Boundary Value Testing

Boundary Value Analysis:

→ A function F of 2 Variables x_1 and x_2, When implemented as a Program, will have some boundaries $a \leq x_1 \leq b$

$c \leq x_2 \leq d$.

→ Strongly typed languages permit explicit definition of such Variable ranges.

↳ Results in faults that are easily revealed by boundary Value testing.

→ Boundary Value analysis focuses on the boundary of the input space to identify testcases.

→ Basic idea:- use input Variable at their minimum, just above the minimum, a nominal Value, just below their maximum and their maximum.

→ This Tool has been successfully integrated with 2 popular front-end CASE tools. The T Tool refers to these Values as min, min +, nom, max-, and max.

→ Based on a critical assumption: Single fault assumption in reliability theory.

→ Thus boundary value analysis testcases are obtained by holding the Values of all but one Variable at their nominal Values. & letting that Variable assume its extreme Values.

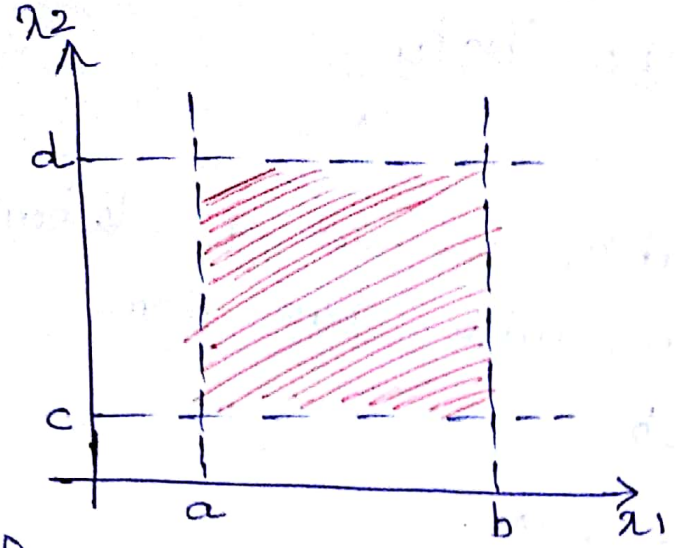


Fig:- Input domain of a function of 2 variables

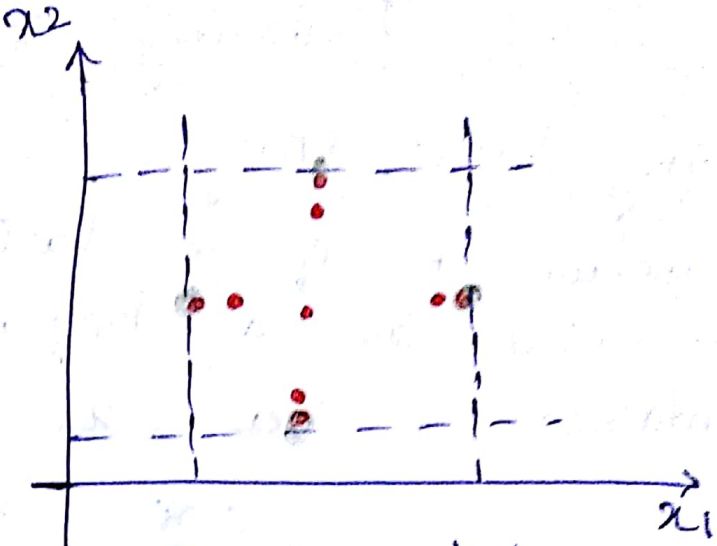


Fig:- Boundary Value analysis testcase for a function of 2 variables.

→ The boundary Value Analysis testcase for function of 2 variables.

$\{ \langle x_{1nom}, x_{2min} \rangle, \langle x_{1nom}, x_{2min+} \rangle, \langle x_{1nom}, x_{2nom} \rangle,$
$\langle x_{1nom}, x_{2max-} \rangle, \langle x_{1nom}, x_{2max} \rangle, \langle x_{1min}, x_{2nom} \rangle,$
$\langle x_{1min+}, x_{2nom} \rangle, \langle x_{1max-}, x_{2nom} \rangle, \langle x_{1max}, x_{2nom} \rangle$

Generalizing Boundary Value Analysis:

Generalized 2 ways

- number of variables
- kind of ranges.

→ For a function of n variables, boundary value analysis yields $4n + 1$ unique testcases.

→ Boundary Value Analysis doesnot make much dense for Boolean variables; the extreme values are TRUE and FALSE, but no clear choice is available for the remaining three.

Limitations of Boundary Value Analysis:

→ Works well when the Program to be tested is a function of several independent Variables that represent bounded physical quantities.

→ Boundary Value Analysis testcases are derived from the extrema of bounded, independent Variables that refer to physical quantities, with no consideration of the nature of the function or of the semantic meaning of the Variables.

Robustness testing:

→ Simple extension of boundary Value analysis.

→ In addition to the five boundary Value analysis values of a Variable, here we also consider, When the extrema are exceeded with a Value slightly greater than the maximum (max +) and Value slightly less than the minimum (min -).

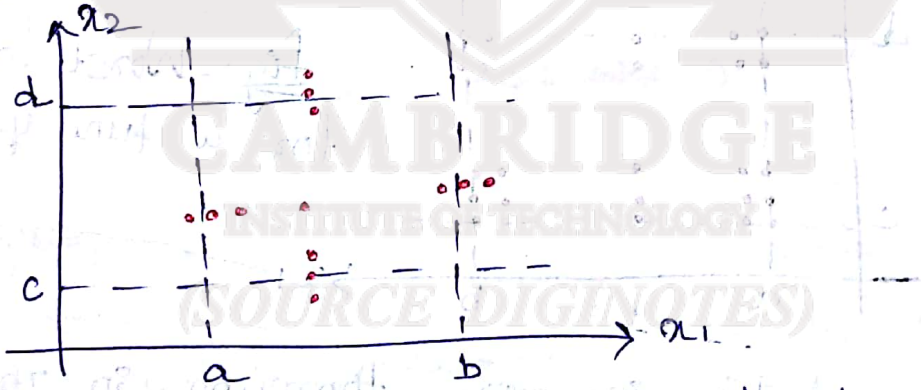


Fig:- Robustness testcase for a function of 2 Variables.

→ The most intresting part of robustness testing is not with inputs, but with outputs.
The main Value of robustness testing is that it forces attention on exception handling.

→ In pascal, for example, if a variable is defined to be within a certain range, values outside that range result in runtime errors that abort normal execution. This raises an interesting question of implementation philosophy: Is it better to perform explicit range checking and use exception handling to deal with robust values, or is it better to stay with strong typing? The exception handling choice mandates robustness testing.

Worst case Testing:

Worst case Analysis: What happens when more than one variable has an extreme value. This idea is used to generate worst case test cases.

→ The result of the 2 variable version of this is shown below:

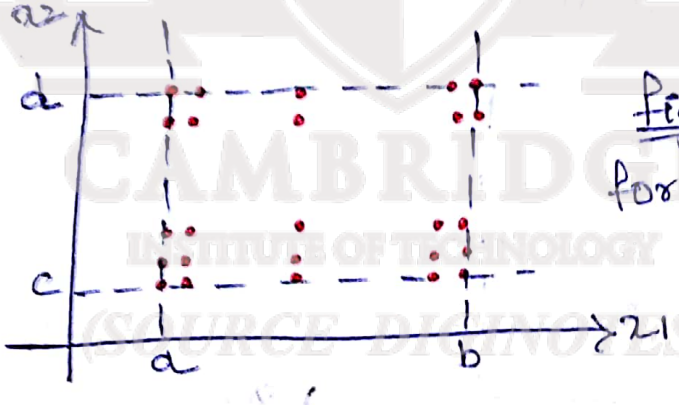


Fig:- Worst case testcase for a func of 2 variab

→ Worst-case testing is more thorough in the sense that boundary value analysis testcases are a proper subset of Worst-case test cases. It also represents much more effort: Worst-case testing for a function of n variables generates 5^n testcases as opposed to $4n+1$ testcases for boundary value Analysis.

→ The best application for Worst case testing is where physical variables have numerous interactions & where failure of the function is extremely costly.

→ For really paranoid testing. We could go to robust worst case testing. This involves the cartesian product of the 7 - elements sets we used in robust testing resulting in 7^n test cases.

→ The below fig shows the robust worst case test cases of 2 variable functions.

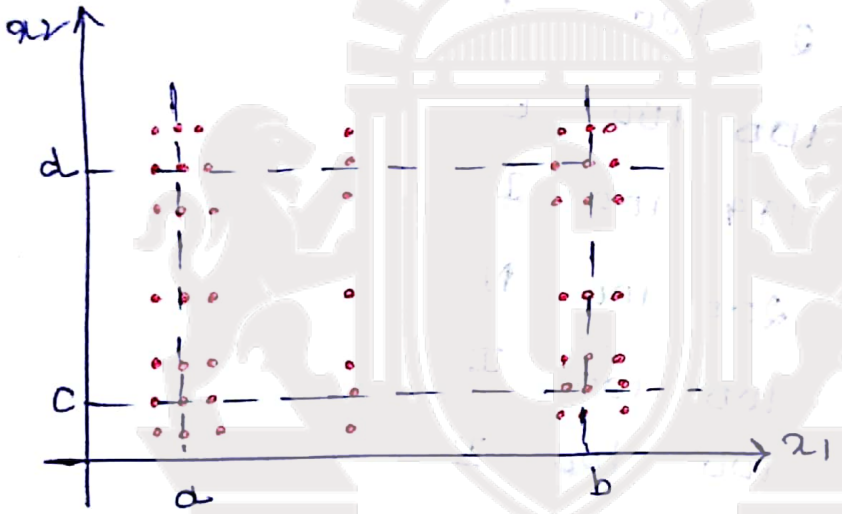


Fig:- Robust Worst case testcases for a func of 2 variables

Special Value Testing:

→ Special value testing occurs when a tester uses domain knowledge, experience with similar programs, and information about "soft spots" to devise testcases.

→ No guidelines are used other than to use 'best engineering judgement'. As a result special value testing is very dependent on the abilities of the testers.

Examples:

→ Triangle Problem Boundary value Analysis testcases.

case	a	b	c	Expected output
1	100	100	1	isosceles.
2	100	100	2	"
3	100	100	100	equilateral
4	100	100	199	isosceles
5	100	100	200	not a triangle
6	100	1	100	I
7	100	2	100	I
8	100	100	100	E
9	100	199	100	I
10	100	200	100	N
11	1	100	100	I
12	2	100	100	I
13	100	100	100	E
14	199	100	100	I
15	200	100	100	N

Test cases for the Triangle Problem:

→ In the Problem. Statement, no conditions are specified on the triangle sides, other than being integers. The lower bounds of the ranges are all 1. upper bound arbitrarily taken as 200.

→ The following table shows the worst testcases in just "one corner" of the input space cube.

Triangle Problem Worst-case test cases:

case	a	b	c	Expected output
1	1	1	1	E
2	1	1	2	N
3	1	1	100	N
4	1	1	199	N
5	1	1	200	N
6	1	2	1	N
7	1	2	2	I
8	1	2	100	N
9	1	2	199	N
10	1	2	200	N
11	1	100	1	N
12	1	100	2	N
13	1	100	100	I
14	1	100	199	N
15	1	100	200	N
16	1	199	1	N
17	1	199	2	N
18	1	199	100	N
19	1	199	199	I
20	1	199	200	N
21	1	200	1	N
22	1	200	2	N
23	1	200	100	N
24	1	200	199	N
25	1	200	200	I

→ Test cases for the NextDate function:

Case	Month	Day	Year	Expected O/P
1	1	1	1812	Jan 2, 1812
2	1	1	1813	Jan 2, 1813
3	1	1	1912	Jan 2, 1912
4	1	1	2011	Jan 2, 2011
5	1	1	2012	Jan 2, 2012
6	1	2	1812	Jan 3, 1812
7	1	2	1813	Jan 3, 1813
8	1	2	1912	Jan 3, 1912
9	1	2	2011	" 2011
10	1	2	2012	" 2012
11	1	15	1812	Jan 16, 1812
12	1	15	1813	" , 1813
13	1	15	1912	" , 1912
14	1	15	2011	" , 2011
15	1	15	2012	" , 2012
16	1	30	1812	Jan 31, 1812
17	1	30	1813	" , 1813
18	1	30	1912	" , 1813
19	1	30	2011	" , 1813
20	1	30	2012	" , 1813
21	1	31	1812	Feb 1, 1812
22	1	31	1813	" , 1813
23	1	31	1912	" , 1912
24	1	31	2011	" , 2011
25	1	31	2012	" , 2012

⇒ Testcases for the commission Problem:

⇒ This time we will look at boundary values for the output range, especially near the threshold points of $1000 and $1800. The output space of the commission is shown in the fig below.

→ The following table shows the worst case in the output space for the commission Problem. These test cases exercise the points at which the commission percentage changes

→ The value below the lowest plane corresponds to sales below the $1000 threshold. The volume between the 2 planes is the 15% commission range. Part of the reason for using the output range to determine test case is that cases from the input range are almost all in the 20% zone.

→ We want to find input variable combinations that stress the boundary values: $100, $1000, $1800 and $1800. These testcases were developed with a spreadsheet saving lot of calculator pecking.

Commission Problem output Boundary Value Analysis

Testcases :

Table 4 Output Boundary Value Analysis Test Cases **Case**

	locks	stocks	barrels	sales	comm	Comment
1	1	1	1	100	10	output minimum
2	1	1	2	125	12.5	output minimum +
3	1	2	1	130	13	output minimum +
4	2	1	1	145	14.5	output minimum +
5	5	5	5	500	50	midpoint
6	10	10	9	975	97.5	border point -
7	10	9	10	970	97	border point -
8	9	10	10	955	95.5	border point -
9	10	10	10	1000	100	border point
10	10	10	11	1025	103.75	border point +
11	10	11	10	1030	104.5	border point +
12	11	10	10	1045	106.75	border point +
13	14	14	14	1400	160	midpoint
14	18	18	17	1775	216.25	border point -
15	18	17	18	1770	215.5	border point -
16	17	18	18	1755	213.25	border point -
17	18	18	18	1800	220	border point
18	18	18	19	1825	225	border point +
19	18	19	18	1830	226	border point +
20	19	18	18	1845	229	border point +
21	48	48	48	4800	820	midpoint
22	70	80	89	7775	1415	output maximum -
23	70	79	90	7770	1414	output maximum -
24	69	80	90	7755	1411	output maximum -
25	70	80	90	7800	1420	output maximum

The following table contains some typical Value test cases for the commission Problem.

case	locks	Stocks	Barrels	Sales	commission	Comment
1	10	11	9	1005	100.75	Bordespoint
2	18	17	19	1795	219.25	BP −
3	18	19	17	1805	221	BP +

Equivalence class Testing:

Equivalence class:

→ They form a partition of a set, where partition refers to a collection of mutually disjoint sets, union of which is the entire set.

→ This has 2 implications of testing:
i) entire set is represented provides a form of completeness
ii) disjointedness ensures a form of non-redundancy

→ A function F, of 2 variables x_1 and x_2. When F is implemented as a program, the input variables x_1 and x_2 will have the following boundaries and intervals within the boundaries:

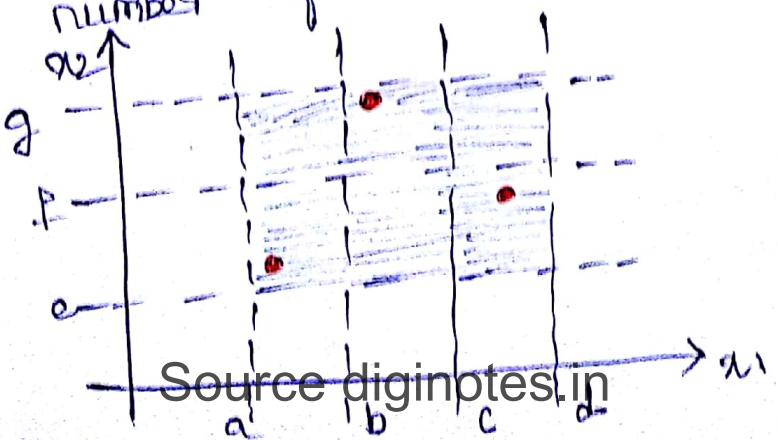
$a \leq x_1 \leq d$, with intervals $[a, b), [b, c), [c, d]$

$e \leq x_2 \leq g$, " " $[e, f), [f, g]$

Where square brackets and parentheses denote respectively closed and open interval endpoints.

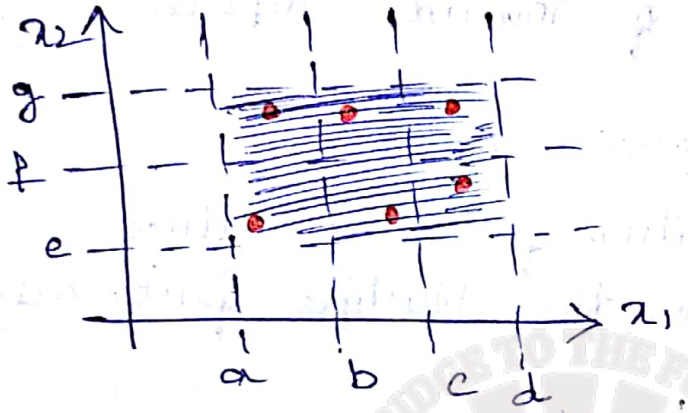
Weak Normal Equivalence class Testing:

→ Accomplished by using one variable from each equivalence class in a test case.
There will be same number of weak equivalence test cases as classes in the largest partition with the number of subsets.



Strong Normal equivalence class Testing:

→ Based on multiple fault assumptions, so we need tests cases from each element of the cartesian product of the equivalence classes.

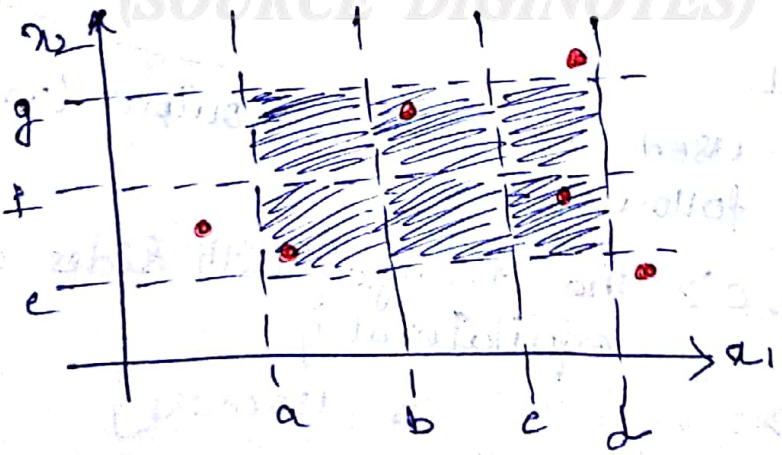


The cartesian product guarantees that we have a notion of completeness in a sense: We cover all the equivalence classes and we have one of each possible combination of inputs.

Weak Robust Equivalence class Testing:

↳ The Robust part comes from consideration of invalid values and the weak part refers to the single fault assumption.

1. for valid inputs, use one value from each valid class.
2. for invalid inputs, use a test case will have one invalid value and the remaining values will all be valid.

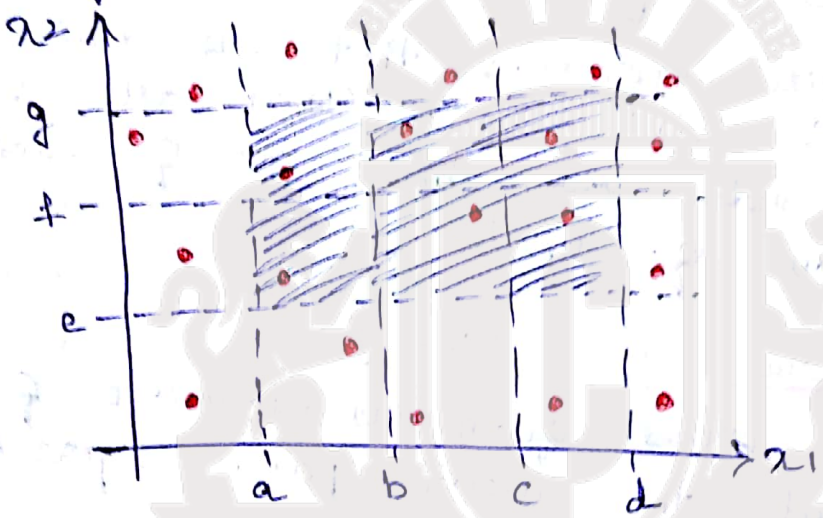


2 Problems:

* Expected outputs for an invalid input is not defined in the specification.
* Strongly typed language eliminates the need for the consideration of invalid inputs.

Strong Robust Equivalence class Testing:

→ Robust – consideration of invalid values
Strong part – Refers to the Multiple fault assumption.



Equivalence class Testcase for the Triangle Problem:

→ 4 Possible outputs can occur:
Not a Triangle
Scalene
Isosceless
Equilateral.

These can be used to identify output (range) equivalence classes as follows:

$R_1 = \{ \langle a, b, c \rangle$: the triangle with sides a, b & c is equilateral$\}$

$R_2 = \{ \langle a, b, c \rangle$: " is isoceles$\}$

$R_3 = \{ \langle a, b, c \rangle$: " Scalene$\}$

$R_4 = \{ \langle a, b, c \rangle$: sides a, b, c don't form a triangle$\}$

Weak Normal equivalence class testcase:

Testcase	a	b	c	Expected output
WN1	5	5	5	equilateral
WN2	2	2	3	isoseles
WN3	3	4	5	Scalene
WN4	4	1	2	not a triangle.

→ Because no valid subintervals of variables a, b, & c exist, the strong normal equivalence class testcases are identical to the weak normal equivalence class testcases.

Weak Robust equivalence test cases. [invalid values]

Testcase	a	b	c	Expected output
WR1	-1	5	5	Value of a is not in the range of Permitted values
WR2	5	-1	5	b
WR3	5	5	-1	c
WR4	201	5	5	a
WR5	5	201	5	b
WR6	5	5	201	c

Strong Robust equivalence class:

Testcase	a	b	c	ED
SR1	-1	5	5	" — a
SR2	5	-1	5	" b
SR3	5	5	-1	" c
SR4	-1	-1	5	" a, b
SR5	5	-1	-1	" b, c
SR6	-1	5	-1	" a, c
SR7	-1	-1	-1	" a, b, c.

Equivalence class Testcases for the NextDate fun:

→ Intervals of Valid Values defined as follows:

$M_1 = \{ month : 1 \leq month \leq 12 \}$

$D_1 = \{ day : 1 \leq day \leq 31 \}$

$Y_1 = \{ Year : 1812 \leq year \leq 2012 \}$

→ The invalid equivalence classes are:

$M_2 = \{ month : month < 1 \}$

$M_3 = \{ " " > 12 \}$

$D_2 = \{ day : day < 1 \}$

$D_3 = \{ day : day > 31 \}$

$Y_2 = \{ Year : Year < 1812 \}$

$Y_3 = \{ Year : Year > 2012 \}$

→ Because the number of Valid classes equals the number of independent variables, only one weak normal equivalence class testcase occurs, and it is identical to the strong normal equivalence class test case:

Case ID	Month	Day	Year	Expected O/P
WN1, SN1	6	15	1912	6/16/1912.

→ Weak Robust testcases:

Case ID	Month	Day	Year	Expected O/P
WR1	6	15	1912	6/16/1912
WR2	-1	15	"	Month not in the range..
WR3	13	15	"	"
WR4	6	-1	1912	Day 1.. 31
WR5	6	32	1912	"
WR6	6	15	1811	Year 1812.. 2012
WR7	6	15	2013	"

Strong Robust equivalence class testcases:

caseID	Month	Day	Year	Expected O/P
SR1	-1	15	1912	
SR2	6	-1	1912	
SR3	6	15	1811	
SR4	-1	-1	1912	
SR5	6	-1	1811	
SR6	-1	15	1811	
SR7	-1	-1	1811	

Equivalence class Testcase for the Commission Problem

The Valid classes of the input Variables are:

L1 = { locks : 1 ≤ locks ≤ 70 }

L2 = { locks = -1 } (occurs if locks = -1 is used to control I/P interaction)

S1 = { stocks : 1 ≤ stocks ≤ 80 }

B1 = { barrels : 1 ≤ barrels ≤ 90 }

The corresponding invalid classes of the input Variables are:

L3 = { locks : locks = 0 or locks < -1 }

L4 = { locks : locks > 70 }

S2 = { stocks : stocks < 1 }

S3 = { stocks : stocks > 80 }

B2 = { barrels : barrels < 1 }

B3 = { barrels : barrels > 90 }

Weak robust testcases:

caseID	locks	stocks	Barrels	expected o/p
WR1	10	10	10	$100
WR2	-1	40	45	Prog terminates
WR3	-2	40	45	Value of lock not in the range 1..70
WR4	71	40	45	Value of Stocks not in the range 1...70
WR5	35	-1	45	Value of Stocks not in the range 1... 80
WR6	35	81	45	"
WR7	35	40	-1	Value of Barrels not in the rang 1.. 90
WR8	35	40	91	"

strong Robust equivalence class:

caseID	locks	stocks	Barrels	Expected o/p
SR1	-2	40	45	Value of locks not in the ran 1..70
SR2	35	-1	45	Value of stocks not in the rang 1..80
SR3	35	40	-2	Value of Barrels not in the ran 1.. 90
SR4	-2	-1	45	Value of locks & Stocks not in t range
SR5	-2	40	-1	Value of locks & Barrels not in the

SR6	35	-1	-1	Value of Stocks & Barrels not in the range
SR7	-2	-1	-1	Value of locks, Stocks, Barrs not in the range.

Decision Table - Based Testing:

→ Ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions.

Table : Portions of Decision Table:

Stub	Rule 1	Rule 2	Rule 3, 4	Rule 5	Rule 6	Rule 7, 8
c_1	T	T	T	F	F	F
c_2	T	T	F	T	T	F
c_3	T	F	–	T	F	–
a_1	x	x		x		
a_2	x				x	
a_3		x		x		
a_4			x			x

→ A decision table has 4 portions:
The left-most column – stub portion
Right – entry portion.
The condition portion is noted by 'c's
Action portion – 'a'.

→ A column in the entry portion is a rule. Rules indicate which actions, if any, are taken for the circumstances indicated in the condition portion of the rule.

→ When condition c_1, c_2 & c_3 are true, action a_1 & a_2 occur. When c_1, c_2 are both true c_3 is false, action a_1, a_3 occur. The entry for c_3 when c_1 is true and c_2 is false is called "dontcare' entry

→ The dont care entry has 2 major Interpretations:
* The condition is irrelevant
* The condition doesnot apply.

→ In case of Binary conditions, the condition Portion of a decision table is a truth table.

→ The completeness property of a decision table guarantee a form of complete testing. Decision table in which all the conditions are binary are called limited entry decision tables. If conditions are allowed to have several values, the resulting tables are called extended entry decision tables.

Technique:

→ To identify testcases with decision tables. We interpret conditions as inputs and actions as outputs. The rules are then interpreted as testcases.

→ In the following table Examples of dont care entries & impossible rule usage. If the integers a, b & c dont constitute a triangle, then we dont even care about possible equalities, as indicated in the first rule. In rules 3, 4 & 6 if 2 pairs of integers are equal, by transitivity, the third pair must be equal; thus the negative entry makes these rules impossible.

c1 : a, b, c form a triangle?
c2 : a = b ?
c3 : a = c ?
c4 : b = c ?
a1 : Not a triangle
a2 : Scalene
a3 : Isosceles
a4 : Equilateral
a5 : Impossible

Table 7.2 Decision Table for the Triangle Problem

c3: a, b, c form a triangle?	N	Y	Y	Y	Y	Y	Y	Y	Y
c2: a = b?	–	Y	Y	N	Y	Y	N	N	N
c3: a = c?	–	Y	N	Y	N	Y	Y	N	N
c4: b = c?	–	Y	N	Y	N	N	Y	Y	N
a1: Not a triangle	X								X
a2: Scalene					X		X	X	
a3: Isosceles		X							
a4: Equilateral			X	X		X			
a5: Impossible									

T T T T
F F F F
T T F F
T F T F

X

a3 : Isosceles

a4 : equilateral

a5 : Impossible

→ The decision table below, illustrates another combination related to technique. The old cond^n is expanded. (C1: a, b, c form a triangle)

C1 : a < b + c ?

C2 : b

C3 : c

C4 :

C5 :

C6 : b = c ?

Table 7.3 Refined Decision Table for the Triangle Problem

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	–	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	–	–	F	T	T	T	T	T	T	T	T
c4: a = b?	–	–	–	T	T	T	T	F	F	F	F
c5: a = c?	–	–	–	T	T	F	F	T	T	F	F
c6: b = c?	–	–	–	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

a1 : Not a Δ^{le}

a2 : Scalene

a3 : Isosceles

a4 : Equilaterale

a5 : Impossible.

→ When cond^n refer to equivalence classes, decision tab have a characteristic appearance. cond^n in the decision table below are from the NextDate Problem; They refer to the Mutually exclusive possibilities for the Month Variable.

→ Because a month is in exactly one equivalence class, we can't ever have a rule in which 2 entries are true.

The dont care entries (–) really mean "Must be false".

Cond R1 R2 R3

c1:

c2:

c3:

Table 7.4 Decision Table with Mutually Exclusive Conditions

Conditions	R1	R2	R3
c1: month in M1?	T	–	–
c2: month in M2?	–	T	–
c3: month in M3?	–	–	T
a1			
a2			
a3			

a1

a2

a3.

→ For limited entry decision tables, if n cond exist, there must be 2^n rules.

We can develop a rule count as follows:

Rules in which no dont care entries occur count as one rule, and each dont care entry in a rule doubles the count of that rule.

→ The rule counts of Table case as shown.

Table 7.5 Decision Table for Table 7.3 with Rule Counts

c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	–	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	–	–	F	T	T	T	T	T	T	T	T
c4: a = b?	–	–	–	T	T	T	T	F	F	F	F
c5: a = c?	–	–	–	T	T	F	F	T	T	F	F
c6: b = c?	–	–	–	T	F	T	F	T	F	T	F
Rule Count	32	16	8	1	1	1	1	1	1	1	1
a1: Not a triangle	X	X	X								X
a2: Scalene							X		X	X	
a3: Isosceles				X							
a4: Equilateral					X	X		X			
a5: Impossible											

Table 7.6 Rule Counts for a Decision Table with Mutually Exclusive Conditions

Conditions	R1	R2	R3
c1: month in M1	T	–	–
c2: month in M2	–	T	–
c3: month in M3	–	–	T
Rule Count	4	4	4
a1			

Table :- Rule counts for Mutually exclusive cond?.

[Refer Table Prev Page]

→ We should only have 8 rules. So we have a Problem. To see where the Problem lies, we expand each of the 3 rules. replacing the "_" entries with T & F possibilities as shown in table below.

Table 7.7 Expanded Version of Table 7.6

Conditions	*1.1*	*1.2*	*1.3*	*1.4*	*2.1*	*2.2*	*2.3*	*2.4*	*3.1*	*3.2*	*3.3*	*3.4*
c1: mo. in M1	T	T	T	T	T	T	F	F	T	T	F	F
c2: mo. in M2	T	T	F	F	T	T	T	T	T	F	T	F
c3: mo. in M3	T	F	T	F	T	F	T	F	T	T	T	T
Rule Count	1	1	1	1	1	1	1	1	1	1	1	1
a1												

→ We have 3 rules in which all entries are T: rules 1.1, 2.1 and 3.1. We also have 2 rules with T, T, F entries: rules 1.2 & 2.2. Similarly rules 1.3 and 3.2 are identical; So are rules 2.3 and 3.3. If we delete the repetitions, We endup with 7 rules; the Missing rule is the one in which all cond^n are false. The result of this Process is shown below.

Decision Table-Based Testing 115

Table 7.8 Mutually Exclusive Conditions with Impossible Rules

	1.1	1.2	1.3	1.4	2.3	2.4	3.4	
c1: mo. in M1	T	T	T	T	F	F	F	F
c2: mo. in M2	T	T	F	F	T	T	F	F
c3: mo. in M3	T	F	T	F	T	F	T	F
Rule Count	1	1	1	1	1	1	1	1
a1: Impossible	X	X	X		X			X

→ The ability to recognize complete decision table puts us in a powerful position with respect to redundancy & inconsistency. The decision table below is redundant - three cond^n & 9 rules exist.

Table 7.9 A Redundant Decision Table

Conditions	1–4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	–	T	T	F	F	F
c3	–	T	F	T	F	F
a1	X	X	X	–	–	X
a2	–	X	X	X	–	–
a3	X	–	X	X	X	X

Table 7.10 An Inconsistent Decision Table

Conditions	1-4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—

→ If the action entries are different as in the Table above (↑), and if this table were to Process a transaction in which c1 is true & both c2 & c3 are false, both r4 rule 4 & 9 apply. We can make 2 observations:

1. Rules 4 & 9 are inconsistent.
2. The decision table is nondeterministic.

→ Rules 4 & 9 are inconsistent because the action sets are different. The whole table is nondeterministic because there is no way to decide whether to apply rule 4 or rule 9.

Testcases for the Triangle Problem:

Table 7.11 Test Cases from Table 7.3

Case ID	a	b	c	Expected Output
DT1	4	1	2	Not a Triangle
DT2	1	4	2	Not a Triangle
DT3	1	2	4	Not a Triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

Testcases for the NextDate function:

→ NextDate Problem illustrates dependencies in the input domain. This makes a perfect example for decision table - based testing.

First try:

Table 7.12 First Try Decision Table with 256 Rules

Conditions												
c1: month in M1?	T											
c2: month in M2?		T										
c3: month in M3?			T									
c4: day in D1?												
c5: day in D2?												
c6: day in D3?												
c7: day in D4?												
c8: year in Y1?												
a1: impossible												
a2: next date												

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year is a leap year}
Y2 = {year : year is not a leap year}

→ To highlight impossible combinations we could make a limited entry decision table. The decision table will have 256 rules, may of which will be impossible.

Second try:

M1 = {month : month has 30 days}
M2 = {month : month has 31 days}
M3 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 28}
D2 = {day : day = 29}
D3 = {day : day = 30}
D4 = {day : day = 31}
Y1 = {year : year = 2000}
Y2 = {year : year is a leap year}
Y3 = {year : year is a common year}

Table 7.13 Second Try Decision Table with 36 Rules

	1	2	3	4	5	6	7	8
c1: month in	M1	M1	M1	M1	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D1	D2	D3	D4
c3: year in	—	—	—	—	—	—	—	—
Rule count	3	3	3	3	3	3	3	3
actions								
a1: impossible				X				
a2: increment day	X	X			X	X	X	
a3: reset day			X					X
a4: increment month			X					?
a5: reset month								?
a6: increment year								?

	9	10	11	12	13	14	15	16	17
c1: month in	M3	M3	M3	M3	M3	M3	M3	M3	M3
c2: day in	D1	D1	D1	D2	D2	D2	D3	D3	D4
c3: year in	Y1	Y2	Y3	Y1	Y2	Y3	—	—	—
Rule count	1	1	1	1	1	1	3	3	3
actions									
a1: impossible						X	X	X	X
a2: increment day		X							
a3: reset day	X		X	X	X				
a4: increment month	X		X	X	X				
a5: reset month									
a6: increment year									

→ The above table illustrates more conditions

Third Try:

M1 = {month : month has 30 days}
M2 = {month : month has 31 days except December}
M3 = {month : month is December}
M4 = {month : month is February}
D1 = {day : 1 ≤ day ≤ 27}
D2 = {day : day = 28}
D3 = {day : day = 29}
D4 = {day : day = 30}
D5 = {day : day = 31}
Y1 = {year : year is a leap year}
Y2 = {year : year is a common year}

Table 7.14 Decision Table for the NextDate Function

	1	2	3	4	5	6	7	8	9	10
c1: month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5
c3: year in	–	–	–	–	–	–	–	–	–	–
actions										
a1: impossible					X					
a2: increment day	X	X	X			X	X	X	X	
a3: reset day				X						X
a4: increment month				X						X
a5: reset month										
a6: increment year										

	11	12	13	14	15	16	17	18	19	20	21	22
c1: month in	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2: day in	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: year in	–				–	–	Y1	Y2	Y1	Y2	–	–
actions												
a1: impossible										X	X	X
a2: increment day	X	X	X	X		X	X					
a3: reset day					X			X	X			
a4: increment month								X	X			
a5: reset month					X							
a6: increment year					X							

We can use the algebra of decision tables to further simplify these 22 testcases. If the action sets of 2 rules in a limited entry decision table are identical, these must be atleast one conditions that allows a rule to be combined with a dont care entry. ~~This is the decision table equivalent of the~~

→ for example rules 1, 2, 3 involve day classes D1, D2, D3 for 30-day months. These can be combined similarly for day classes D1, D2, D3 and D4 in the 31-day month rules and D4 & D5 for February. The result is the following table.

Table 7.15 Reduced Decision Table for the NextDate Function

	1–3	*4*	*5*	*6–9*	*10*
c1: month in	M1	M1	M1	M2	M2
c2: day in	D1, D2, D3	D4	D5	D1, D2, D3, D4	D5
c3: year in	—	—	—	—	—
actions					
a1: Impossible			X		
a2: Increment day	X			X	
a3: reset day		X			X
a4: Increment month		X			X
a5: reset month					
a6: Increment year					

	11–14	*15*	*16*	*17*	*18*	*19*	*20*	*21, 22*
c1: month in	M3	M3	M4	M4	M4	M4	M4	M4
c2: day in	D1, D2, D3, D4	D5	D1	D2	D2	D3	D3	D4, D5
c3: year in	—	—	—	Y1	Y2	Y1	Y2	—
actions								
a1: Impossible							X	X
a2: Increment day	X		X	X				
a3: reset day		X			X	X		
a4: Increment month					X	X		
a5: reset month		X						
a6: Increment year		X						

Table 7.16 Decision Table Test Cases for NextDate

Case ID	*Month*	*Day*	*Year*	*Expected Output*
1–3	April	15	2001	April 16, 2001
4	April	30	2001	May 1, 2001
5	April	31	2001	Impossible
6–9	January	15	2001	January 16, 2001
10	January	31	2001	February 1, 2001
11–14	December	15	2001	December 16, 2001
15	December	31	2001	January 1, 2002
16	February	15	2001	February 16, 2001
17	February	28	2004	February 29, 2004
18	February	28	2001	March 1, 2001
19	February	29	2004	March 1, 2004
20	February	29	2001	Impossible
21, 22	February	30	2001	Impossible

The corresponding testcases are shown in the table above.