SOFTWARE TESTING METHODOLOGIES

UNIT-(I)

- INTRODUCTION:
  - PURPOSE OF TESTING
  - DICHOTOMIES
  - MODEL FOR TESTING
  - CONSEQUENCES OF BUGS
  - TAXONOMY OF BUGS
- FLOW GRAPHS AND PATH TESTING
  - BASIC CONCEPTS OF PATH TESTING
  - PREDICATES
  - PATH PREDICATES AND ACHIEVABLE PATHS
  - PATH SENSITIZING
  - PATH INSTRUMENTATION
  - APPLICATION OF PATH TESTING.

* JNTUK Previously Asked Questions.

1. Is prevented bug better than a detected and corrected bug? Justify. What is the purpose of Testing?
2. State and explain various Dichotomies in Software Testing?
3. (a) Write in detail about Structural bugs and data bugs?
   (b) Write a short notes on requirements? features and functionality of bugs?
4. (a) Draw and explain model of testing. Is complete testing Posible? (b) What is the importance and consequences of Bugs?

5. (a) what is meant by program's control flow? How is it useful for path testing?

(b) State and explain various path selection rules?

6. (a) Discuss Traversal marker with an example?

(b) write in detail about heuristic procedure for sensitizing paths?

7. (a) state and explain various kinds of predicate blindness with example?

(b) write in detail about predicate interpretation and predicate coverage?

8. (a) Explain the process of achieving (C1+C2) coverage?

(b) Explain various loops with an example?

9. write a short notes on Taxonomy of Bugs?

INTRODUCTION TO SOFTWARE TESTING

- It is a process used to identify the correctness, completeness quality of developed computer software.
- It also helps to identify errors, missing requirements.
- It can be done either manually or using software tools.
- Testing is done by "Testers".
- Software testing is a verification and validation process.

PURPOSE OF TESTING

Testing definition:

" Testing is the process of executing a program with the intent of finding errors".

(or)

" Testing is the process of evaluating system & System components to verify that it meets requirements.

Myth : "Good programmers write code without bugs".

In reality : In 100 lines of source code 2-3 errors occur.

* 1. productivity and quality in software:

- In Testing process, from starting stage to ending stage different flaws (errors) occur, at that situations product is discarded (change) & back again rework.
- Productivity means sum of costs of meterials + rework + discarded components + quality assurance.
- Quality is depends on quality attributes ie Reliability, usability, performance, security, efficiency etc.

2. Goals of Software Testing :

1. Short term goal
2. Long term goal
3. post implementation goals.

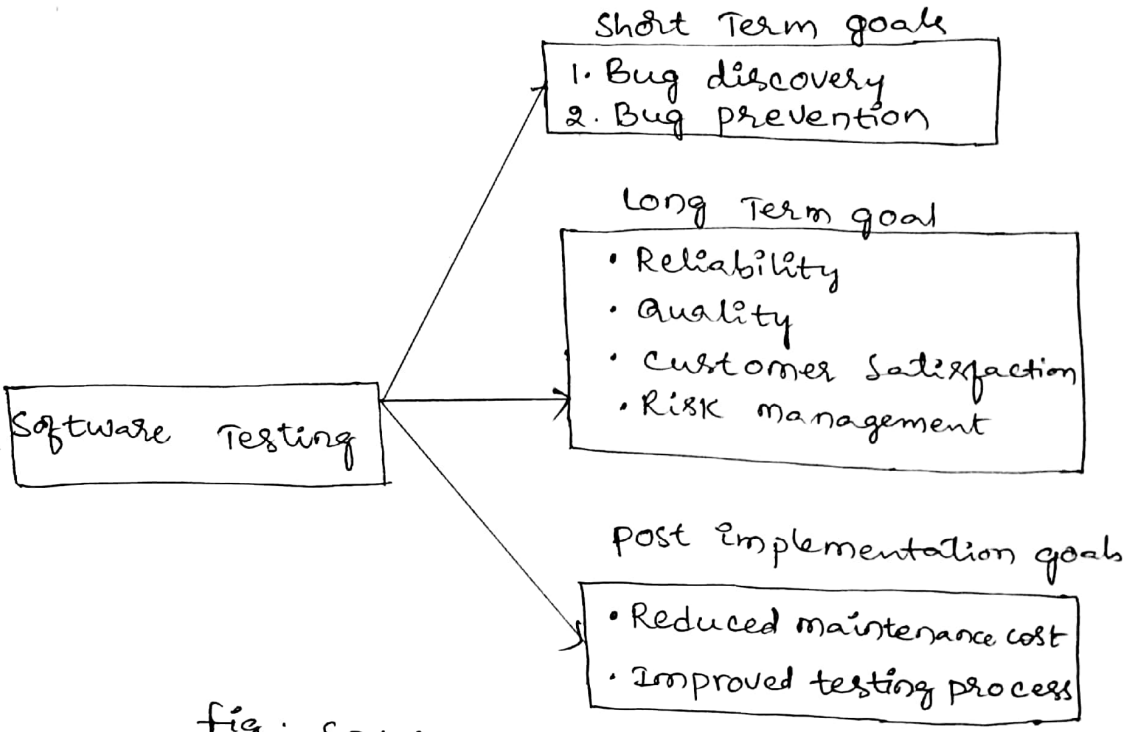


fig : Software Testing goals.

* Bug prevention is better than Bug discovery.

→ A prevented bug is better than a detected and corrected bug.

→ Bug prevention is to identifying the potential risks (predefined errors) (or) feature errors.
→ Bug prevention is the primary goal of software Testing.
→ In bug prevention no rework.

→ A bug discovery has rework.

→ Failed to ~~bug~~ prevent bugs, to discover bugs based on Symptoms.

→ Bug discovery is a process that helps in determining errors in every phases of software development.

3. phases in Tester's mental life :

phase 0 Thinking:

"There is no difference between Testing and debugging".

phase 1 Thinking:

"Software works progress differencess between Testing and debugging".

phase 2 Thinking:

"Software doesn't work".

phase 3 Thinking:

"Risk is reduced".

phase 4 Thinking:

"What makes Software Testable".

* 4. Test design:

→ Basically test design is the act of creating and writing test suites for testing a Software.

→ Test analysis and identifying test conditions gives us a generic idea for testing.

→ Test case generation (Template)

Test case id	purpose	inputs	expected o/p	Actual o/p
TC1				

Different stages (phases) in Software Testing.

1. Debugging oriented phase
2. Demonstration oriented phase
3. Destruction oriented phase
4. Evolution oriented phase
5. prevention oriented phase
6. process oriented phase.

DICHOTOMIES:

Dichotomies means differences (in Testing Terminologies)

* 1. Differences between Testing and debugging.

Testing	Debugging
1. Purpose: program has bugs	1. purpose: Errors that caused program failure.
2. Start: Known conditions, outcome is expected	2. Start: unknown conditions, output is not expected.
3. Testing can be planned and designed	3. Debugging can't be planned.
4. Testing proves programmer's failure.	4. programmer's justification
5. In testing, design and coding Knowledge is not required.	5. In debugging, design and coding knowledge is required.
6. Testing can be done by Tester (or) Automated tools.	6. Debugging can be done by developer and debugging Automated tools dream.

2. Function Vs structure.

Function	Structure.
1. Fuctional testing is treated as "Black box Testing".	1. Structural testing is treated as "White box Testing".
2. It is an external behaviour of system.	2. It is an internal behaviour of system.
3. Both are usefull it have some limitations.	3. Both are usefull it have some limitations.
4. They target different bugs. detect all bugs, but takes infinite time (more times)	4. Bug's can't be completely detected.

* 3. Designer Vs Testers

Designer	Tester
1. Who design the test, that person is called as "Designer".	1. Who ~~design~~ Teste the code, that person is called as "Tester".
2. In function testing designer and tester both are different persons.	2. In structural testing designer and tester both are merged.

* 4. Modularity Vs Efficiency:

Modularity	Efficiency
1. Module is nothing but, in a system smallest component.	1. In modularity, integration process, efficiency is required.
2. Smallest module integration is difficult but program is understanding.	2. Smaller modules has less efficiency required.
3. Largest module integration is easy, but program is not understanding.	3. Larger modules has more efficiency required.
4. Each module has use an independent test cases.	

* 5. Small Vs Large.

Small	Large.
1. constructing program that has few components, written by few programmers.	1. Constructing program that has many components, written by many programers.
2. EX: In Offices, Homes.	EX: Software companies.

* 6. Builder Vs Buyer.

Builder	Buyer.
1. Who is design the system and its accountable to buyer.	1. Who buys systems for profits, services.
2. User: final uses who are benefici by S/W. Tester: Who tests the S/W.	2. Operator who has to live with builder mistakes.

• MODEL FOR TESTING :

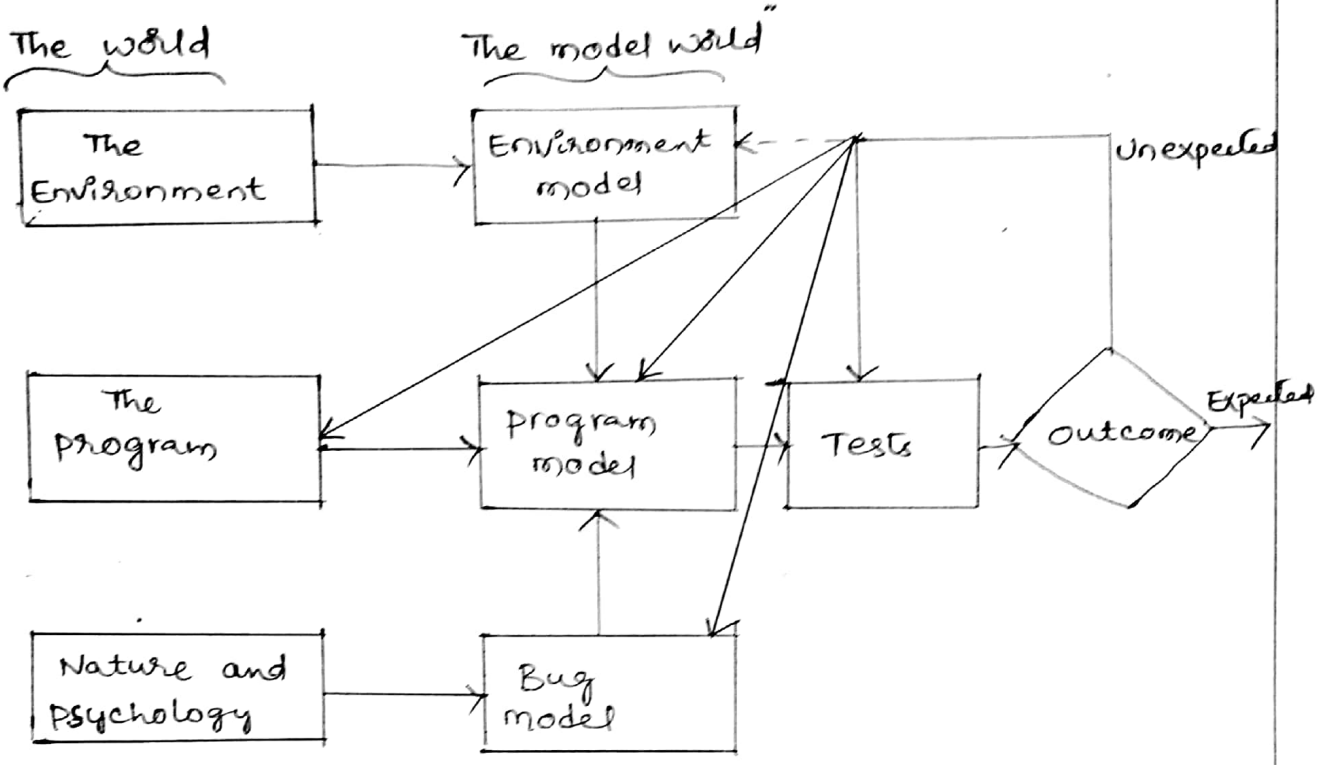


figure : A model for Testing.

Above figure is a model of testing process. It includes Three models.

1. A model of the environment
2. A model of the program
3. A model of the expected bugs.

1. Environment :
   - → A program environment is the hardware and software required to make it run.
   - → It includes all programs, the programs under tests such as OS, linkage, editor, compiler.

2. Program :
   - → complicated to understand in detail.
   - → It deal with a simplified overall view.
   - → The programs are developed by the "developer".

3. Bug :
   - → Bug model categorize the bugs as initialized, wrong variables etc. Bug locality, Kinds of bugs.

Test : Different testing strategies are "unit testing", "Integration Testing".

Outcome : Results is Success (expected) end, Otherwise it will be test environment & program.

. CONSEQUENCES OF BUGS:

Consequences : How bugs may affects users.

1. Mild → Misspelled output or mal-aligned print-out.
2. Moderate → outputs are redundant impacting performance.
3. Annoying → Systems behaviour is dehumanizing for ex: beep sound.
4. Disturbing → Legitimate transactions refused. EX; ATM
5. Serious → Losing track of transactions, Accountability Lost.
6. Very serious → System does another transaction instead of requested
   EX: credited another account.
7. Extreme → The problems aren't limited to a few users.
8. Intolerable → Long term unrecoverable corruption of Database.
9. Catastrophic → System fails shuts down
10. Infectious. → corrupts Other Systems.

• TAXONOMY OF BUGS

Taxonomy means different categories & classifications.

1. Requirement, features and functionality bugs.
2. Structural bugs
2. Data bugs
4. coding bugs
5. Interface, Integration and System bugs
6. Test and test design bugs.

1. Requirements & Specifications

→ Requirements Incompleteness
→ Ambiguous
→ Lack of clarity on requirements → Analyst assumption not Known to the designer.

Feature & functionality Bugs:

→ Specification problems create feature bugs.

→ Wrong feature bug has design implications.

→ Missing feature is easy to detect & correct.

2. Structural Bugs : structural bugs are

ⓐ control & sequence bugs

ⓑ logic bugs

ⓒ processing bugs

ⓓ Initialization bugs

ⓔ Data flow bugs

3. Data bugs : Data bugs are

ⓐ Generic Data bugs

ⓑ Dynamic data Vs static data

ⓒ Information, parameter and control bugs

ⓓ Contents, structure & Attributes related bugs.

4. coding bugs : coding errors

ⓐ Syntax errors

ⓑ Coding errors

ⓒ Logical errors

ⓓ Documentation bugs

5. Interface, Integration and System bugs.

ⓐ External interface ⓑ Internal interfaces ⓒ H/W architecture bugs ⓓ S/W architecture bugs ⓔ OS bugs ⓕ Integration bugs ⓖ System bugs

6. Test & Test design bugs : ⓐ Test debugging ⓑ Test quality assurance. ⓒ Test execution automation ⓓ Test design automation

FLOW GRAPH AND PATH TESTING:

. BASICS OF PATH TESTING:

- path testing is the set of paths are properly chosen then we have achieved some measure of tests.
- For example, pick paths to assure that every source statement has been executed at least once.
- path testing require complete knowledge of the program structure.
- path testing is used by programmers to unit testing, their own code.

. path testing ~~Rules~~ Guidelines:

~~Rule~~ (1)
——— : Write structured code (conditional & iterational) statements.

↓

~~Rule~~ (2)
——— : Structured code converted into "control flow graph".

~~Rule~~ (3)
——— : Find cyclomatic complexity.

~~Rule~~ (4)
——— : Determine independent paths

~~Rule~~ (5)
——— : Design test cases.

* Control flow graph :

- The control flow graph is a graphical representation of a program.
- Notations Used in a flow graph :

ⓐ Node
ⓑ Edge
ⓒ Decision node
ⓓ Junction node
ⓔ path.
ⓕ segment

Node : It is denoted by a circle.

Edge : This notation is used for joining multiple nodes.

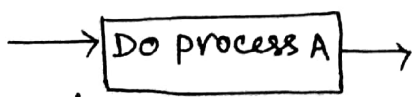
Decision Node : A node that consists of more than one arrow links and producing from it.

Junction Node : A node that consists of more than one arrow comming into it.

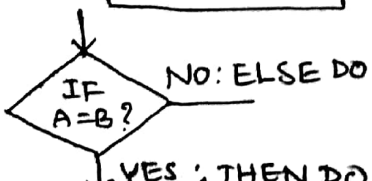
path : path defined as a series of statements.

segment : sequence of links.

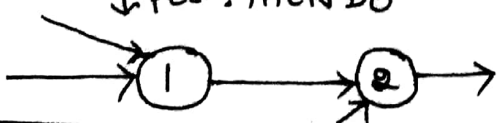
Process Block :



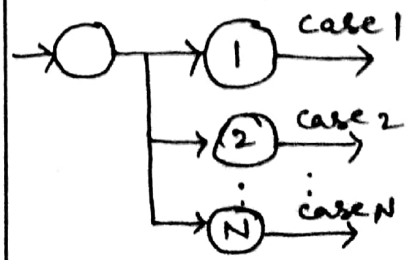
Decisions :



Junctions :



CASE STATEMENTS



Process Blocks :

A sequence of program statements uninterrupted by decisions or functions with a single entry and single exit.

Junction :

A point in the program where control flow can merge

Ex: GOTO, JUMP.

Decisions :

A program point at which the control flow can diverge.

(or) Conditions.

Ex: IF statements.

Case statements :

A multi-way branch or decision.

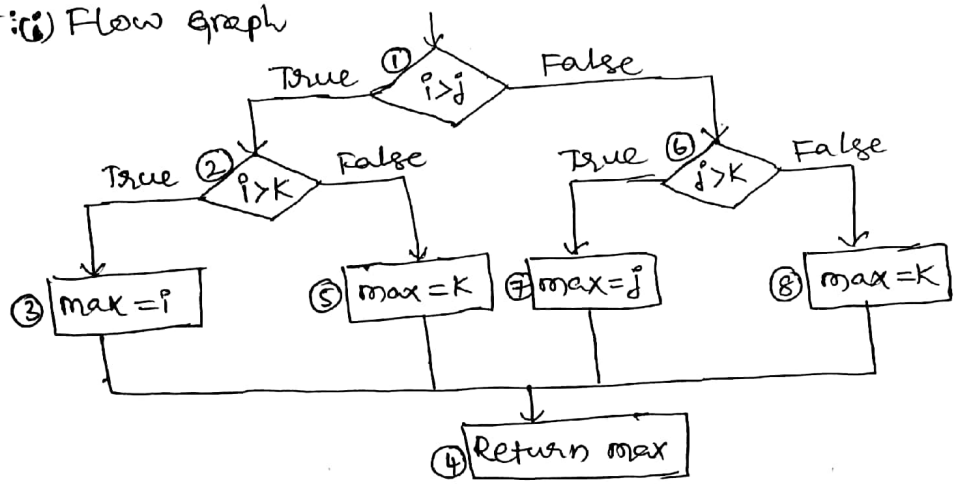
Ex: switch cases, multiple GO to statements.

Control flow graph	Flow chart.
1. Compact representation of the program	1. Usually a multi page description
2. Focus on inputs, outputs and control flow into and out of the block.	2. Focus on the process steps inside.
3. Inside details of a process block are not shown	3. Every part of the process block are drawn.

Ex: consider the following program segment, find Basic path testing?

```
1. int max(int i, int j, int K)
2. {
3. int max;
4. if(i>j) then
5. if(i>K) then max=i;
6. else max=K;
7. else if(j>K) max=j
8. else max=K
9. return(max);
10. }
```

Sol: (i) Flow Graph



(ii) Cyclomatic complexity.

$$V(G) = e - n + p$$
$$= 10 - 8 + 2$$
$$= 4$$

(iii) Independent paths :
1 → 2 → 3 → 4 — max i
1 → 2 → 5 → 4 — max K
1 → 6 → 7 → 4 — max j
1 → 6 → 8 → 4 — max K

(iv) Test cases :

Test Case Id	Inputs			Expected o/p.	Remarks
	i	j	K		
TC1	5	3	2	5	max i
TC2	5	3	6	6	max K
TC3	3	5	2	5	max j
TC4	3	5	6	6	max K

path selection criteria :

A minimal set of paths to be able to do complete testing.

→ Each pass through a routine from entry to exit, as one traces through it, is a potential path.

path testing criteria :

1. path testing :

Execute all possible control flow paths through the program. It implies 100% path coverage.

2. Statement Testing :

Execute all statements in the program atleast once under the some test. It depends on true or false statements. It is denoted by 'C1' (node coverage).

3. Branch Testing :

Execute enough tests to assure that every branch alternative has been exercised at least once under test. It is denoted by 'C2' (Link coverage) (extention of loop testing). is branch testing.

* path Selection Rules :

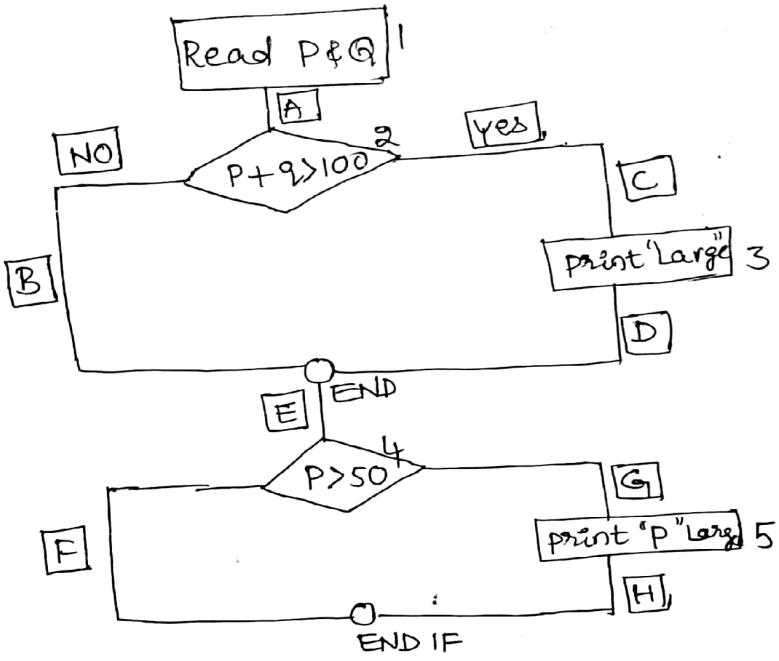
1. pick the simplest and functionally sensible entry/exit paths.
2. pick additional paths as small variations from previous paths (no loops).
3. pick additional paths but without an functional meaning (only to achieve C1 + C2) coverage.
4. Be comfortable with the chosen paths.

Example

Example : Given the following code, how much minimum. Number of test cases is required for statement coverage and branch coverage using path testing.

```
read p and q
if p+q>100
then print "large" endif
if p>50
then print "p large" endif.
```

Solution :



Statement coverage : "All true paths".

S.NO	Testcase Id	Input values P	Q	output	path coverage
1	TC1	70	50	"P" large	1A→2C→3D→E→4G→5H

Branch coverage " To calculate entire True path & entire False path.

S.NO	Test Case Id	Inputs P	Q	output	path coverage
1	TC1	70	50	P is large	1A→2C→3D→E→4G→5H
2	TC2	30	40	—	1A→2B→E→4F
path coverage 3.	TC3	70	80	—	1A→2B→E→4G→5H
4	TC4	40	80	—	1A→2C→3D→E→4F

Loop Testing

Loop testing is defined as the process of testing the loops present in a program. It is the extension of the branch coverage. Types of loop testing.

1. Simple loop
2. Nested loop
3. concatenated loop.

Simple loop: It consists of a single loop.

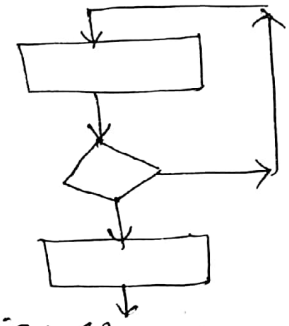


fig: Simple loop.

Nested loop: A loop which consists of two or more loops is called "nested loop".

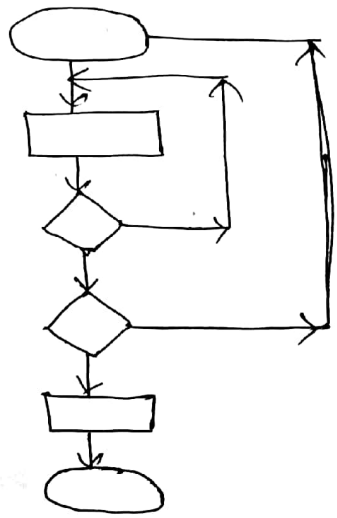


fig: Nested loop.

concatenated loops

concatenated loops are those loops that are linked together.

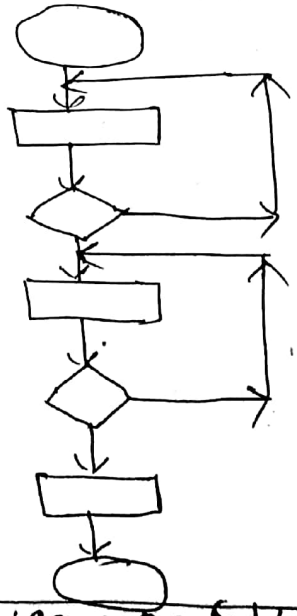
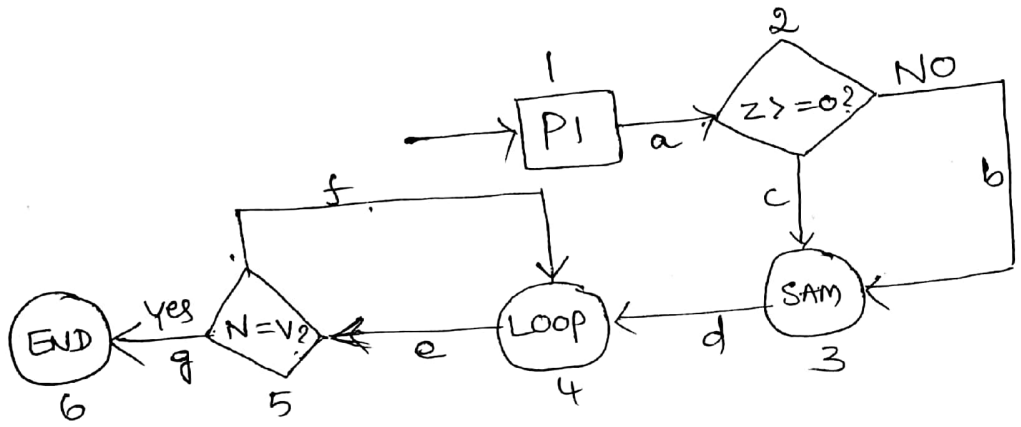


fig: concatenated loops.

* Picking enough paths for achieving C1+C2 coverage.



1. Does every decision have Y & N (C2)?
2. Are all cases of case statement marked (C2)?
3. Is every three way branch covered (C2)?
4. Is every link covered at least once (C1)?

* PREDICATES:

path: A sequences of process link (& nodes).

Predicates: The logical feenction evaluated at a decision
Ex: True or False (Binary, boolean).

compound predicate:
Two & more predicates combined with AND, OR

path predicates:
Every path corresponds to a succession of True/false values for predicates traversed on that path.

- A predicate associated with a path

"X > 0 is True" AND "W is either negative or equal to 122" is True.

- Predicate interpretation:

→ The symbolic substitution of operations along the paths in order to express the predicate in terms of input vector is called "predicate interpretations".

→ An input vector is a set of inputs to a routine arranged as a one dimensional array.

→ An interpretive trace program is one that executes every statement.

Ex:

```
INPUT X,Y
    ON X GOTO A,B,C
A: Z:=7 @ GOTO H
B: Z:=-7 @ GOTO H
C: Z:=0 @ GOTO H
H: DO SOMETHING
K: IF X+Z > 0 GOTO GOOD ELSE GOTO BETTER
```

~~INPUT X~~
~~IF X<0 THEN Y:=2~~
~~ELSE Y:=1~~
~~IF X+Y* Y>0 THEN..~~

Predicate expression:

A: $X5 > 0$
B: $X_1 + 3X_2 + 17 >= 0$
C: $X3 = 17$
D: $X4 - X_1 >= 14 X2$
E: $X6 < 0$
F: $X_1 + 3X_2 + 17 >= 0$
G: $X3 = 17$
H: $X4 - X_1 >= 14X2$

Converting into the predicate expression form.

$ABCD + EBCD \Rightarrow (A+E)BCD$

predicate blindness :

1. Assignment blinding
2. Equality blinding
3. self blinding.

Assignment blinding :

A buggy predicate seems to work correctly as the specific value choosen in an assignment statement works with both the correct & buggy predicate.

correct

```
X := 7
IF Y>0 THEN...
```

BUGGY

```
X := 7
IF X+Y>0 THEN ...
```

(check for Y=1)

Equality blinding :

When the path selected by a prior predicate results in a value that works both for the correct & buggy predicate.

correct

```
IF Y=2 THEN ..
  IF X+Y>3 THEN ...
```

BUGGY

```
IF Y=2 THEN
IF X>1 THEN
```

(check for any X>1)

Self blinding :

When a buggy predicate is a multiple of the correct one and the result is indistinguishable along that path.

correct

```
X := A
IF X-1>0 THEN ....
```

BUGGY

```
X := A
IF X+A-2>0 THEN..
```

(check for any X>A)

Achievable Paths :

1. Objective is to select & test just enough paths to achieve a satisfactory notion of test completeness such as C1+C2.
2. Extract the program's control flow graph & select a set of tentative covering paths.
3. Trace the path through, multiplying the individual compound predicates to achieve a boolean expression.
   Example : (A+BC)(D+E)
4. Multiply & obtain sum of products form of the path predicate expression : AD+AE+BCD+BCE.

path sensitization :

It's the act of finding a set of solutions to the path predicate expression.

Heuristic procedures :

choose an easily sensitizable path set, & pick hard-to-sensitize paths to achieve more coverage.

→ Identify all the variables that affect the decisions.
→ process dependency as an equation, function
→ for correlated variables, express the logical, arithmatic,
→ Identify correlated predicates and document the nature of the correlation as for variable.
→ start path selection with uncorrelated & independent predicate paths.
→

Example of path sensitization :

1. Simple Independent Uncorrelated predicates.

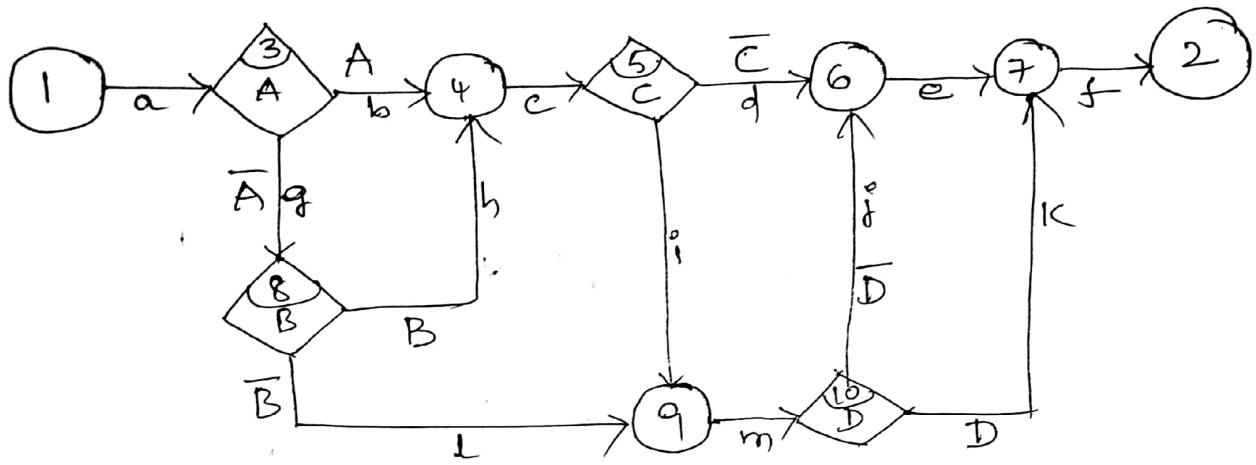


fig: predicate Notation.

path	predicate values
abcdef	A $\bar{C}$
aghcimkf	$\bar{A}$ B C D
aglmjef	$\bar{A}$ $\bar{B}$ $\bar{D}$

A Simple case of solving inequalities

path	predicate values
abcdef	A $\bar{C}$
abcimjef	A C $\bar{D}$
abcimkf	A C D
aghcdef	$\bar{A}$ B $\bar{C}$
aglmkf	$\bar{A}$ $\bar{B}$ $\bar{D}$

(obtained by the procedure for finding a covering set of paths).

* PATH INSTRUMENTATION:

1. path instrumentation is what we have to do confirm that the outcome was achieved by the intended path.

2. co-incidental corrections: The coincidental correctness stands for achieving the desired outcome for wrong resons.

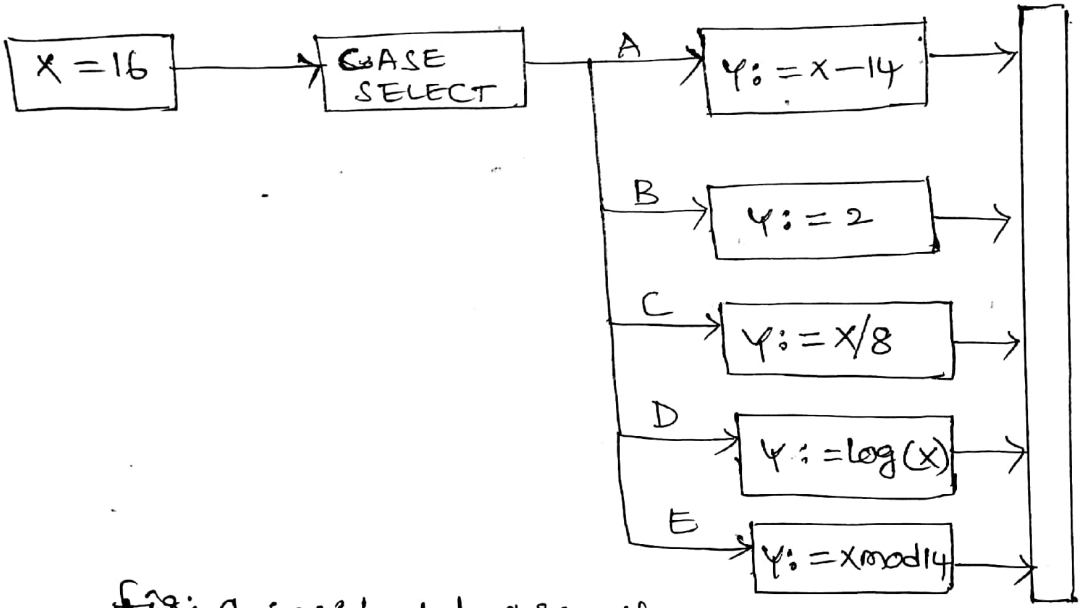


fig: Coincidental corrections.

The above figure is an example of a routine that, chosen input values (X=16), yields the same outcome (Y=2) no matter which case we select.

The 5 cases could be totally jumbled and still the outcome would be the same.

* Link markers (or) Traversal markers.

A simple and effective form of instrumentation is called a "Traversal marker" (or) Single Link markers.

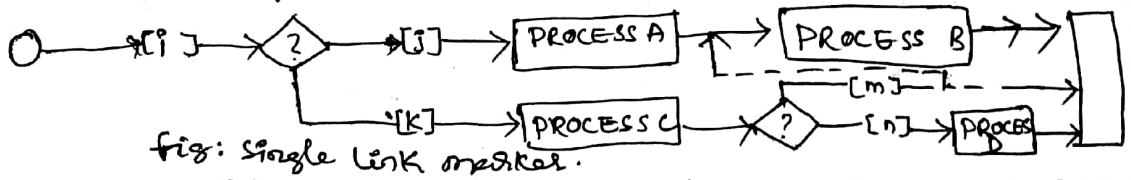


fig: Single link marker.

- we intended to traverse the ikm path, but because of rampaging GOTO in the middle of the m link, we goto process B, if coincidental correctness is against, outcome will be the same. & won't know about bug.

• path selection criteria achieving $C_1 + C_2$ coverage.

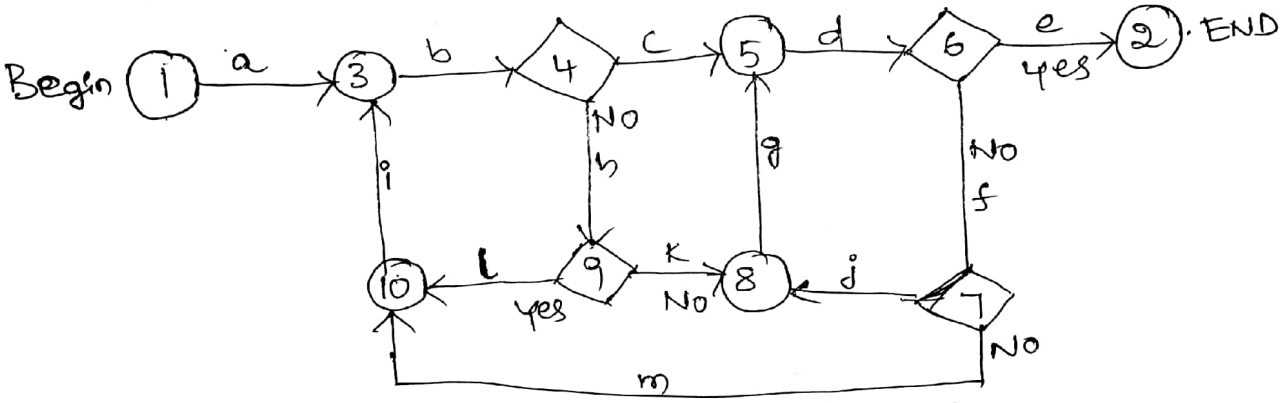
path selection criteria implements on

(a) statement coverage

(b) Branch coverage

(c) path coverage.

Ex: path selection

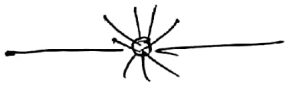


1. Draw the control flow graph on a single sheet of paper.
2. Does every decision have a YES and NO in its column? (C2)
3. Has every case of all case statements been marked? (C2)
4. Is every Three way branch covered (C2)
5. Is every link covered at least once (C1)

PATHS	Decisions				Process link
	4	6	7	9	a b c d e f g h i j k l m
abcde	yes	yes			✓ ✓ ✓ ✓ ✓
aibhkgde	NO	yes		NO	✓ ✓ ✓ ✓ ✓ ✓
abhlibcde	NO, yes	yes		yes	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
abcdfjgde	yes	No, yes	yes	•	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
abcdfmibcde	yes	No, yes	NO		✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

• Applications of path Testing:

1. Higher code coverage
   - (a) statement coverage
   - (b) Branch coverage
   - (c) path coverage.
2. Unit testing
3. Integration Testing.
4. maintenance Testing.
5. cyclomatic complexity.



UNIT-II

- TRANSACTION FLOW TESTING:
  - TRANSACTION FLOWS
  - TRANSACTION FLOW TESTING TECHNIQUES
- DATA FLOW TESTING:
  - BASICS OF DATA FLOW TESTING
  - STRATEGIES IN DATA FLOW TESTING
  - APPLICATIONS OF DATA FLOW TESTING

* TRANSACTION FLOW TESTING INTRODUCTION

A transaction consists of a sequence of operations, some of which are performed by a system.

EX: A transaction for an online information retrival system consists of the following steps.

- Accept input
- Validate input
- Transmit acknowledgement to requester
- Search file
- Request directions from user
- processing request
- Update file
- Transmit output.

• TRANSACTION FLOW TESTING TECHNIQUES

→ Transaction flow technique is an static testing.

→ Static testing is defined as without having the need to execute the system (or) software.

Transaction flow Techniques are

(i) Inspection
(ii) Review
(iii) Walkthroughs.

(i) Inspection :

Inspection is defined as the process of checking the software at regular intervals inorder to identify the errors and rectify the errors.

Inspection process :

Inspection team members are

(a) Author → Author is a designer & programmer
(b) Inspector → peer member of Team
(c) moderator → one of the key person of the team
(d) recorder. → who is maintaining the document of the software product.

(ii) Review :

Review is nothing but to conducting more than one people among the software development life cycle (SDLC) stages.

Ex: Analysis review, design review, coding review, Testing review.

(iii) Walkthroughs :

Walkthroughs is nothing but before going to deliver the product to the customer once again check the product (review). Here walkthroughs errors checked.

* DATA FLOW TESTING : (OR) DATA FLOW MODEL

Data flow ~~technique~~ Testing is a white box testing technique.

Definition :

Data flow testing is the name given to a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects.

Advantages :

→ Data flow testing uses the control flow graph to explore the unreasonable things that can happen to data (data flow anomalies).

→ consideration of data flow anomalies leads to test path selection strategies that fill the gaps between complete path testing and branch and statement testing.

Ex: $e = (m+n+p+q) * (m+n-(p+q))$

Step (1) : Given m, n, p, q are control flow with von-neumans paradigms.

$a = m+n$

$b = p+q$

$c = a+b$

$d = a-b$

$e = c*d$

Step ②

Begin

par do

Read m, n, p, q

par do

a = m + n

b = p + q

End par

par do

c = a + b

d = a - b

End par

par do

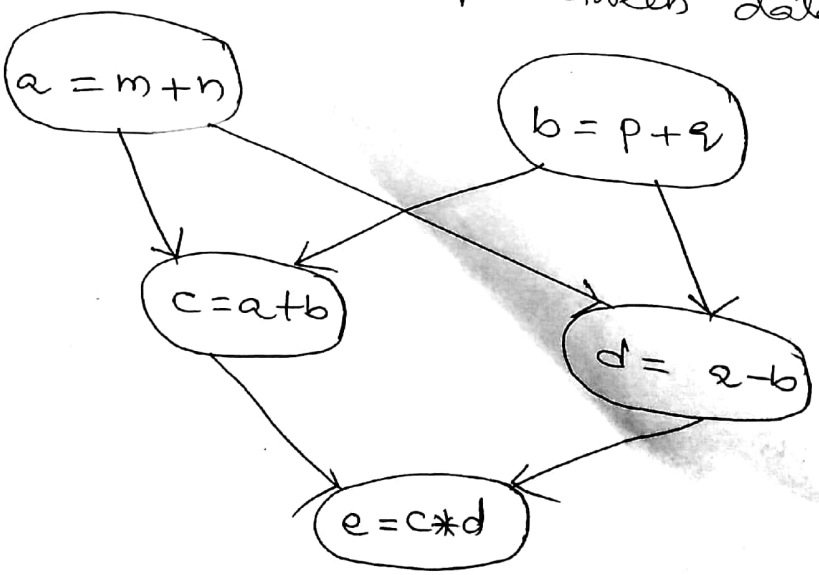
e = c * d

End par

end.

Step ⑧

Inter relationship between data items.



Data flow testing and Data flow criteria

Different categories of data objects:

1. Define Objects (d)
2. Used Objects (U)
3. Killed Objects (K)
4. Calculation (or) computation Objects (C)
5. predicate data Objects (P)

1. Defined Objects (d):

An object (variable) is defined when appears in a data declaration.

2. Used Objects (U):

An object is used when it is part of a computation or a predicate.

3. Killed Objects (K):

An object is released or finalized.

4. Computation Objects (C):

An Object is used an calculation & computation.

5. Predicate data Objects (P):

An Object is used in a conditional or iterational statements.

* Write an example to find out definition and Uses for the following program.

Eg:

⇒
```
1. read (x, y);
2. z = x + 2;
3. if (z < y)
4.     w = x + 1;
   else
5.     y = y + 1;
6. print (x, y, w, t).
```

Sol

S.NO	Define	C- Uses	P-Uses
1.	x, y	—	—
2.	z	x	—
3	—	—	z, y
4.	w	x	—
5	y	y	—
6	x, y, w, z	—	—

Data flow Anomalies :

Data flow anomalies defines those data usage pattern that lead to incorrect in correct code execution. This can be either represented used in single character or double character.

(1) Single character anomalies :

They are represented as follows.

(a) ~x

This anomaly indicates that all the prior actions are not of interest to x.

(b) x~

This anomaly indicates that all the post actions are not of interest to x.

Here x, can be any state of the data object.

Anomaly	Description.
~d	first define
~U	first Use
~K	First Kill
D~	Define last
U~	Use last
K~	Kill last.

Two character Anomalies

Anomaly	Description
du	define use
dk	define kill
ud	use define
uk	use kill
ku	kill use
kd	kill define
dd	define define
uu	use use
kk	kill kill

* **Slices**

A (static) program slice is a part of a program (Ex: a selected set of statements) with respect to a given variable X.

* **Dices**

A program dice is a part of a slice in which all statements which are known to be correct have been removed.

Data Flow Testing Strategies

(a) All-du-paths (ADUP) — strongest data flow testing strategy. it is super set

(b) All-Uses (AU) —— every use variable there exists a path

(c) All-p-Uses (APU+C) — predicate use.

(d) Some-p-Uses

(d) All-c-Uses & Some-p-Uses (AcU+P) — computation and predicate use

(e) All-predicate-Uses (APU) — It taken out from APU+C

(f) All-computational Uses (AcU) — It taken from AcU+P.

(g) All-Definition (AD). — Use of variable predicate & computational use.

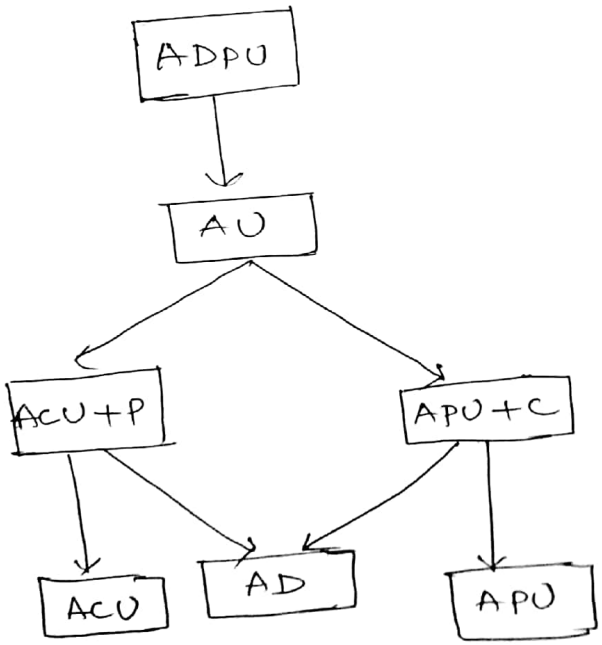


fig: Data flow Testing strategies.

APPLICATIONS OF DATA FLOW TESTING

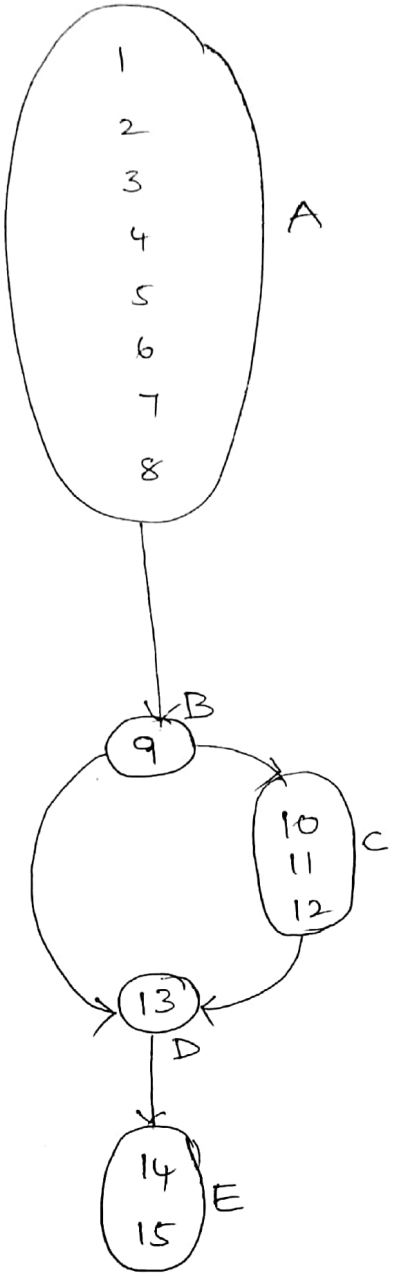
Ex: consider a program to input two numbers and print them in ascending order given below. Find all du paths and identify those du-paths that are not feasible. Also find all dc paths and generate the test cases for all paths (dc path and non dc path)

program

```
#include <stdio.h>
#include <conio.h>
1. void main()
2. {
3 int a, b, t;
4. clrscr();
5. printf ("enter first number");
6. scanf ("%d", &a)
7. printf ("enter second number");
8. scanf ("%d", &b);
9. if (a<b) {
10. t = a;
11. a=b;
12 b=t;
13 }
14 printf ("%d %d", a, b);
15. getch(); }
```

Solution

(i) control flow Graph

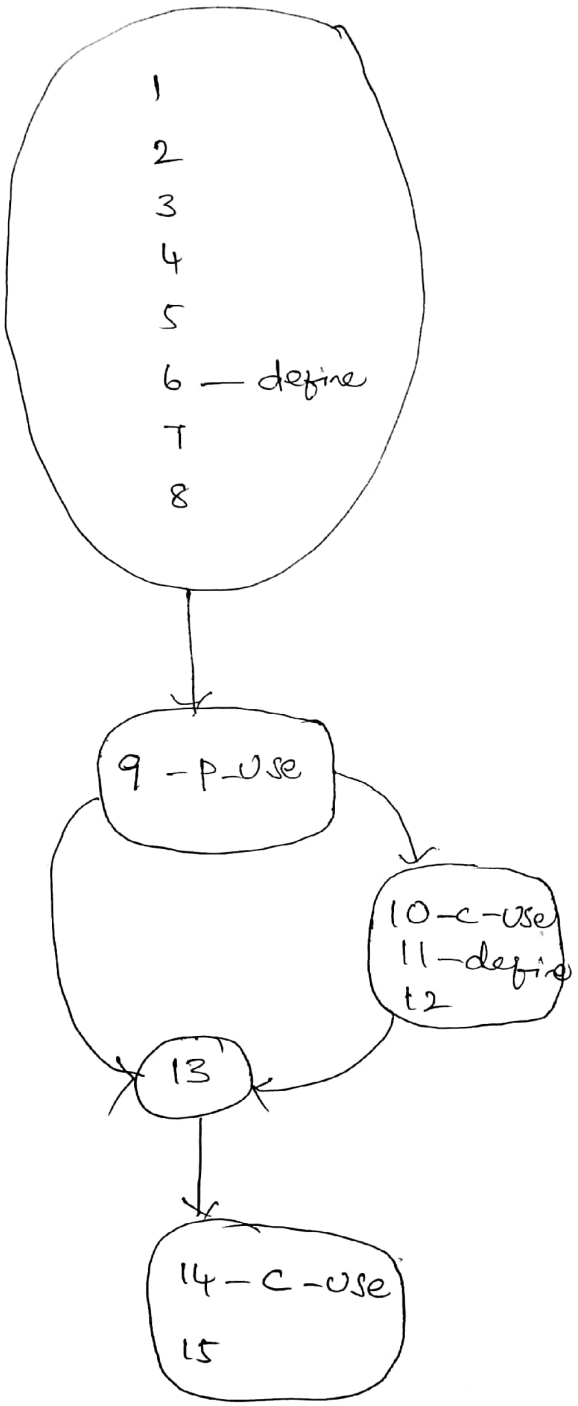


(ii) Independent paths

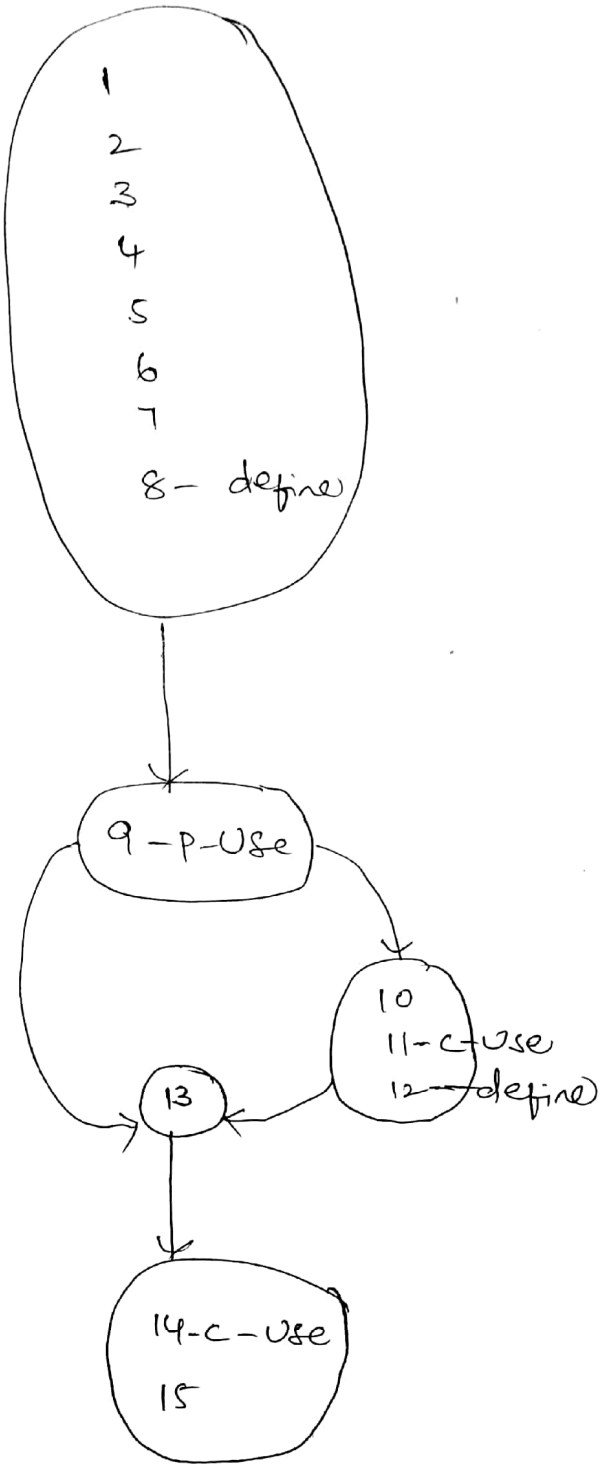
A – B – C – D – E

A – B – D – E

(iii) Data Flow Graph for Variable "a".



(iv) Data Flow Graph for variable "b".



Variable	Defined At	Used At
a	6, 11	9, 10, 14
b	8, 12	11, 14

Data Flow Testing paths

Strategy	a	b
All Uses (AU	6-9 6-9-10 11-12-14	8-9 8-9-10-11 12-14
All P Uses (APU)	6-7-8-9	8-9
All C-Uses (ACU	6-7-8-9-10 11-12-13-14 6-7-8-9-13-14	8-9-10-11 8-9-13-14 12-13-14
All du paths (ADUP)	6-7-8-9-10-11-12-13-14 6-7-8-9-13-14-15 11-12-13-14-15	8-9-10-11-12-13-14 8-9-13-14-15 12-13-14-15

Applications of path Testing (&) Data flow Testing

1. Higher code coverage
2. Unit testing
3. Integration Testing
4. Cyclomatic complexity
   (a) Statement coverage
   (b) branch coverage
   (c) path coverage.

JNTUK previously Asked questions

1 (a) Define Transaction & Transaction flow testing with an example?

(b) What is meant by Transaction flow testing? Discuss its significances?

2 (a) What is meant by data flow model? Discuss various components of it?

* (b) Write about data flow testing strategies in detail?

③ What is meant by data flow anomalies? How data flow testing will explore them?

④ Distinguish between control flow & Transaction flow?

⑤ Write applications of data flow testing?

Short Answer Type questions

① Write data flow anomalies?

② Define slicing & Dicing?

UNIT-III

- DOMAIN TESTING:
  - DOMAINS AND PATHS
  - NICE & UGLY DOMAINS
  - DOMAIN TESTING
  - DOMAINS AND INTERFACES TESTING
  - DOMAIN AND TESTABILITY
- PATHS
- PATH PRODUCTS AND REGULAR EXPRESSIONS:
  - PATH PRODUCTS
  - PATH EXPRESSION
  - REDUCTION PROCEDURE
  - APPLICACTIONS
- REGULAR EXPRESSIONS & FLOW ANOMALY DETECTION.

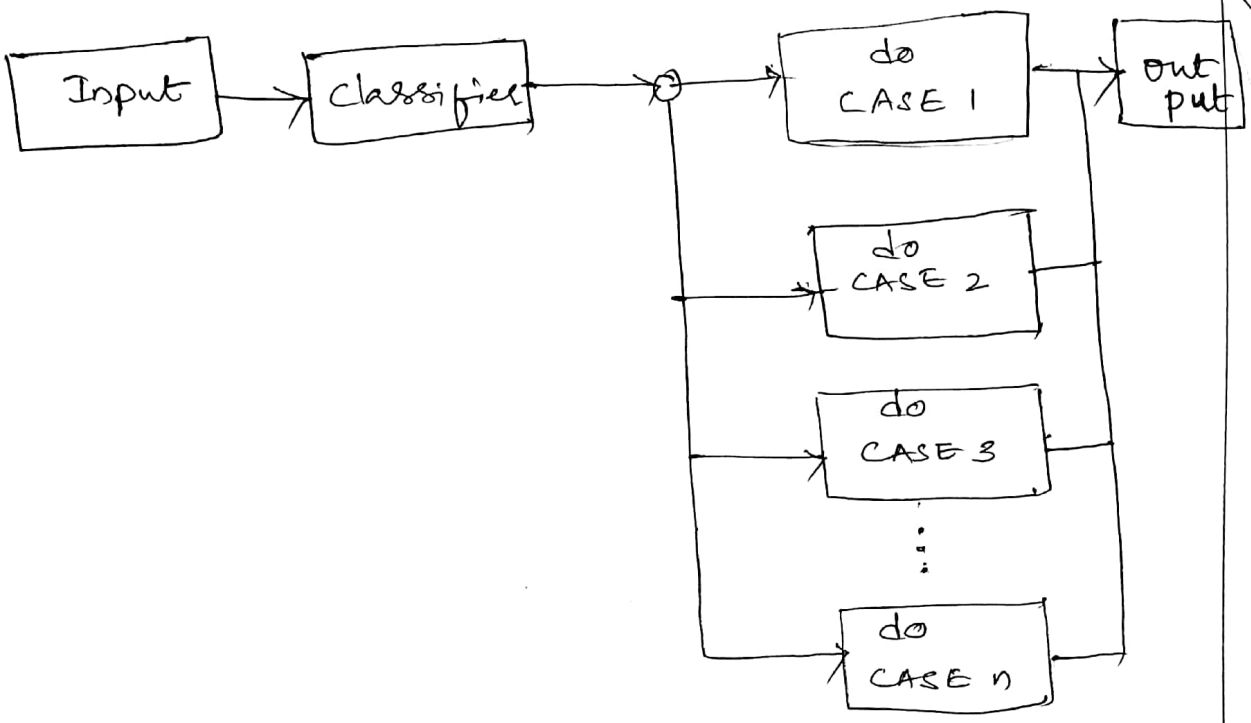
DOMAIN TESTING:

→ Domain means the mathematical representation of boundaries.

→ Domain testing is a functional Testing, there is no need to require structural knowledge.

→ In domain testing uses different values of variables & predicates.

→ Domain testing we are using domain classifier.



Domain Testing equivalence

Example : Equivalence class partitioning based on the quadratic equation is $d = b^2 - 4ac$.

d has two different real roots if $d > 0$
$b < 0$

the equation has two identical real roots if $d = 0$

Test Cases

Test case id	inputs				Expected o/p
	a	b	c	d	
TC1	1	2	1	d=0	d = 0
TC2	1	2	2	d<0	d < 0
TC3	1	4	1	d>0	d > 0

NICE DOMAIN

→ Nice (good) domain is better than ugly (bad) domain.

→ Bug frequency is lesser than ugly domain

→ Nice domain properties are

1. Linear
2. Complete
3. Systematic (consistency)

Nice Two dimensional domains

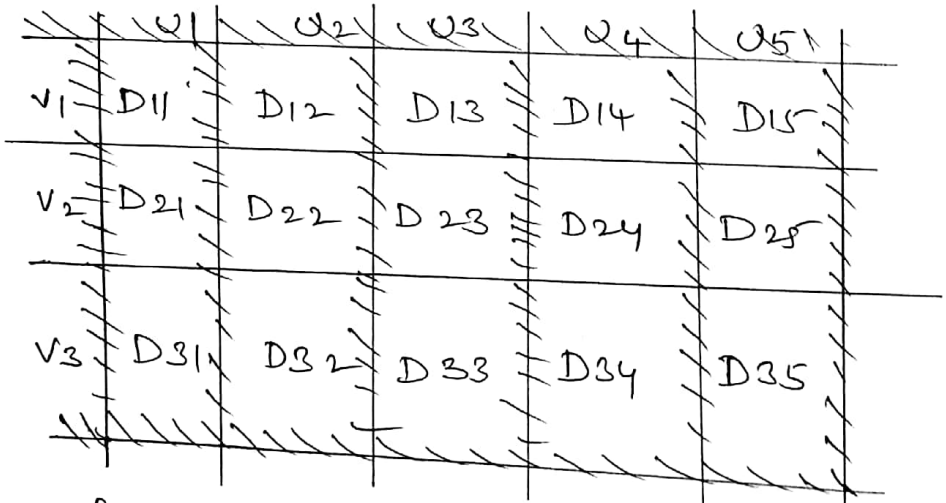


fig:. Nice Two dimensional Domains

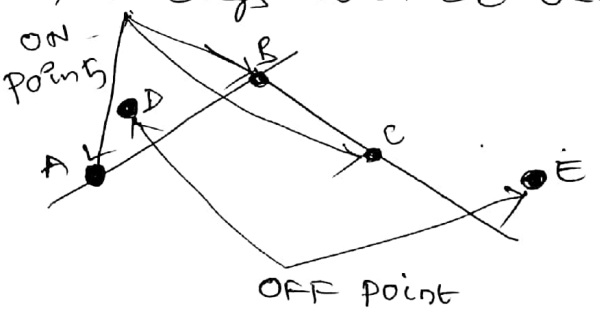
Ugly domain

→ Ugly (bad) domain means bad specifications (OFF points)

→ Ugly domains are more bugs will be occured

Ex: ① ambiguity

② ~~domain~~



→ Ugly domain has missing boundaries, overlap.

complete _____

Incomplete - - - - - - -

Generic domain bugs (or) errors

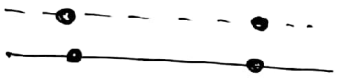
① Shifted boundary

② Tilted boundary

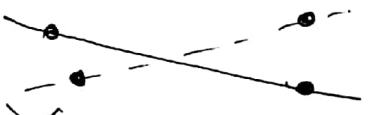
③ Extra boundary.

④ Missing boundary

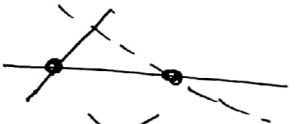
1. Shifted boundary:



2. Tilted boundary:



3. Extra boundary:



4. missing boundary:

DOMAIN AND INTERFACE TESTING :

→ Recall that we defined integration testing as testing the correctness of the interface between two components A & B.

→ Interface between any two components is considered as a subroutine call.

→ We are looking for bugs in that "Call" when we do interface testing.

. Domain and range

The set of outputs values produced by a function is called the range of the function, in contrast with the domain.

→ Interface testing requires that we select the output values of the calling routine i.e callers range must be compatible with the called routine's domain.

→ An interface test consists of exploring the correctness of the following mappings :

Caller domain - - → Caller range (Caller unit test)

Caller range - - → Called domain (integration test)

Called domain - - → Called range (Called unit test).

Closure compatibility:

Assume that the caller's range and the called domain spans the same numbers for example, 0 to 17.

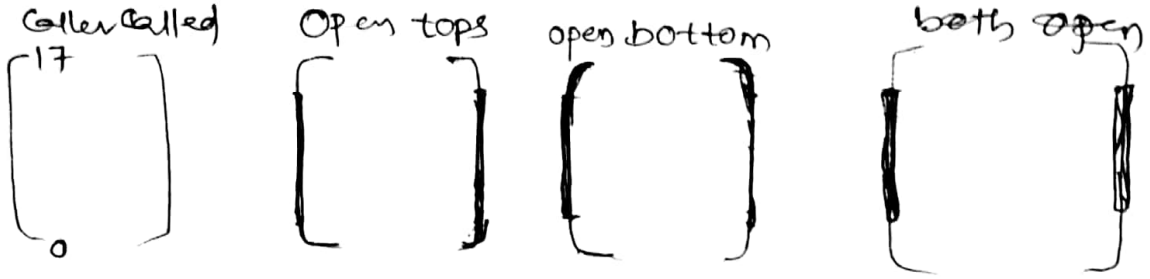


fig: Range/Domain closure compatibility.

Domain and Testability

1. Linearizing transformations
   - (a) polynomials ($x, x^2, x^3, \ldots$)
   - (b) Logarithmic Transforms
   - (c)
2. coordinate transformations.

* Domain closure

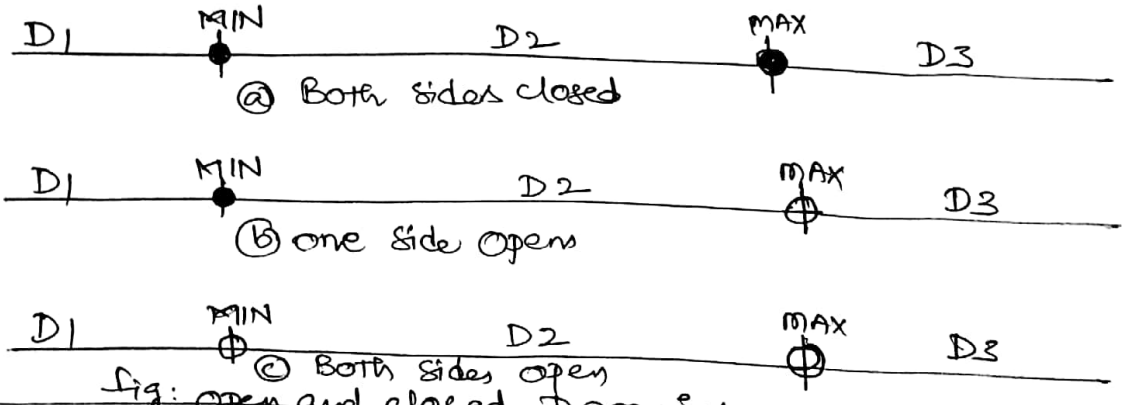


fig: Open and closed Domains

path products :

The name of a path that consists of two successive path segments is conveniently expressed by the concatenation of path product of the segments names.

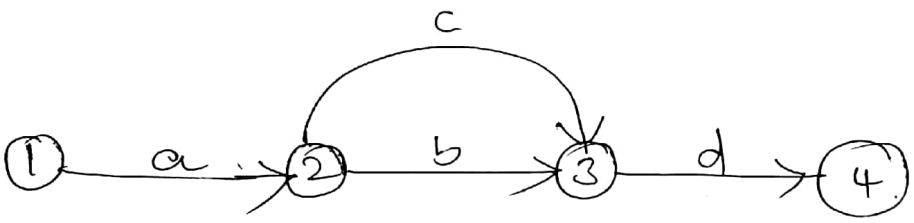
Ex① If X and Y are defined as

X = abcde

Y = fghij then paths corresponding to X followed by Y is denoted by

XY = abcdefghij

Ex②



abd, acd

Ex③

abc, abbc, abbbc ....

Path expression :

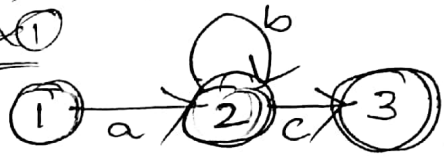
Consider a pair of nodes in a graph and the set of paths between those node.

→ Denote that set of paths by upper case letter such as X, Y.

The member of paths set can be listed as follows.

Ex(1)

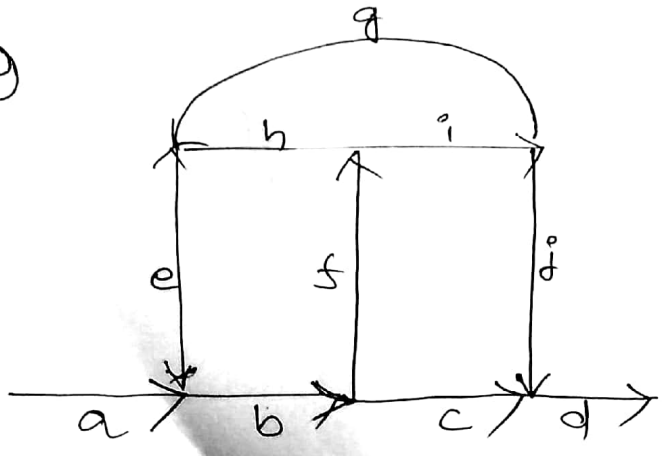
ac, abc, abbc, abbbc+...



* Any expression that consists of path names and OR's and which denotes a set of paths between two nodes is called "path expression".

ac+abc+abbc+abbbc+...

Ex(2)



abcd+abfijd+abfigebcd+abfhgbcd.

loops

$a^1 = a$

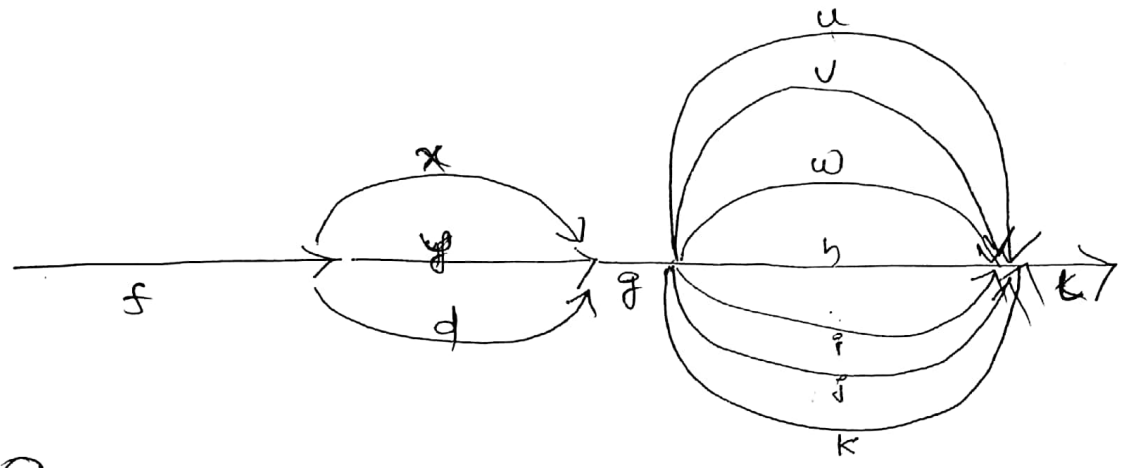
$a^2 = aa$

$a^3 = aaa$

$\vdots$

$a^n = aaa\ldots n$

Ex:



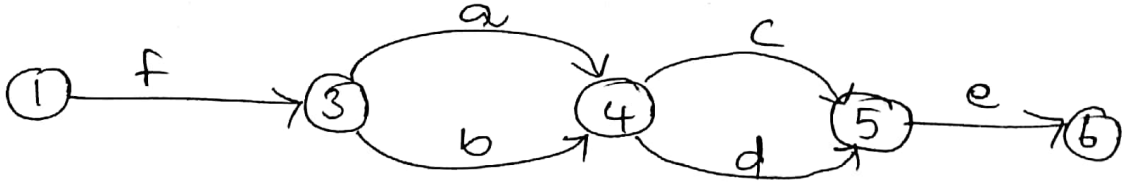
Rule (1)

parallel links add.

$$f(x+y+d)g(u+v+w+h+i+j+k)l$$

Distributed laws

$$A(B+C) = AB+AC$$



$\Rightarrow f(a+b)(c+d)e$

$\Rightarrow (fa+fb)(ce+de)$

$\therefore acef + adef + bcef + fbde.$

Associative law

Ex: $(x+y)+z = x+(y+z)$
$= x+y+z$

Ex: $A(BC) = (AB)C = ABC$

Absorption law

$x+x = x$

$y+y = y$.

Reduction procedure

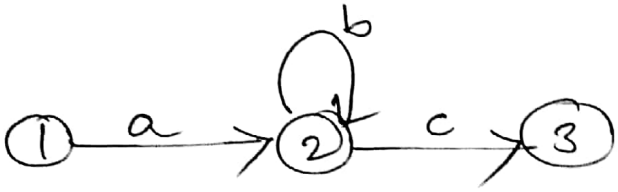
node by node reduction procedure.

Step ① — combine all serial links by multiplying their path expression

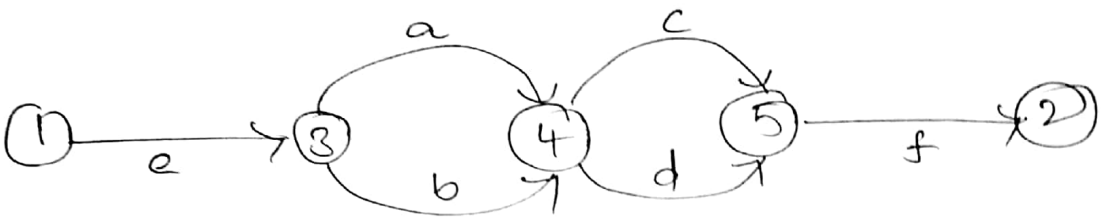
Step ② — combine all parallel links by adding their path expression

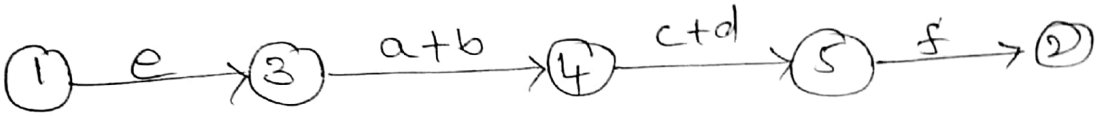
Step ③ — remove all self loops replace links of the form x^*.

Ex①

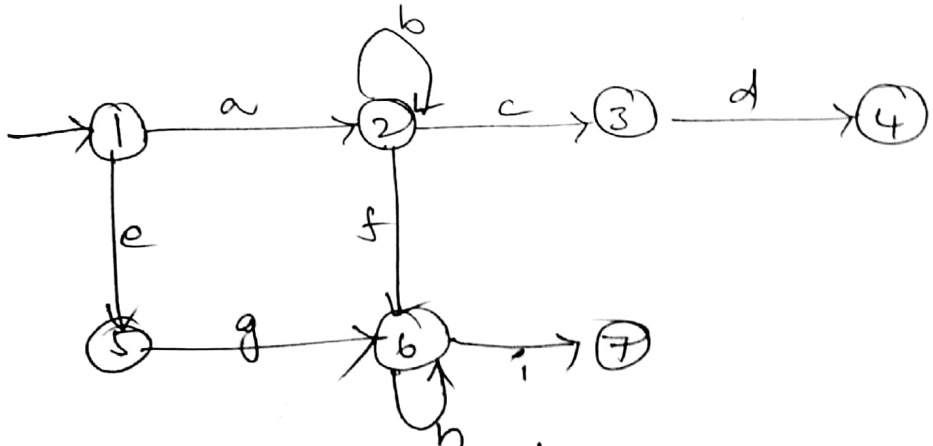


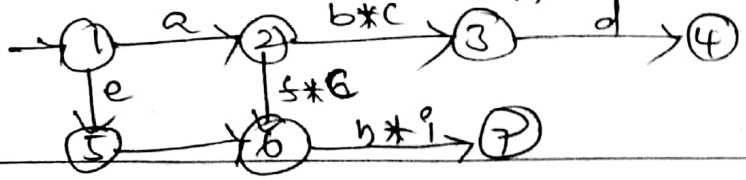
Ex②



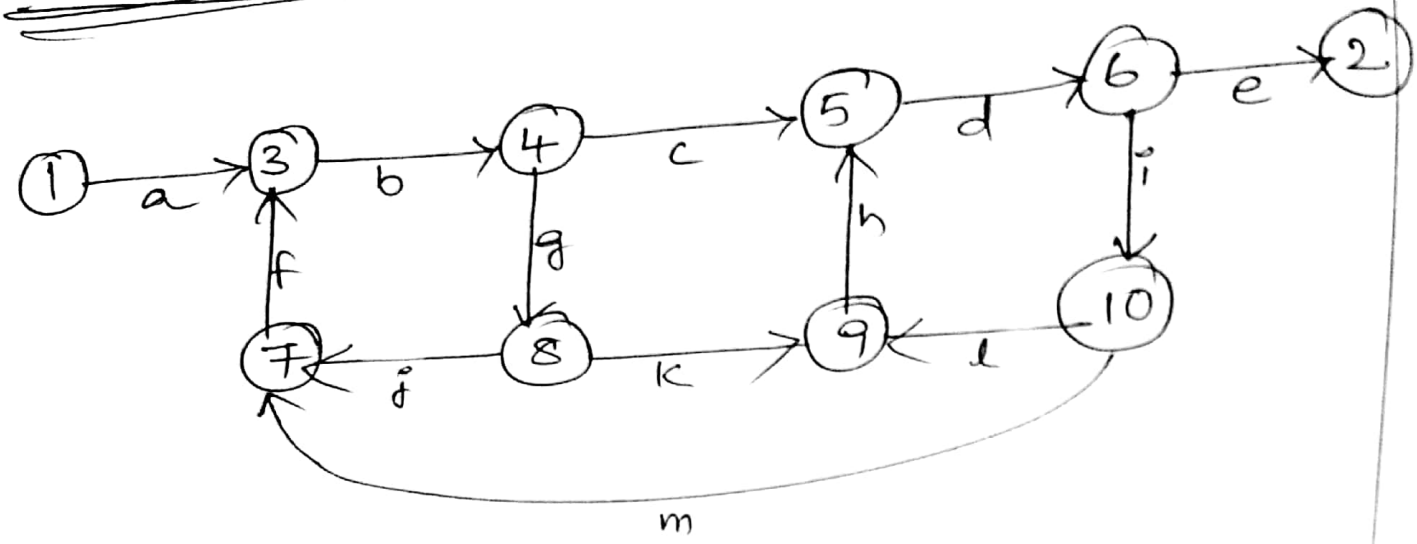


Ex③ :

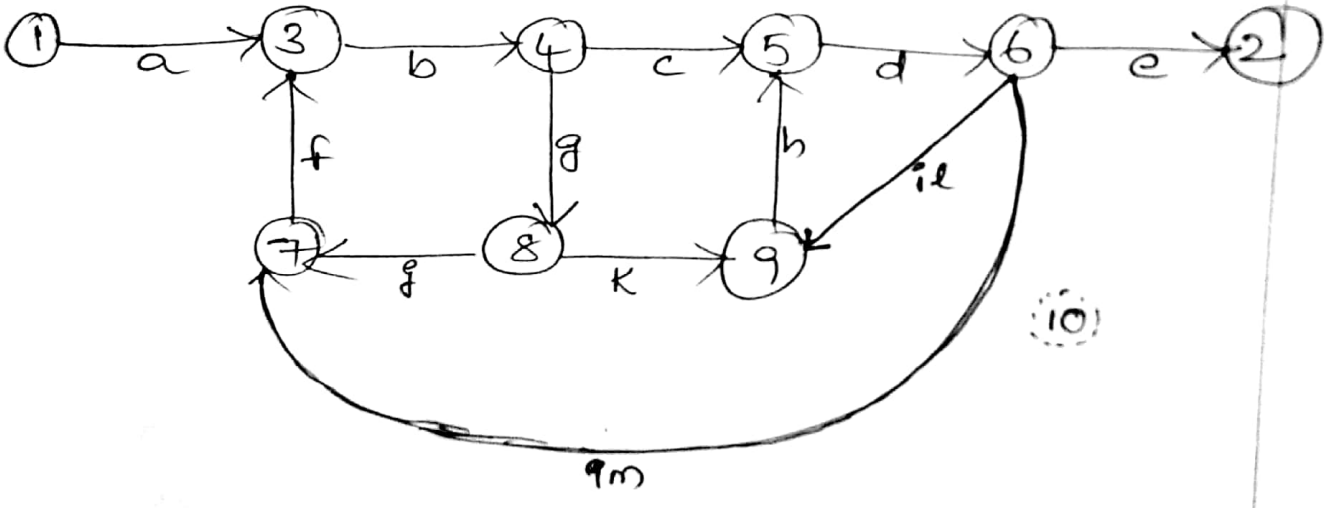




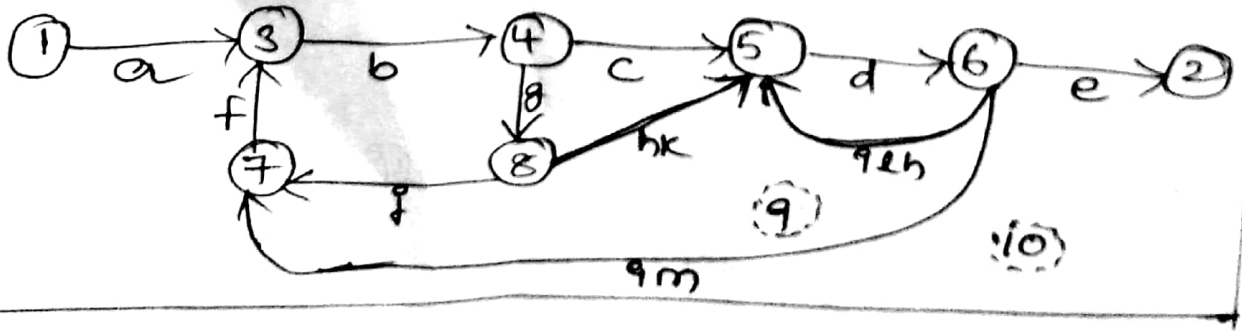
* Apply reduction procedure algorithm



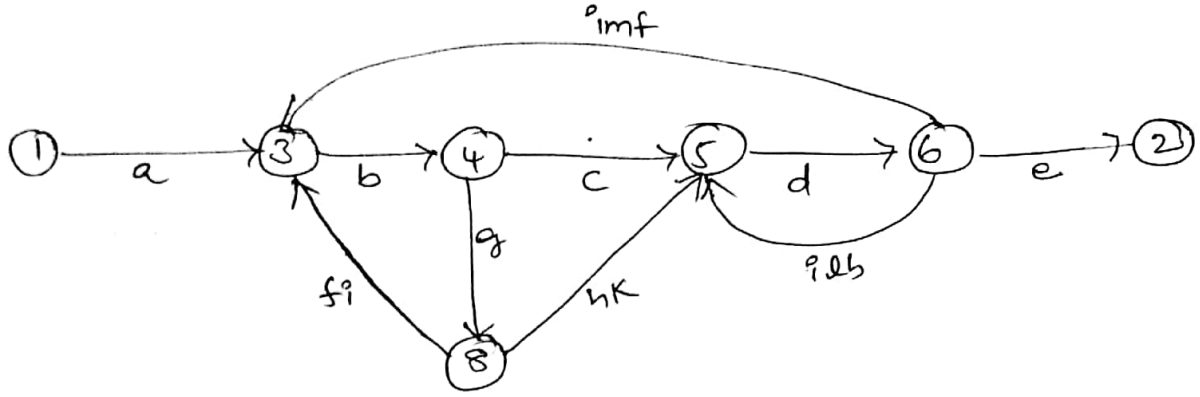
Step ① Remove node 10



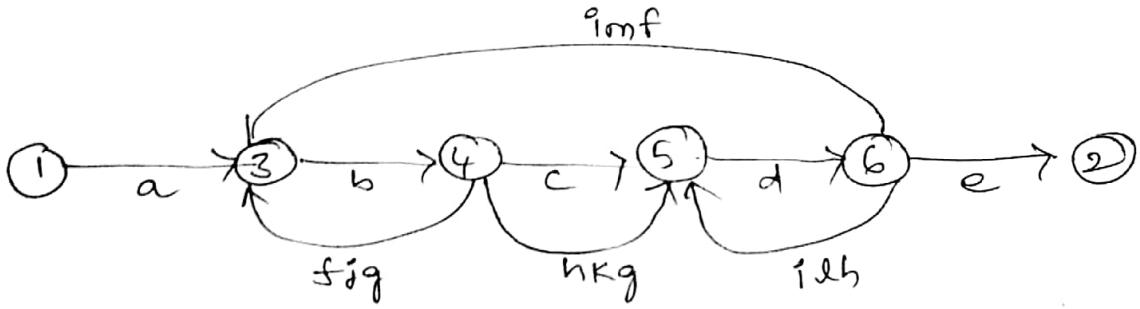
Step ② Remove node 9



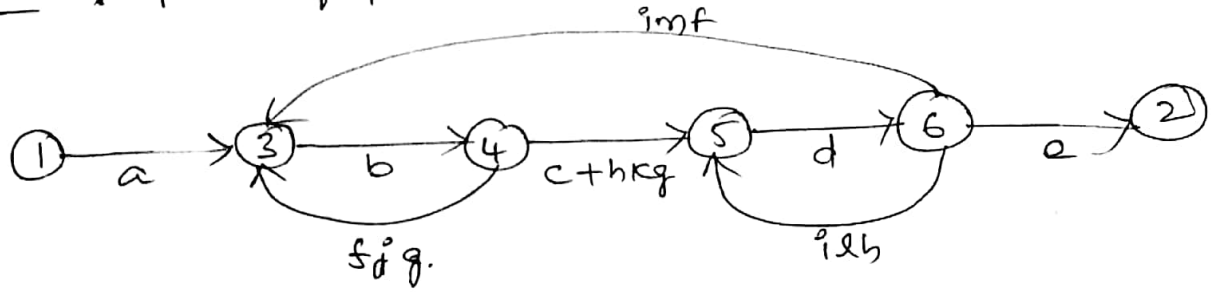
Step (3) Remove node 7.



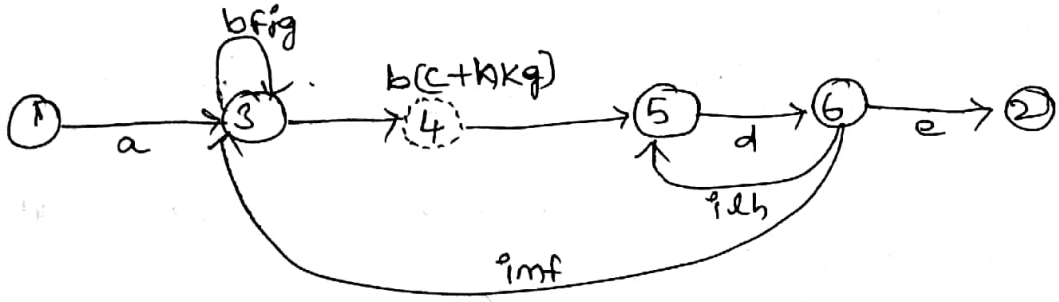
Step (4) Remove node 8



Step (5) : pair of parallel links between node 4 & 5.

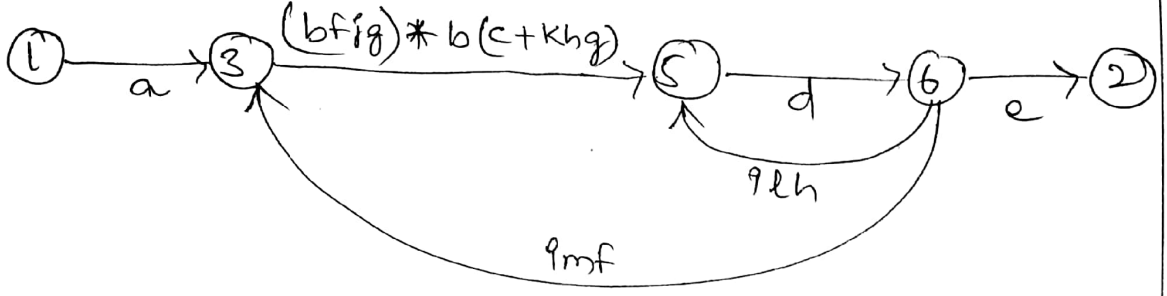


Step (6) : Remove node 4 leads to a loop term



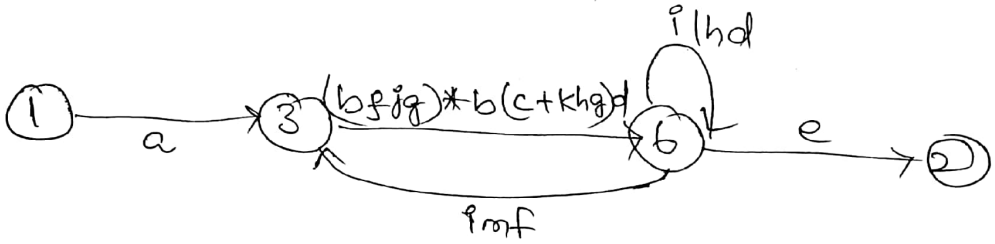
Step (7)

: The process by applying the loop-removal step as follows.



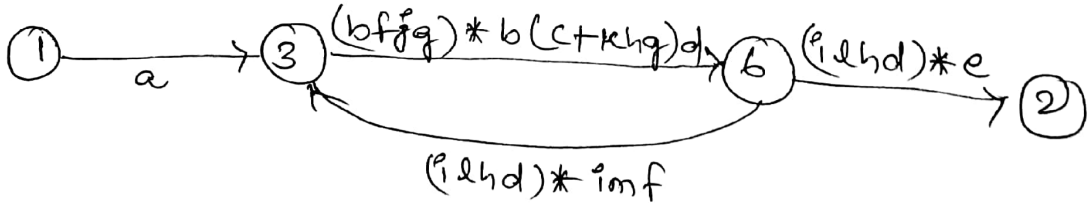
Step (8)

: Remove node 5



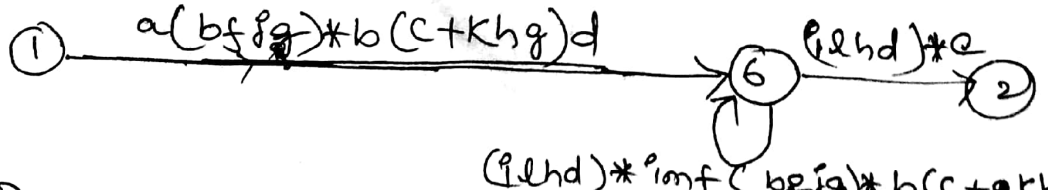
Step (9)

: Remove the loop at node 6 to yields



Step (10)

: Remove node 3 to yields



Step (11) Removing the loop and then node 6 result in the following expression

1 a(bfjg)*b(c+khg)d((glhd)*imf(bjgf)*b(c+gkhg)d)*(glhd)*e 2

Applications :

→ The purpose of node removal algorithm is to present one very generalized concept.

→ The path expression and way of getting it every application follows this common pattern.

→ convert the program & graph into a path expression.

→ Replace the link names by link weights for the property of interest.

→ The path expression has been converted to an expression in some algebra such as ordinary algebra, regular expressions, boolean algebra.

There are 3 categories cases in a flow graph.

case	path expression	weight expression
parallel link	$a+b$	$w_a + w_b$
Serial links	ab	$w_a w_b$
loops	a^*	$\sum_{i=0}^{n} w_{a^i}$

Regular expression and flow anomaly detection:

→ The generic flow anomaly detection problem is that of looking for a specific sequence of operations considering all possible paths through a routine. The operations are SET and RESET, denoted by S and R respectively.

→ You now have a regular expression that denotes all possible sequences of operations in that graph.

Ex: $A = PP$

$B = SRR$

$C = RP$

$T = SS$

The Theorem states that SS will appear in $PP(SRR)^{*}RP$ if it appers in $PP(SRR)^{n}RP$.

$A = P + PP + PS$

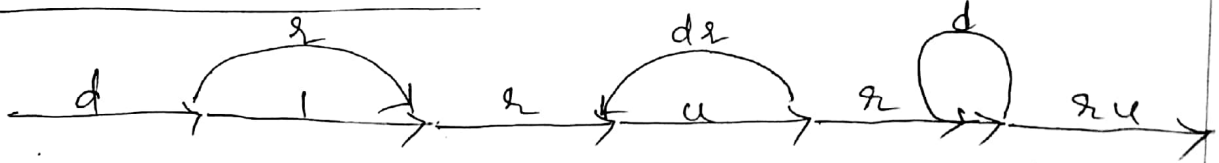
$B = PSR + PS(R + PS)$

$C = RP$

$T = P^4$.

P^4 Sequence in $AB^{n}C$? The Theorem states that we have $(P + PP + PS)[PSR + PS(R + PS)]^{*}RP$.

Data flow Testing example.



The expression is $d(R+1)R[1 + (udR)^{*}uR(1 + d^{*}RU$

$\therefore (dRR + dR)(1 + udRudR)(uRU + uRd^{*}RU)$.

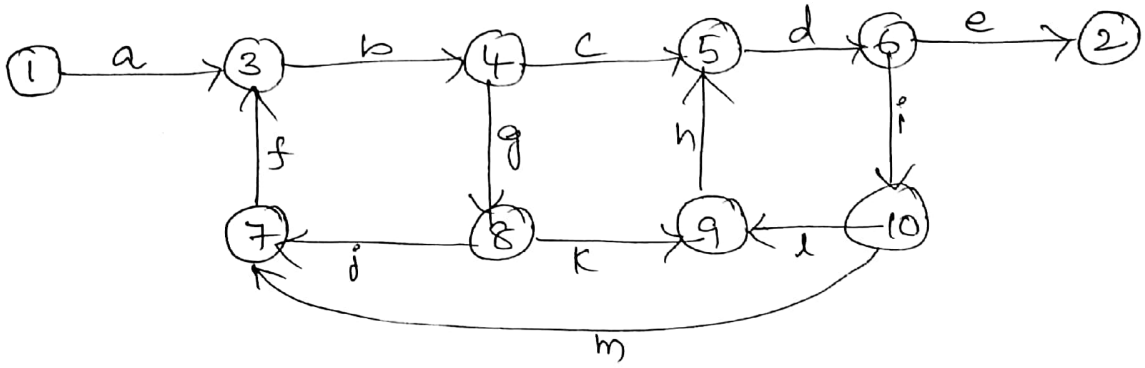
Regular expressions has 3 notations ① Union ② Concatenation ③ Kleen closure.

Short Answer Type questions

1. What is path products?
2. Write regular expressions?
3. Define domain testing?
4. ~~Define string testing~~

Essay Type questions

* 1. Apply reduction procedure algorithm to the following graph.



2. What is meant by domain testing? discuss
   - (a) Nice and ugly domain
   - (b) Write a short note on domain dimensions?
3. Discuss in detail domains and interface testing?
4. (a) many different bugs can result in domain errors? explain?
   - (b) Write about domain closure and domain dimensions?

(5) ~~Discuss about domains and ~~interface~~ testing~~

(5) Explain regular expressions and flow anomaly detection with example?

(6)

(a) With a neat diagram explain the schematic representation of domain Testing?

(b) Explain

(i) Linear domain boundaries

(ii) Non linear domain boundaries

(iii) Complete domain boundaries

(iv) In-Complete domain boundaries?

UNIT-IV

- SYNTAX TESTING:
  - WHY
  - WHAT AND HOW
  - A GRAMMAR FOR FORMATS
- TEST CASE GENERATION
- IMPLEMENTATION AND APPLICATIONS AND TESTABILITY TIPS.

- LOGIC BASED TESTING
  - OVERVIEW
  - DECISION TABLES
- PATH EXPRESSIONS
- KV CHARTS AND SPECIFICATIONS.

SYNTAX TESTING:

Syntax Testing is a type of black box testing techniqe which is used to examine the format and the grammar of the data inputs used in the software application.

Different grammar formats & notations such as BNF, CFG, Regular expressions, graph algebric representations.

→ Syntax Testing consists of the following steps.

① Identify the target language or format

② Define the syntax format of the language. such as BNF (Backus-Naur form)

③ Test and debug the syntax to assure that it is complete and consistent.

A GRAMMAR FOR FORMATS:

Every input has a syntax.

① BNF Notations

ⓐ The elements

every input can be considered as if it were a string of characters.

alpha-characters ::= A|B|C|D|E|....|Z

numerals ::= 1|2|3|....|9

zero ::= 0

Signs ::= b|#|$|%|&|;|:|,|.|?

Space ::= Sp

ⓑ BNF Operators

words : AB, DE, XY ...

nonwords : AAA, A, 11, ..

② CFG

E → E+E
E → E*E
E → id

TEST CASE GENERATION:

A Test case is defined as a set of actions executed to verify a particular feature or functionalities of the software application.

Ex: Test case generation implementation:

GMAIL

Check login functionality and then many possible test cases are

Test case 1:

Check results on entering valid User Id & password.

Testcase 2:

Check results on entering invalid User id & password.

Test case 3:

Check response when User id is empty & login button is pressed, and many more test cases.

Implementation and Application :

1. Execution automation
2. Manual execution
3. Design Automation

Testability Tips

1. Compiler overview
   - lexical analysis
   - Passing
   - code production
2.

2. Typical softwares
3. Seperation of phases.

* LOGIC BASED TESTING :

Logic is one of the most often used words in programmers vocabularies but one of their least used techniques.

Boolean algebra is to logic as arithmetic is to mathematics. Without it, the tester & programmer is cut off from many test and design techniques and tools that is capable those techniques.

Decision tables and KV charts are the best logic based testing techniques.

DECISION TABLE :

→ Decision table is one of the logic based System.

→ It consists of two partitions

(i) Condition Stubs

(ii) Action Stubs.

→ In condition Stubs, condition entries (TRUE/FALSE/I (Immeterial)) are maintaining at rules.
I → means not consider the condition & statement.

→ In Action Stubs, Action entries (results) are maintained at rules.

→ In decision table each column of the table, is a rule that specifies the condition under which the action named in the action stub will take place.

Ex:

A Program calculate the total salary of an employe with the condition that if the working hours are less than or equal to 48 hours, then give normal salary, The hours over 48 on normal working days are calculated at the rate of 1.25 of salary. However. on holidays & sundays the hours are calculated at rate of 2.00 time of the salary define test case Using decision table Testing?

Sol

Decision Table

		Rules		
		Rule1	Rule2	Rule3
Condition stubs	C1: Working hours ≤ 48	T	I	F
	C2: Working hours > 48	F	T	T
	C3: On holidays/ sundays	I	F	T
Action stubs	A1: Normal salary	X		
	A2: 1.25		X	
	A3: 2.00			X

Test cases

Test case Id	Inputs	Days	expected o/p.
TC1	48	MON-FRI	Normal Salary
TC2	56	MON-FRI	1.25
TC3	70	MON-SUN	2.00.

EX (2)

Decision Table of Printer Trouble Shooter.

		Rules			
		R1	R2	R3	R4
Condition Stubs	C1: Printer does not print	Y	Y	I	Y
	C2: A red light is flashing	Y	Y	I	N
	C3: Printer is unrecognized	Y	N	Y	N
Action Entries	A1: Check the printer computer cable.	X	X		X
	A2: Ensure printer S/W installation	X		X	
	A3: Check for paper jam	X	X		

fig: Decision Table of printer Trouble shooter.

* PATH EXPRESSION :

→ path expression is defined as the some of all individual product terms.

→ It is a logic based testing.

→ It is a structural testing when its applied to structure (control flow graph).

→ A predicate is implemented as a process whose outcome is truth functional values, such as

0 — $\bar{A}$
1 — A true values.

→ The purpose of path expression flow graph

Converts path expression into boolean algebra
Using predicates truth values (A & $\bar{A}$).
True branch label indicates A.
False branch label indicats $\bar{A}$.

→ The path expressions followed by different boolean algebric laws or rules.

✱ The following laws (rules) of boolean algebra:

1. $A+A=A$

   $\bar{A}+\bar{A}=\bar{A}$

2. $A+1=1$

3. $A+0=A$

4. $A+B=B+A$ (commutative law)

5. $A+\bar{A}=1$

6. $AA=A$

   $\bar{A}\bar{A}=\bar{A}$

7. $A\times 1=A$

8. $A\times 0=0$

9. $AB=BA$

10. $A\bar{A}=0$

11. $\bar{\bar{A}}=A$

12. $\bar{0}=1$

13. $\bar{1}=0$

14. $\overline{A+B}=\overline{AB}$ (Demorgan Theorem)

15. $\overline{AB}=\bar{A}+\bar{B}$

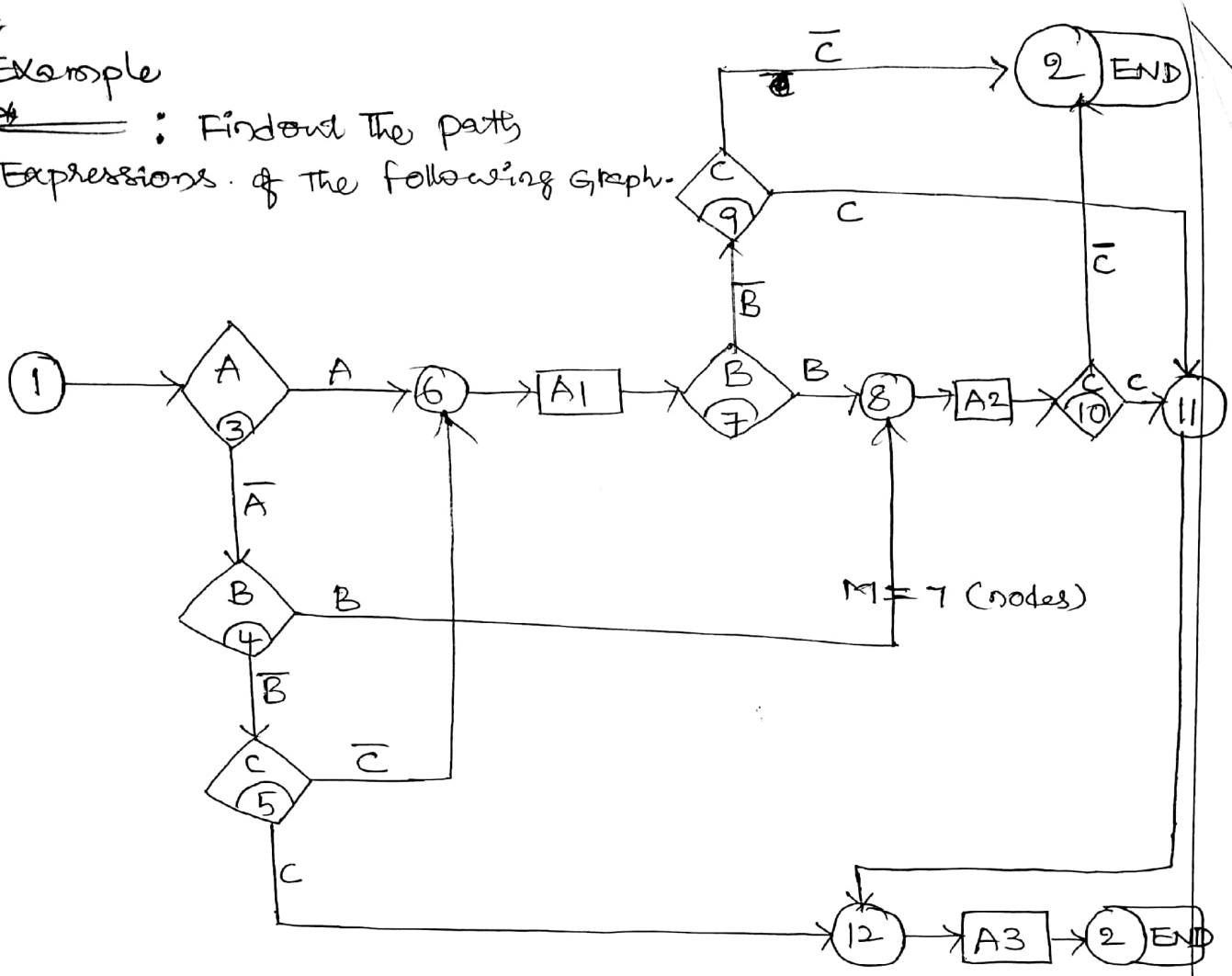
16. $A(B+C)=(AB+AC)$ (Destributive law)

17. $(AB)C=A(BC)$ (multiplication is associative)

18. $(A+B)+C=A+(B+C)$

19. $A+\bar{A}B=A+B$ (Absorption law (20) $A+AB=A$.

Example : Findout the path Expressions of the following Graph.



Solution

$$N6 = A + \bar{A}\bar{B}\bar{C}$$

$$N8 = (N6)B + \bar{A}B$$

$$= AB + \bar{A}\bar{B}\bar{C}B + \bar{A}B$$

(Substitution) rule. distributive law, commutative law

$$N11 = (N8)C + (N6)\bar{B}C$$

$$N12 = N11 + (N8)\bar{C} + (N6)\bar{B}\bar{C}$$

KV charts (Karnough Vietch) :

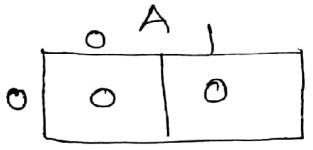
→ Karnough Vietch chart is much similar to "Karnough maps".

→ The main importance of KVC is to reduce a boolean algebric manipulation to graphical representation.

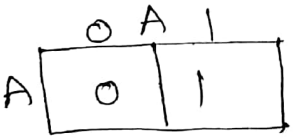
→ In this chart we are using single variable two variables, 3 variables, and 4 variables.

Single variable :

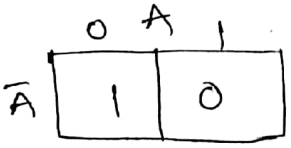
All the boolean functions of a single variable and their equivalence representation of a KV chart shown in the following figures.



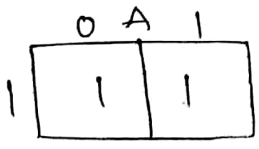
→ The function is never True



→ The function is True when A is true.



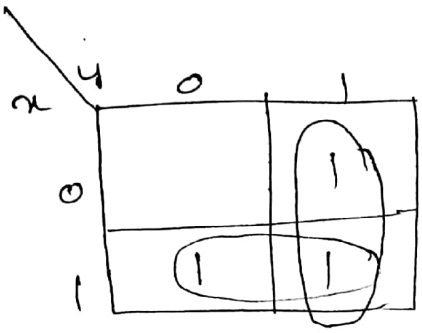
→ The function is true when A is false.



→ The function is always true.

fig: KV charts for Functions of a single variable.

Two variable : → We are using 2-variables in a K-map i.e x, y and we are taking 0, 1

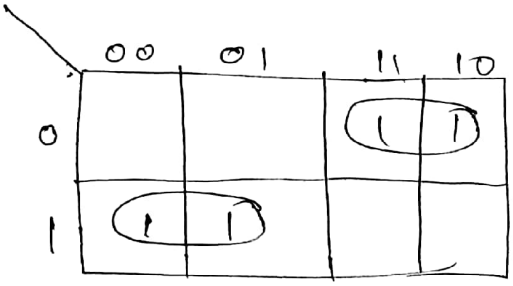


∴ $x + y$

Three variables :

We are using 3-variables in a K-map i.e x, y and z.

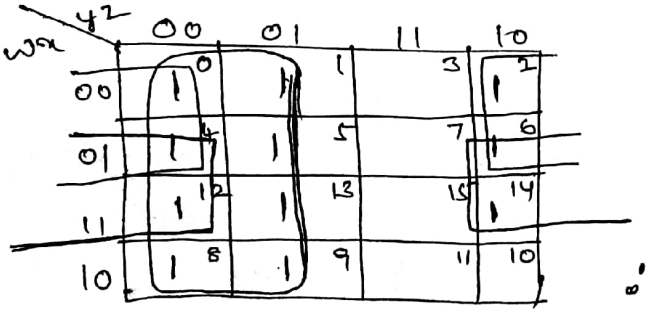
Ex: $F(x,y,z) = \Sigma(2,3,4,5)$



∴ $\bar{x}y + x\bar{y}$

Four variables : We are using 4-variables in a K-map i.e x, y, z, w.

Ex: $F(x,y,z,w) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$



∴ $\bar{y} + \bar{w}\bar{z} + x\bar{z}$.

SPECIFICATIONS :

→ Enter the boolean expressions in a KV chart and check for consistency. If the specifications are consistent, there will be no overlaps.

* Finding and Translating the logic

"IF predicate THEN action".

⇒ predicates are written using the AND, OR, NOT and boolean activities, IF, THEN, IMMATERIAL, EXCLUSIVE OR.

* Ambiguites and contradictions.

* Don't care and impossible terms.

* Testability Tips

Canonical form : It has various successive stages.

(1) predicate Calculator.

(2) Logic analyzer

(3) Domain processor.

(4) A Balm for programmers

(5) How Big, How small?

(6) switches, Flags, and unreachable paths

(7) Essential and Inessential finite state Behavior.

Hardware Logic Testing:

→ Hardware testing is easier than software testing.

→ Hardware testing automation is 10 to 15 years ahead of software testing automation, hardware testing methods and its associated theory is fertile ground for software testing methods.

→ The Hardware designers look more like programmers each day.

Knowledge based Testing

→ The Knowledge-based system also expert System or "artificial intelligence" systems.

→ Knowledge based systems incorporate knowledge from knowledge domain such as medicine, law or civil engineering into a database.

→ The users data is processed through the rule base to yield conclusions and requests for more data. The processing is done by a program called the 'inference engine.

* Stub

"Test stubs are programs that simulate the behaviour of software components or modules".

Short Answer Type questions

1. What is syntax testing?
2. What is logic based testing?
2. Define path expression with examples?
4. Define test case?
5. What is meant by predicate?

Essay Type questions

1 (a) What are decision tables? Do you think decision tables as a basic for test case design justify?

(b) Define
(i) Hardware logic testing
(ii) Knowledge based testing.

2 Write a short notes on path expression and What are the rules for Boolean Algebra? Illustrate the rules to the following expression and explain

$$N6 = A + \bar{A}\bar{B}\bar{C}$$
$$N8 = (N6)B + \bar{A}B = AB + \bar{A}\bar{B}\bar{C}B + \bar{A}B$$
$$N11 = (N8)C + (N6)\bar{B}C$$
$$N12 = N11 + \bar{A}\bar{B}C$$
$$N2 = N12 + (N8)\bar{C} + (N6)\bar{B}\bar{C}$$

(3) Reduce the following functions using K-maps.

$$F(A,B,C,D) = P(4,5,6,7,8,12,13) + d(1,15)$$

(4)

(a) How can we determine paths in domains in logic based testing?

(b) How can we cast the specifications into sentences of the following form explain?

"IF predicate THEN action".

UNIT-V

- STATE
  - STATE GRAPHS AND TRANSITION TESTING
    - STATE GRAPHS
    - GOOD & BAD STATE GRAPHS
    - STATE TESTING
    - TESTABILITY TIPS
  - GRAPH MATRICES AND APPLICATIONS
    - MOTIVATIONAL OVERVIEW
    - MATRIX OF GRAPH
    - RELATIONS
    - POWER OF A MATRIX
    - NODE REDUCTION ALGORITHM.

* STATE :

A state is defined as "a combination of circumstances" attributes belonging (or) to a person (or) Thing.

For example : A moving automobile whose engine is running can have the following states. w.r.t to its transitions.

→ A state can be denoted by the symbol circle ○.

STATE GRAPH :

→ A state graph is defined as a graphical representation of the system behaviour.

→ It consists of set of states, input transitions, output.

→ The state graph is used as a functional testing tool for state testing to identify the state bugs and transition bugs.

Inputs :

A state graph takes inputs that may be values or variables provided to state.

Transition :

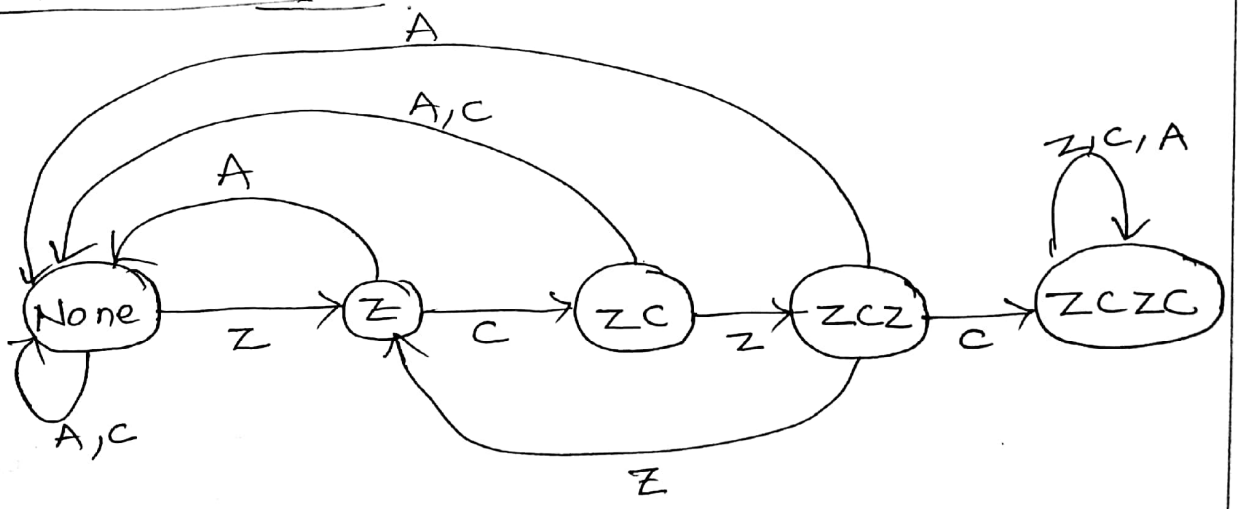
The links joining the state are transitions.

⟶ .

Output :

Based on the input operations performed on the states.

Example of state graph :



State table :

As state graph has a large no. of states and Transitions.

Definitions :

A state table is defined as "tabular representation of the state graph".

→ It consists of rows and columns to store the information of state graph.

→ A state table must specify a state in a each row.

→ A state table must specify an input condition in each column.

State graph can be converted into the state table.

	→ Inputs ←		
STATES	A	C	Z
None	None	None	Z
Z	None	ZC	Z
ZC	None	None	ZCZ
ZCZ	None	ZCZ	Z
ZCZC	ZCZC	ZCZC	ZCZC

fig: state table.

FSM (Finite State Machine)

A finite state machine is an abstract device that can be represented by a state graph having a finite no. of states and a finite no. of transitions between states.

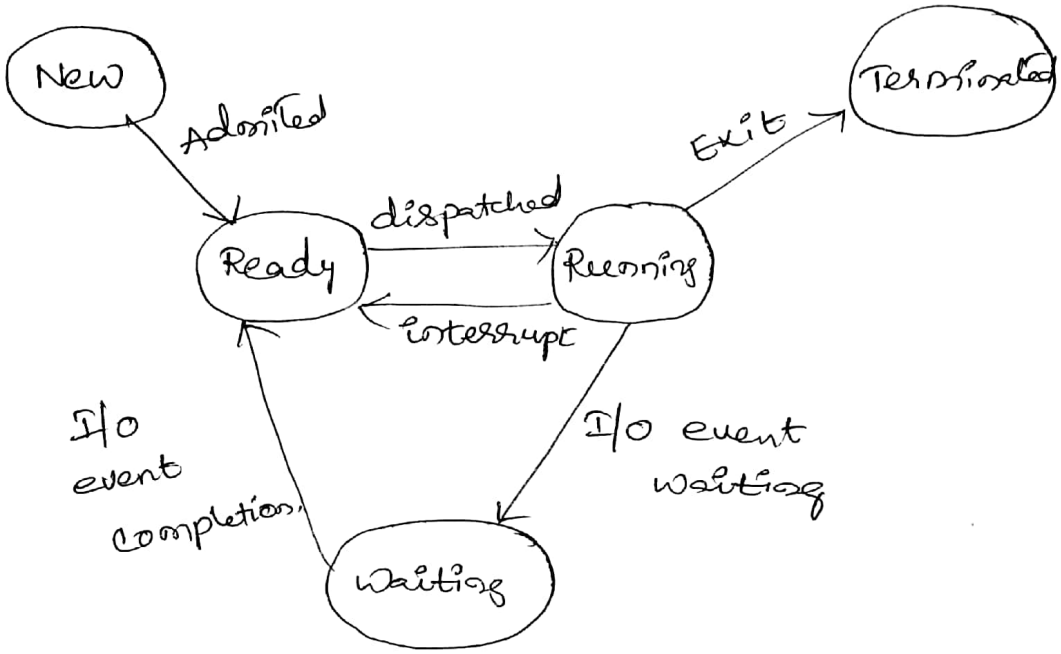


fig: State graph.

State table

States	Inputs					
	Admitted	dispatched	interrupt	I/O event waiting	I/O event completion	Exit
New	Ready	New	New	New	New	New
Ready	Ready	Running	Ready	Ready	Ready	Ready
Running	Running	Running	Ready	Waiting	Running	Terminate
Waiting	Waiting	Waiting	Waiting	Waiting	Ready	Waiting
Terminated	Terminate	Terminate	Terminate	Terminate	Terminate	Terminate

* Good state graph and bad state graph

Good state graph :

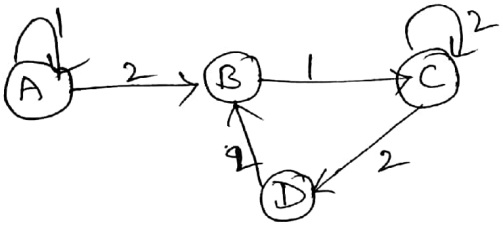
→ It is used in software testing design.

→ The total no. of states = The product of the possibilities of the factors that may make the states.

→ Every transition there is one output action is specified.

→ equivalence and completness obtained.

Ex:



Bad state graph :

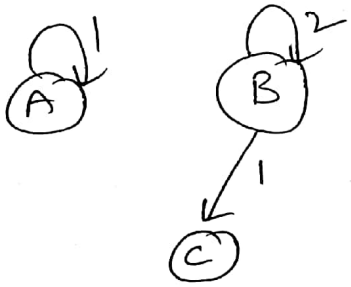
→ In bad state graph communication factors may appear to be impossible.

→ un reachable states (codes)

→ Dead code states

→ Transitioning bugs, Output errors.

Ex: (1)



Ex (2)

Equivalent states :

→ Two states are equivalent if every sequence of inputs starting from one state produces exactly the same sequence of outputs when started from the other state.

Ex:

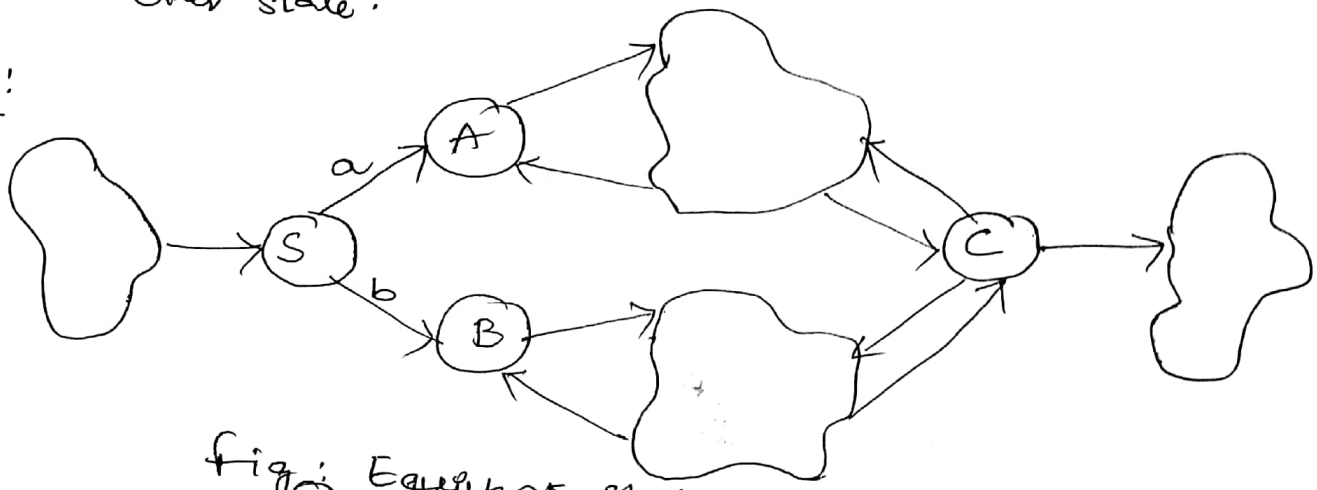


fig ④ Equivalent states.

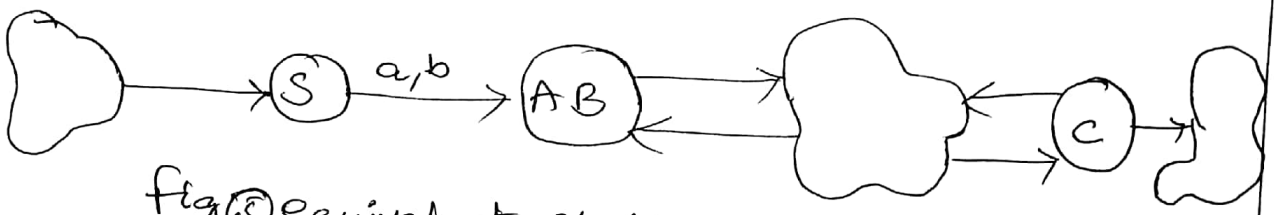


fig ⑤ equivalent states of fig ④ merged.

Unreachable states

→ An unreachable state is like unreachable code.

→ A state that no input sequence can reach.

→ An unreachable state is not impossible, just as unreachable code is not impossible.

Dead states :

→ A dead state is a state that once entered cannot be left.

→ This is not necessarily a bug, but it is suspicious.

→ Dead code means no use code.

State bugs
_____ : State bugs are different ways.

① Impossible states

② Equivalent states

③ Wrong no. of states.

Transition bugs
_____ :

① Unspecified and contradictory Transition.

② Unreachable states

③ Dead states.

④ output errors

⑤ Encoding errors (bugs)

⑥ Incomplete / inconsistant.

STATE TESTING
_____ :

During the process of state testing different symptoms can be occured.

1. Wrong no. of states.
2. Wrong transition for a given state input combination.
3. Wrong output for a given transition.
4. duplicate states.
5. States or sets of states that have become dead.
6. States or sets of states that have become unreachable.

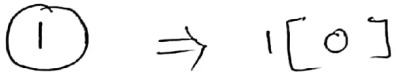
Graph matrix :

→ A Square matrix [ ] in which the no. of rows and no. of columns equal to that of the total no. of nodes then such matrix is referred as graph matrix.

→ In graph matrix each row and column interacting represents the respective rows nodes and column nodes.

→ The no. of rows and column of a matrix will always be equal to the nodes in a graph.

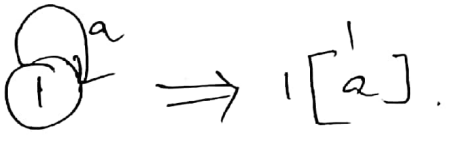
Example :

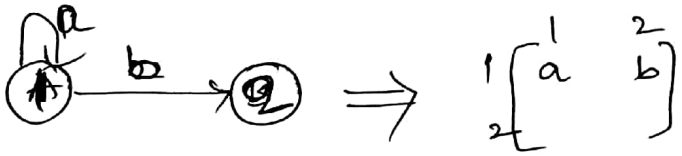


nodes in graph

→ This representation has no path segments from node ① to node ①.

→ The transition between any two nodes can be shown at one and only one place.



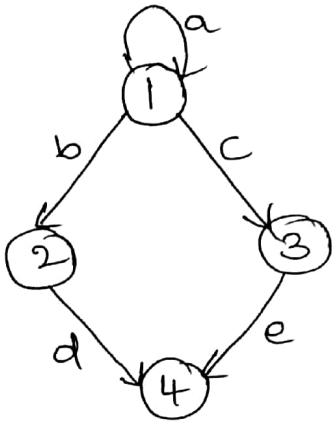


Connection matrix

→ A connection matrix is a matrix that contains link weight between two nodes.

→ These link weights basically provides information regarding the control flow.

Ex! = consider the following graph matrix.



Connection matrix

	1	2	3	4	row weight
1	a	b	c	0	3
2	0	0	0	1	1
3	0	0	0	1	1
4	0	0	0	0	0

0 - represents empty.

1 - represents connect.

Cyclomatic complexity of connection matrix:

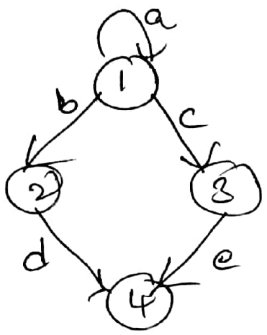
Definition:

The cyclomatic compleaity obtained by subtraction 1 from the total no. of entries in each row and ignoring rows with no. of entities.

→ We obtain the equivalent no. of decisions for each row.

→ adding these values and then adding 1 to the sum yields.

Ex: Graph



Connection matrix

	1	2	3	4	
1	1	1	1	0	3 − 1 = 2
2	0	0	0	1	1 − 1 = 0
3	0	0	0	1	1 − 1 = 0
4	0	0	0	0	0 − 0 = 0

∴ Cyclomatic complexity = 2 + 0 + 0 + 0

= 2

∴ Finally add 1 to cyclomatic complexity

2 + 1 = (3)

Relations

→ A relation is a property that exists between two objects.

Ex: A is connected to B, That means node (A) is connected to node (B), it represents aRb where R is a relation in b/n a & b.

Relations such as $a<b$, $a>b$, $a<=b$, $a>=b$, $a=b$.

The relations are different types.

1. Transitive relations
2. Reflexive relations
3. Symmetric relations
4. Anti-Symetric relation.
5. Equivalence relation.
6. partial Ordering relation

(1) Transitive relation :

A transitive relation is defined as the relation between any two objects implies

If A>B and B>C Then A>C.

Ex: A=10, B=8, C=6 Then 10>8 and 8>6 Then 10>6 True.

(2) Reflexive relation :

A relation 'R' is reflexive if for every 'a'.

aRa.

A reflexive relationship is equivalent to self loop at every node.

Ex: aRa 

(3) Symetric relation :

A relation 'R' is Symetric, if for every a & b,

aRb = bRa.

→ A Symetric relation means that if there is a link from a to b Then there is also link from b to a.

4. Anti-Symetric relation :

A relation 'R' is anti symetric if, for every a&b if aRb and bRa, Then a=b

5. Equivalence relation :

→ The equivalence relation is a relation that satisfy Transitive, reflexive & Symetric relation properties

6. partial ordering relation :

→ The partial ordering relation that satisfies

Transitive

reflexive

Symetric relation properties.

Ex:

⇒ loop free.

There is a atleast one maximum element and there is atleast one maximum element.

* power of a matrix :

Given a square matrix A, for n being a non-negative integer, A^n is defined as the product taking A and multiplying it by itself 'n' times.

$n = 2$; A^n That means $A^2 \Rightarrow A \times A$.

Ex: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$$A^2 = A \times A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1+6 & 2+8 \\ 3+12 & 6+16 \end{bmatrix}$$

$$= \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}.$$

* matrix powers and products :

Given a matrix whose entries are a_{ij}, the square of that matrix is obtained by replacing every entry with

$$a_{ij} = \sum_{K=1}^{n} a_{iK} a_{Kj}$$

more generally, given two matrices A & B, with entries a_{iK} and b_{Kj}, respectively. Their product is a new matrix C, whose entries are c_{ij},

where $c_{ij} = \sum_{K=1}^{n} a_{iK} b_{Kj}$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42}$$

$$c_{13} = a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} + a_{14}b_{43}$$

. . . .

. . . . .

. . . . .

$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42}$$

. . . .

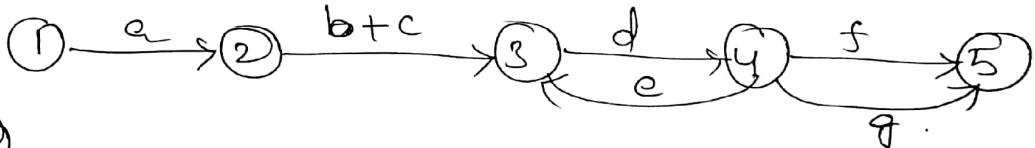
. . . .

. . . .

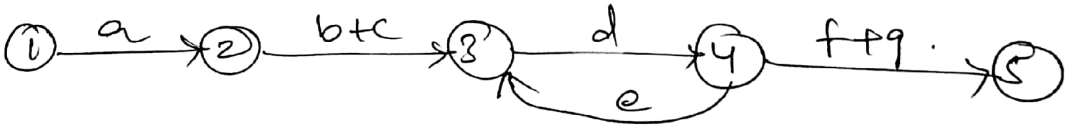
$$c_{44} = a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44}.$$

Node reduction algorithm :

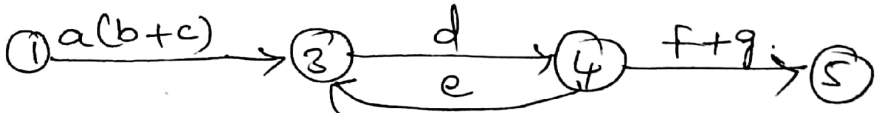
Step 1 — remove parallel loop at node (2) & node (3).



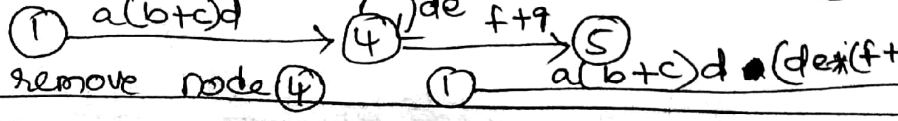
Step 2 — remove parallel loop at node (4) & (5)



Step 3 — remove node (2)



Step 4 — loop free node (3) and remove node (3)



Step 5 — remove node (4)

partitioning Algorithm:

Consider any graph over a Transitive relation. The graph may have loops. We would like to partition the graph by grouping nodes in such a way that every loop is contained with one group & another.

procedures

1. We might want to embed the loops within a subroutine so as to have a resulting graph which is loop-free at the top level.
2. many graphs with loops are easy to analyze if you know where to break the loops.
3. ~~write your own~~ I can recognize loops.

Example

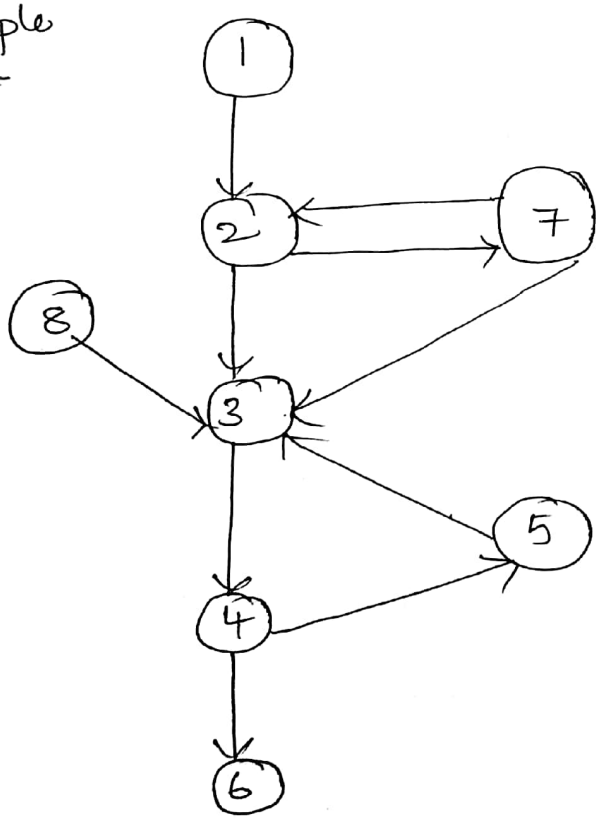
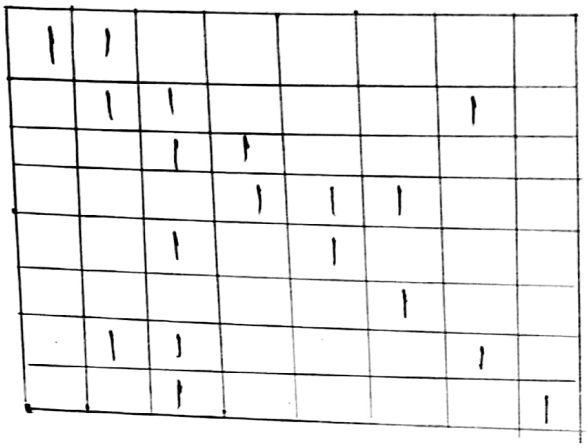
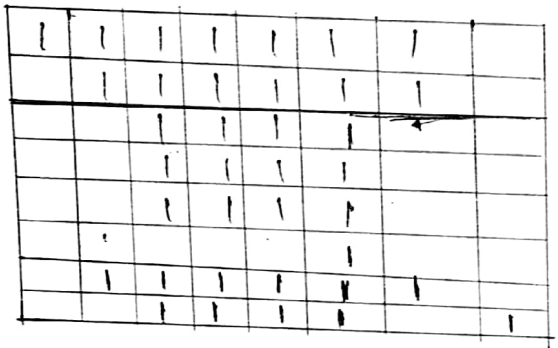


fig: Graph.

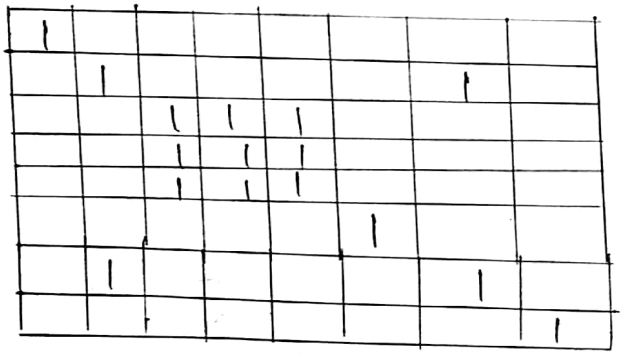
The relation matrix is



The Transitive closure matrix is



Intersection with its Transpose yields

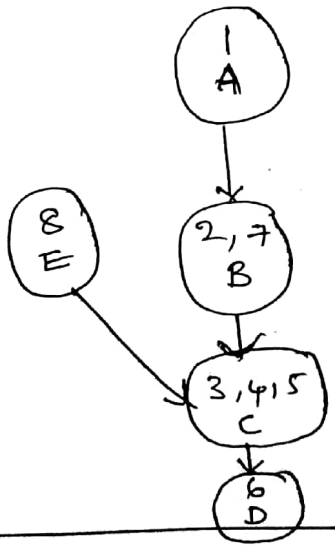


The algorithm leads to the following equivalent node sets,

A = [1]
B = [2, 7]
C = [3, 4, 5]
D = [6]
E = [8]

Whose graph is

	A	B	C	D	E
A	1	1			
B		1	1		
C			1	1	
D				1	
E			1		1



* Node reduction Optimization procedure.

→ The optimum order for node reduction is to do lowest degree node first.

→ The idea is to get the lists as short as possible as quickly as possible.

→ Node of degree 3 (one in and two out or two in and one out) reduce the total link count by one link when removed.

→ A degree 4-node keeps the link count the same, and all higher degree nodes increases the link count.

Short Answer Type questions :

1. Define state graph
2. Describe matrix of a graph
3. Explain State ~~of~~ Testing?
4. Write power of a matrix.

Essay Type questions

1. (a) Define state testing? What is the impact of Bugs in state Testing?
   (b) Explain ~~unreachable~~ unreachable states and dead states in detail?
2. (a) Explain about good state and bad state graph?
   (b) Write in detail about equivalent states?

(3) The behaviour of a finite state machine is 'invariant under all encoding' justifies?

(4) Explain the following in connections to states graph

(a) Inputs and Transitions
(b) Outputs
(c) State tables
(d) Matrix of a graph

(5)
(a) Explain briefly node reduction algorithm?
(b) How can a node reduction optimization be done?

(6)(a) Write a partition algorithm
(b) Write about loops in matrix representation.

(7)(a) Write about matrix powers and products?
(b) Explain cross term reduction and node term reduction optimization.

UNIT-VI

- SOFTWARE TESTING TOOLS:
  - Introduction to Testing,
  - Automated Testing,
  - Concept of Automation Testing
  - Introduction to list of tools like Winrunner, Load runner,
  - Jmeter,
  - About Win Runner
  - Using Win runner,
  - Mapping The GUI
  - Recording Test,
  - Working with Test
  - Enchancing Test.
  - Checkpoints
  - Test script Language (TSL)
  - Running and Debugging Tests
  - Analysing results
  - Batch Tests
  - Rapid Test script Wizard.

* Need for Automation tools (or) Advantages of Automation tools

1. Reducing the Testing effort (cost, schedule time)
2. Human mistakes avoidance
3. Reducing the overall software cost.
4. Improving the software testing process based on latest automation tools.

⑤ Reducing the involvement of tester in executing tests.

⑥ performance and reliability assurances provided these tools.

Disadvantages of Automation tools

① cost more expensive

② Automation tools ~~can't~~ can't debugging

Guidelines for automation tools:

① select the tools based on organization need.

② Tools are tested based on application prototype.

Automation tools are the software testing tools such as:

① win runner

② load runner

③ selenium

* Introduction to win runner tool:

→ win runner is mercury enterprise legacy automated testing tool.

→ win runner is a Test automation tool, design to help customer.

→ As win runner runs tests, it simulates a user by moving and clicking the GUI objects by entering keyboard inputs → It is a licensed software, cost more expensive.

Features of Win runner :

→ Win runner is a functional regression testing.

→ Windows platform dependent (only runs on windows operating systems).

Win runner environment :

(1) windows

(2) Web applications → Ex: Vb, Java, C++

(3) Other Technologies → Ex: SAP, DRACLE

✱ Win runner Testing process :

1. Create a GUI map (Objects)

2. Create tests (Test script language (TSL))

3. Debug Tests

4. Runs Tests.

Load runner tool

1. Load runner is a Software Testing tool from microfocus.
2. It is used to test application.
3. It measures system behaviour and performance under load.
4. It can simulate thousands of user concurrently using applications softwares.
5. This tool mainly used in recording, Analysing and performance of key components.
6. It is a licenced software, cost more expensive.

Selenium tool

1. Selinium is an Open source Web automation tool.
2. It is a ~~free~~ freeware software testing tool, without paying any cost.
3. Nowadays widely used tool in the market.
4. It can automate across multiple OS like, Windows, MAC & Linux and browser like. firefox, chrome, IE.
5. Selenium Test script can be written in Programing languages like Java, C#, Python, Perl, Java script.

* Jmeter Introduction :

1. The Apache Jmeter is pure Java open source software, which was first developed by Stefano mazzocchi of the Apache software foundation.

2. It designed to load test functional behaviour and measure performance of web applications.

3. Jmeter is used for testing web applications or FTP application.

4. Nowadays, it is used for a functional Test, database server test.

Mapping the GUI in winrunner tool :

1. The GUI map file contains the logical names and physical descriptions of GUI objects.

2. Win runner stores information it learns about a window or Object in a GUI map.

3. GUI map provides a centralized Object repository. allowing testers to verify and modify and replay objects.

* Test script language (TSL)

1. Test scripts are used in automated testing.
2. Sometimes, a set of instructions written in a human language, used in manual testing, it is also called Test script.
3. Test script languages are Java script, Vb, html, perl.

Running and Debugging Tests:

Run Test

1. Test files or individual tests may be selected in the Testing tool and run with the Run tests, button or Using the items in the right click context menu.

2. To stop running tests, press abort tests in the testing tool.

Debug Tests

1. To enable test debugging, select debug.
2. Enable debugging from the Test Complete main menu.
3. While your test is running, you can switch to Test Complete, just as you would switch to any windows application.
4. A breakpoint is a location in your script or keyword test where you want the script.

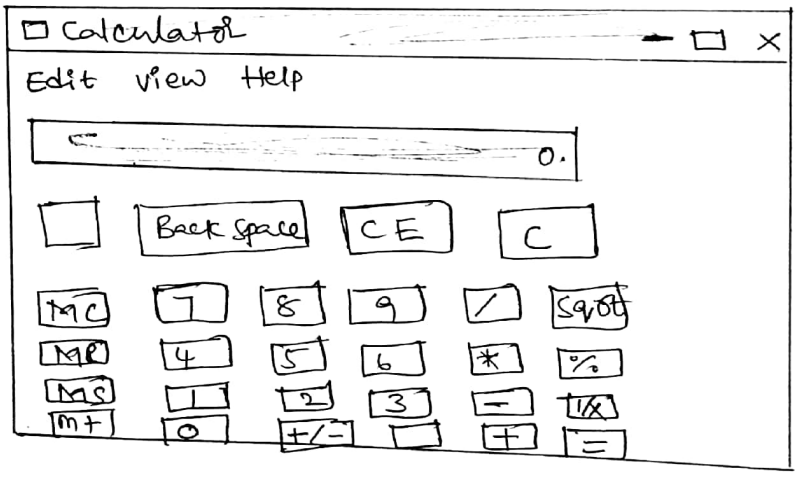
* Batch Tests

* A batch file (.bat) is used in DOS and Windows which is an unformatted text file that consists of series of commands to be executed by the command line interpreter.

* Rapid test script wizard

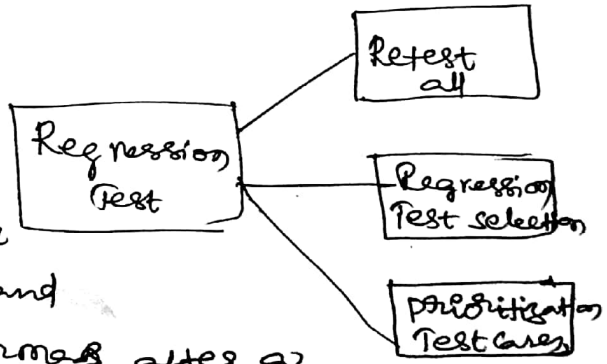
1. The Rapid test script wizard is the fastest way of performing the test process.
2. It systematically opens up all the windows in the applications,
3. Stores the learnt information in the GUI map file. (like clicking a button, selecting the menu item)

Ex: Let us apply this wizard on the Calculator application whose GUI is shown in figure.



* Regression Testing

Regression Testing is a re-running functional and non functional tests to ensure that previously developed and tested software still performs after a change.



Short Answer Type questions :

(1) What are the advantages of win runner tool?
(2) Give any four benefits of automation of tests tools?
(3) List out various automation tools?
(4) Write about batch tests.
(5) Write a short notes on TSL (Test script language)?
(6) Write about Jmeter?
(7) Explain GUI map?
(8) What are the advantages of load runner?
(9) What is Regression Testing?

Essay Type Questions

(1)
(a) List and explain various guidelines automation tools?
(b) What are the advantages and disadvantages of automation tools testing?

(2) Write a short notes on
(a) win runner tool
(b) Load runner tool.

(3) Write a short notes on
(a) Running and debugging Tests
(b) Batch Tests
(c) Rapid Test Script wizard
(d) mapping GUI
(e) Jmeter.
(f) TSL.