Combinatorial Testing

NPTEL

Combinatorial Testing: Motivation

- The behavior of a program may be affected by many factors:
  - **Input parameters,**
  - **Environment configurations (global variables),**
  - **State variables. ..**
- Equivalence partitioning of an input variable:
  - Identify the possible types of input values requiring different processing.
- If the factors are many:
  - It is impractical to test all possible combinations of values of all factors.

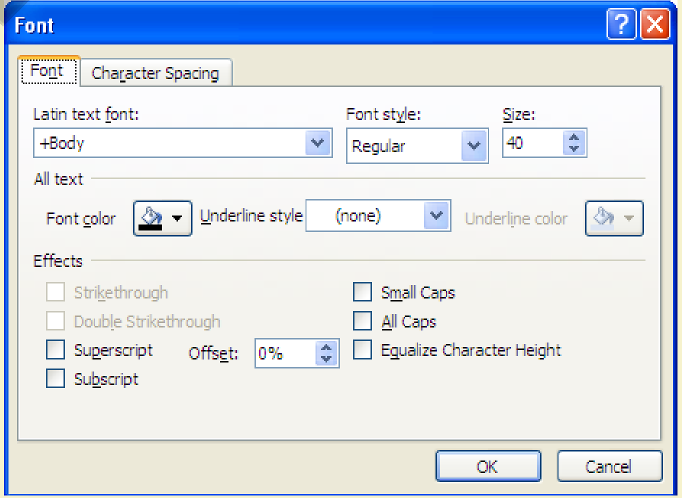
Combinatorial: Relating to, or involving combinations

NPTEL

Combinatorial Testing: Motivation

- Many times, the specific action to be performed depends on the value of a set of Boolean variable:
  - Controller applications
  - User interfaces

NPTEL


Font

Font | Character Spacing

Latin text font: +Body
Font style: Regular
Size: 40

All text
Font color
Underline style: (none)
Underline color

Effects
☐ Strikethrough
☐ Double Strikethrough
☐ Superscript Offset: 0%
☐ Subscript
☐ Small Caps
☐ All Caps
☐ Equalize Character Height

OK Cancel

Combinatorial Testing

- Several combinatorial testing strategies exist:
  - Decision table-based testing
  - Cause-effect graphing
  - Pair-wise testing (reduced number of test cases)

NPTEL

- Applicable to requirements involving conditional actions.
- This is represented as a decision table:
  - Conditions = inputs
  - Actions = outputs
  - Rules =test cases
- Assume independence of inputs
- Example
  - If c1 AND c2 OR c3 then A1

Decision table-based Testing (DTT)

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No	No	Yes

NPTEL

Combinations

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No	No	Yes

Conditions

Actions

NPTEL

- A decision table consists of a number of columns (rules) that comprise all test situations
- Example: the triangle problem
  - 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: Right angled

Sample Decision table

	r1	r2	...				rn
C1	0	1					0
c2	-	1					0
C3	-	1					1
C4	-	1					0
a1	1	0					0
a2	0	0					1
a3	0	0					0
a4	0	1					0
a5	0	0					

NPTEL

Test cases from Decision Tables

Test Case ID	a	b	c	Expected output
TC1	4	1	2	Not a Triangle
TC2	2888	2888	2888	Equilateral
TC3	?	|	)	Impossible
TC4				
…				
TC11				

C1: a, b,c form a triangle

C2: a=b

C3: a= c

C4: b= c

NPTEL

More Complete Decision Table for the Triangle Problem

Conditions											
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
Actions											
A1: Not a Triangle	X	X	X								
A2: Scalene											X
A3: Isosceles							X		X	X	
A4: Equilateral				X							
A5: Impossible					X	X		X			

NPTEL

Test Cases for the Triangle Problem

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

NPTEL

Decision Table – Example 2

Printer Troubleshooting

Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

NPTEL

Quiz: Develop BB Test Cases

- Policy for charging customers for certain in-flight services:

If the flight is more than half-full and ticket cost is more than Rs. 3000, free meals are served unless it is a domestic flight. Otherwise, no meals are served. Meals are charged on all domestic flights.

NPTEL

Fill all combinations in the table.

		POSSIBLE COMBINATIONS							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than Rs.3000 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS									

NPTEL

Analyze column by column to determine which actions are appropriate for each combination

		POSSIBLE COMBINATIONS							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than Rs. 3000 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	*serve meals*					Y	Y	Y	Y
	free							Y	

NPTEL

Reduce the table by eliminating redundant columns.

		POSSIBLE COMBINATIONS							
CONDITONS	more than half-full	N	N	N	N	Y	Y	Y	Y
	more than Rs. 3000 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	serve meals					X	X	X	X
	free							X	

NPTEL

Final solution

		Combinations			
CONDITONS	more than half-full	N	Y	Y	Y
	more than 3000 per seat	-	N	Y	Y
	domestic flight	-	-	N	Y
ACTIONS	serve meals		X	X	X
	free			X	

NPTEL

Assumptions regarding rules

–Rules need to be complete:

- **That is, every combination of decision table values including default combinations are present.**

–Rules need to be consistent:

- That is, there is no two different actions for the same combinations of conditions

NPTEL

Guidelines and Observations

- Decision table testing is appropriate for programs:
  - There is a lot of decision making
  - Output is a logical relationship among input variables
  - Results depend on calculations involving subsets of inputs
  - There are cause and effect relationships between input and output
- Decision tables do not scale up very well

NPTEL

Quiz: Design test Cases

- Customers on a e-commerce site get following discount:
  - A member gets 10% discount for purchases lower than Rs. 2000, else 15% discount
  - Purchase using SBI card fetches 5% discount
  - If the purchase amount after all discounts exceeds Rs. 2000/- then shipping is free.

NPTEL

Cause-effect Graphs

- Overview:
  - Explores combinations of possible inputs
  - Specific combination of inputs (causes) results in outputs (effects)
  - Represented as nodes of a cause effect graph
  - The graph also includes constraints and a number of intermediate nodes linking causes and effects

NPTEL

- If depositing less than Rs. 1 Lakh, rate of interest:
  - 6% for deposit upto 1 year
  - 7% for deposit over 1 year but less than 3 yrs
  - 8% for deposit 3 years and above
- If depositing more than Rs. 1 Lakh, rate of interest:
  - 7% for deposit upto 1 year
  - 8% for deposit over 1 year but less than 3 yrs
  - 9% for deposit 3 years and above

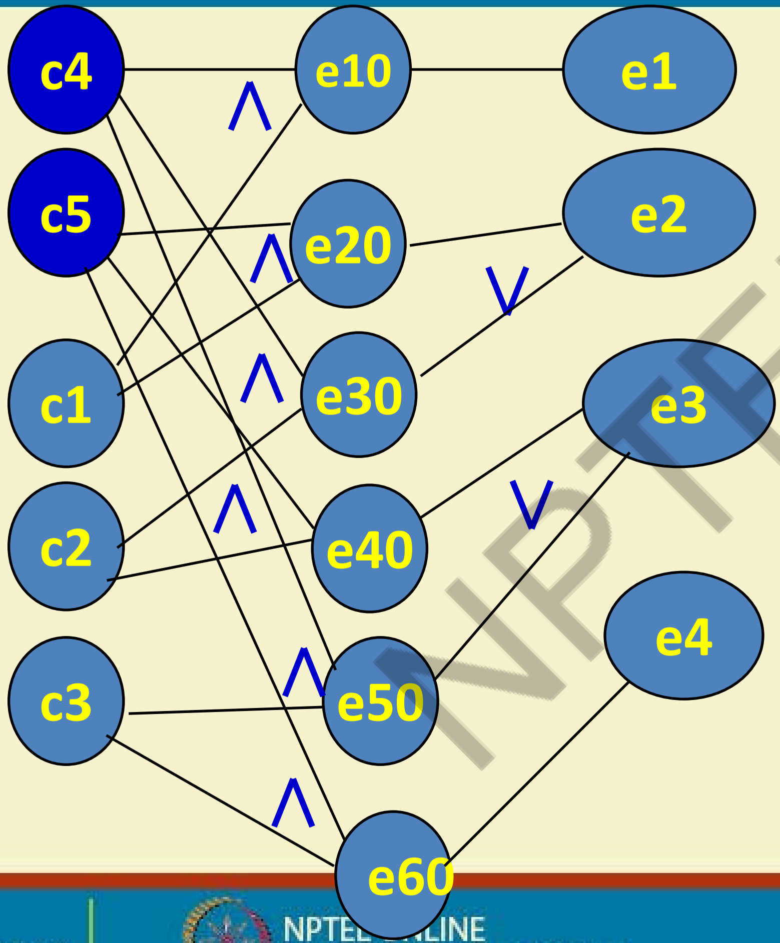
Cause-Effect Graph Example

NPTEL

Cause-Effect Graph Example

Causes	**Effects**
C1: Deposit<1yr	e1: Rate 6%
C2: 1yr<Deposit<3yrs	e2: Rate 7%
C3: Deposit>3yrs	e3: Rate 8%
C4:Deposit <1 Lakh	e4: Rate 9%
C5: Deposit >=1Lakh	

NPTEL



Cause-Effect Graphing

NPTEL

Develop a Decision Table

C1	C2	C3	C4	C5	e1	e2	e3	e4
1	0	0	1	0	1	0	0	0
1	0	0	0	1	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	0	0	1	1	0	1	0

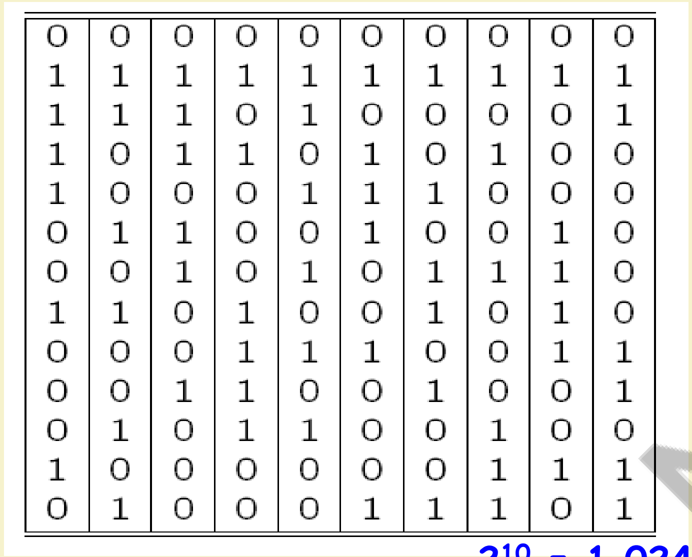
- Convert each row to a test case

NPTEL

Pair-wise Testing

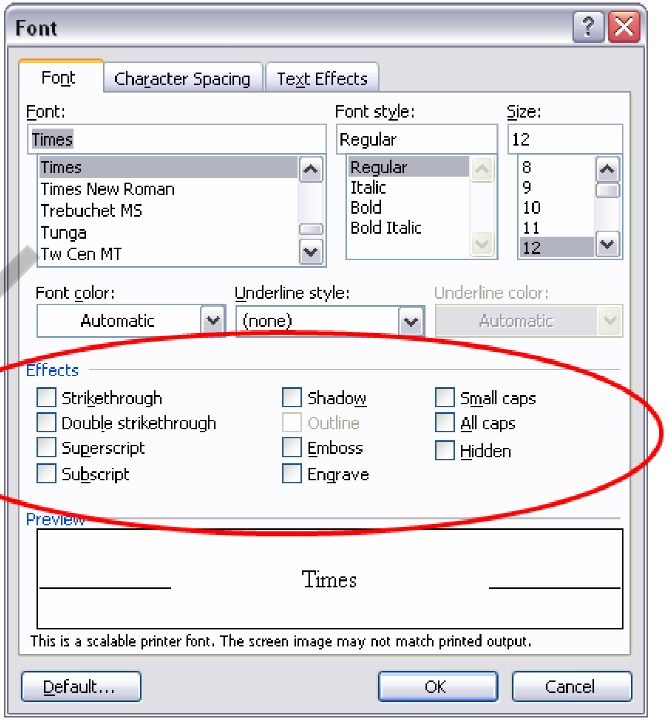
NPTEL

Combinatorial Testing of User Interface



0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

0 = effect off
1 = effect on



Font

Font | Character Spacing | Text Effects

Font: Times
Font style: Regular
Size: 12

Times, Times New Roman, Trebuchet MS, Tunga, Tw Cen MT
Regular, Italic, Bold, Bold Italic
8, 9, 10, 11, 12

Font color: Automatic
Underline style: (none)
Underline color: Automatic

Effects

- [ ] Strikethrough
- [ ] Double strikethrough
- [ ] Superscript
- [ ] Subscript
- [ ] Shadow
- [ ] Outline
- [ ] Emboss
- [ ] Engrave
- [ ] Small caps
- [ ] All caps
- [ ] Hidden

Preview

Times

This is a scalable printer font. The screen image may not match printed output.

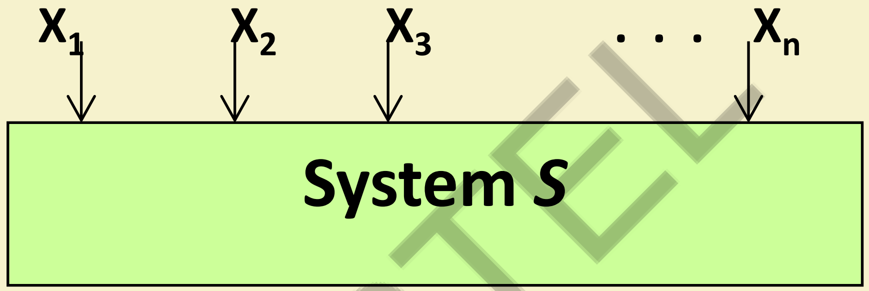
Default... | OK | Cancel

2^{10} = 1,024 tests for all combinations

* 10 ^{3} = 1024 * 1000 …. Just too many to tests

NPTEL

Combinatorial Testing Problem



•Combinatorial testing problems generally follow a simple input-process-output model;

•The “state” of the system is not the focus of combinatorial testing.

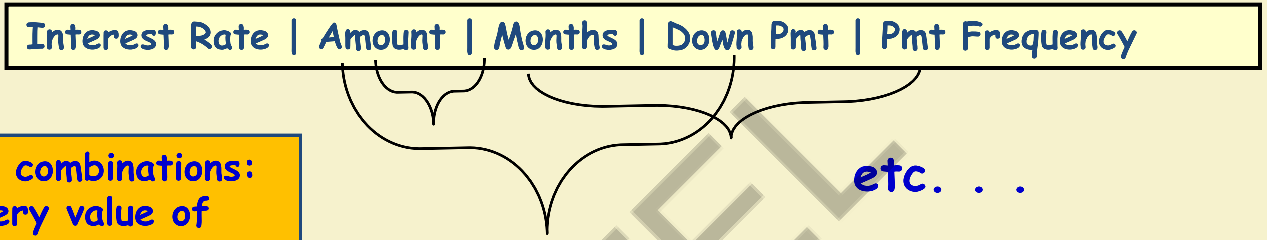
NPTEL

t-way Testing

- Instead of testing all possible combinations:
  - A subset of combinations is generated.
- Key observation:
  - **It is often the case that a fault is caused by interactions among a few factors.**
- t-way testing can dramatically reduce the number of test cases:
  - but remains effective in terms of fault detection.

NPTEL

t-way Interaction Testing



All combinations: every value of every parameters

All pairs: every value of each pair of parameters

etc. . . .

t-way interactions: every value of every t-way combination of parameters

NPTEL

Pairwise Testing

Pressure	Temperature	Velocity	Acceleration	Air Density
A	T1	1	10	1.1
B	T2	2	0	2.1
	T3	3	20	3.1
		4	0	
		5		
		6		

Pressure	Temperature
A	T1
A	T2
A	T3
B	T1
B	T2
B	T3

NPTEL

Pairwise Reductions

Number of inputs	Number of selected test data values	Number of combinations	Size of pair wise test set
7	2	128	8
13	3	1.6×10^{6}	15
40	3	1.2×10^{19}	21

NPTEL

Fault-Model

- **A t-way interaction fault:**
  - Triggered by a certain combination of t input values.
  - A simple fault is a 1-way fault
  - Pairwise fault is a t-way fault where t = 2.
- **In practice, a majority of software faults consist of simple and pairwise faults.**

NPTEL

Single-mode Bugs

- The simplest bugs are single-mode faults:
  - **Occur when one option causes a problem regardless of the other settings**
  - **Example:** A printout is always gets smeared when you choose the duplex option in the print dialog box
    - Regardless of the printer or the other selected options

NPTEL

Double-mode Faults

- **Double-mode faults**
  - Occurs when two options are combined
  - **Example:** The printout is smeared only when duplex is selected and the printer selected is model 394

NPTEL

Multi-mode Faults

- **Multi-mode faults**
  - Occur when three or more settings produce the bug
  - This is the type of problems that make complete coverage necessary

NPTEL

Example of Pairwise Fault

```
• begin
  – int x, y, z;
  – input (x, y, z);
  – if (x == x1 and y == y2)
      • output (f(x, y, z));
  – else if (x == x2 and y == y1)
      • output (g(x, y));
  – Else          // Missing (x == x2 and y == y1) f(x, y, z) – g(x, y);
      • output (f(x, y, z) + g(x, y))
• end
```

- **Expected**: x = x1 and y = y1 => f(x, y, z) – g(x, y);

  x = x2, y = y2 => f(x, y, z) + g(x, y)

NPTEL

Example: Android smart phone testing

• Apps should work on all combinations of platform options, but there are 3 x 3 x 4 x 3 x 5 x 4 x 4 x 5 x 4 = 172,800 configurations

HARDKEYBOARDHIDDEN_NO
HARDKEYBOARDHIDDEN_UNDEFINED
HARDKEYBOARDHIDDEN_YES

KEYBOARDHIDDEN_NO
KEYBOARDHIDDEN_UNDEFINED
KEYBOARDHIDDEN_YES

KEYBOARD_12KEY
KEYBOARD_NOKEYS
KEYBOARD_QWERTY
KEYBOARD_UNDEFINED

NAVIGATIONHIDDEN_NO
NAVIGATIONHIDDEN_UNDEFINED

NAVIGATIONHIDDEN_YES

NAVIGATION_DPAD
NAVIGATION_NONAV
NAVIGATION_TRACKBALL
NAVIGATION_UNDEFINED
NAVIGATION_WHEEL

ORIENTATION_LANDSCAPE
ORIENTATION_PORTRAIT
ORIENTATION_SQUARE
ORIENTATION_UNDEFINED

SCREENLAYOUT_LONG_MASK
SCREENLAYOUT_LONG_NO
SCREENLAYOUT_LONG_UNDEFINED

SCREENLAYOUT_LONG_YES

SCREENLAYOUT_SIZE_LARGE
SCREENLAYOUT_SIZE_MASK
SCREENLAYOUT_SIZE_NORMAL

SCREENLAYOUT_SIZE_SMALL
SCREENLAYOUT_SIZE_UNDEFINED

TOUCHSCREEN_FINGER
TOUCHSCREEN_NOTOUCH
TOUCHSCREEN_STYLUS
TOUCHSCREEN_UNDEFINED

NPTEL

White-Box Testing

NPTEL

What is White-box Testing?

- White-box test cases designed based on:
  - Code structure of program.
  - White-box testing is also called structural testing.

NPTEL

White-Box Testing Strategies

- **Coverage-based:**
  - Design test cases to cover certain program elements.
- **Fault-based:**
  - Design test cases to expose some category of faults

NPTEL

- Several white-box testing strategies have become very popular :
  - Statement coverage
  - Branch coverage
  - Path coverage
  - Condition coverage
  - MC/DC coverage
  - Mutation testing
  - Data flow-based testing

White-Box Testing

NPTEL

Why Both BB and WB Testing?

Black-box	White-box
• Impossible to write a test case for every possible set of inputs and outputs • Some code parts may not be reachable • **Does not tell if extra functionality has been implemented.**	• Does not address the question of whether a program matches the specification • Does not tell if all functionalities have been implemented • **Does not uncover any missing program logic**

NPTEL

Coverage-Based Testing Versus Fault-Based Testing

- Idea behind coverage-based testing:
  - Design test cases so that certain program elements are executed (or covered).
  - Example: statement coverage, path coverage, etc.
- Idea behind fault-based testing:
  - Design test cases that focus on discovering certain types of faults.
  - Example: Mutation testing.

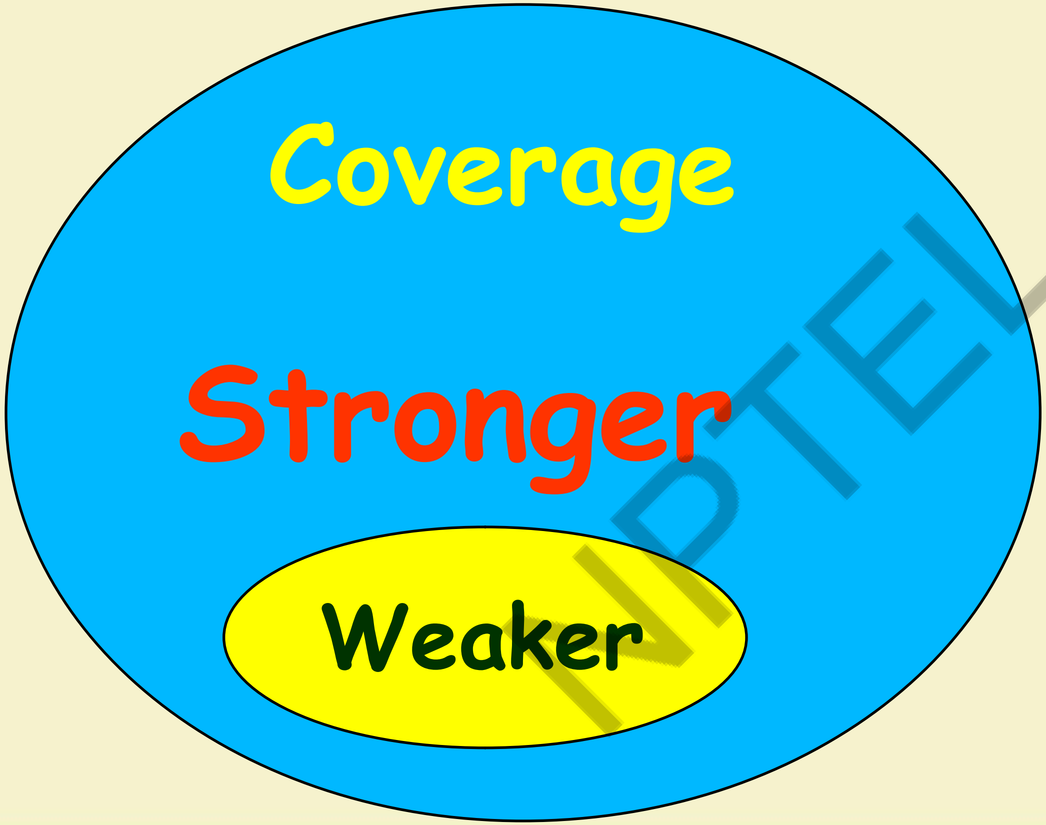
NPTEL

- **Statement:** each statement executed at least once
- **Branch:** each branch traversed (and every entry point taken) at least once
- **Condition:** each condition True at least once and False at least once
- **Multiple Condition:** All combination of Condition covered
- **Path**:
- **Dependency:**

Types of program element Coverage

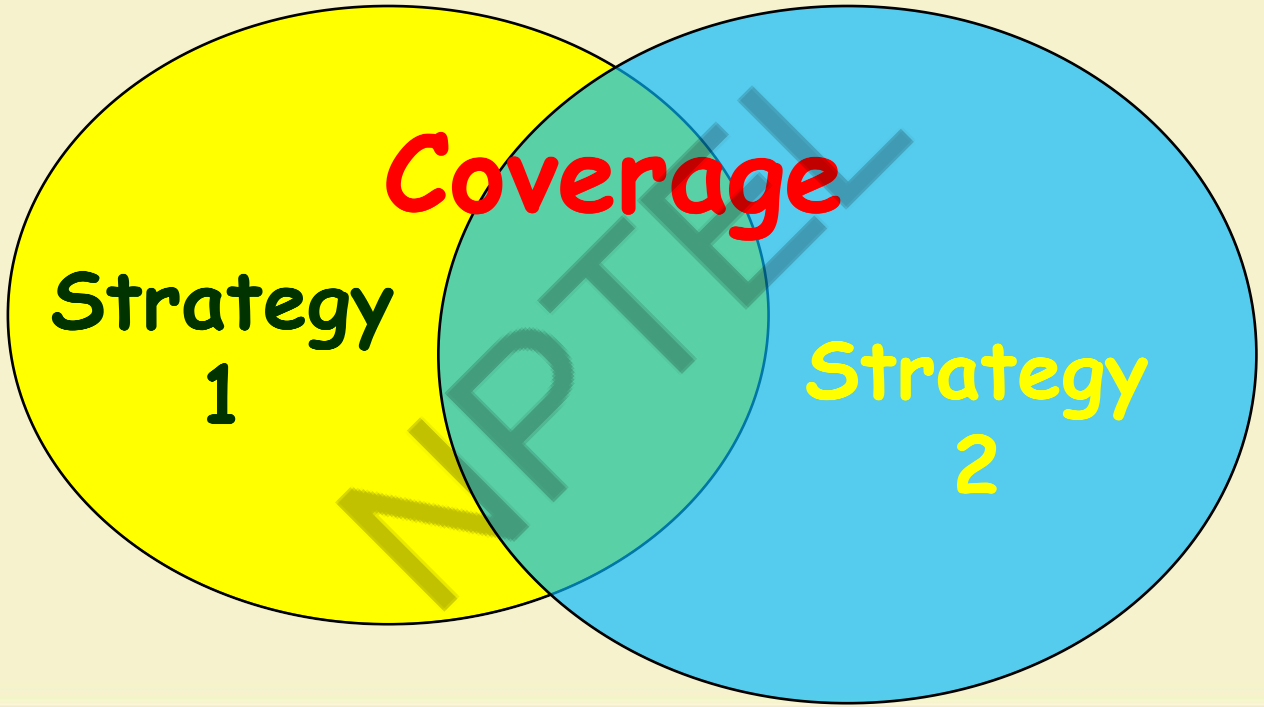
NPTEL

Stronger and Weaker Testing



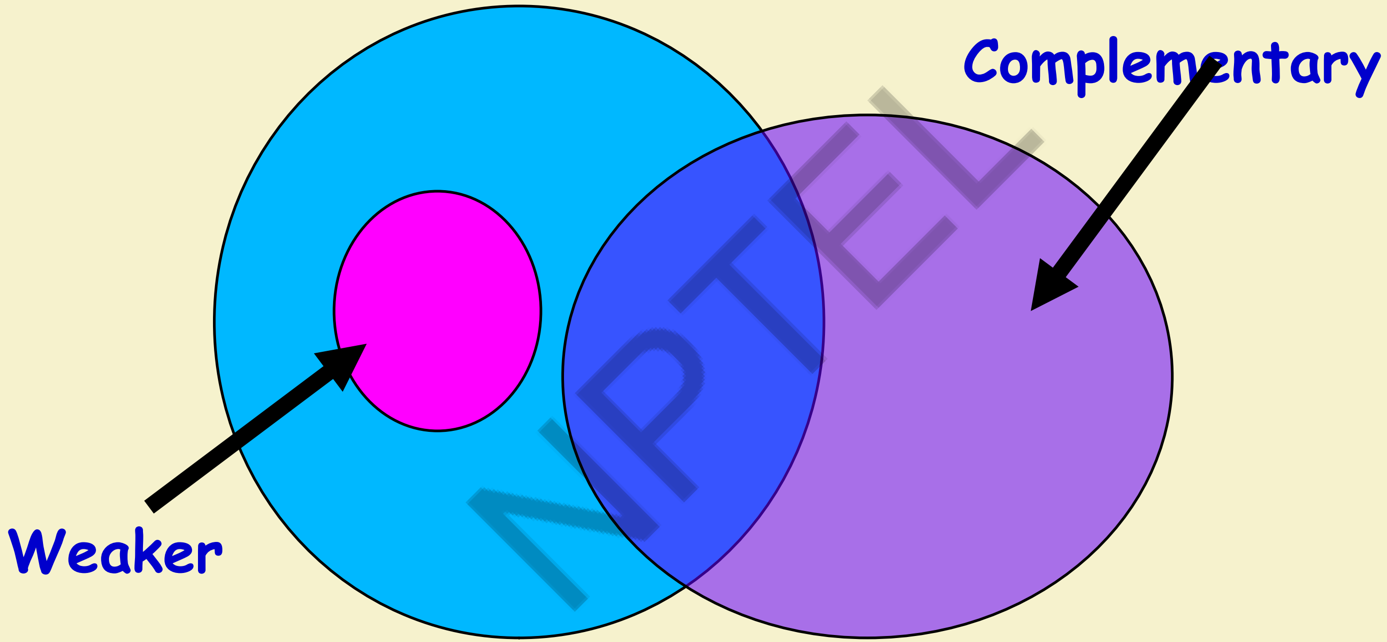
NPTEL

Complementary Testing



NPTEL

Stronger, Weaker, and Complementary Testing



NPTEL

Statement Coverage

- Statement coverage strategy:
  - Design test cases so that every statement in the program is executed at least once.

NPTEL

Statement Coverage

- The principal idea:
  - –Unless a statement is executed,
  - –We have no way of knowing if an error exists in that statement.

NPTEL

Statement Coverage Criterion

- However, observing that a statement behaves properly for one input value:
  - **No guarantee that it will behave correctly for all input values!**

NPTEL

Statement Coverage

- Coverage measurement:

$$\frac{\text{\# executed statements}}{\text{\# statements}}$$

- **Rationale:** a fault in a statement can only be revealed by executing the faulty statement

NPTEL

Example

- int f1(int x, int y){
- 1 while (x != y){
- 2 if (x>y) **then**
- 3 x=x-y;
- 4 else y=y-x;
- 5 }
- 6 return x; }

Euclid's GCD Algorithm

NPTEL

Example

```
int f1(int x,int y){
1 while (x != y){
2    if (x>y) then
3          x=x-y;
4    else y=y-x;
5  }
6 return x;        }
```

Euclid's GCD Algorithm

NPTEL

Euclid's GCD Algorithm

- By choosing the test set {(x=3,y=3),(x=4,y=3),(x=3,y=4)}
  - All statements are executed at least once.

NPTEL

Branch Coverage

- **Also called decision coverage.**
- Test cases are designed such that:
  - Each branch condition
    - Assumes true as well as false value.

NPTEL

Example

```
int f1(int x,int y){
1 while (x != y){
2     if (x>y) then
3         x=x-y;
4     else y=y-x;
5  }
6 return x;       }
```

NPTEL

Example

- Test cases for branch coverage can be:
- {(x=3,y=3),(x=3,y=2), (x=4,y=3), (x=3,y=4)}

NPTEL

Branch Testing

- **Adequacy criterion:** Each branch (edge in the CFG) must be executed at least once
- Coverage:

$$\frac{\#\ \text{executed branches}}{\#\ \text{branches}}$$

NPTEL

Quiz 1: Branch and Statement Coverage: Which is Stronger?

- Branch testing guarantees statement coverage:
  - A stronger testing compared to the statement coverage-based testing.

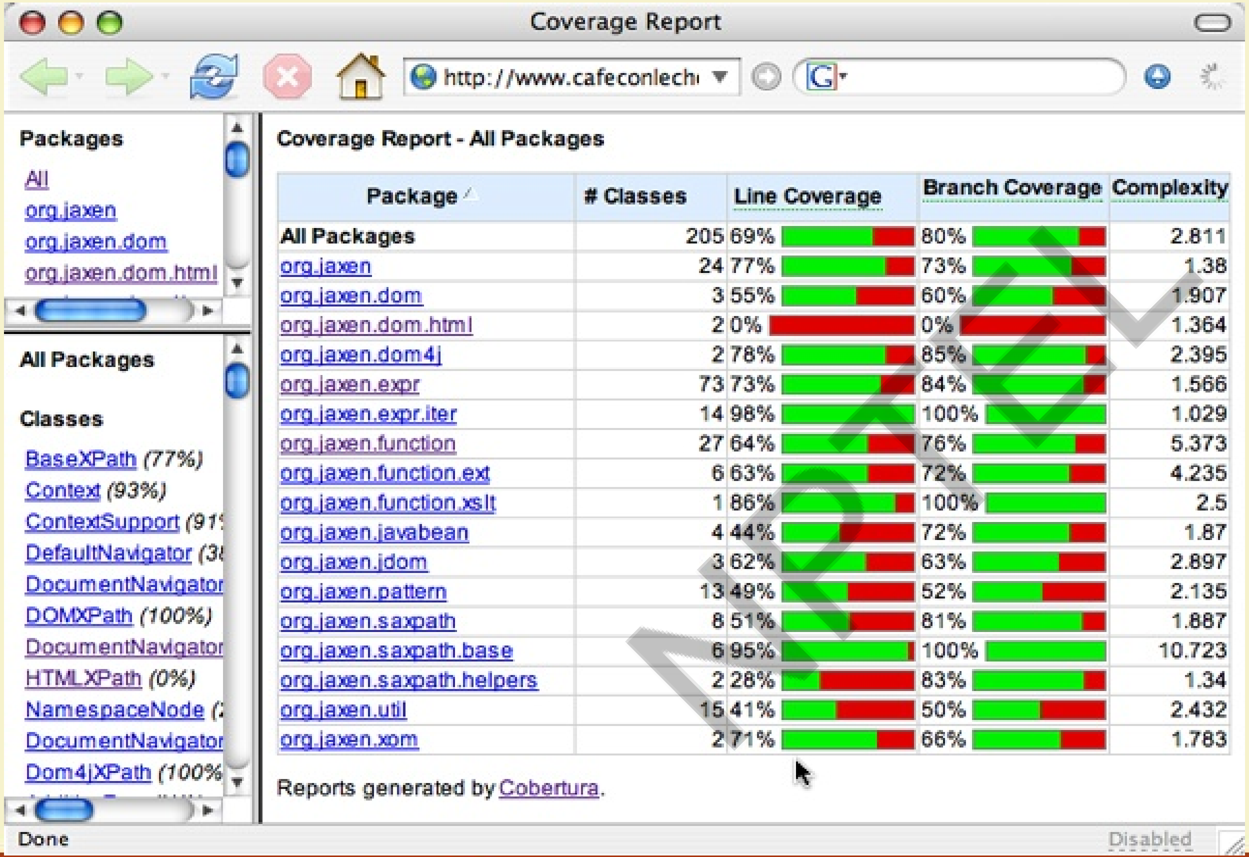
NPTEL

Stronger Testing

- Stronger testing:
  - –Superset of weaker testing
  - –A stronger testing covers all the elements covered by a weaker testing.
  - –Covers some additional elements not covered by weaker testing

NPTEL

Sample Coverage Report

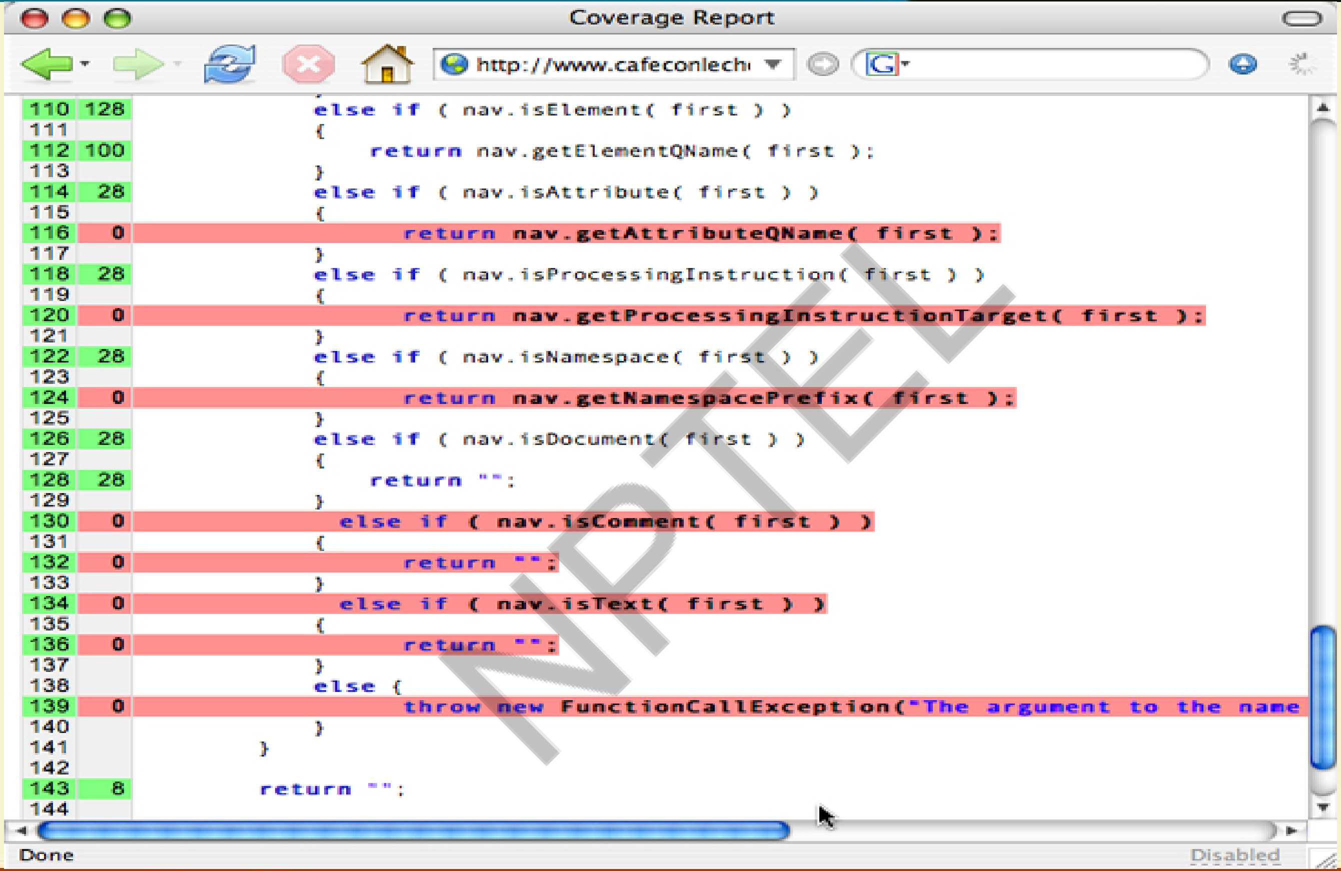


Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	205	69%	80%	2.811
org.jaxen	24	77%	73%	1.38
org.jaxen.dom	3	55%	60%	1.907
org.jaxen.dom.html	2	0%	0%	1.364
org.jaxen.dom4j	2	78%	85%	2.395
org.jaxen.expr	73	73%	84%	1.566
org.jaxen.expr.iter	14	98%	100%	1.029
org.jaxen.function	27	64%	76%	5.373
org.jaxen.function.ext	6	63%	72%	4.235
org.jaxen.function.xslt	1	86%	100%	2.5
org.jaxen.javabean	4	44%	72%	1.87
org.jaxen.jdom	3	62%	63%	2.897
org.jaxen.pattern	13	49%	52%	2.135
org.jaxen.saxpath	8	51%	81%	1.887
org.jaxen.saxpath.base	6	95%	100%	10.723
org.jaxen.saxpath.helpers	2	28%	83%	1.34
org.jaxen.util	15	41%	50%	2.432
org.jaxen.xom	2	71%	66%	1.783

Reports generated by Cobertura.

NPTEL



Coverage Report

http://www.cafeconlech

```
110  128            else if ( nav.isElement( first ) )
111                  {
112  100                return nav.getElementQName( first );
113                  }
114   28            else if ( nav.isAttribute( first ) )
115                  {
116    0                   return nav.getAttributeQName( first );
117                  }
118   28            else if ( nav.isProcessingInstruction( first ) )
119                  {
120    0                   return nav.getProcessingInstructionTarget( first );
121                  }
122   28            else if ( nav.isNamespace( first ) )
123                  {
124    0                   return nav.getNamespacePrefix( first );
125                  }
126   28            else if ( nav.isDocument( first ) )
127                  {
128   28                return "";
129                  }
130    0              else if ( nav.isComment( first ) )
131                  {
132    0                   return "";
133                  }
134    0              else if ( nav.isText( first ) )
135                  {
136    0                   return "";
137                  }
138                  else {
139    0                   throw new FunctionCallException("The argument to the name
140                  }
141              }
142
143    8         return "";
144
```

Done

Disabled

NPTEL

Statements vs Branch Testing

- Traversing all edges of a graph causes all nodes to be visited
  - So a test suite that satisfies branch adequacy criterion also satisfies statement adequacy criterion for the same program.
- The converse is not true:
  - A statement-adequate (or node-adequate) test suite may not be branch-adequate (edge-adequate).

NPTEL

- Statement coverage
- Branch coverage (aka decision coverage)
- Basic condition coverage
- Condition/Decision coverage
- Multiple condition coverage
- MC/DC coverage
- Path coverage
- Data flow-based testing
- Mutation testing

White-box Testing

NPTEL

All Branches can still miss testing specific conditions

- Assume failure occurs when c==DIGIT

```
if((c == ALPHABET) || (c ==DIGIT))
```

- Branch adequacy criterion can be satisfied by c==alphabet and c==splchar
  - **The faulty sub-expression might not be tested!**
  - Even though we test both outcomes of the branch

NPTEL

Basic Condition Coverage

- Also called condition coverage or simple condition coverage .
- Test case design: **((c = = ALPHABET) || (c= = DIGIT))**
  - –Each component of a composite conditional expression
    - Made to assume both true and false values.

NPTEL

Basic Condition Testing

- **Simple or (basic) Condition Testing:**
  - Test cases make each atomic condition assume T and F values
  - Example: **if (a>10 && b<50)**
- **Following test inputs would achieve basic condition coverage**
  - **a=15, b=30**
  - **a=5, b=60**
- Does basic condition coverage subsume decision coverage?

NPTEL

Example: BCC

- Consider the conditional expression
  - **((c1.and.c2).or.c3):**
- Each of c1, c2, and c3 is exercised with all possible values,
  - That is, given true and false values.

NPTEL

Basic condition testing

- Adequacy criterion: each basic condition must be executed at least once
- Coverage:

$$\frac{\text{\# truth values taken by all basic conditions}}{2 * \text{\# basic conditions}}$$

NPTEL

Is BCC Stronger than Decision Coverage?

- Consider the conditional statement:
  - **If(((a>5).and.(b<3)).or.(c==0)) a=10;**
- Two test cases can achieve basic condition coverage: (a=10, b=2, c=2) and (a=1, b=10, c=0)
- **BCC does not imply Decision coverage and vice versa**

NPTEL

Condition/Decision Coverage Testing

- **Condition/decision coverage:**
  - Each atomic condition made to assume both T and F values
  - Decisions are also made to get T an F values
- **Multiple condition coverage (MCC):**
  - Atomic conditions made to assume all possible combinations of truth values

NPTEL

MCC

- Test cases make Conditions to assume all possible combinations of truth values.
- Consider: **if (a || b && c) then …**

Test	a	b	c
(1)	T	T	T
(2)	T	T	F
(3)	T	F	T
(4)	T	F	F
(5)	F	T	T
(6)	T	T	F
(7)	F	F	T
(8)	F	F	F

Exponential in the number of basic conditions

NPTEL

Multiple Condition Coverage (MCC)

- Consider a Boolean expression having n components:
  - **For condition coverage we require 2^n test cases.**
- MCC testing technique:
  - Practical only if n (the number of component conditions) is small.

NPTEL

MCC for Compound conditions: Exponential complexity

(((a || b) && c) || d) && e

$2^5=32$

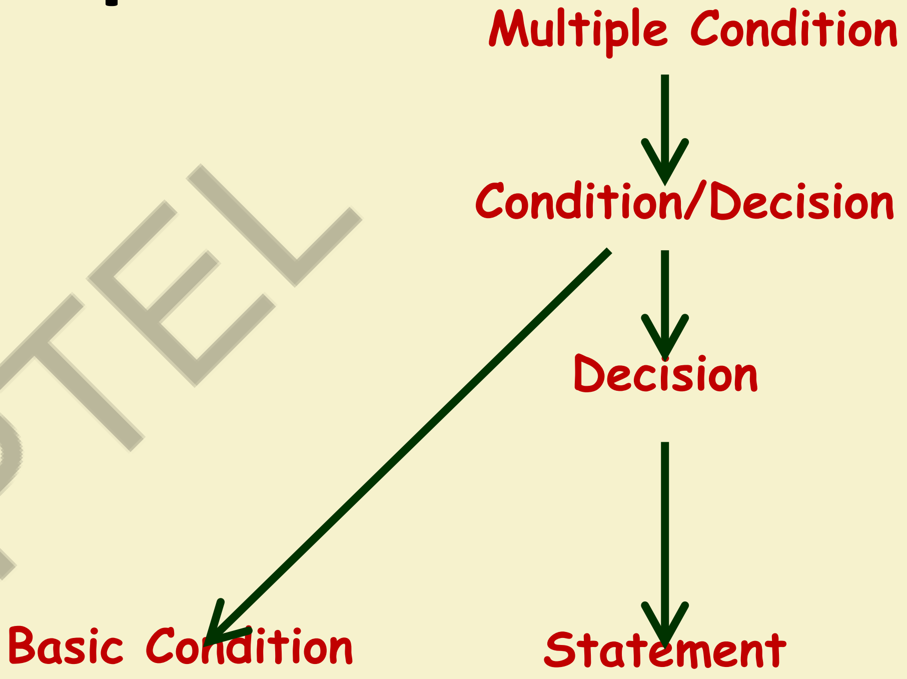
Test Case	a	b	c	d	e
(1)	T	—	T	—	T
(2)	F	T	T	—	T
(3)	T	—	F	T	T
(4)	F	T	F	T	T
(5)	F	F	—	T	T
(6)	T	—	T	—	F
(7)	F	T	T	—	F
(8)	T	—	F	T	F
(9)	F	T	F	T	F
(10)	F	F	—	T	F
(11)	T	—	F	F	—
(12)	F	T	F	F	—
(13)	F	F	—	F	—

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

NPTEL

Subsumption

- Condition testing:
  - Stronger testing than branch testing.
- Branch testing:
  - Stronger than statement coverage testing.



NPTEL

Shortcomings of Condition Testing

- **Redundancy of test cases:** Condition evaluation could be compiler-dependent:
  - **Reason: Short circuit evaluation of conditions**
- **Coverage may be Unachievable:** Possible dependencies among variables:
  - Example: **((chr==`A´)||(chr==`E´))** can not both be true at the same time

NPTEL

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)

NPTEL