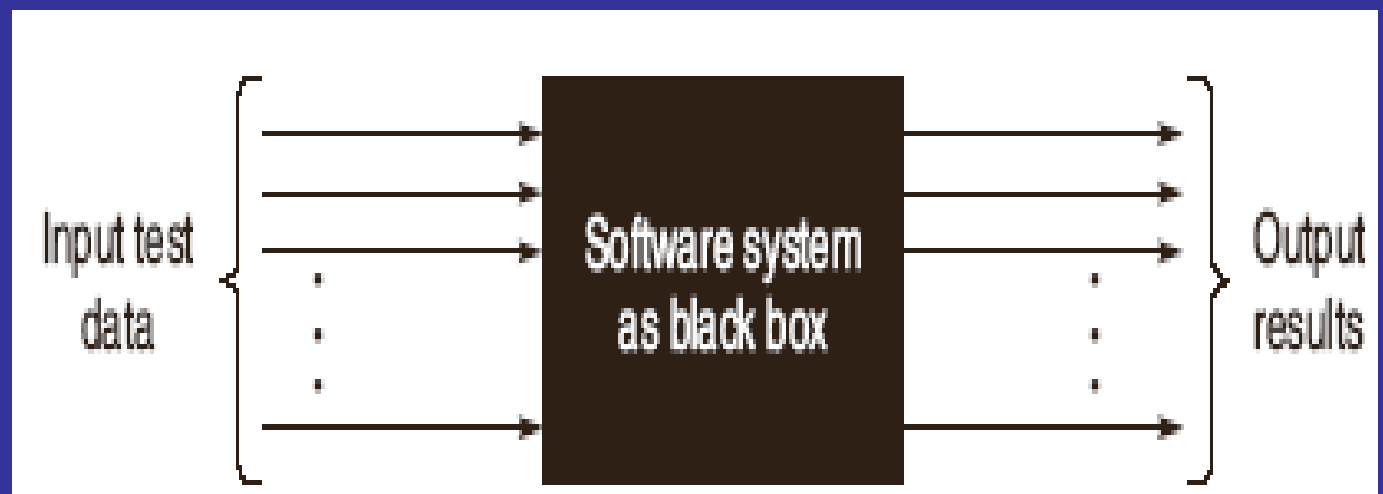
OXFORD
HIGHER EDUCATION

Dynamic Testing: Black Box Testing Techniques

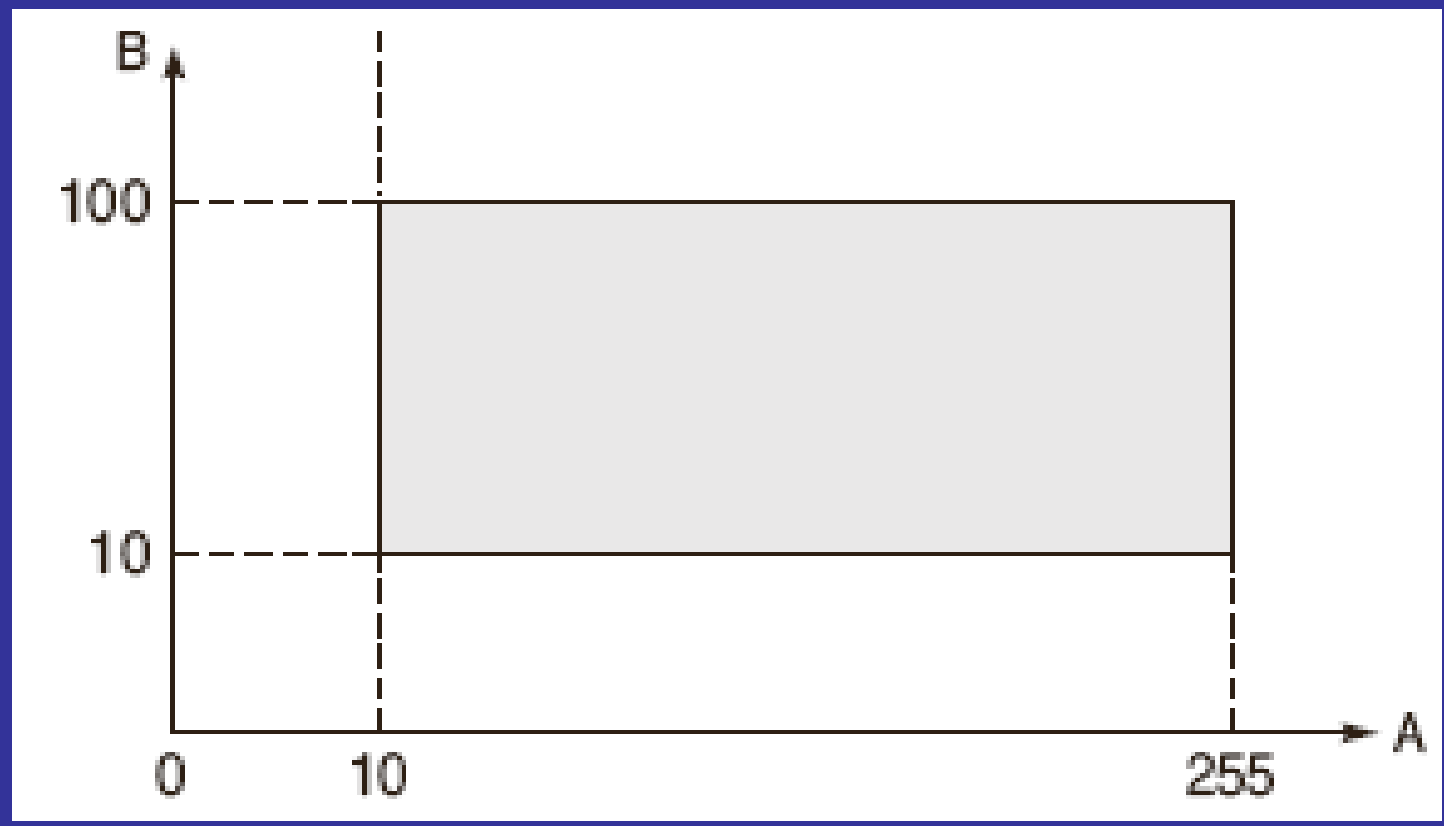
Black Box Testing



Black Box Testing

- To test the modules independently.
- To test the functional validity of the software
- Interface errors are detected.
- To test the system behavior and check its performance.
- To test the maximum load or stress on the system.
- Customer accepts the system within defined acceptable limits.

Boundary Value Analysis (BVA)



Boundary Value Analysis (BVA)

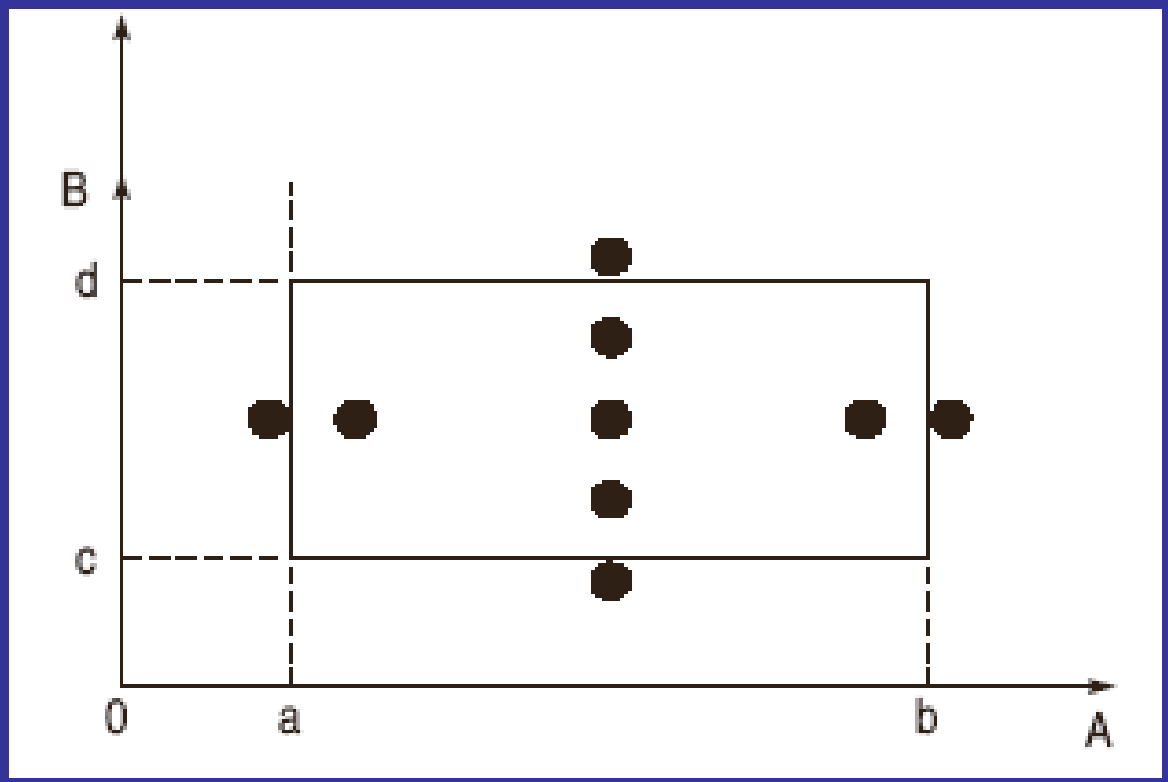
- Boundary Value Checking (BVC)
- Robustness Testing
- Worst Case Testing

Boundary Value Checking

- 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-)

Boundary Value Checking

- Anom, Bmin
- Anom, Bmin+
- Anom, Bmax
- Anom, Bmax-
- Amin, Bnom
- Amin+, Bnom
- Amax, Bnom
- Amax-, Bnom
- Anom, Bnom



- 4n+1 test cases can be designed with boundary value checking method.

Robustness Testing Method

A value just greater than the Maximum value (Max+)

A value just less than Minimum value (Min-)

- When test cases are designed considering above points in addition to BVC, it is called Robustness testing.

- Amax+, Bnom
- Amin-, Bnom
- Anom, Bmax+
- Anom, Bmin-
-
- It can be generalized that for n input variables in a module, 6n+1 test cases are designed with Robustness testing.

Worst Case Testing Method

- When more than one variable are in extreme values, i.e. when more than one variable are on the boundary. It is called Worst case testing method.

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

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. $A_{\max-}, B_{\max-}$

Example

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

- 1) Test cases using BVC

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

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

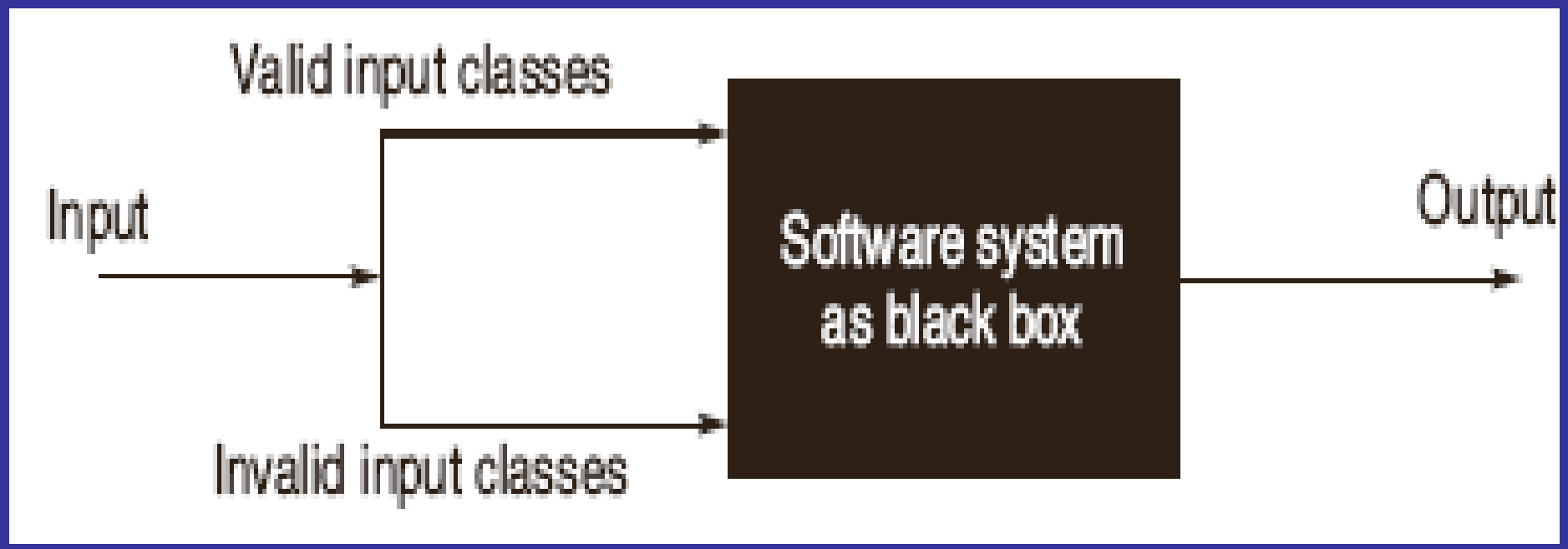
Example

- Test Cases Using Robust Testing

Test Case ID	Integer Variable	Expected Output
1	0	Invalid input
2	1	Not a prime number
3	2	Prime number
4	100	Not a prime number
5	99	Not a prime number
6	101	Invalid input
7	53	Prime number

Min value = 1
Min^- value = 0
Min^+ value = 2
Max value = 100
Max^- value = 99
Max^+ value = 101
Nominal value = 50–55

Equivalence Class Testing



Example

- A program reads three numbers A, B and C with range [1,50] and prints largest number. Design all test cases for this program using equivalence class testing technique.

I1 = {<A,B,C> : 1 ⩽ A ⩽ 50}
I2 = {<A,B,C> : 1 ⩽ B ⩽ 50}
I3 = {<A,B,C> : 1 ⩽ C ⩽ 50}
I4 = {<A,B,C> : A < 1}
I5 = {<A,B,C> : A > 50}
I6 = {<A,B,C> : B < 1}
I7 = {<A,B,C> : B > 50}
I8 = {<A,B,C> : C < 1}
I9 = {<A,B,C> : C > 50}

Test case ID	A	B	C	Expected result	Classes covered by the test case
1	13	25	36	C is greatest	I_1, I_2, I_3
2	0	13	45	Invalid input	I_4
3	51	34	17	Invalid input	I_5
4	29	0	18	Invalid input	I_6
5	36	53	32	Invalid input	I_7
6	27	42	0	Invalid input	I_8
7	33	21	51	Invalid input	I_9

Example

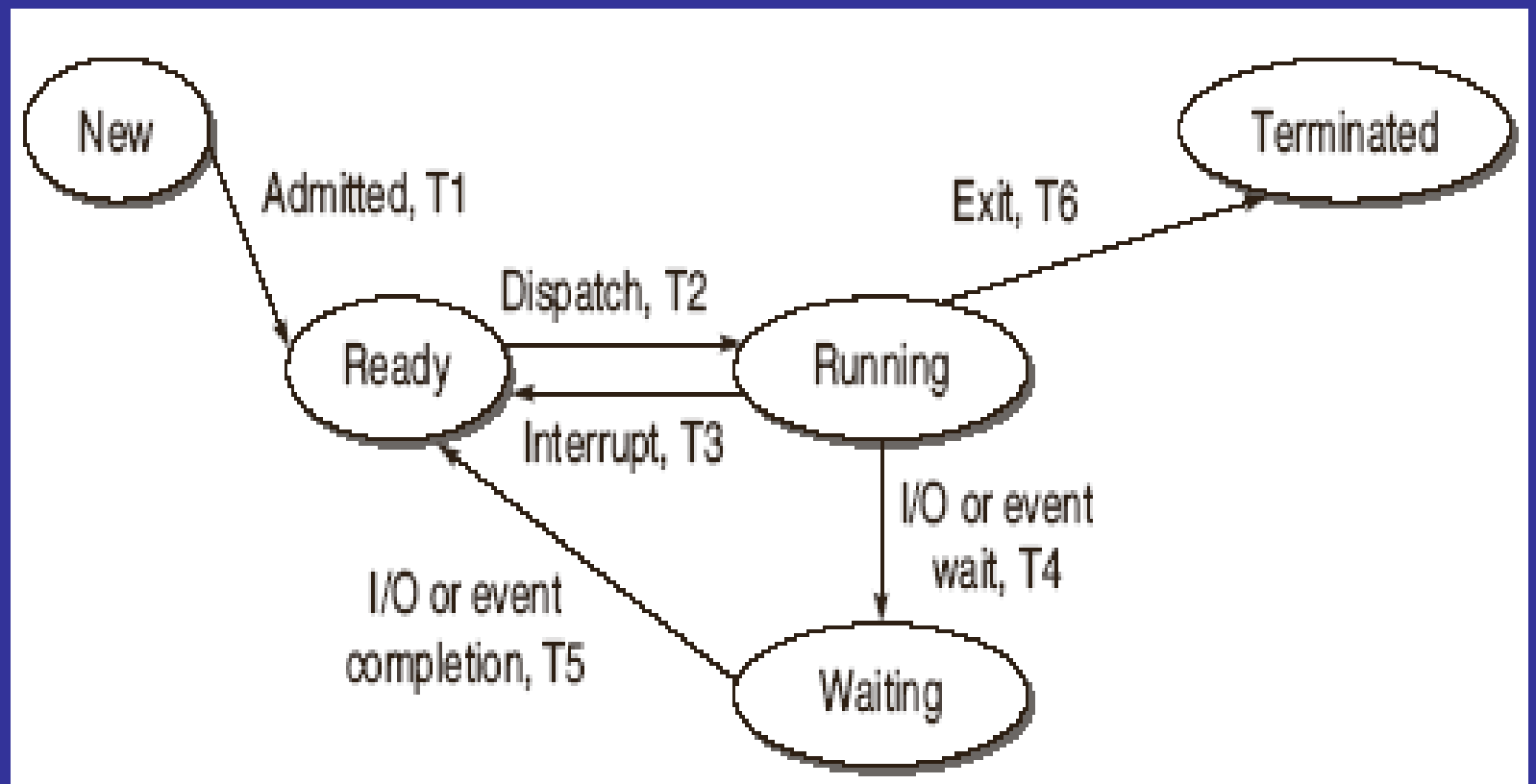
- I1 = {<A,B,C> : A > B, A > C}
- I2 = {<A,B,C> : B > A, B > C}
- I3 = {<A,B,C> : C > A, C > B}
- I4 = {<A,B,C> : A = B, A ≠ C}
- I5 = {<A,B,C> : B = C, A ≠ B}
- I6 = {<A,B,C> : A = C, C ≠ B
- I7 = {<A,B,C> : A = B = C}

Test case ID	A	B	C	Expected Result	Classes Covered by the test case
1	25	13	13	A is greatest	I_1, I_5
2	25	40	25	B is greatest	I_2, I_6
3	24	24	37	C is greatest	I_3, I_4
4	25	25	25	All three are equal	I_7

State Table Based Testing

Finite State Machine (FSM)

State Transition Diagrams or State Graph



State Table Based Testing

- *State Table*

State\Input Event	Admit	Dispatch	Interrupt	I/O or Event Wait	I/O or Event Wait Over	Exit
New	**Ready/ T1**	New / T0	New / T0	New / T0	New / T0	New / T0
Ready	Ready/ T1	**Running/ T2**	Ready / T1	Ready / T1	Ready / T1	Ready / T1
Running	Running/T2	Running/ T2	**Ready / T3**	**Waiting/ T4**	Running/ T2	**Terminated/T6**
Waiting	Waiting/T4	Waiting / T4	Waiting/T4	Waiting / T4	**Ready / T5**	Waiting / T4

State Table Based Testing

Test Case ID	Test Source	Input		Expected Results	
		Current State	Event	Output	Next State
TC1	Cell 1	New	Admit	T1	Ready
TC2	Cell 2	New	Dispatch	T0	New
TC3	Cell 3	New	Interrupt	T0	New
TC4	Cell 4	New	I/O wait	T0	New
TC5	Cell 5	New	I/O wait over	T0	New
TC6	Cell 6	New	exit	T0	New
TC7	Cell 7	Ready	Admit	T1	Ready
TC8	Cell 8	Ready	Dispatch	T2	Running
TC9	Cell 9	Ready	Interrupt	T1	Ready
TC10	Cell 10	Ready	I/O wait	T1	Ready
TC11	Cell 11	Ready	I/O wait	T1	Ready
TC12	Cell 12	Ready	Exit	T1	Ready
TC13	Cell 13	Running	Admit	T2	Running
TC14	Cell 14	Running	Dispatch	T2	Running
TC15	Cell 15	Running	Interrupt	T3	Ready
TC16	Cell 16	Running	I/O wait	T4	Waiting
TC17	Cell 17	Running	I/O wait over	T2	Running
TC18	Cell 18	Running	Exit	T6	Terminated
TC19	Cell 19	Waiting	Admit	T4	Waiting
TC20	Cell 20	Waiting	Dispatch	T4	Waiting
TC21	Cell 21	Waiting	Interrupt	T4	Waiting
TC22	Cell 22	Waiting	I/O wait	T4	Waiting
TC23	Cell 23	Waiting	I/O wait over	T5	Ready
TC24	Cell 24	Waiting	Exit	T4	Waiting

Decision Table Based Testing

		ENTRY				
Condition Stub		**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**...**
	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 Based Testing

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 the test cases using decision table testing.

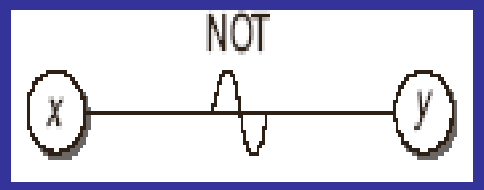
ENTRY				
		Rule 1	Rule 2	Rule3
Condition Stub	C1: Working hours > 48	I	F	T
	C2: Holidays or Sundays	T	F	F
Action Stub	A1: Normal salary		X	
	A2: 1.25 of salary			X
	A3: 2.00 of salary	X		

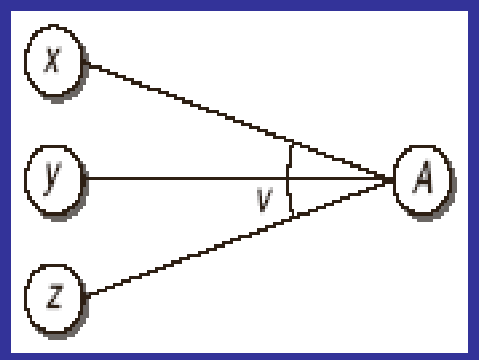
Decision Table Based Testing

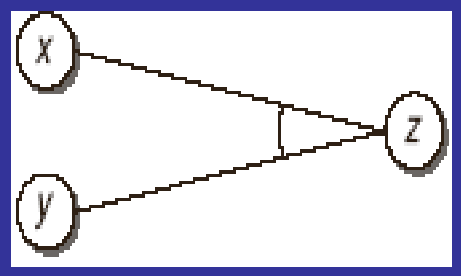
Test Case ID	Working Hour	Day	Expected Result
1	48	Monday	Normal Salary
2	50	Tuesday	1.25 of salary
3	52	Sunday	2.00 of salary

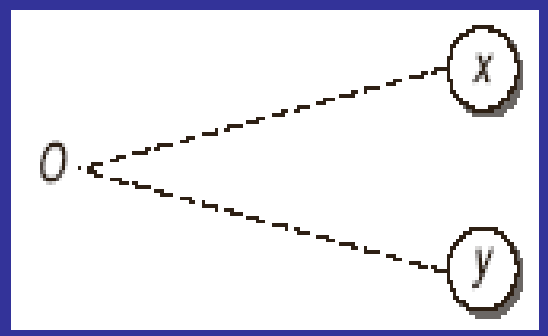
Cause-Effect Graphing based Testing

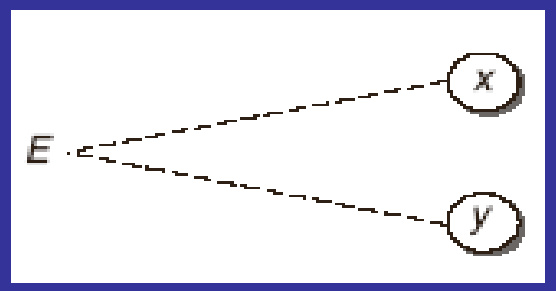
Basic Notations for Cause-Effect Graph

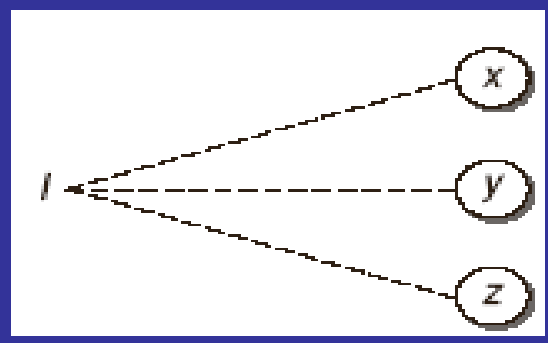


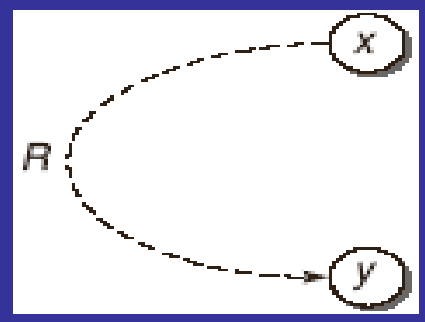


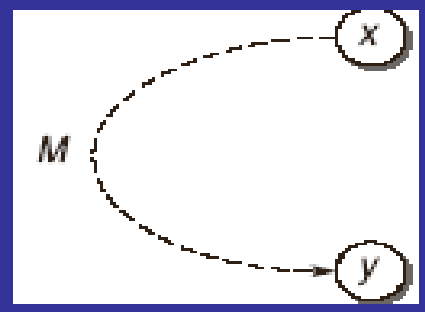












Cause-Effect Graphing based Testing

OXFORD

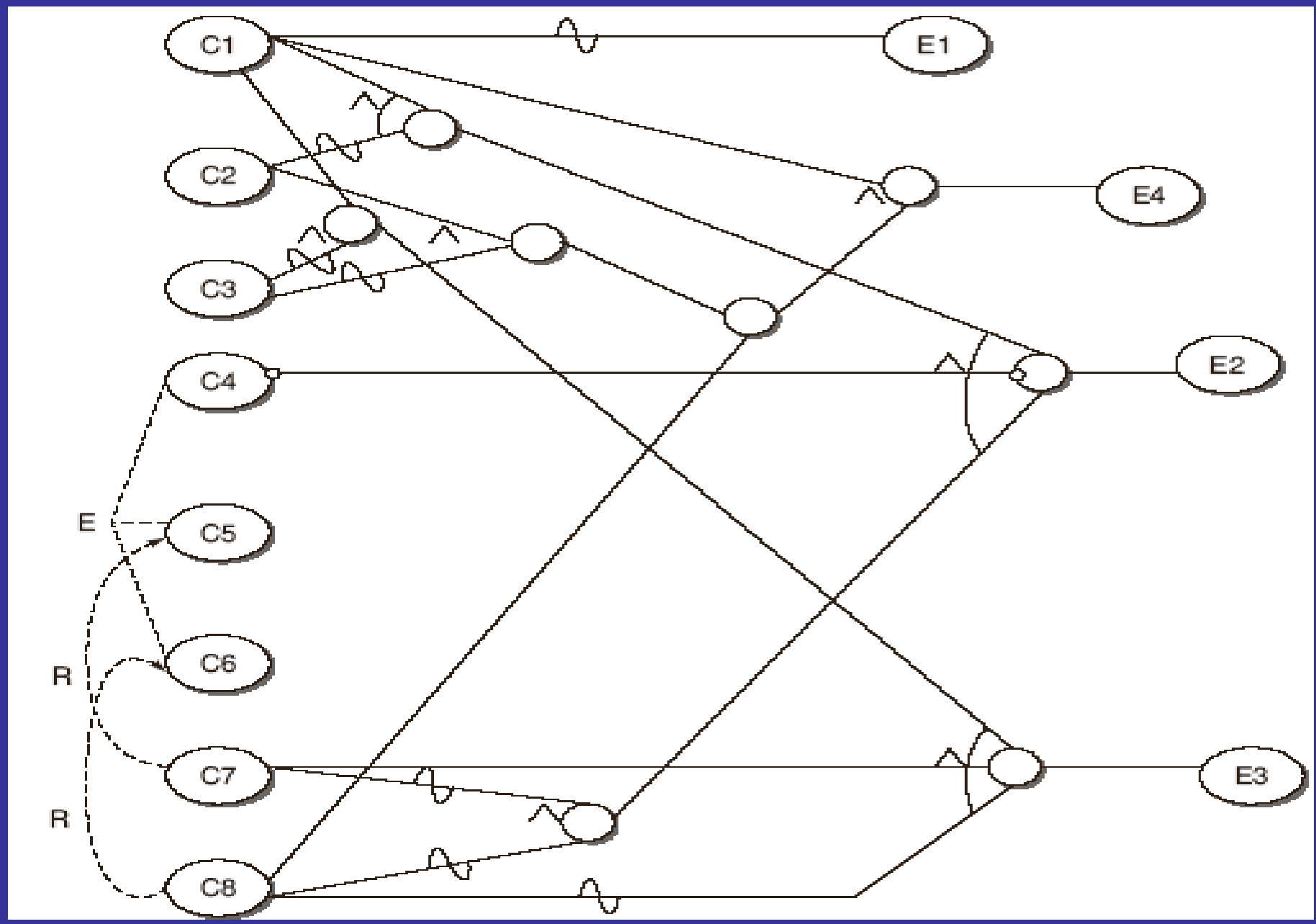
Example

- A program has been designed for the determination of nature of roots of a quadratic equation. Quadratic equation takes three input values from the range [0,100]. Design all test cases using Cause-Effect graphing technique.

Cause-Effect Graphing based Testing

- C1: a ≠ 0
- C2: b = 0
- C3: c = 0
- C4: D > 0 where D is b2 − 4 * a * c
- C5: D < 0
- C6: D = 0
- C7: a = b = c
- C8: a = c = b/2
- E1: Not a quadratic equation
- E2: Real Roots
- E3: Imaginary Roots
- E4: Equal Roots

Cause-Effect Graphing based Testing



Cause-Effect Graphing Based Testing

	R1	R2	R3	R4	R5	R6	R7
C1: a ≠ 0	T	T	T	T	T	T	F
C2: b = 0	F	I	I	T	F	F	I
C3: c = 0	I	F	I	T	F	F	I
C4: D > 0	T	F	F	F	F	F	I
C5: D < 0	F	T	F	F	T	F	I
C6: D = 0	F	F	T	T	F	T	I
C7: a = b = c	F	I	F	F	T	F	I
C8: a = c = b/2	F	F	I	F	F	T	I
A1: Not a quadratic equation							X
A2: Real roots	X						
A3: Imaginary roots		X			X		
A4: Equal roots			X	X		X	

Test Case ID	a	b	c	Expected Output
1	1	50	50	Real roots
2	100	50	50	Imaginary roots
3	1	6	9	Equal
4	100	0	0	Equal
5	99	99	99	Imaginary
6	50	100	50	Equal
7	0	50	30	Not a quadratic equation

Error Guessing

- Error guessing is the method used when all other methods fail or it is the method for some special cases which need to be tested.
- It means error or bug can be guessed which do not fit in any of the earlier defined situations. So test cases are generated for these special cases.