DECODE®

A Guide for Engineering Students

Sub. Code : CS615PE

Software Testing Methodologies

JNTU H-R18 - B.TECH., III-II (CSE / IT) Professional Elective - III

- Written by Popular Authors of Text Books of Technical Publications
- Covers Entire Syllabus
- Question - Answer Format
- Exact Answers & Solutions
- Fill in the Blanks with Answers for Mid Term Exam
- MCQs with Answers for Mid Term Exam

SOLVED JNTU QUESTION PAPERS

- Feb. 2017
- Apr. 2018
- Dec. 2018
- May 2019
- Dec. 2019
- Sept. 2021 (R18)
- Mar. 2022 (R18)

first edition : may 2022

Price : ₹ 240/-

ISBN 978-93-5585-055-3

9 789355 850553

TECHNICAL® PUBLICATIONS

SINCE 1993 An Up-Thrust for Knowledge

Dr. Rachna K. Somkunwar

SYLLABUS

Software Testing Methodologies - (CS615PE)

UNIT - I

Introduction : Purpose of testing, Dichotomies, model for testing, consequences of bugs, taxonomy of bugs Flow graphs and Path testing : Basics concepts of path testing, predicates, path predicates and achievable paths, path sensitizing, path instrumentation, application of path testing. **(Chapter - 1)**

UNIT - II

Transaction Flow Testing : transaction flows, transaction flow testing techniques. Dataflow testing : Basics of dataflow testing, strategies in dataflow testing, application of dataflow testing. Domain Testing : domains and paths, Nice & ugly domains, domain testing, domains and interfaces testing, domain and interface testing, domains and testability. **(Chapter - 2)**

UNIT - III

Paths, Path products and Regular expressions : path products & path expression, reduction procedure, applications, regular expressions & flow anomaly detection.
Logic Based Testing : overview, decision tables, path expressions, kv charts, specifications. **(Chapter - 3)**

UNIT - IV

State, State Graphs and Transition testing : state graphs, good & bad state graphs, state testing, Testability tips. **(Chapter - 4)**

UNIT - V

Graph Matrices and Application : Motivational overview, matrix of graph, relations, power of a matrix, node reduction algorithm, building tools. (Student should be given an exposure to a tool like JMeter or Win-runner). **(Chapter - 5)**

TABLE OF CONTENTS

UNIT - I

1 Introduction

1.1 Introduction : Purpose of Testing, Dichotomies, Model for Testing, Consequences of bugs, Taxonomy of Bugs

Q.1 What is software testing and write its purpose?
OR What is meant by testing? Why we need it ?

☞ [JNTU : Part A, April-18, Dec.-19, Marks 2]

Ans. : Testing is the practice of manually or automatically exercising or assessing a system or system components to ensure that they meet stated requirements.

Purpose:

At least half of the time and effort required to create a functional program is spent on testing.

- MYTH : Good programmers build bug-free code. (It's incorrect!!!)
- Even well-written have 1-3 defects per hundred statements, according to history.

Q.2 What is productivity and quality in software ?

Ans. :

- Every manufacturing stage in the manufacture of consumer goods and other products, from component to final stage, is subjected to quality control and testing.
- If defects are identified at any point throughout the manufacturing process, the product is either rejected or returned for rework and correction.
- Productivity is calculated as the sum of material, rework and discarded component costs, as well as the cost of quality assurance and testing.
- There is a tradeoff between quality assurance and manufacturing costs : if not enough time is spent on quality assurance, the reject rate will be high and the net cost will be high as well. Inspection costs will dominate if inspection is good and all errors are identified as they happen, then the net cost will suffer once again.
- Testing and quality assurance costs for 'manufactured' things can range from 2 % in consumer products to 80 % in equipment such as spaceships, nuclear reactors, and aircrafts, where failures pose a life-threatening threat. Software, on the other hand, has a negligible manufacturing cost.
- The cost of defects accounts for the majority of software costs : the cost of discovering them, the cost of fixing them, the cost of building tests to find them and the cost of running those tests.
- In the case of software, quality and productivity are indistinguishable due to the low cost of a software copy.
- Quality assurance, which includes testing and test design, should also focus on bug prevention. A bug that is prevented is preferable to one that is discovered and fixed.

Q.3 List the goals of software testing.

☞ [JNTU : Part A, May-19, Marks 2]

Ans.: Quality assurance includes testing and test design. Testing and test design do not prevent bugs, but they should be able to detect and correct them.

Tests should provide a clear diagnosis, allowing faults to be quickly fixed.

A bug that is prevented is preferable to one that is discovered and fixed.

Test then code

The ideal test activity would be so effective in preventing bugs that it would be unnecessary to test.

A test design must include all expectations, a detailed description of the test technique and the findings of the actual test.

Q.4 Explain the phases of testers life.

Ans. : Phases of a tester's mental life can be divided into five categories :

1. **0th Phase :** (Until 1956: Debugging Oriented) the terms "testing" and "debugging" are interchangeable. In the early days of software development, until testing became a discipline, phase 0 thinking was the standard.
2. The goal of testing in **Phase 1** (1957-1978: Demonstration Oriented) is to demonstrate that software works. During the late 1970s, this was a hot topic. This failed because as testing progresses, the likelihood of demonstrating that software works 'decreases.' To put it another way, the more we test, the more likely that we have to identify a bug.
3. The goal of testing in **Phase 2** (1979-1982 : Destruction Oriented) is to demonstrate that software does not work. This also failed since the software will never be launched because one or more bugs will be discovered. Furthermore, a bug that has been fixed may result in the emergence of a new bug.
4. **Phase 3 :** (1983-1987 : Evaluation Oriented) the goal of testing is to lower the perceived risk of not working to an acceptable level (Statistical Quality Control). The idea is that testing improves the product to the extent that it catches issues and fixes them. When there is sufficient confidence in the product, it is released.
5. **4th Phase :** (1988-2000 : Prevention Oriented) the factor that is taken into account here is testability. One motivation is to cut down on testing time. Checking testable and non-testable code is another reason. Code that is easy to test has fewer bugs than code that is difficult to test. The major goal here is to identify the testing approaches that will be used to test the code.

Q.5 Testing isn't the only thing that needs to be done.

Ans. : First, we must review, inspect, read, conduct walkthroughs and test. The following are the principal methods in decreasing order of effectiveness :

1. Methods of inspection : Walkthroughs, desk checks, formal inspections and code reading are all part of the process. These methods appear to be as effective as testing; however the bugs found aren't as numerous.
2. Style of design : We avoid a bugs, it provides testability, transparency and clarity.
3. Methods of Static Analysis : Strong typing and type checking are included. It gets rid of a whole class of bugs.
4. Languages : Certain types of bugs can be reduced by using the source language. Programmers discover new types of faults in new languages, hence the bug rate appears to be unaffected by the programming languages utilized.
5. Design methodology and development environment : Design methodology can help weavoid a variety of problems. The development process that was employed, as well as the setting in which the technique was implemented

Q.6 State and explain various dichotomies in software testing. ☞ **[JNTU : Part B, May-18, Marks 10, Sept.-21, March-22, Marks 8]**

Ans. : 1. Testing Versus Debugging :

Many individuals mistakenly believe that testing and debugging is the same thing. The goal of testing is to demonstrate that a program has flaws. The goal of testing is to identify the error or misunderstanding that caused the program to fail, as well as to develop and implement program improvements to solve the problem.

Debugging normally follows testing, but the aims, methods and most importantly, psychology is all different.

2. Function versus Structure :

Tests can be created from either a functional or a structural standpoint. The program or system is considered as a black box in functional testing. It is given inputs and its outputs are checked for conformity to a set of rules. Functional testing considers the user's perspective, focusing on the program's functionality and features rather than the program's implementation.

The implementation specifics are examined in structural testing. Structural testing is dominated by factors such as programming style, control technique, source language, database design and code specifics.

Both structural and functional tests have advantages and disadvantages and they target various types of issues. All problems can be detected by functional tests, but it would take an unlimited amount of time to do so. Structural tests are inherently limiting and even if fully conducted, they cannot uncover all problems.

3. Designer versus Tester :

The tester is the one who actually tests the code, while the test designer is the one who creates the tests. The designer and tester are most likely not the same individual during functional testing. The tester and the coder merge into one person during unit testing.

Because software designers' tests are inclined toward structural considerations by nature, they suffer from the limits of structural testing.

4. Modularity versus Efficiency :

A module is a system's discrete, well-defined, tiny component. Smaller components are more difficult to integrate, while larger modules are more difficult to comprehend. Modularity can be found in both tests and systems. Testing may and should be broken down into modular components as well. Separate modules can be tested using small, independent test cases.

5. Small versus Large :

Large-scale programming entails putting together that are made up of many different components authored by many different programrs. Small-scale programming is something we do for ourselves in the quiet of our own offices. With increasing size, qualitative and quantitative changes occur, as do testing methodologies and quality criteria.

6. Builder versus Buyer :

The majority of software is created and utilized by the same company. This situation, however, is dishonest since it obscures accountability. There can be no responsibility if the function Object() { [native code] } and the buyer are not separated.

The following are examples of distinct roles / users in a system :

- **Builder :** The person who creates the system and is responsible to the customer.
- **Buyer :** Who pays for the system in the hopes of making money from the services it provides ?
- **User :** The system's ultimate beneficiary or sufferer. The user's best interests are also protected.
- **Tester :** Who is hell-bent on destroying the builder ?
- **Operator :** Who is responsible for the blunders of the builders, the buyers' hazy (unclear) specifications, the testers' oversights and the complaints of the users ?

Q.7 Differentiate between debugging and testing.

Ans. : The table below illustrates a few key distinctions between testing and debugging.

Sr. No.	Debugging	Testing
1	Debugging begins with perhaps unknown initial conditions and can only be predicted statistically in the end.	Testing begins with well-defined conditions, follows set methods and yields predictable results.
2	Debugging procedures and duration cannot be limited in this way.	Testing may be planned, organized and scheduled, and it should be.
3	Debugging is a method of reasoning.	A demonstration of inaccuracy or apparent correctness is called testing.

4	Debugging is a programmer's reward (justification).	Testing demonstrates a programmer's inability.
5	Debugging necessitates intuitive leaps, experimentation and a sense of liberation.	Testing should attempt to be predictable, uninteresting, restricted, inflexible, and inhuman in its execution.
6	Debugging is impossible without a thorough understanding of the design.	Without design expertise, a lot of testing can be done.
7	An insider must do the debugging.	Testing can frequently be done by a third party.
8	Debugging that is automated is still a pipe dream.	The execution and design of many tests can be automated.

Q.8 Explain model of testing with suitable diagram.

Ans. : A model of the testing process is shown in the diagram. It consists of three models : an environment model, a program model and a model of the expected bugs.

- **Environment :**

  The hardware and software required to run a program are referred to as the program's environment. Communication lines, other systems, terminals, and operators may all be part of the online system's environment.

  All that interact with and are utilized to build the program under test are included in the environment, including the operating system, linkage editor, loader, compiler and utility functions.

  It's not a good idea to blame faults on the surroundings because the hardware and firmware are reliable.

- **Program :**

  The majority of programs are too complex to fully comprehend.

  The program's notion must be streamlined in order to be tested.

  If the simple model of the program fails to explain the unexpected behavior, we may need to revise it to include more facts and details. If that doesn't work, we will have to change the program.

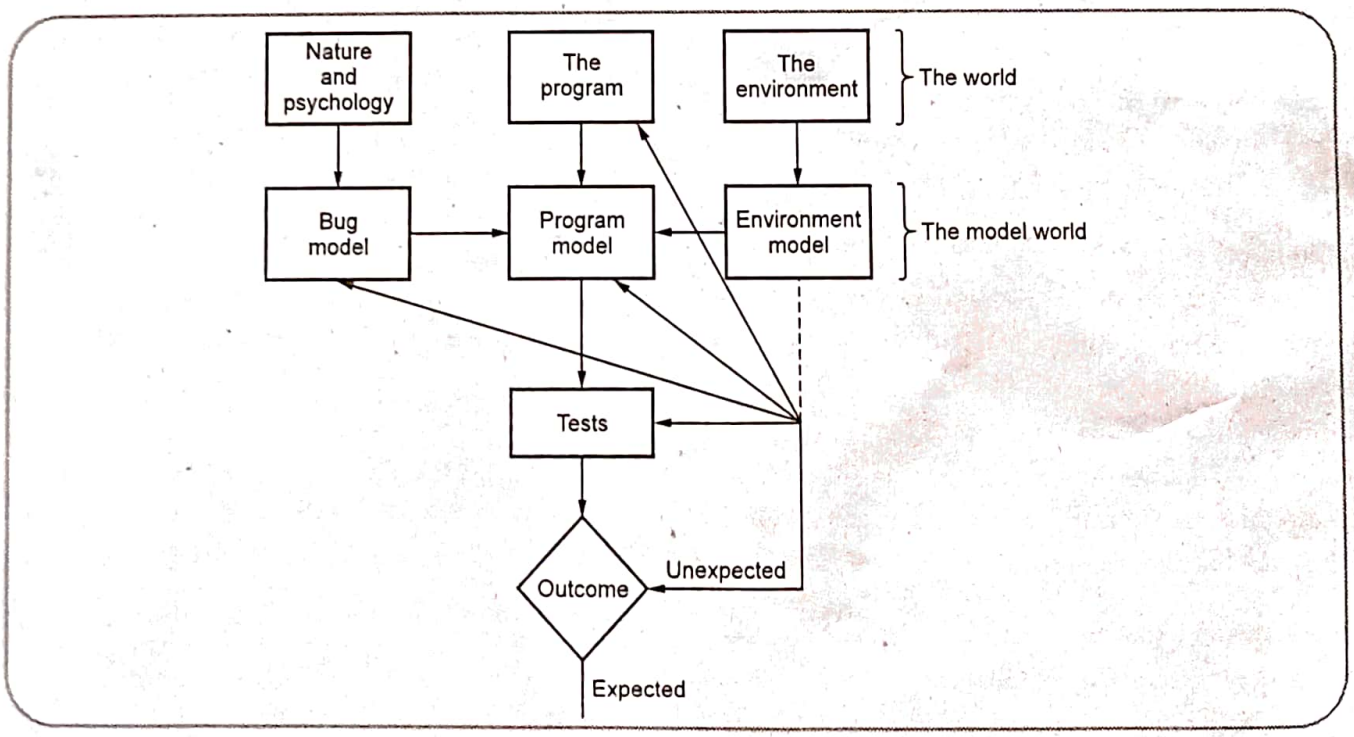


Fig. Q.8.1 Model for testing

- **Bugs :**

  Bugs are more cunning (deceiving but dangerous) than we ever imagined.

  An unexpected test result may cause us to reconsider our definition of an issue and our bug model.

  Many testers have positive ideas about flaws, yet they are frequently unable to test successfully and justify the dirty tests that most applications require.

- **Tests :**

  Tests are formal procedures that require the preparation of inputs, the prediction of outcomes, the documentation of tests, the execution of commands and the observation of results. All of these faults are susceptible to recurrence.

- **Role of models :**

  Creating, selecting, investigating, and refining models is the art of testing. The amount of distinct models we have on hand, as well as their capacity to explain a program's behavior, determines our ability to go through this procedure.

Q.9 Write a short note on : Optimistic notions about bugs

Ans. :

1. The Nice, Tame and Logical Bug Hypothesis : The idea that bugs are nice, tame and logical. (Benign: Non-harmful.)
2. Problem Locality Hypothesis : The idea that a bug found in a component only impacts the behavior of that component.
3. Control Bug Dominance : The assumption that defects are dominated by flaws in program control structures (if, switch, etc.).
4. Code/Data Separation : The idea that bugs are aware of the distinction between code and data.
5. Lingua Salvatore Est. : The concept that language syntax and semantics (e.g., Structured Coding, Strong Typing and so on) eliminate the majority of defects.
6. Corrections Persist : The erroneous notion that a bug that has been fixed will remain fixed.
7. Silver Bullets : The erroneous idea that X (Language, Design technique, representation and environment) provides bug immunity.
8. Sadism Suffices : The widely held belief (particularly among independent testers) that a sadistic streak, a lack of cunning, and intuition are enough to eliminate most problems. Tough bugs necessitate special methods and techniques.
9. Angelic Testers : The idea that testers are better at designing tests than programmers is at writing code.

Q.10 What are feature bugs?

[JNTU : Part A, Dec.-18, Marks 2]

Ans. : A bug is an informal term for a defect, which signifies that software or an application is not performing as expected. A software bug can also be referred to as an issue, error, fault, or failure in software testing. When developers made a mistake or made an error while developing the product, the issue appeared.

- A feature can be incorrect, absent, or unneeded as a result of a specification error (serving no useful purpose). It's easier to spot and fix a missing feature or case. A misplaced feature could have far-reaching design ramifications.
- Removing functionality could make the software more complicated, take more resources and lead to more issues.

Specification and feature bug remedies :

- The majority of feature defects stem from issues with human-to-human communication. One option is to employ formal specification languages or systems at a high level. Such languages and methods provide temporary assistance but do not solve the problem in the long run.
- Short-term Support : Specification languages make it easier to formalize requirements and analyze inconsistencies and ambiguities.

- Long-term Support : Assume we have a fantastic specification language that can be used to build clear, complete specs with clear, complete tests and consistent test criteria.

Q.11 What are structural bugs? Explain.

☞ [JNTU : Part B, Dec.- 18, Marks 5]

OR What are control and sequence bugs? How they can be caught ? ☞ [JNTU : Part B, May-19, Marks 5, Sept.-21, Marks 7]

Ans. : Structural bugs are divided into several categories :

1. Control and sequence bugs :

- Paths left out, unreachable code, improper loop nesting, incorrect loop-back or loop termination criteria, missing process steps, duplicated processing, unnecessary processing, rampaging, GOTO's, ill-conceived (not properly planned) switches, spaghetti code and worst of all, pachinko code are all examples of control and sequence bugs.
- Control flow defects exist for a variety of reasons, one of which is that this domain is receptive (supportive) to theoretical treatment.
- Another reason for control flow defects is the widespread use of outdated code, particularly ALP and COBOL code, which is dominated by control flow bugs.
- Testing, particularly structural testing, more precisely path testing mixed with a bottom line functional test based on a specification, catches control and sequence issues at all levels.

2. Logic bugs :

- Logic errors, particularly those resulting from a misunderstanding of how case statements and logic operators behave individually and in combinations.
- Boolean expressions in deeply nested IF-THEN-ELSE structures are also evaluated.
- Processing bugs are those that are components of logical (i.e. Boolean) processing that aren't related to control flow.
- Control-flow bugs are those that are components of a logical expression (e.g., a control-flow statement) that is used to direct the control flow.

3. Processing bugs :

- Arithmetic bugs, algebraic bugs, mathematical function evaluation bugs, algorithm selection flaws, and general processing bugs are all examples of processing bugs.
- Incorrect conversion from one data representation to another, ignoring overflow, improper usage of greater-than-or-equal, and so on are examples of processing problems.
- Although these bugs are common (12 percent), they are usually detected through competent unit testing.

4. Initialization bugs :

- Bugs in the initialization process are prevalent. Initialization bugs can be inefficient and unnecessary.
- In general, superfluous bugs aren't damaging, but they can wreak havoc on performance.
- Common initialization problems include forgetting to initialize variables before first use, assuming they are already initialized, initializing to the incorrect format, representation, or type and so on.
- Declaring all variables explicitly, as in Pascal, can help solve some initialization issues.

5. Data-Flow bugs and anomalies :

- The majority of initialization issues are a subset of data flow anomalies.
- A data flow anomaly arises when we expect to do anything unusual with data along a particular path, such as utilizing an uninitialized variable, attempting to use a variable before it exists, changing and then not saving or using the result, or initializing twice without an intermediate usage.

Q.12 What is bug ? Explain various kinds of bugs.

☞ [JNTU : Part B, Dec.-19, Marks 5]

Ans. : Bugs are more cunning (deceiving but dangerous) than we ever imagined.

- An unexpected test result may cause us to reconsider our definition of an issue and our bug model.
- Some positive ideas about flaws that many programmers or testers have are frequently ineffective for testing and justifying the dirty tests that most applications require.

1. Functional bugs

- Functional bugs are bugs that affect a software component's functionality. A Login button that does not allow users to login, an Add to Cart button that does not update the cart, a search box that does not answer to a user's query and so on.
- In simple terms, a functional bug is any component in an app or website that does not work as planned.
- When testers undertake complete functional testing for their apps or websites in real-world scenarios, such vulnerabilities are frequently discovered. To prevent delivering poor user experiences in the production environment, teams must guarantee that all functional defects are handled early on.

2. Logical bugs

- A logical defect causes software to function erroneously and breaks its intended workflow. These flaws can cause unexpected software behavior, including crashes. Poorly written code or misreading of business logic are the most common causes of logical bugs.
- Assigning a value to the wrong variable is an example of a logical bug.
- Dividing two numbers rather than adding them together produces unexpected results.

3. Workflow bugs

- Workflow defects are related to a software application's user journey (navigation). Consider the case of a website where a user must complete a form about their medical history. Following the completion of the form, the user has three options :

  1. Save 2. Exit and Save 3. Previous Page

- If the user selects "Save and Exit" from the available options, the user means to save the entered data and then exit. However, if we click the Save and Depart button and then exit the form without saving the data, we will have a workflow bug.

4. Unit level bugs

- Bugs at the unit level are fairly prevalent and they're usually easy to fix. Developers undertake unit testing after the initial modules of software components are completed to guarantee that the little batches of code are working properly. This is where developers run upon a variety of flaws that were ignored during the developing process.
- Because developers deal with a very tiny amount of code, unit level issues are easy to isolate. Furthermore, because reproducing these errors takes less effort, engineers can pinpoint the particular bug and solve it quickly.
- If a developer writes a single-page form, for example, a unit test will check that all of the input fields accept proper inputs and that the buttons are working. Developers run into a unit-level bug if a field doesn't accept the correct characters or integers.

5. System - level Integration bugs

- When two or more units of code authored by different developers fail to interface with each other, system-level integration issues occur. Inconsistencies or incompatibilities between two or more components are the most common causes of these issues. Because engineers must study a greater section of code, such issues are harder to track down and solve. Replicating them is also time-consuming.
- System-level integration flaws might include memory overflows and inefficient interfaces between the application UI and the database.

6. Out of bound bugs

- When a system user interacts with the UI in an unanticipated way, Out of Bound Bugs appear. These issues occur when a user inputs a value or parameter that is outside the scope of its intended use, such as submitting a much higher or smaller number or an undefined data type as an input value. During functional testing of web or mobile apps, these issues frequently appear in form validations.

Q.13 What are consequences of bugs ?

Ans. : BUGS' IMPORTANCE : The importance of bugs is determined by their frequency, cost of remedy, installation cost and repercussions.

- What is the frequency of that type of bug? Pay special attention to the most common bug kinds.
- Cost of Correction : How much does it cost to fix a bug once it has been discovered? The price is determined by a combination of two factors : (1) the discovery price (2) the cost of correcting the problem. When a defect is detected later in the development cycle, these costs skyrocket. The cost of correction is also determined by the size of the system.
- Cost of installation : The cost of installation is proportional to the number of installations : low for a single user application, but more for distributed systems. Fixing and spreading a single issue could cost more than the entire system's development cost.
- Consequences : What are the bug's ramifications? The implications of bugs can range from minor to fatal.
- BUGS' Repercussions : A bug's consequences can be measured in human terms rather than computer words. On a scale of one to ten, the following are some of the effects of a bug :
- Mild : The bug's symptoms visually (gently) irritate us; a misspelt output or a misplaced printout.
- Moderate : The outputs are ambiguous or redundant. The problem has a negative influence on the system's performance.
- Annoying : The system's behavior is demeaning as a result of the bug. Names are truncated or arbitrarily edited, for example.
- It refuses to handle valid (approved / legal) transactions, which is alarming. We can't get money from the ATM like credit card in not active.
- It loses track of its transactions, which is serious. Not only the transaction itself, but the fact that it happened at all. Accountability has vanished.
- Very Serious : The flaw causes the system to process transactions incorrectly. The system credits our paycheck to another account or converts deposits to withdrawals rather than losing it.
- Extreme : The issues aren't confined to a small number of users or transaction kinds. Instead of being irregular and infrequent, they are frequent and arbitrary) or for unusual instances.
- Intolerable : The database suffers from long-term unrecoverable corruption that is difficult to detect. The possibility of shutting down the system is being seriously considered.
- Catastrophic : Because the system fails, the decision to shut down is taken away from us.
- What could be more infectious than a broken system? One that corrupts other systems while remaining unaffected; one that erodes the social and physical environment; one that melts nuclear reactors and initiates war.

Q.14 What are the factors involved in bug severity ?

Ans. :

1. Cost of correction : Not that important because catastrophic problems are easy to fix, whereas tiny bugs can take a long time to debug.
2. Dependence on Context and Application : Severity is determined by the context and application in which it is employed.

3. Creating Culture Dependency : What matters is determined by the program authors' cultural ambitions. Vendors of test tools are more concerned about problems in their software than game developers.
4. User Culture Dependency : The severity of a situation is also influenced by the user's culture. Users who are inexperienced with PC software go berserk over flaws, but professionals (experts) may just overlook them.
5. The software development phase : The severity of the problem is determined by the stage of development. As it gets closer to field use, any bug becomes more serious and the longer it has been around, the more terrible it becomes.

Q.15 Explain taxonomy of bugs ?

☞ [JNTU : March-22, Marks 5]

Ans. : Taxonomy of bugs :

1) **Requirements, Features and Functionality Bugs;** Refer Q.10.
2) **Structural bugs :** Refer Q.11.
3) **Data bugs :** All defects arising from the specification of data items, their formats, the quantity of such objects and their initial values are classified as data bugs.

   Data bugs are at least as common as programming bugs, but they are frequently ignored.

   Data is being migrated by code : As software evolves, more and more control and processing functions are being placed in tables.

   As a result, there is a growing realization that fixing flaws in code is only half the battle and that data issues require equal attention.
4) **Coding bugs :** Any type of coding fault can result in any of the other types of defects.

   If the source language translator has effective syntax checking, syntax problems are often unimportant.

   We should expect a lot of logic and coding faults if a program has a lot of syntax errors.

   Documentation issues are also considered coding bugs because they can lead maintenance programmers astray.
5) **Interface, integration and system bugs :**
   1. External interfaces are the means by which we communicate with the rest of the world.

      Devices, actuators, sensors, input terminals, printers, and communication lines are among them.

      Robustness should be the key design criterion for any interface with the outside world.

      A protocol should be used for all external interfaces, whether human or machine. It's possible that the protocol is incorrect or that it's been implemented incorrectly.

      Invalid timing or sequence assumptions relating to external signals are examples of other external interface problems. External input or output formats are misunderstood. There isn't enough tolerance for faulty input data.
   2. Internal interfaces : Internal interfaces are similar to exterior interfaces in principle but are more controlled. Communication routines are a great example of internal interfaces.

      The system must adapt to the external environment, but the internal environment, which consists of interfaces with other components, can be negotiated.

      Internal and external interfaces both share the same issue.

      Integration bugs : Bugs relating to the integration of operational and tested components, as well as the interfaces between them, are known as integration bugs.

      Inconsistencies or incompatibilities between components cause these issues.

      Data structures, call sequences, registers, semaphores, communication links, and protocols are examples of communication mechanisms can result in integration issues.

Integration defects are not a large bug category (9%), but they are costly because they are frequently discovered late in the game and demand adjustments in multiple components and/or data structures.

System bugs : System bugs are any flaws that can't be attributed to a single component or their simple interactions, but instead come from the sum of interactions between multiple components such as , data, hardware, and operating systems.

There can be no effective system testing unless all components and integrations have been thoroughly tested.

System flaws are uncommon (1.7 percent), but they are critical because they are frequently discovered after the system has been deployed.

6) **Test and test design bugs :**

Testing : No one is immune to bugs, including testers. Complex situations and datasets are required for testing. They need the execution of code or the equivalent, and as a result, they may have errors.

If the specification is correct, it has been appropriately interpreted and implemented and a competent test has been established; yet, the criterion used to evaluate the software's behavior may be erroneous or impossible. As a result, adequate test criteria must be devised. The more complex the criteria, the more likely they are to contain errors.

Q.16 Define Static Vs Dynamic bugs.

Ans. : Dynamic data is transitory. Whatever their purpose, their lifespan is often limited to the processing duration of a single transaction. A storage object can be used to store information.

Different types of dynamic data exist, each having its own set of qualities, attributes, and residues.

Leftover junk in a shared resource causes dynamic data problems. There are three options for dealing with this :

1) The user cleans up after themselves
2) The resource management cleans up in general
3) There is no clean up.

The shape and substance of static data are both fixed. They can be found directly or indirectly in the source code or database, such as a number, a string of characters, or a bit pattern.

Static data bugs will be solved through compile-time processing.

Information, parameter, and control :

Static or dynamic data can perform one of three functions, or a combination of functions: parameter, control, or information.

Q.17 Define Content, Structure and Attributes :

Ans. : The content of a data structure can be a bit pattern, a character string, or a number.

Content is nothing more than a bit pattern that has no significance unless it is decoded by a hardware or software processor. All data defects cause content to be corrupted or misinterpreted.

The size, shape, and numbers that describe the data object, which is the memory address used to store the content, are referred to as structure. (For example, a two-dimensional array.)

Attributes are the semantics associated with the contents of a data object, and they are related to the specification. (A number, an alphanumeric string, or a subroutine, for example.) Because things become less formal as we move from content to attributes, the severity and sophistication of issues rises.

Q.18 Explain different categories of Interface, Integration, and System Bugs :

Ans. :

1. External interfaces :

- External interfaces are the tools that allow us to communicate with the rest of the world.
- Devices, actuators, sensors, input terminals, printers and communication lines are among them.
- Robustness should be the key design criterion for any interface with the outside world.

- A protocol should be used for all external interfaces, whether human or machine. It's possible that the protocol is incorrect or that it's been implemented incorrectly.
- Invalid timing or sequence assumptions relating to external signals are examples of other external interface problems.
- External input or output formats are misunderstood.
- There isn't enough tolerance for faulty input data.

2. Internal interfaces :

- Internal interfaces are similar to exterior interfaces in principle but are more regulated.
- Communication routines are a great example of internal interfaces.
- The system must adapt to the external environment, but the internal environment, which consists of interfaces with other components, can be negotiated.
- Internal and external interfaces both share the same issue.

3. Hardware architecture :

- Hardware architecture bugs are caused by a lack of understanding of how the hardware works.
- Address generation errors, i/o device operation/instruction errors, waiting too long for a response, erroneous interrupt handling, and so on are examples of hardware architectural flaws.
- The solution to problems with hardware architecture and interfaces is twofold : (1) Proper programming and testing are essential. (2) Hardware interface software is centralized in built by hardware interface specialists.

4. Operating system problems :

- Operating system bugs are a mix of hardware architecture and interface bugs, usually caused by a misunderstanding of what the operating system is supposed to perform.
- For all operating system calls, employ operating system interface specialists and explicit interface modules or macros.
- This method may not completely remove bugs, but it will help to isolate them and make testing easier.

5. Software architecture:

- Interactive bugs are the type of software architecture bugs.
- Unit and integration testing of routines can pass without revealing such issues.
- Many of them are load-dependent and their symptoms only appear when the system is under stress.
- For example, here's a bug sample : Assume there will be no interruptions. Interrupts not being blocked or unblocked, Assumption that memory and registers were or were not initialized and so on.
- For these issues, careful module integration and stress testing of the final system are useful methods.

6. System-level control and sequence bugs :

Ignored timing is one of these problems. Assuming that events occur in a predetermined order, Working with data before all of the data from the disc has arrived Missing, incorrect, redundant, or wasteful process stages, and waiting for an impossible combination of criteria. Highly structured sequence control is the cure for these bugs. Internal, specialized sequence control systems are beneficial.

7. Resource management issues :

- Memory is partitioned into dynamically allocated resources such as buffer blocks, queue blocks, task control blocks, and overlay buffers.
- Memory resource pools are subdivided from external mass storage units such as discs.
- Some resource management and usage flaws include : Required resource not being obtained, wrong resource being utilized, Resource already being used, Resource deadlock and so on.

- Remedy for resource management : A bug-prevention design solution is always better to a bug-detection test procedure.
- In resource management, the solution is to keep the resource structure simple : the fewest types of resources, the fewest pools and no private resource management.

8. Integration bugs :

- Integration bugs are issues with the integration of working and tested components, as well as the interfaces between them.
- Inconsistencies or incompatibilities between components cause these issues.
- Data structures, call sequences, registers, semaphores and communication links and protocols are examples of communication mechanisms can result in integration issues.
- Integration flaws are not a large bug category (9 percent), but they are costly because they are frequently discovered late in the game and demand adjustments in multiple components and/or data structures.

9. System bugs :

- System flaws are any bugs that can't be traced back to a single component or their simple interactions, but instead are the result of a complex set of interactions involving multiple components such as data, hardware and operating systems.
- There can be no effective system testing unless all components and integrations have been thoroughly tested.
- System flaws are uncommon (1.7 percent), but they are critical because they are frequently discovered after the system has been deployed.

Q.19 Explain in detail about test and design bugs and its remedies.

Ans. : Testing : no one is immune to bugs, including testers. Tests necessitate complex scenarios and databases, as well as code or the equivalent to perform, and as a result, they may contain defects. If the specification is correct, it has been appropriately interpreted and implemented and a competent test has been established; yet, the criterion used to evaluate the software's behavior may be erroneous or impossible. As a result, adequate test criteria must be devised. The more complex the criteria, the more likely they are to contain errors.

Remedies : The following are some test bug remedies:

1. Test debugging : The first line of defence against test errors is to test and debug the tests. When compared to program debugging, test debugging is easier since tests, when properly designed, are simpler than programs and do not require efficiency sacrifices.
2. Test quality assurance : Programmers have the right to inquire about how quality is verified through independent testing.
3. Automated test execution : The history of bug elimination and prevention in software is inextricably linked to the history of programming automation aids. To limit the occurrence of programming and operation errors, assemblers, loaders, and compilers are developed. Various test execution automation technologies nearly remove test execution bugs.
4. Test design automation : Much of test design can and has been automated, just as much of software development has been. Automation, whether for software or testing, reduces the number of bugs for a certain productivity rate.

1.2 : Flow Graphs and Path Testing : Basics Concepts of Path Testing, Predicates, Path Predicates and Achievable Paths, Path Sensitizing, Path Instrumentation, Application of Path Testing

Q.20 Explain different data object states in data flow graphs. ☞ [JNTU : Part B, Dec.-19, Marks 5]

Ans. : State and Use of Data Objects : Data Objects can be produced, terminated and used. They can be utilized in two ways : (1) As part of a calculation

(2) As a control flow predicate these options are represented by the icons below :

a. Defined : d - defined, created, initialized etc.

b. Killed or undefined : k - Killed, un-defined, released etc.

c. Usage : u - used for something, (c - used in calculations, p - used in a predicate)

d. Defined : When an object appears in a data declaration, it is clearly specified. Alternatively, when it occurs on the assignment's left hand side, it is implied. It can also be used to denote the opening of a file. It has been decided to allocate a dynamically allocated object. Something has been pushed to the top of the stack. A record has been made. Undefined (k) : When an object is released or otherwise made inaccessible, it is killed on undefined.

When the contents of the container are no longer known with total certainty (perfection), objects that have been dynamically allocated are returned to the availability pool. Records are being returned and after it has been popped, the former top of the stack. An assignment statement can instantly kill and redefine. If we execute a new assignment such as A : = 17, for example, we have killed A's prior value and redefined it.

A (u) Usage : When a variable occurs on the right hand side of an assignment statement, it is utilized for computation (c). The contents of a file record are read or written. When it appears directly in a predicate, it is utilized in a Predicate (p).

Q.21 List the elements of flow graph and explain each element with suitable diagram.

☞ [JNTU : Part B, Dec.-19, Marks 5]

OR Discuss various flow graph elements with their notations. ☞ [JNTU : Part B, April-18, Marks 5]

Ans. : Flow Graph Elements : A flow graph contains four different types of elements.

(1) Process block (2) Decisions

(3) Junctions (4) Case statements

1. Process block :

A process block is a set of program statements that are not interrupted by decisions or junctions.

- It is a set of statements in which if any one of the block's statements is performed, the entire block is executed.
- A process block is a single or hundreds of statements long piece of straight line code.
- There is only one way in and out of a process. A single statement or instruction, a sequence of statements or instructions, a single entry/exit subroutine, a macro or function call, or a combination of these can make up a macro or function call.

2. Decisions :

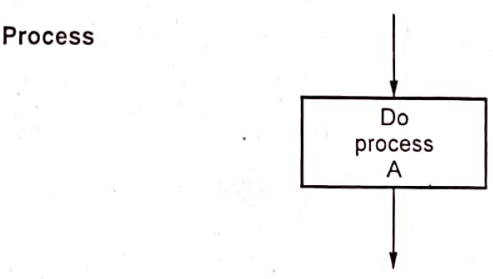
- A decision is a control flow divergence point in a program.
- Conditional branch and conditional skip instructions in machine language are instances of judgments.
- In control flow, most decisions are two-way, but some are three-way branches.

3. Case statements :

- A case statement is a decision tree with multiple paths.
- A jump table in assembly language and the PASCAL case statement are both examples of case statements.
- There are no differences between decisions and case statements in terms of test design.

4. Junctions :

- In a program, a junction is a point when the control flow can join.
- Junctions include the target of an ALP jump or skip command, as well as a label that is a GOTO target.



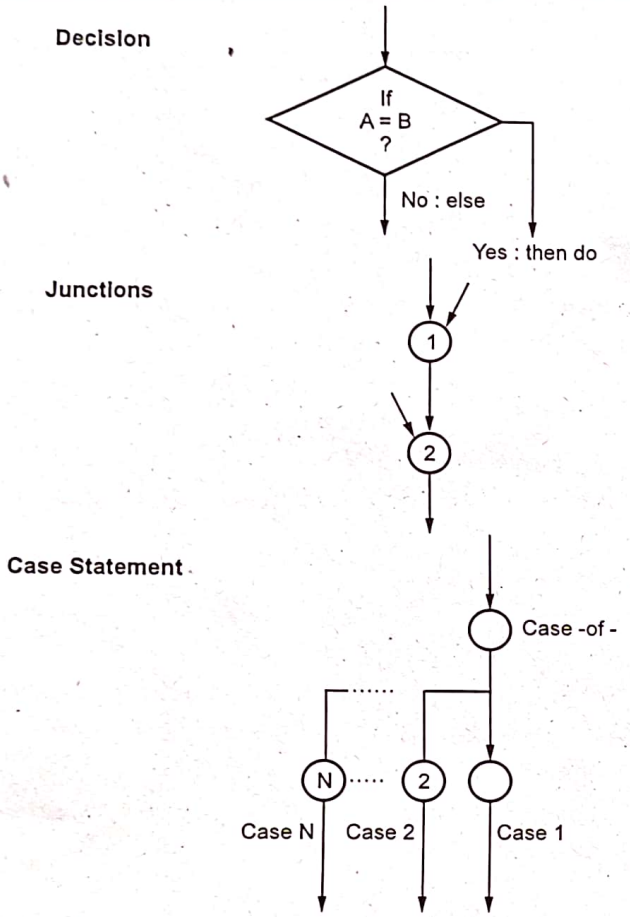


Fig. Q.21.1 Elements of flow graph

Q.22 Describe notational evolution of control flow graph with example. ☞ [JNTU : Part B, Dec.-18, Marks 5]

Ans. : The control flow graph depicts the program's structure in a simplified manner. Changes in nomenclature made during the construction of control flow graphs :

- The process boxes were unnecessary. Every line has an implicit method for linking junctions and making judgments.
- We don't need to know the details of the decisions; we just need to know that there is a branch; and we don't need to know the names of the target labels; we just need to know that they exist. As a result, we may simply replace them with numbers.
- To further grasp this, we'll look at an example (Fig. Q.22.1) written in Programming Design Language, a FORTRAN-like Programming Language (PDL). For better comprehension, the program's flowchart (Fig. Q.22.2) and flowgraph (Fig. Q.22.3) are included below.
- The standard one-for-one classical flowchart is presented in Fig. Q.22.2 as the first stage in translating the program to a flowchart. It's worth noting that the level of complexity has increased, the level of clarity has reduced, and we've had to introduce auxiliary labels (LOOP, XX and YY) that have no program counterpart. We consolidated the process steps in Fig. Q.22.3 and replaced them with a single process box. A control flow graph is now available.
- However, this representation is still excessively cluttered. We simplify the notation even more to get at Fig. Q.22.4, where we can see the control flow for the first time.

```
CODE* (PDL)
          INPUT X, Y
          Z : = X + Y
          V : = X – Y
          IF Z > = ∅ GOTO SAM
JOE :     Z : = Z – 1
SAM :     Z : = Z + V
          FOR U = ∅ TO Z
          V(U),U(V) : = (Z + V) * U
          IF V(U) = ∅ GOTO JOE
          Z : = Z – 1
          IF Z = ∅ GOTO ELL
          U : = U + 1
          NEXT U
          V(U – 1) : = V(U + 1) + U(V – 1)
          ELL:V(U + U(V)) : = U + V
          IF U = V GOTO JOE
          IF U > V THEN U : = Z
           Z : = U
           END
```

* A contrived horror

Fig. Q22.1 Program Example (PDL)

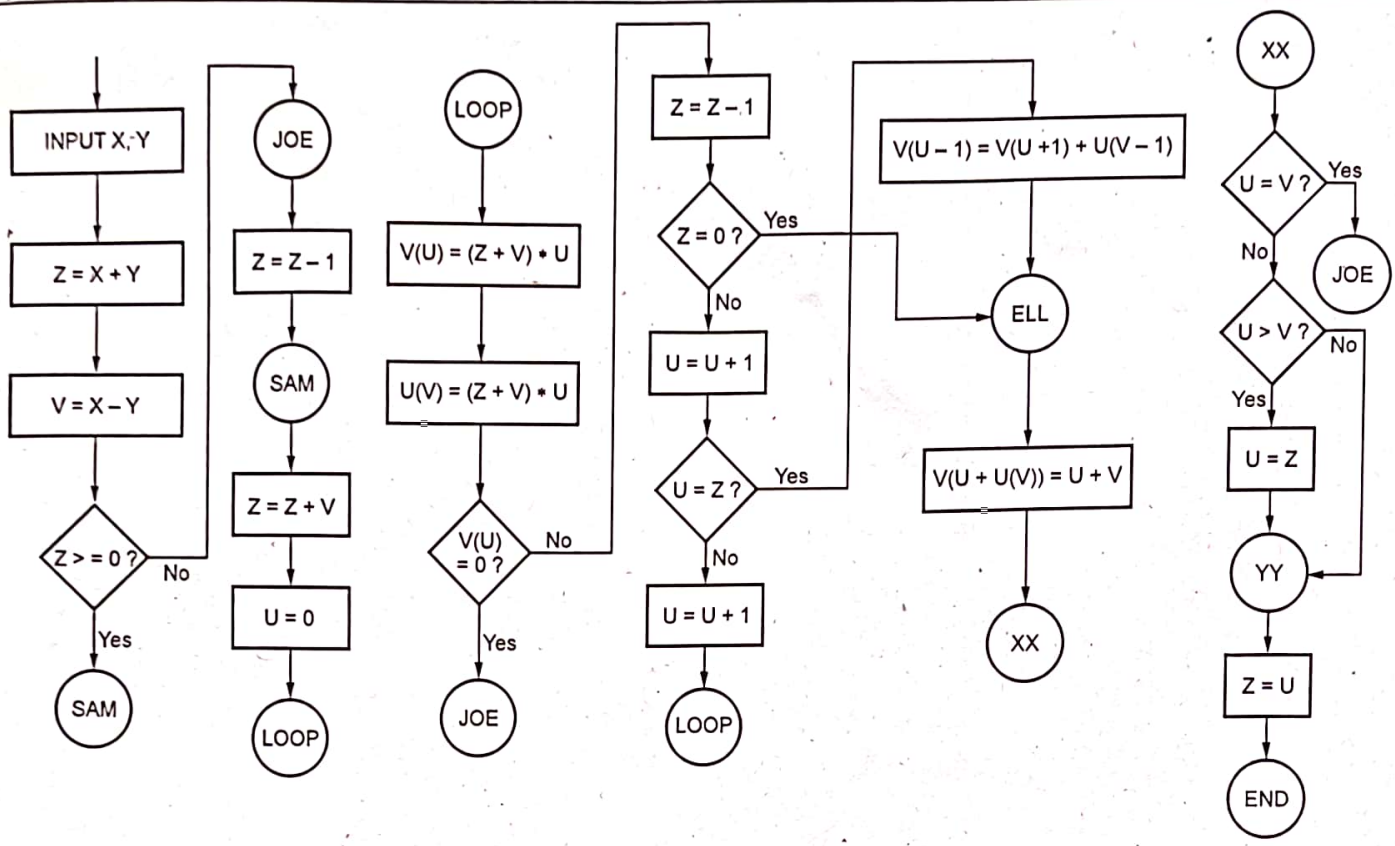


Fig. Q.22.2 : One-to-one flowchart for example program in Fig. Q.22.1

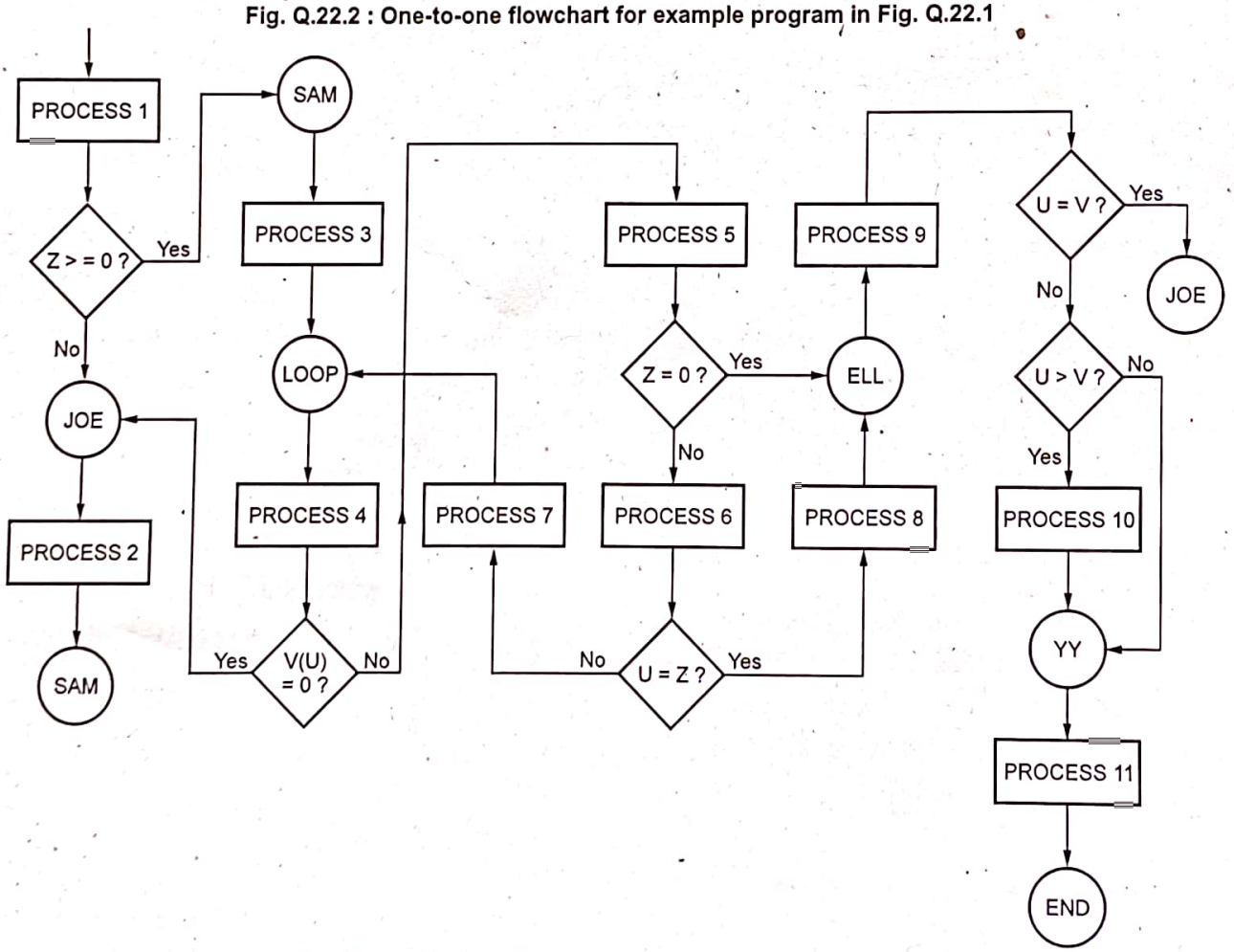


Fig. Q.22.3 Control Flow graph for example in Fig. Q.22.1

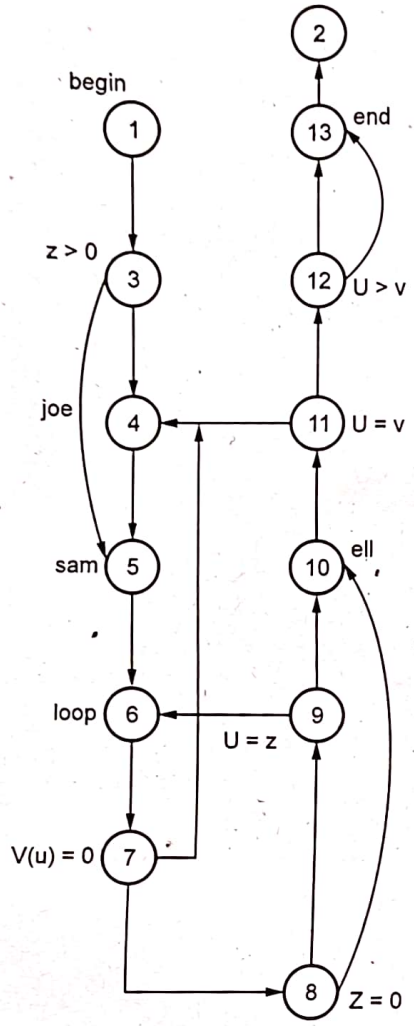


Fig. Q.22.4 Simplified flow graph notation

Fig. Q.22.5 shows the final transformation, which removes the node numbers for a more straightforward depiction. The best method to work with control flow graphs is to utilize the simplest representation feasible - that is, only as much information as we need to correlate back to the source program or PDL.

Q.23 What is meant by program's control flow ? How is it useful for path testing ?

☞ [JNTU : Part B, April-18, Marks 5]

Ans.: Data flow testing is a set of test methodologies that involves choosing paths through a program's control flow to investigate sequences of events involving the status of variables or data objects.

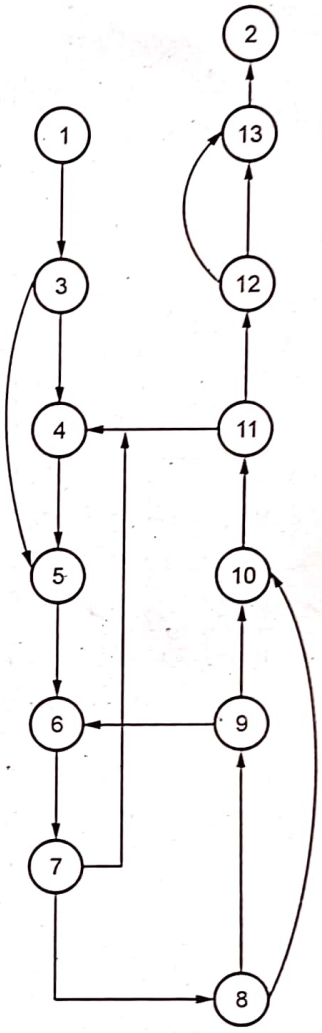


Fig. Q.22.5 Even simplified flow graph notation

The points at which variables get values and the points at which these values are used are the focus of dataflow testing.

The control flow graph of a program is designed to determine a collection of linearly independent paths of execution in the path testing approach. The number of linearly independent paths is determined using cyclomatic complexity and then test cases are generated for each path.

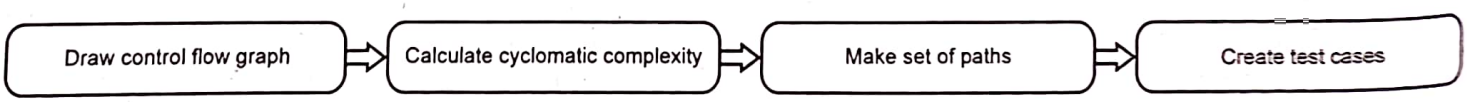


Fig. Q.23.1 Control flow in path testing

Q.24 Explain linked list representation of flow graph.

Ans. : The subtleties of the control flow inside a program are often inconvenient, despite the fact that graphical representations of flow graphs are enlightening.

Each node in a linked list representation has a name, and each link in the flow graph has an entry on the list. Only the data relevant to the control flow is displayed.

```
1     (BEGIN)     : 3
2     (END)       : Exit, no outlink
3     (Z>Ø?)      : 4 (FALSE)
                  : 5 (TRUE)
4     (JOE)       : 5
5     (SAM)       : 6
6     (LOOP)      : 7
7     (V(U)=Ø?)   : 4 (TRUE)
                  : 8 (FALSE)
8     (Z=Ø?)      : 9 (FALSE)
                  : 10 (TRUE)
9     (U=Z?)      : 6 (FALSE) = LOOP
                  : 10 (TRUE) = ELL
10    (ELL)       : 11
11    (U=V?)      : 4 (TRUE) = JOE
                  : 12 (FALSE)
12 (U > V?)       : 13 (TRUE)
                  : 13 (FALSE)
13                : 2 (END)
```

Fig. Q.24.1 Notation for a linked list control flow graph

Q.25 Explain flowgraph - Program correspondence.

Ans. : A flow graph, like a topographic map, is a visual depiction of a program rather than the program itself. Because many program structures, such as if-then-else constructs, are made up of a combination of decisions, junctions and processes, we can't always correlate the elements of a program in a unique way with flow graph parts.

It is not always possible to convert a flow graph piece to a statement and vice versa.

(as seen in Fig. Q.25.1)

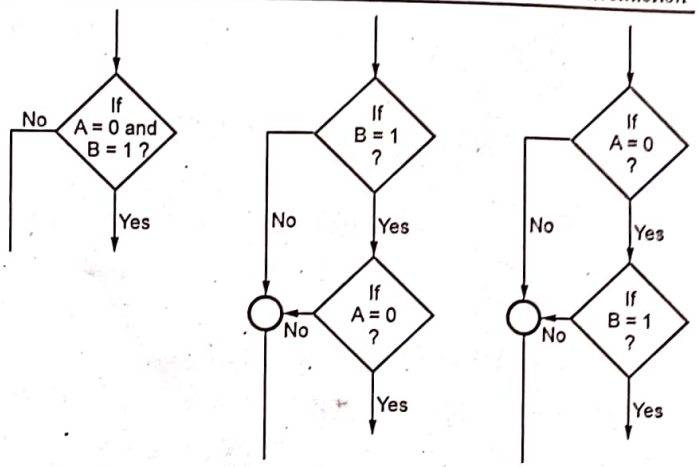


Fig. Q.25.1 Flow diagrams with different logic

(Statement "IF (A = 0) AND (B = 1) THEN . . .")

Incorrect translation from flow graph to code while development can result in defects and bad translation during test design can result in missing test cases and bugs that aren't identified.

Q.26 Define : Flowgraph and flowchart generation.

Ans. : Flowcharts can be

1. Handwritten by the programmer in one of two ways.
2. Generated automatically by a flowcharting program using a mechanical analysis of the source code.
3. A flow charting program, based in part on structural analysis of the source code and in part on directives given by the programmer, produces semi-automatically.

There are only a few control flow graph generators available.

Q.27 What is path testing ? Give a note on path selection and predicates.

☞ [JNTU : Part B, Dec.-18, Marks 5]

Ans. :

- Path testing refers to a group of test methodologies that include carefully selecting a set of test pathways through software.
- We have accomplished some level of test thoroughness if the set of pathways is suitably chosen. Choose enough pathways, for example, to

ensure that each source statement is executed at least once.

- Path testing is the most ancient of all structural test testing
- Path testing is especially useful for unit testing new software. It's a structural method.
- It necessitates a thorough understanding of the program's structure.
- Programmers use it to unit test their own code the most.
- As the size of the software aggregation under test grows larger, path testing's effectiveness gradually deteriorates.

Path selection :

- Using the same criteria we used for structural path testing, choose a set of covering pathways (c1+c2).
- As with control flow graphs, select a covering set of pathways based on functionally meaningful transactions.
- Find the most circuitous, long and unusual path from the transaction flow's entry to exit.

Path predicate :

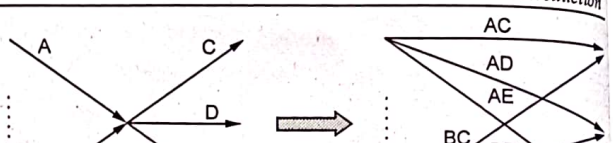
A Path Predicate is a predicate that is related with a path. "x is greater than zero," "x+y>=90," and "w is either negative or equal to 10 is true" are examples of predicates whose truth values lead the routine to follow a particular path.

Q.28 Define cross and parallel term in path testing.

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. :

- The cross-term step is the most basic step in the reduction method; it removes a node, bringing the total number of nodes down by one. We ultimately get down to one entry and one exit node after several applications of this phase. The scenario at a random node that has been designated for removal is depicted in the diagram below :

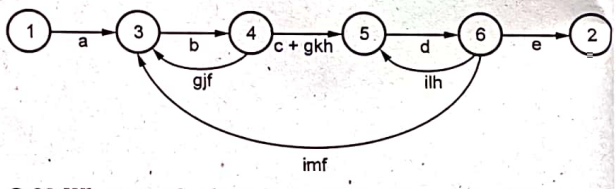


From the above diagram, one can infer :

$$(a + b)(c + d + e) = ac + ad + ae + bc + bd + be$$

Parallel term:

Node 8 was removed, resulting in a pair of parallel linkages connecting nodes 4 and 5. Combine them to form a path expression for an analogous link with the path expression c+gkh, as follows :



Q.29 What are the limitations of path testing.

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. : Though the list of advantages provided by path testing is not full, it is necessary to discuss the few disadvantages/drawbacks provided by this testing in order to gain a comprehensive understanding of its method and features. These are the drawbacks :

- Path testing demands qualified and skilled testers with in-depth understanding of programming and code;
- When the product becomes more complicated, it is difficult to test all paths with this type of testing technique.

Q.30 Explain independence of variables and predicates.

Ans. : The truth values of path predicates are determined by the values of input variables, either directly or indirectly.

If the value of a variable does not change as a result of processing, it is said to be independent of it.

The variable is process dependent if its value can change as a result of the processing.

A process dependent predicate is one whose truth value can vary as a result of processing, while a

process independent predicate is one whose truth value does not change as a result of processing.

A predicate's process dependence does not always imply that the input variables on which it is based are also dependent.

Q.31 Explain correlation of variables and predicates.

Ans. : When the values of two variables cannot be described independently, they are said to be correlated.

Uncorrelated variables are those whose values can be given separately and without limitation.

Correlated predicates are a pair of predicates whose outcomes are dependent on one or more factors in common.

The predicate X==Y, for example, is followed by the predicate X+Y == 8. If we choose X and Y values to meet the first criterion, we may have changed the truth value of the second predicate.

Only if all of the predicates in a routine are uncorrelated is any path through it possible.

Q.32 Explain testing blindness.

Ans. : Testing Blindness is a pathological (destructive) condition in which the desired path is attained for the incorrect reason.

There are three types of blindness testing :

1. **Assignment blindness :** When a flawed predicate appears to work successfully because the exact value specified for an assignment statement works with both the correct and wrong predicate, this is known as assignment blindness.

   For Example :

Correct buggy	Correct buggy
X = 7, if Y > 0	X = 7, if X+Y > 0
then ...	then ...

- If Y=1 is set in the test case, the desired path is followed in both cases, but there is still a defect.

2. **Equality blindness :**

- Equality blindness arises when a prior predicate selects a path that yields a value that is valid for both the correct and buggy predicate.

For Example :

Correct buggy	Correct buggy
X = 2, if X+Y > 3	X = 2, if X > 1
then ...	then ...

- If y=2, the first predicate compels the rest of the path, resulting in the proper and buggy versions taking the same path at the second predicate for any positive value of x.

3. **Self - blindness :** When the buggy predicate is a multiple of the proper predicate, it becomes indistinguishable along that path.

   For example :

Correct Buggy	Correct Buggy
X = A, if X-1 > 0	X = A, if X+1-2 > 0
then ...	then ...

Q.33 Explain in detail about path sensitization

☞ **[JNTU : Part B, Dec.-19, Marks 5, Sept.-21, Marks 8]**

Ans. : Path sensitization is the process of identifying a set of solutions to a path predicate expression.

- Most regular pathways are simple to sensitize, with 80 percent to 95 percent transaction flow coverage (c1+c2) being common.
- Sensitization is the act of defining the transaction; the remaining little proportion is typically quite challenging. If the easy paths have sensitization issues, look for a defect in the transaction flows or a design flaw.

Heuristic procedures for sensitizing paths :

1. Instead of picking paths without thinking about how to sensitize them, try to pick a covering path set that is easy to sensitize and only pick hard to sensitize paths as needed to accomplish coverage.
2. Make a list of all the variables that influence the decision.
3. Determine whether or not the predicates are dependent or independent.
4. Begin with uncorrelated, independent predicates when choosing a path.

5. If coverage cannot be obtained with independent uncorrelated predicates, use correlated predicates to enlarge the path set.
6. If coverage is still lacking, expand the cases to include those with dependent predicates.
7. Finally, employ dependent, associated predicates.

Q.34 Write the applications of path testing.

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. :

1. Components' Integration, coverage and paths

- Unit testing, particularly for new software, relies heavily on path testing methodologies. All called and co-requisite components are replaced with stubs before the new component is tested as a standalone unit. A low-level component simulator that is more reliable than the real thing. The integration difficulties are clarified by path testing. C1 coverage varies between 50 and 85 percent at the system level. Because it is hard to monitor C2 coverage without disturbing the system's performance, we did not provide statistics for C2 coverage in system testing.
- A formal course of study that must be completed concurrently with another.

2. New code :

- New code should always be exposed to sufficient path testing in order to get the desired results. C2
- When it's evident that the stub's bug potential is much lower than that of the called components, stubs are employed. Stubs will not be used to replace old, reliable components. Paths within called components are given some thought. In most cases, we will aim to employ the shortest entry/exit path possible to complete the task.

3. Maintenance :

- The distinction between maintenance testing and new code testing is significant. Maintenance testing, on the other hand, is a whole other scenario. It entails system modifications that are accommodated as needed. The updated component is initially subjected to path testing.

4. Path testing with C1+C2 for rehosting

- When C1+C2 coverage is combined with automatic or semiautomatic structural test generators, we get a very powerful and effective rehosting procedure.
- Software gets rehosted because maintaining the environment in which it operates is no longer cost effective.
- The goal of rehosting is to change the operating environment, not the software that is rehosted. By comparison, rehosting from one COBOL environment to another is simple. Rehosted software can be tweaked to increase efficiency and/or include new features that were difficult to implement in previous settings. The old environment's test suites (collection) and its outcomes become the specification for the rehosted software.

Q.35 Explain achievable path in detail.

☞ [JNTU : Part A, Dec.-18, Marks 3]

Ans. :

Aspect of testing	Manual	Automation
Test execution	QA testers do it manually.	Using automation tools and scripts, this is completed automatically.
Test efficiency	It's time-consuming and inefficient.	More testing in less time, with more efficiency
Types of tasks	Completely manual tasks	The majority of tasks, including real-time user simulations, may be automated.
Test coverage	It's difficult to provide adequate test coverage.	It's simple to ensure that more tests are covered.

Q.36 Explain achievable and unachievable paths in detail.

Ans. :

1. We wish to choose and test a sufficient number of paths in order to obtain a satisfactory definition of test completeness, such as C1+C2.
2. Select a collection of tentative covering paths from the program's control flow graph.
3. Interpret the predicates along the path as needed to represent them in terms of the input vector for any path in that set. Individual predicates are complex or can become compound as a result of interpretation in general.
4. Follow the path, multiplying the different compound predicates to obtain a boolean statement like :

   (A+BC) (D+E) (FGH) (IJ) (K) (l) (L)
5. To get a sum of products form, multiply the expression out.

   ADFGHIJKL+AEFGHIJKL+BCDFGHIJKL+BCEF GHIJKL
6. Each product term refers to a collection of inequalities that, when solved, result in an input vector that directs the routine down the specified path.
7. Find a set of input values for the chosen path by solving any of the inequality sets.
8. If we can come up with a solution, the path is feasible.
9. The path is unattainable if you can't discover a solution to any of the sets of inequalities.

   PATH SENSITIZATION is the process of identifying a set of solutions to the path predicate expression.

Q.37 Explain path instrumentation in detail.

Ans. :

1. Path instrumentation is the process of confirming that the chosen path produced the desired result.
2. Co-incidental Correctness : This term refers to attaining the intended result for the incorrect cause.

The Fig. Q.37.1 shows an example of a method that, regardless of which scenario we choose, produces the same result (Y = 2) for the (unfortunately) chosen input value (X:=16). As a result, the tests chosen in this manner will not indicate whether or not we have achieved coverage. The five situations may, for example, be completely mixed up and the result would still be the same. Path instrumentation is what we need to perform to ensure that the intended path produced the desired result.

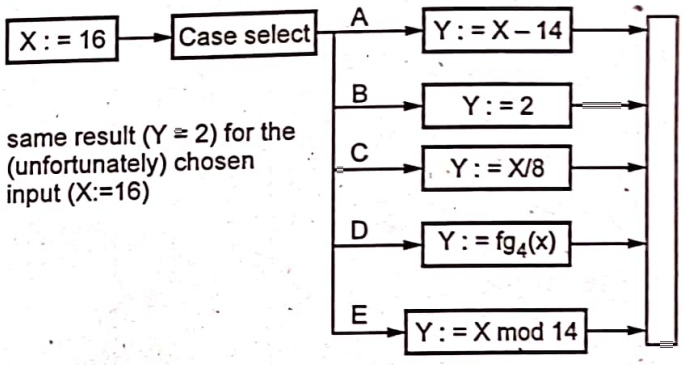


Fig. Q.37.1

The following are examples of instrumentation methods :

1. **Trace interpretation program :** An interpretative trace programme runs each statement sequentially, recording the intermediate values of all calculations, the statement labels traversed and so forth. We have all the information we need to confirm the conclusion and, more importantly, to confirm that it was attained through the planned method if we run the tested procedure under a trace.

   The problem with traces is that they provide significantly more data than we require. In fact, establishing the path from the enormous output dump provided by a typical trace software is more work than simulating the computer by hand to check the path.
2. **Traversal marker or link marker :** A traversal marker or link marker is a simple and useful form of instrumentation.
   - Use a lowercase letter to name each link.

- Instrument the links so that the name of the link is captured when it is clicked.
- If there are no defects, the sequence of letters produced from the routine's input to its exit should exactly match the path name.

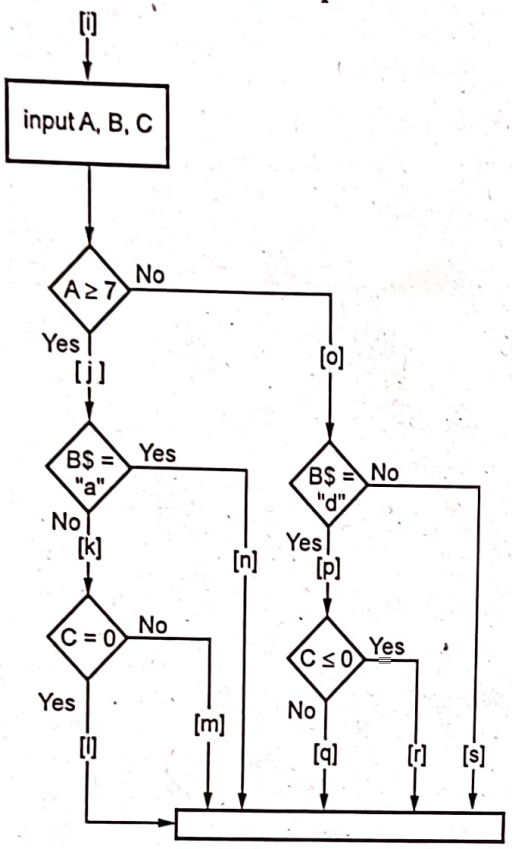


Fig. Q.37.1 : Single link marker instrumentation

Two link marker approach : To overcome the drawbacks of the single link marker method, use two markers per link : one at the start and one at the conclusion. The path name and the beginning and end of the link are now specified by the two link markers.

Link counter : Counters are a less disruptive (and less revealing) instrumentation method. We just increment a link counter instead of pushing a unique link name into a string when the link is browsed. The path length is now confirmed to be as expected. The same issue that led to the development of double link markers also led to the development of double link counters.

Fill in the Blanks for Mid Term Exam

Q.1 _____ is another name for structural testing.

Q.2 The system's final recipient is referred to as the _____.

Q.3 Although no two systems are identical, they can share _____ of the code.

Q.4 _____ are the four layers of testing.

Q.5 _____ are included in evaluating the seriousness of bugs.

Q.6 The most crucial factor to consider when creating an interface is its _____.

Q.7 The programs environment includes both _____.

Q.8 Bugs have a wide range of repercussions, from _____ to _____.

Q.9 The process of combining components to generate a bigger component is known as _____.

Q.10 _____ and _____ are two objectives for testing.

Q.11 In terms of process design, _____ are favoured.

Q.12 _____ begins with conditions that are well-known.

Q.13 _____ and _____ test cases are the two forms of test cases.

Q.14 The _____ is a graphical depiction of the control structure of a program.

Multiple Choice Questions for Mid Term Exam

Q.1 According to the first part of the testers' mental lives _____.

- [a] testing shows software works
- [b] testing shows software doesn't work
- [c] testing=debugging
- [d] testing is not an act

Q.2 Errors in syntax are common.

[a] data bugs [b] logic bugs

[c] coding bugs [d] structure bugs

Q.3 The initial purpose of testing is to determine _____.

[a] bug detection [b] bug prevention

[c] bug correction

[d] to calculate the cost of the bug

Q.4 SQA is _____.

[a] system quality assistance

[b] software quality assistance

[c] system quality assurance

[d] software quality assurance

Q.5 What are the objectives of testing ?

[a] to find the error

[b] to show program has bugs

[c] to correct the error

[d] none of these

Q.6 Functional testing is _____.

[a] black box [b] white box

[c] glass box [d] open box

Q.7 Most defects are eliminated by language syntax and semantics, according to this belief.

[a] control bug dominance [b] data separation

[c] lingua salvator estimate [d] angelic testers

Q.8 Which one is not the consequence of bugs ?

[a] serious [b] extreme

[c] infectious [d] none

Q.9 The pesticide paradox is _____.

[a] testing shows all bugs

[b] complexity of software grows to the limit of managerial ability

[c] a method to find bugs leaves subtler bugs

[d] testing doesn't show all bugs

Q.10 There are no cures for testbugs bugs _____.

[a] debugging

[b] design automation

[c] test execution automation

[d] test consequences

Q.11 An example of bad test case design is _____.

[a] logic bugs [b] data bugs

[c] processing bugs [d] requirement bugs

Q.12 Other than testing, there are a few ways to avoid bugs.

[a] reviews [b] syntax checking

[c] inspections [d] all the above

Q.13 The project's program staff should be _____.

[a] 10-20 [b] 20-30

[c] 25-45 [d] 15-20

Q.14 Testing the entire system is _____.

[a] system testing [b] complete testing

[c] integration testing [d] software testing

Q.15 Can a bug free product delivery be guaranteed with testing ?

[a] No [b] Yes

Q.16 Acceptance test is done _____.

[a] after accepting a system

[b] to know users need

[c] to know whether to accept a system or not

[d] to know the ability of the programmer

Answer Keys for Fill in the Blanks :

Q.1	White box testing	Q.2	user
Q.3	75 %	Q.4	Component, integration, system, and acceptance testing
Q.5	Correction installation, and consequential costs	Q.6	robustness
Q.7	hardware and software	Q.8	mild, catastrophic
Q.9	integration	Q.10	Bug prevention, bug detection
Q.11	flowgraphs	Q.12	Testing
Q.13	Formal, informal	Q.14	control flowgraph

Answer Keys for Multiple Choice Questions :

Q.1	a	Q.2	c	Q.3	b
Q.4	d	Q.5	b	Q.6	a
Q.7	c	Q.8	d	Q.9	c
Q.10	d	Q.11	a	Q.12	d
Q.13	b	Q.14	a	Q.15	a
Q.16	c				

END...

UNIT - II

2 Transaction Flow Testing

2.1 Transaction Flows, Transaction Flows Techniques, Dataflow Testing : Basics of Dataflow Testing, Strategies in Dataflow Testing, Application of Dataflow Testing

Q.1 Explain data flow Vs transaction flow.

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. : Data flow testing : It is used to discover the following issues -

1. To locate a variable that is referenced but not defined,
2. To locate a variable that has been specified but has never been utilized,
3. De-allocating a variable before it is utilized is a method of locating a variable that has been defined several times.
4. Expensive and time-consuming procedure
5. It necessitates a working knowledge of programming languages.

Testing the flow of transactions

1. From the perspective of a system user, a transaction is a unit of work.
2. A transaction is a set of processes, some of which are carried out by a system, others by people or devices outside the system.
3. Transactions start with birth, i.e., they are born as a result of some external act.
4. When the transaction processing is completed, the transaction is removed from the system.

Q.2 What is transaction flow testing ? Explain with example.

☞ [JNTU : Part B, Dec.-19, Marks 5, March-22, Marks 7]

Ans. :

- A transaction is a unit of labor from the perspective of a system user.
- A transaction is a set of processes, some of which are carried out by a system, others by people or devices outside the system.
- Transactions start with birth, i.e., they are born as a result of some external act.

Transaction flow testing techniques :

1. Get the transactions flows :

- Complex systems that process a large number of varied, complex transactions should include explicit representations of the transaction flows, or something similar.
- Because transaction flows are similar to control flow graphs, we should expect to see them at increasing levels of detail.
- Detailed transaction flows are an essential pre-requisite to the logical design of a system's functional test.
- The system's design documentation should include an overview section that covers the primary transaction flows.

2. Inspections, reviews and walkthroughs :

- For system reviews or inspections, transaction flows are a natural agenda item.

- We should do the following during the walkthroughs :

1. Talk about enough transaction types to cover 98 percent to 99 percent of the transactions the system is expected to handle.
2. Instead of using technical words, discuss pathways across flows in functional terms.
3. Request that the designers link each flow to the specification and demonstrate how that transaction, either directly or indirectly, is related to the requirements.

- As path testing is the cornerstone of unit testing, make transaction flow testing the cornerstone of system functional testing.
- Using the same criteria we used for structural path testing, choose a set of covering pathways (c1+c2).
- As with control flow graphs, select a covering set of pathways based on functionally meaningful transactions.
- Find the most circuitous, long and unusual path from the transaction flow's entry to exit.

3. Path selection :

- For loops, extreme values and domain borders, choose additional flow pathways.
- Create extra test cases to ensure that all births and deaths are accurate.
- As soon as feasible, publish and distribute the selected test paths through the transaction flows so that they can have the most impact on the project.

4. Path sensitization :

- Most regular pathways are simple to sensitize, with 80 percent to 95 percent transaction flow coverage (c1+c2) being common.
- The remaining little fraction is frequently challenging.
- The act of defining the transaction is known as sensitization. If the easy paths have sensitization issues, look for a defect in the transaction flows or a design flaw.

5. Path instrumentation :

- In transaction flow testing, instrumentation is more important than in unit path testing.
- The information about a transaction's course must be preserved with the transaction, and it can be kept by a central transaction dispatcher or by individual processing modules.
- Such traces are provided by operating systems or a running log in some systems.

Q.3 What is transaction instrumentation in transaction flow ? Explain with example.

☞ **[JNTU : Part B, Dec.-19, Marks 5]**

Ans. : In transaction flow testing, instrumentation is more important than in unit path testing. The information about a transaction's course must be retained with the transaction and it can be kept by a central transaction dispatcher or by individual processing modules.

- As a representation of a system's processing, transaction flows are introduced.
- Functional testing is carried out using the same approaches that were used to create control flow graphs.
- To an independent system tester, transaction flows and transaction flow testing are what control flows and path testing are to a coder.
- The transaction flow graph is used to develop a software behavioral model that leads to functional testing.
- The transaction flow graph is a representation of the system's behavior structure (functionality).

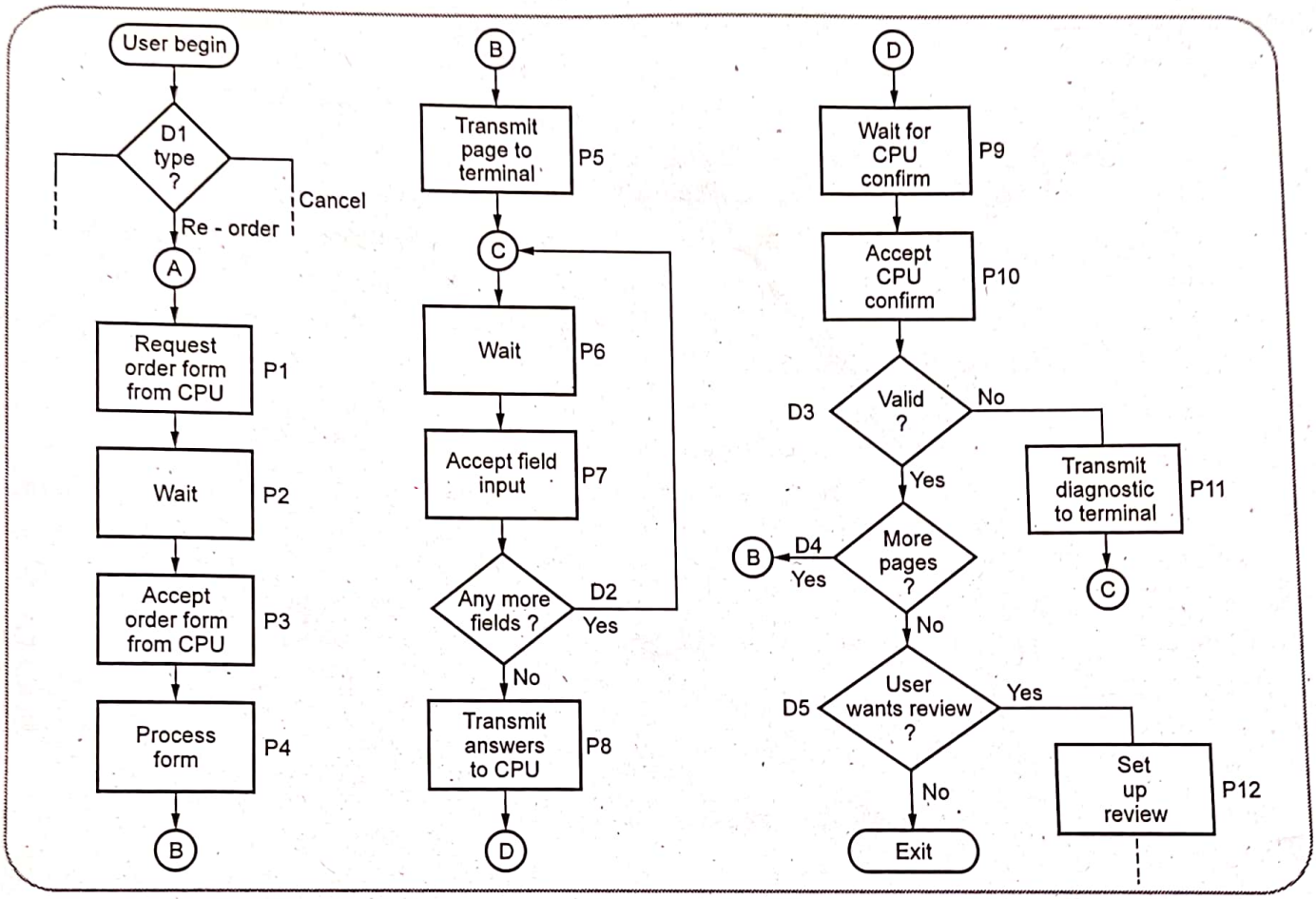


Fig. Q.3.1 Transaction flow

Q.4 What is meant by transaction flow testing. Discuss its significance. ☞ **[JNTU : Part B, April-18, Marks 5]**

Ans. : Refer Q.2.

Significance : Transaction flows are critical for defining the requirements of complex systems, particularly online systems. A large system, such as an air traffic control or airline reservation system includes thousands of different transaction flows rather than hundreds.

Q.5 Distinguish between control flow and transaction flow. ☞ **[JNTU: Part B, May-19, Marks 5]**

Ans. : Transaction flow : Transaction flows are critical for defining the requirements of complex systems, particularly online systems. A large system, such as an air traffic control or airline reservation system includes thousands of different transaction flows rather than hundreds.

A transaction flow is the path a payment travels from beginning to end, from approval to settlement. When a customer buys anything with a credit card, the transaction goes via a stream or transaction flow, going through a number of players before being settled.

Control flow :

Control-flow testing is a type of structural testing that employs the control flow of a program as a model. Control-flow testing is focused on picking a set of test pathways through the program with care. The paths selected are used to attain a given level of testing thoroughness.

Q.6 Discuss about complication in transaction - flow testing. ☞ [JNTU : Part B, Dec.-18, Marks 5]

Ans. :

- Transactions have a distinct identity from the time they are created to the time they are finished in simple circumstances.
- Transactions can give birth to others in many systems and transactions can also merge.

1. Births : The decision symbol, or nodes with two or more out links, can be interpreted in three distinct ways. A decision, a biosis, or mitosis is all possibilities.
2. Decision : In this case, the transaction will choose one of the two options, but not both. (See Fig. Q.6.1 (a) for more information.)
3. Biosis : In this case, the incoming transaction gives birth to a second transaction, both of which continue on their own routes while the parent preserves its identity. (See Fig. Q.6.1 (b)).
4. Mitosis : In this step, the parent transaction is deleted and two new ones are formed. (See Fig. Q.6.1 (c) for further information.)

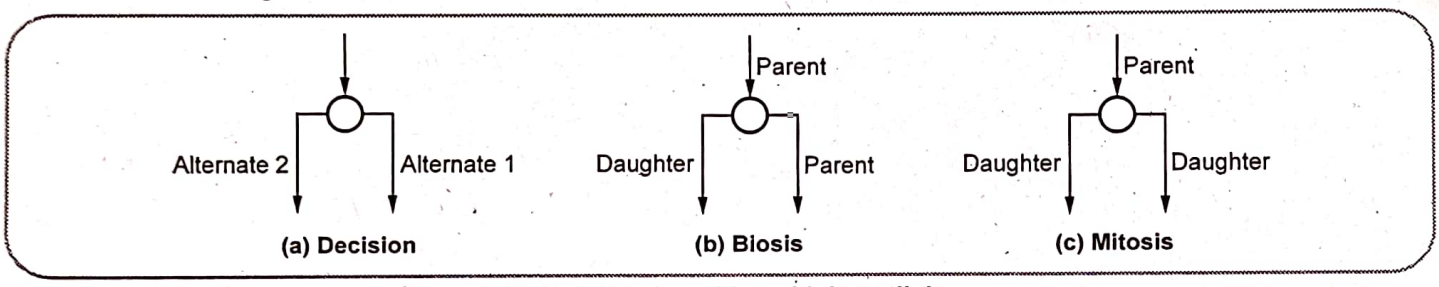


Fig. Q.6.1 Nodes with multiple outlinks

5. Mergers : Transaction flow junction points can be just as disruptive as splits in the flow. There are three different sorts of intersections : (1) Common intersection (2) Consumption (3) Affirmation

1 Ordinary junction : An ordinary junction, similar to a control flow graph's junction. A transaction can arrive via one of the two links. (See Fig. Q.6.1 (a) for more information.)

2 Absorption : The predator transaction absorbs the prey transaction in the absorption case. The prey has vanished, but the predator has not. (See Fig. Q.6.1 (b) for more information.)

3 Conjugation : In the conjugation case, the two parent transactions are combined to create a new daughter. This instance is referred to as conjugation, in keeping with the biological theme. (See Fig. Q.6.1(c) for further information.)

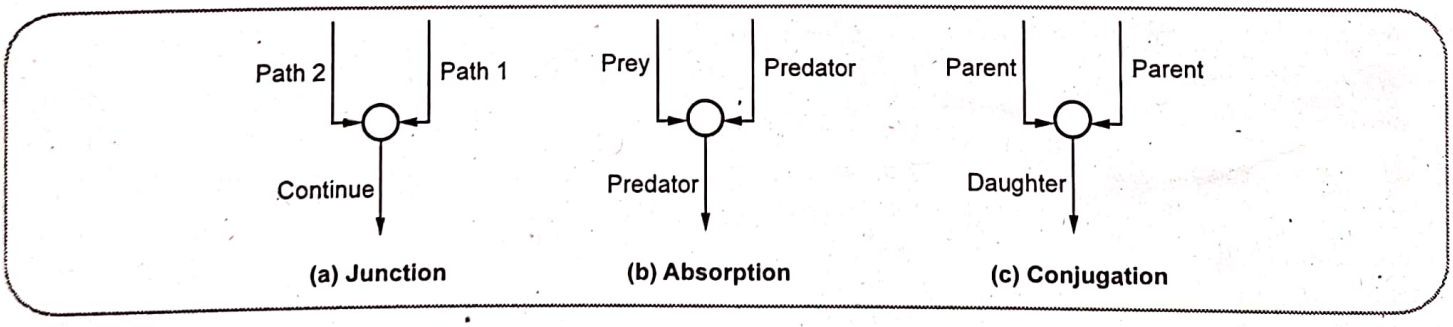


Fig. Q.6.2 Transaction flow junctions and mergers

Q.7 Explain transaction flow graph in detail with its implementation. ☞ [JNTU : Sept.-21, Marks 8]

Ans. :

- As a representation of a system's processing, transaction flows are introduced.
- The techniques used to create control flow graphs are then used to functional testing.
- To the independent system tester, transaction flows and transaction flow testing are what control flows and path testing are to the programmer.
- The transaction flow graph is used to develop a program's behavioral model, which leads to functional testing.
- The transaction flow graph is a representation of the system's behavior structure (functionality).

Transaction flows :

- A transaction is a unit of work from the perspective of a system user.
- A transaction is a set of processes, some of which are carried out by the system, others by people or devices outside the system.
- Transactions start with birth-that is, they are born as a result of an external event.
- The transaction is no longer in the system once it has been processed.

The following actions or tasks might make up a transaction for an online information retrieval system :

- Accept (tentative) input validate (tentative) input (birth)
- Send an acknowledgement to the requester.
- Carry out input processing
- Find a file
- Obtain directions from the user.
- Accept suggestions.
- Validate the data.
- Request is being processed.
- File should be updated.
- Output to the internet
- Make a note of the transaction in the log and clean it up (death)

Q.8 Explain transaction - flow graph implementation with example ☞ [JNTU : Part B, Dec.-18, Marks 5]

Ans. : As a representation of a system's processing, transaction flows are introduced.

- Functional testing is carried out using the same approaches that were used to create control flow graphs.
- To an independent system tester, transaction flows and transaction flow testing are what control flows and path testing are to a coder.
- The transaction flow graph is used to develop a program's behavioral model, which leads to functional testing.
- The transaction flow graph is a representation of the system's behavior structure (functionality).

The following is an example of a transaction flow :

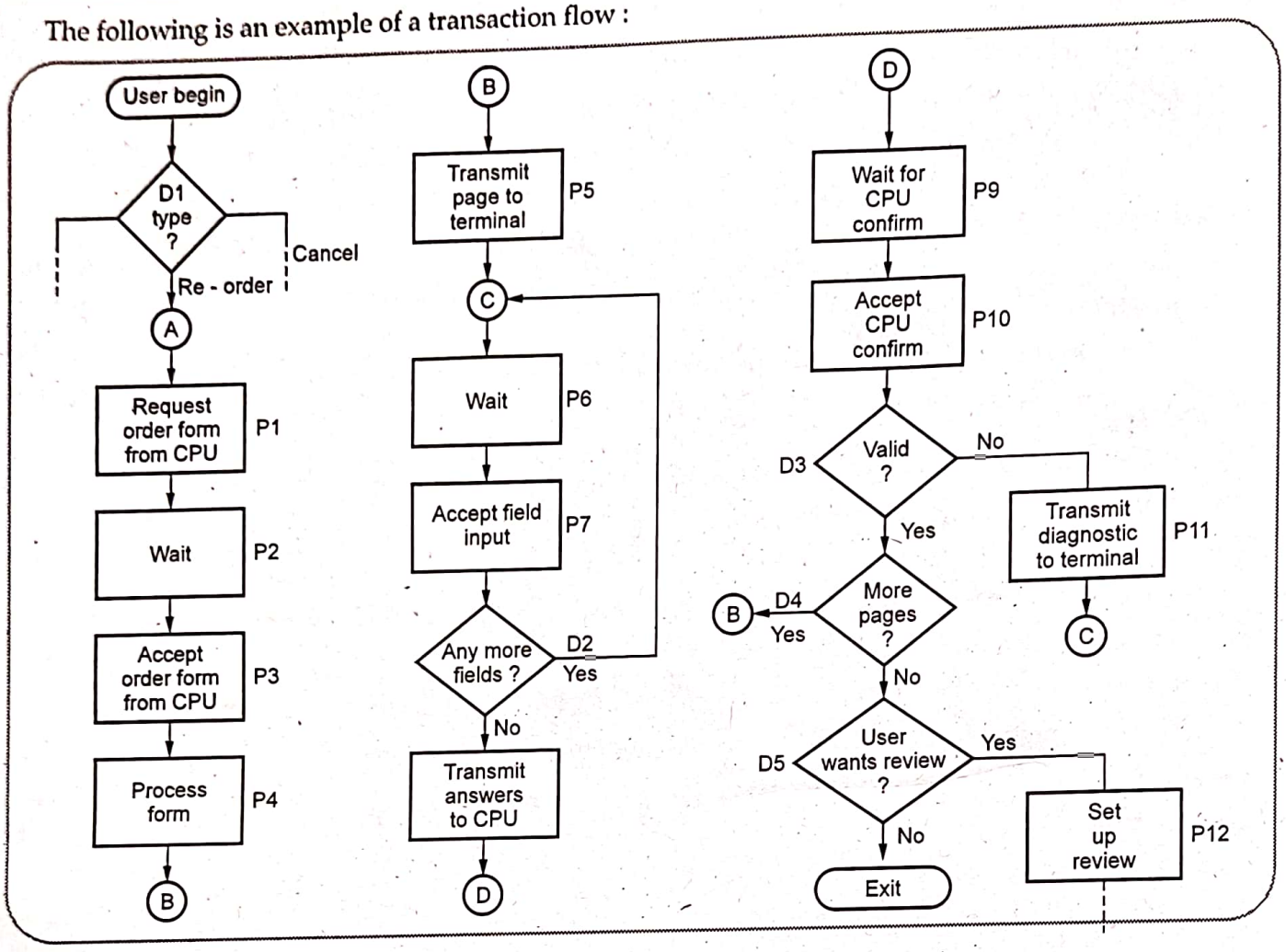


Fig. Q.8.1 An example of a transaction flow

- Transaction flows are critical for defining the requirements of complex systems, particularly online systems.
- A large system, such as an air traffic control or airline reservation system, contains thousands of various transaction flows rather than hundreds.
- In comparison to control flow graphs, the flows are represented by very basic flow graphs, many of which have a single straight - through path.
- After user input mistakes, the most typical loop is used to request a retry. For example, an ATM system will let the user to attempt three times before taking the card away the fourth time.

Q.9 What is data flow anomaly ?

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. : Anomaly :

Anomaly is a term used in software testing to describe a result that differs from what was expected. This behavior can be caused by a document or by the tester's own ideas and experiences.

An anomaly can also allude to a usability issue, because even if the test ware follows the specification, it can still be made more usable. The anomaly is sometimes referred to as a fault or bug.

Data flow anomalies :

While performing box testing or static testing, data flow anomalies are discovered. Anomalies in data flow are expressed by two characters dependent on the sequence of events. we have used defined (d), slain (k) and put to use (u). Based on these three sequences of activities, there are nine possible combinations : dd,

dk, du, kd, kk, ku, ud, uk, uu. The table below clearly demonstrates which of these combinations are acceptable and which are suspected of being anomalous.

Combination	Explanation	Possibilities of anomaly
dd	The data objects were defined twice.	Non - harmful but suspicious
dk	The data object was defined, but it was killed without being used.	Bad programming techniques
du	The data object has been defined and is being used.	NOT an anomaly
kd	Removed the data object and renamed it	NOT an anomaly
kk	I killed and re-killed the data object.	Bad programming techniques
ku	The data object was killed and then it was used.	Defect
ud	The data object was used and it was redefined.	NOT an anomaly
uk	The data object was used, and it was killed.	NOT an anomaly
uu	The data object was used, and it was reused	NOT an anomaly

Possibilities of anomalies :

1. -k :- This is possibly unusual because the variable had not been defined from the entrance to this point on the path. We're eradicating a variable that doesn't exist in the first place.
2. -d :- all right. This is only the first step in this journey.
3. -u :- It's possible that this is an anomaly. If the variable is global and has already been defined, this is not unusual.
4. k- :- This isn't unusual. The variable was killed as the last action on this path.
5. d- :- It's possible that this is an anomaly. On this path, the variable was defined but not used. However, this might be a universal definition.
6. u- :- This isn't unusual. On this path, the variable was used but not killed. Although this sequence is not unusual, it does indicate a common issue. This could be an instance where a dynamically allocated object was not returned to the pool after use, where d and k stand for dynamic storage allocation and return, respectively.

Q.10 Explain about data flow anomaly graph with example. ☞ **[JNTU : Part A, Dec.-18, Marks 3];**

Ans. : According to the data flow anomaly model, an item can be in one of four states :

1. K : Undefined, has been killed before, does not exist
2. D : A term that has been defined but has yet to be applied to anything.
3. The letter U has been utilized in computation or as a predicate.

4. A-Anamolous

Anomaly is a term used to describe something that is out of the ordinary.

The state of the variable is expressed by capital letters (K, D, U, A), which should not be confused with the program action, which is denoted by lower case letters.

Data that isn't forgiving - Flow anomaly flow graph : An unforgiving model in which a variable can never return to a condition of grace once it has become abnormal.

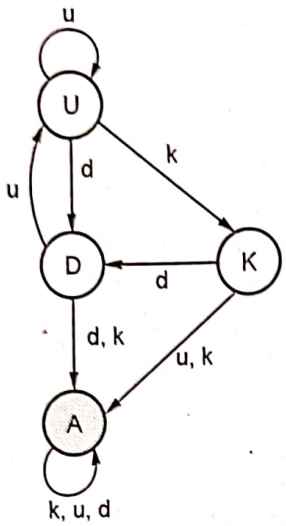


Fig. Q.10.1 Unforgiving data flow anomaly state graph

Flow anomaly - Forgiving data flow graph : The forgiving model is an alternative paradigm in which the anomalous condition can be redeemed (recovered).

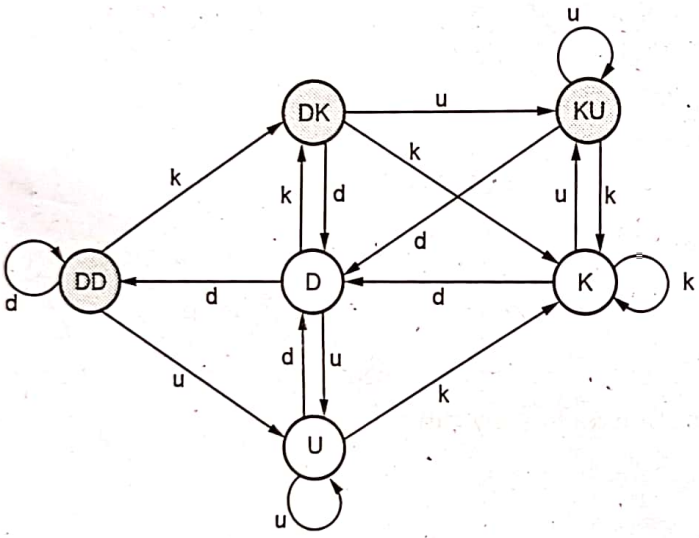


Fig. Q.10.2 Forgiving data flow anomaly state graph

The kk sequence is not anomalous in this graph, which has three normal and three anomalous states. This state graph differs from Fig. Q.10.1 in that redemption is a possibility. Any of the three anomalous states can be used to return the variable to a productive working state with the right action.

The purpose of displaying this alternative anomaly state graph is to explain how the characteristics of an anomaly are affected by factors such as language, application, context and even our mood. In theory, each circumstance necessitates the creation of a new definition of data flow abnormality (e.g., a new state graph). At the very least, make sure the anomalous term used in the theory or included in a data flow anomaly test tool is correct for our case.

Q.11 Explain static Vs dynamic anomaly detection.

Ans. : Static analysis is the process of analyzing source code without actually running it. The effect of static analysis, for example, is source code syntax error detection.

Dynamic analysis is performed on the fly while the program is running and is based on intermediate values generated by the program. The dynamic outcome, for example, is a division by zero warning.

If a problem, such as a data flow abnormality, can be found using static analysis methods, it belongs in the language processor, not testing.

In today's language processors, there is a lot more static analysis for data flow analysis for data flow irregularities.

Language processors that compel variable declarations, for example, can discover (-u) and (ku) anomalies. However, there are numerous things for which existing static analysis concepts are insufficient.

Q.12 Why static analysis isn't enough ?

Ans. : There are numerous topics for which existing static analysis concepts are insufficient. They are as follows :

Dead variables : Although it is frequently easy to establish whether a variable is dead or alive at a specific point in the program, the broader problem remains unresolved.

Arrays : Arrays are difficult because the array is defined or killed as a single object, yet references to individual points inside the array are made. Because array pointers are frequently computed dynamically, there is no method to validate the value of the pointer using a static analysis. Because dynamically allocated arrays in many languages contain garbage unless explicitly initialized, -u anomalies are possible.

Records and pointers : The array problem and the challenge with pointers are two different types of multipart data structures. The similar issue exists with

records and references to them. Furthermore, we generate files and their names dynamically in many apps and there's no way to know whether such objects are in the appropriate state on a particular path, or even if they exist at all, without executing them.

Dynamic subroutine and function names in a call : A dynamic variable in a call is the name of a subroutine or function. What is supplied, whether it is a list of subroutine names or a collection of data objects, is built along a certain path. Without running the path, there's no way to tell if the call is correct or not.

False anomalies : Anomalies are path - specific. Even a "obvious bug" like ku may not be a problem if the path along which the anomaly exists is impossible to follow. Such "anomalies" are fictitious. Unfortunately, establishing whether or not a path is feasible is an unsolved task.

Recoverable anomalies and alternate state graphs : An anomaly is defined by its context, application and semantics. How does the compiler know which model we are refering to ? It can't since the term "anomaly" is not precisely defined. We may or may not agree with the built - in anomalous definition of the language processor.

Concurrency, Interrupts, System Issues : Most anomaly issues get significantly more sophisticated as we move away from the simple single task un-processor context and start thinking in terms of systems.

Q.13 Write the applications of data flow testing.

☞ [JNTU : Part A, April-18, Marks 3];

OR What is meant by testing ? Write about any two applications of data flow testing.

☞ [JNTU : Part A, May-19, Marks 3];

Ans. : According to studies, faults discovered with 90 percent "data coverage" are twice as often as bugs discovered through 90 percent branch coverage.

Even when process flow testing is not assisted by automation, it is found to be beneficial.

It necessitates additional record - keeping in order to track the status of the variables. Computers make it easier to keep track of these variables, which cut down on testing time. Compilers can also incorporate data flow testing tools.

Q.14 Explain data flow machines

Ans. : Data flow machines are divided into two categories, each with its own architecture.

1. Von Neumann machines
2. Multi-Instruction, Multi-Data Machines (MIMD)

Von Neumann machine architecture

Most modern computers are Von-Neumann machines, which have interchangeable instructions and data storage in the same memory units.

The Von Neumann machine architecture executes one micro instruction at a time in the following sequence :

- Retrieve a command from memory
- Fetch operands
- Interpret instruction
- Process or run the program
- Save the results
- Increase the program counter
- GOTO 1

MIMD Architecture (Multi-Instruction, Multi-Data machines)

- These machines can fetch several instructions and objects at the same time.
- They can also perform arithmetic and logical operations on different data objects at the same time.
- The compiler is in charge of deciding how to order them.

Q.15 Explain data flow graphs and data object state and its usage.

Ans. : Data flow graphs

- The data flow graph is a graph consisting of nodes and directed links.
- We will use a control graph to show what happens to data objects of interest at that moment.
- Our objective is to expose deviations between the data flows we have and the data flows we want.

Data object state and its usage

It is possible to create, kill and use data objects. They can be applied in two different ways :

- When making a calculation
- As a condition of a control flow

These options are represented by the icons below :

- d- Formed, defined, initialized, etc.
- k- Undefined, killed and released
- u- Is a word that has a specific meaning ?
- c- Is a symbol that is used in calculations ?
- In a predicate, p - is used.

Defined :

- When an object occurs in a data declaration, it is clearly defined; when it appears on the left hand side of the assignment, it is implicitly defined.
- It can also be used to denote the opening of a file.
- It has been decided to allocate a dynamically allocated object.
- Something has been pushed to the top of the stack.
- A record has been made.

Killed or Undefined :

1. When an object is relinquished or otherwise made inaccessible, it is killed on undefined.
2. When the contents of the container are no longer certain.
3. Objects that have been dynamically allocated are returned to the availability pool.
4. Record retrieval
5. The old stack's top after it has been popped.
6. An assignment statement can instantly kill and redefine.

Usage :

1. When a variable occurs on the right hand side of an assignment statement, it is used for computation (c).
2. A record in a file is read or written.
3. When it appears directly in a predicate, it is employed in a predicate (p).

Q.16 Compare data flow and path flow testing strategies. ☞ **[JNTU : Part B, April-18, May-19, Marks 5];**

Ans. : Path-flow and data-flow testing methodologies are compared in this diagram. The arrows indicate that the strategy at the tail of the arrow is more powerful than the strategy at the head.

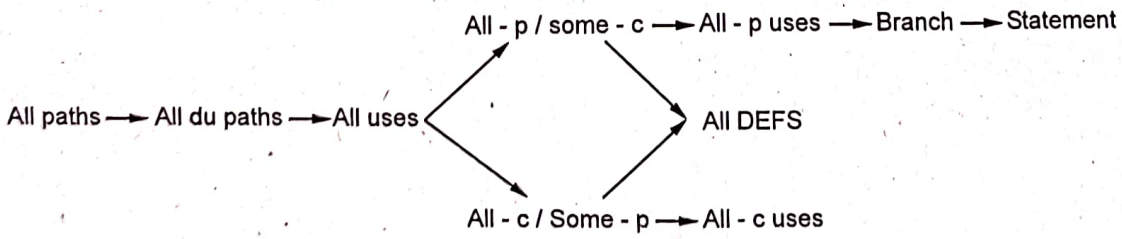


Fig. Q.16.1 Relative strength of structural test strategies

The more interesting hierarchy for practical applications is on the right-hand side of this graph, along the path from "all paths" to "all statements."

Although ACU+P is more powerful than ACU, both are inferior than predicate - biased techniques. It's also worth noting that "all definitions" isn't the same as ACU or APU.

Q.17 Discuss in detail data flow testing strategies.

☞ [JNTU : Part B, May-19, Marks 5];

Ans. : Data flow testing strategies :

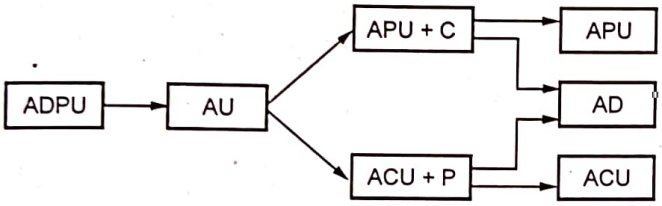


Fig. Q.17.1 Data flow testing

The test selection criteria are as follows :

1. All-definitions : Choose a thorough path that includes the def-clear path from node I to node I for each variable x and node I in such a way that x has a global declaration in node I.
   - Having a p-use of x or y on an edge (j,k)
   - Node j with a global c-use of x or Edge (j,k) with a p-use of x
2. All c-uses : Choose a thorough path that includes the def-clear path from node I to all nodes j that have a global c-use of x in j for every variable x and node I in such a way that x has a global declaration in node i.
3. All p-uses : Pick a thorough path that includes the def-clear path from node I to all edges (j,k) with p-use of x on edge for every variable x and node I in such a way that x has a global declaration in node i. (j,k).
4. All p-uses/Some c-uses : this criterion is similar to the all p-uses criterion, except it lowers to some c-uses criterion when variable x has no global p-use, as shown below.
5. Some c-uses : Choose a thorough path that includes the def-clear path from node I to some nodes j that have a global c-use of x in node j for each variable x and node I in such a way that x has a global declaration in node i.
6. All c-uses/Some p-uses : This criterion is similar to the all c-uses criterion, only it reduces to some p-uses when variable x has no global c-use, as seen below :
7. A comprehensive path including def-clear paths from node I to some edges (j,k) with a p-use of x on edge : For each variable x and node I in such a way that x has a global declaration in node I choose a comprehensive path including def-clear paths from node I to some edges (j,k) with a p-use of x on edge (j,k).
8. All uses : This is a mixture of the all p-uses and all c-uses criteria.
9. All du-paths : Choose a comprehensive path that includes all du-paths from node I for each variable x and node I in such a way that x has a global declaration in node i.
   - To all nodes j with a global c-use of x in j and a global c-use x in j and
   - Having a p-use of x on all edges (j, k), on (j,k)

Q.18 Discuss about the data flow model

☞ [JNTU : Part B, Dec.-18, Marks 5];

Ans. : The data flow model is based on the control flow graph of the program, which should not be confused with the data flow graph. Each connection is annotated with symbols (such as d, k, u, c and p) or sequences of symbols (such as dd, du, ddd) that signify the sequence of data operations performed on that link with regard to the variable of interest. Link weights are a type of annotation. The structure of the control flow graph is the same for all variables; the weights vary.

Components of the model :

1. There is a node with a unique name for each statement. Except for exit and entry nodes, every node has at least one outlink and at least one inlink.

2. To complete the graph, dummy exit nodes are added at the outgoing arrowheads of exit statements (e.g., END, RETURN). Entrance nodes, on the other hand are dummy nodes added at entry statements (for example, BEGIN) for the same reason.
3. The right sequence of data-flow operations for simple statements (statements with only one outlink) is weighted in the outlink of such statements. It's worth noting that the sequence can contain multiple letters. In most languages, for example, the assignment expression A := A + B is weighted by cd or perhaps ckd for variable A. Anomalies can occur in languages that allow numerous simultaneous assignments and/or compound statements. The sequence must match the order in which the object code for that variable will be executed.
4. The p - use(s) on every outlink, relevant to that outlink, are weighted with the predicate nodes (e.g., IF-THEN-ELSE, DO WHILE, CASE).
5. Every set of simple statements (for example, a set of nodes with one inlink and one outlink) can be substituted by a pair of nodes with the concatenation of link weights as link weights.
6. The weight of a link is defined by the sequence of activities on that link for that variable if there are multiple data - flow actions on that link for that variable.
7. A link having numerous data - flow actions on it, on the other hand, can be substituted by a series of equivalent connections, each with only one data-flow action for any variable.

Q.19 Explain data-flow testing with an example. Explain its generalizations and limitations.

☞ **[JNTU : Part B, April-18, Marks 5];**

Ans. : Data flow testing refers to a group of test methodologies that involve choosing paths across a program's control flow in order to investigate sequences of events relating to the status of data objects.

Example 1 : Choose enough pathways, to ensure that every data item has been initialized before usage or that all defined objects have been used.

OR

Example 2 :

This code contains eight statements. We can't cover all 8 statements in a single route with this code because if 2 is valid, then 4, 5, 6, 7 aren't traversed and if 4 is valid, then 2 and 3 aren't traversed.

As a result, we shall evaluate two paths in order to cover all of the claims.

x = 1
Path – 1, 2, 3, 8
Output = 2

When x is set to 1, it proceeds to step 1 to assign x to 1 and then to step 2, which is incorrect because x is less than 0 (x > 0 and x = – 1). It will then proceed to step 3 and then to step 4; because step 4 is true (x = 0 and their x is smaller than 0), it will proceed to step 5 (x1), which is true and then to step 6 (x = x+1), where x is increased by 1.

So,

$$x = -1+1$$
$$x = 0$$

x become 0 and it goes to step 5(x < 1),as it is true it will jump to step

6 (x = x + 1)

$$x = x+1$$
$$x = 0+1$$
$$x = 1$$

Because x is now 1, it will proceed to step 5 (x1), where the condition will be false and it will proceed to step 7 (a = x + 1), where it will set a = 2 because x is now 1. The value of an at the end is 2. And at step 8, we obtain 2 as an output.

Generalization

1. The programmer has the ability to run a variety of tests on data values and variables. Data flow testing is the name for this form of testing.

There are two types of data flow testing ; Static data flow testing and dynamic data flow testing.

2. Static data flow testing is looking at the source code without running it.
3. Data flow anomalies are discovered during static data flow testing.
4. From source code, dynamic data flow determines program pathways.

Limitations :

1. Programming knowledge is required of testers.
2. It takes a long time
3. Expensive procedure.

Q.20 Explain the terms dicing, data-flow and debugging. ☞ **[JNTU : Part B, April-18, Marks 5];**

Ans. : Data flow :

Data flow testing refers to a group of test methodologies that involve choosing paths across a program's control flow in order to investigate sequences of events relating to the status of data objects.

Dicing :

A program dice is a section of a slice that has been stripped of all assertions that are known to be correct. To put it another way, a dice is made from a slice by adding data gathered through testing or experimentation (e.g., debugging).

The debugger initially restricts her scope to past statements that could have resulted in the erroneous value at statement I (the slice), then discards those that have been proven right by testing.

Debugging :

Debugging can be characterized as an iterative process in which slices are refined further by dicing, with the dicing information derived from ad hoc testing aiming primarily at removing possibilities. When the dice have been reduced to only one problematic statement, debugging is complete.

Q.21 Explain different data object states in data flow graphs. ☞ **[JNTU : Part B, Dec.- 19, Marks 5]**

Ans. : State and use of data objects : Data objects can be produced, terminated and used. They can be utilized in two ways : (1) As part of a calculation (2) As a control flow predicate these options are represented by the icons below :

Defined : d - defined, created, initialized etc.

Killed or undefined k : k - killed, undefined, released etc.

Usage U : used for something (c - used in calculations)

(p - used in a predicate)

1. (d) Defined : When an object appears in a data declaration, it is clearly specified. Alternatively, when it occurs on the assignment's left hand side, it is implied. It can also be used to denote the opening of a file. It has been decided to allocate a dynamically allocated object. Something has been pushed to the top of the stack. A record has been made. Undefined (k) : When an object is released or otherwise made inaccessible, it is killed on undefined.

When the contents of the container are no longer known with total certainty (perfection). Objects that have been dynamically allocated are returned to the availability pool. Records are being returned. After it has been popped, the former top of the stack. An assignment statement can instantly kill and redefine. If we execute a new assignment such as A : = 17, for example, we have killed A's prior value and redefined it.

Usage (u) : When a variable occurs on the right hand side of an assignment statement, it is utilized for computation (c). The contents of a file record are read or written. When it appears directly in a predicate, it is utilized in a Predicate (p).

2.2 Domain Testing : Domains and Paths, Nice & Ugly Domains, Domain Testing, Domains and Interfacing Testing, Domains and Testability

Q.22 Explain predicates of domain testing with examples. ☞ **[JNTU : Part B, Dec.-19, Marks 5, Sept.-21, Marks 7]**

OR Define domain testing with example. ☞ **[JNTU : Part A, April-18, May-19, Marks 2]**

Ans. :

- Predicates are considered to be understood in terms of input vector variables in domain testing.
- If structure is subjected to domain testing, predicate interpretation must be based on actual pathways through the routine, i.e. the implementation control flow graph.
- When domain testing is used on specifications, the interpretation is based on a data flow graph for the routine; however, because of the nature of specifications, no interpretation is usually required because the domains are stated directly.
- There is at least one path through the procedure for each domain.
- If the domain consists of disconnected sections or is defined by the union of two or more domains, there may be more than one path.
- The borders of domains are defined. The majority of domain problems occur at domain boundaries.
- There is at least one predicate for each boundary that indicates which integers belong to the domain and which do not.
- We know that integers bigger than zero belong to the ALPHA processing domain(s), while zero and lower numbers belong to the BETA domain in the line IF x>0 THEN ALPHA ELSE BETA (s).
- No matter how many variables define a domain, it may have one or more borders. The domain is the inside of a circle of radius 4 around the origin if the predicate is $x^2 + y^2 < 16$. Similarly, a spherical domain with one boundary but three variables might be defined.
- Domains are typically characterized by a large number of boundary segments and as a result, a large number of predicates. The domain's limits are defined by the set of interpreted predicates traveled on that path (i.e., the path's predicate expression).

Q.23 Compare domain testing and interface testing.

☞ **[JNTU : Part B, Dec.-19, Marks 5]**

Ans. : Domain testing :

- Because domains are defined by their borders, domain testing focuses on or near those boundaries.
- Define a test strategy for each situation after classifying what can go wrong with bounds. Select a sufficient number of points to test for all known types of boundary errors.
- Because each boundary serves at least two domains, test points used to inspect one domain can also be utilized to inspect adjacent domains. Remove any test points that are redundant.
- Run the tests and discover if any boundaries are wrong and, if so, how, using posttest analysis (the tedious part).
- Perform enough tests to ensure that every domain's border is verified.

Interface testing :

Bugs in interface testing are more likely to be caused by single variables than by unusual combinations of two or more variables. Confirm the range of the caller, the domain span of the called routine and the closure of each domain declared for a variable by testing each input variable independently of the other input variables. Because there are two borders to test and the domain is one dimensional, each boundary requires one on and one off point, for a total of two on points and two off points for the domain. Select the proper off points for the closure (COOOOI). Begin with the domains of the called routines and produce test points in accordance with the domain - testing method used in component testing for that routine.

We need to be very competent in mathematics to perform this without using tools for more than one variable at a time.

Q.24 Define domain testing. Explain about nice domains in detail. ☞ **[JNTU : Part B, Dec.-18, Marks 10]**

Ans. : Domain testing : Refer Q.23

Nice domains :

What are the origins of these domains ?

An imperfect iterative process aiming at achieving (user, customer, voter) happiness has established and will continue to develop domains.

- Domains that have been implemented cannot be incomplete or inconsistent. Every input will be processed (rejection is a process) and this could take a long time. Domains that are inconsistent will be made consistent.
- Specified domains, on the other hand, may be incomplete and/or inconsistent. In this context, incomplete indicates that there are input vectors for which no path has been given and inconsistent means that at least two contradicting specifications exist over the same segment of the input space.
- Linear, Complete, Systematic and Orthogonal, Consistently closed, Convex and simply connected are some significant qualities of good domains.
- Domain testing is as simple as testing gets to the degree that domains have these qualities.
- The frequency of bugs is lower in nice domains than in ugly domains.

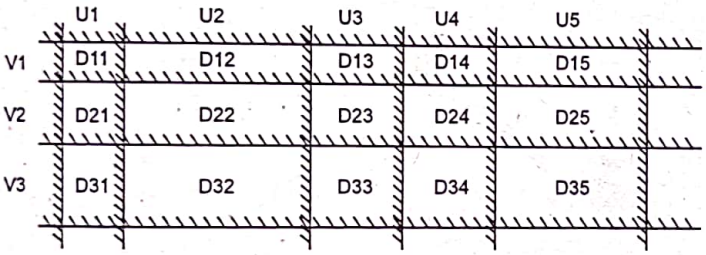


Fig. Q.24.1 Nice two-dimensional domains

Q.25 State and explain various restrictions at domain testing processes.

☞ [JNTU : Part B, May-18, 19, Marks 10]

OR Write about restrictions of domain testing.

☞ [JNTU : Part B, Dec.-18, Marks 5]

Ans. : Other testing methodologies, including domain testing, have limitations. Among them are the following :

- Co-incidental correctness : Domain testing isn't very good at detecting defects where the result is accurate for the wrong reasons. We may mistake an inappropriate border if we are bothered by coincidental correctness. When dealing with routines that have binary results (i.e., TRUE/FALSE), this signals a weakness for domain testing.
- Representative outcome : One type of partition testing is domain testing. Partition - testing methodologies partition the program's input space into domains, with each domain containing all inputs that are comparable (not equal, but equivalent) in the sense that every input reflects all inputs in that domain.
- If a test shows that the selected input is accurate, processing is assumed to be correct and other inputs within that domain are anticipated (possibly unjustifiably) to be correct. Partition testing encompasses most test approaches, whether functional or structural and so makes this representative outcome assumption. For x = 2, for example, x2 and 2x are equivalent, but the functions are not. Rather than x2 against 2x, functional distinctions between adjacent domains are usually simple, such as x + 7 versus x + 9.

Q.26 What is loop free software ?

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. : Free looping software : Loops are difficult to test in domains. The problem with loops is that each iteration can produce a different predicate expression (after interpretation), which could result in a change in the domain boundary.

Q.27 Write about linear vector space.

☞ [JNTU : Part A, Dec.-19, Marks 2]

Ans. : Most domain testing studies assume linear borders, which isn't a bad assumption because most boundary predicates are linear in practice.

Q.28 What is domain span ?

☞ [JNTU : Part A, Dec.-18, Marks 2]

Ans. : The domain span of a single variable is the range of numbers between (and including) the smallest and biggest value. We want (at the very least) compatible domain spans and compatible closures for each input variable (Compatible but need not be equal).

In contrast to the domain, which is the set of input values over which the function is defined, the range of

a function is the set of output values produced by the function.

Q.29 Discuss about domain dimensionality.

☞ [JNTU : Part A, Dec.-18, Marks 3]

Ans. :

- Every input variable expands the domain by one dimension.
- On a number line, one variable specifies domains.
- Planar domains are defined by two variables.
- Solid domains are defined by three variables.
- Each new predicate slashes through previously established domains, halving them.
- Thus, planes are cut by lines and points, volumes by planes, lines and points and n-spaces by hyper planes; every boundary slices through the input vector space with a dimensionality less than the dimensionality of the space.

Q.30 Write about nice and ugly domains and give examples to each domain

☞ [JNTU : Part B, Dec.-19, Marks 5]

OR Define nice and ugly domains.

☞ [JNTU : Part A, May-19, Marks 2]

OR In what a nice domain differs from ugly domains.

☞ [JNTU : Part A, April-18, Marks 2]

Ans. : Nice domain : Refer Q. 24.

Ugly domain :

Some domains are ugly from the start, while others are made worse by poor specs.

- Every programmers' simplification of unsightly domains can be good or detrimental.
- In their pursuit for elegant solutions, programmers will "simplify" crucial complexity. Testers in quest of great insights will miss key situations because they will be blind to necessary complexity.
- If the ugliness is due to faulty specifications and the programmer's simplification is harmless, the programmer has turned ugly into beautiful.
- However, if the domain's complexity is critical (for example, the tax code), such "simplifications" are defects.
- Nonlinear boundaries are so uncommon in everyday programming that no knowledge exists on how programmers might "fix" them if they're necessary.

Q.31 With a neat diagram, explain the schematic representation of domain testing.

☞ [JNTU : Part B, April-18, Marks 5]

Ans. :

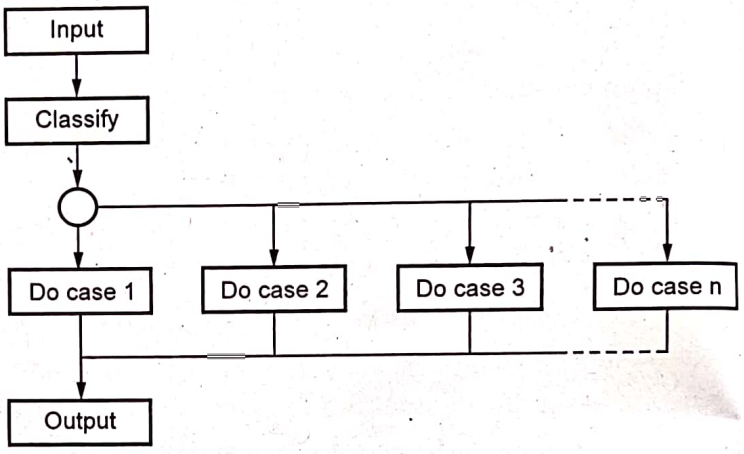


Fig. Q.31.1 Schematic representation of domain testing

- A procedure must identify the input and get it flowing in the appropriate direction before it can do anything.
- An invalid input (for example, a value that is too large) is simply a 'reject' processing case.
- Instead of performing calculations, the input is passed to a fictitious subroutine. We focus on the categorization component of the routine rather than the calculations in domain testing.
- This approach does not require structural knowledge; only a consistent, comprehensive statement of input values for each scenario is required.
- We may deduce that there must be at least one path for each case to be processed.

Q.32 Define domains and paths.

Ans. :

- A domain is a set of possible values for an independent variable or the variables of a function in mathematics.

- Domain testing seeks to establish whether the categorization is valid or not using programs as input data classifiers.
- Domain testing can be done using specifications or similar implementation data.

Q.33 Define domains and testability.

Ans. : A domain is a set of possible values for an independent variable or the variables of a function in mathematics.

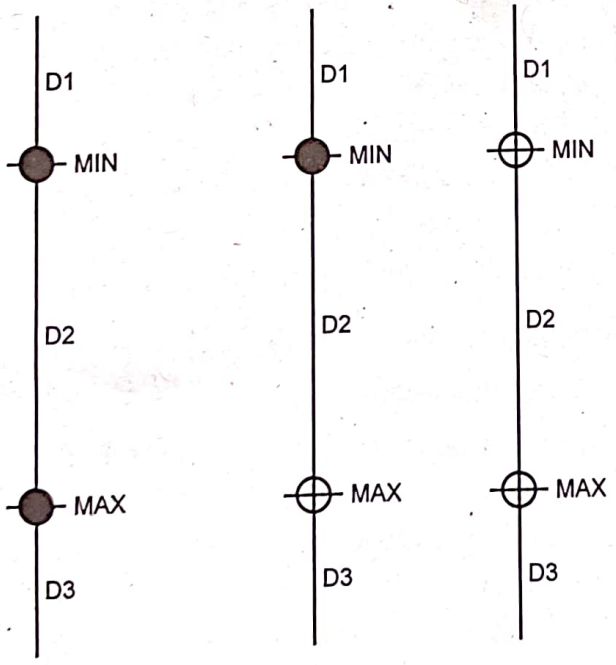
Domain testing is a software testing technique in which selecting a small number of test cases from a nearly infinite group of test cases.

Q.34 Define domain closure.

Ans. : If the points on a domain border belong to the domain, the boundary is closed with regard to the domain.

- The boundary is considered to be open if the boundary points pertain to another domain.
- The figure depicts three scenarios for a one-dimensional domain - that is, a domain defined by a single input variable; let's call it x.

The importance of domain closure stems from the fact that incorrect domain closure issues are common. When $x > 0$ was intended, for example, $x >= 0$ was used instead.



(a) Both sides closed (b) One side open

Fig. Q.34.1 Open and closed Domains

Q.35 Explain bug assumption in detail.

Ans. : The domain testing bug assumption is that processing is fine, but the domain definition is incorrect.

- Incorrectly implemented domains result in improper bounds, which may lead to incorrect control flow predicates.
- Domain errors can be caused by a variety of issues. Here are a few examples :

1. **Domain errors :** Double zero representation : Boundary mistakes for negative zero are widespread in computers or languages that have a distinct positive and negative zero.
2. **Floating point zero check :** A floating point number can only equal zero if it was previously defined as zero, or if it is subtracted from itself or multiplied by zero. As a result, the floating point zero check must be performed on an epsilon value.
3. **Contradictory domains :** A domain that has been implemented can never be ambiguous or contradictory, whereas a domain that has been specified can. When a domain specification is contradictory, it signifies that at least two ostensibly separate domains overlap.
4. **Ambiguous domains :** When a domain's union is incomplete, it's called an ambiguous domain. There are holes or missing domains in the given domains. A common ambiguity is not stating what happens to points on the domain border.
5. **Over specified domains :** His domain can get overloaded with so many requirements that it becomes null. Another way to describe it is that the domain's path is impossible to follow.
6. **Boundary errors :** Errors that occur at or near a domain's boundary. Boundary closure bug, for example, moved, skewed, missing or additional boundary.
7. **Closure reversal :** This is a rather typical bug. $>=$ is used to define the predicate. The programmer implements the logical complement and wrongly uses $=$ for the new predicate; for example, $x >= 0$ is incorrectly negated as $x = 0$, causing boundary

values to shift to adjacent domains.

8. **Faulty logic :** Compound predicates, in particular, are vulnerable to erroneous logic transformations and poor simplification. If the predicates specify domain boundaries, improper logic manipulations can lead to a variety of domain defects.

Q.36 What are the restrictions of domain testing ?

Ans. : Other testing methodologies, including domain testing, have limitations. Among them are the following :

- **Co-incidental correctness :** Domain testing isn't excellent at detecting defects where the conclusion is right but for the incorrect reasons. We may mistake an inappropriate border if we are bothered by coincidental correctness. When dealing with routines that have binary results (i.e., True/False), this signals a weakness for domain testing.
- **Representative outcome :** One type of partition testing is domain testing. partition - testing methodologies partition the program's input space into domains, with each domain containing all inputs that are comparable (not equal, but equivalent) in the sense that every input reflects all inputs in that domain.
- If a test shows that the selected input is accurate, processing is assumed to be correct and other inputs within that domain are anticipated (possibly unjustifiably) to be correct. Partition testing encompasses most test approaches, whether functional or structural and so makes this representative outcome assumption. For x = 2, for example, x^2 and 2^x are equivalent but the functions are not. Rather than x^2 against 2^x, functional distinctions between adjacent domains are usually simple, such as x + 7 versus x + 9.

Q.37 What is simple domain boundaries and compound predicates :

Ans. : Compound predicates in which each portion specifies a different boundary are not a problem : For example, x > = 0 AND x < 17 specifies two domain boundaries with a single compound predicate. Consider the following compound predicate that requires only one boundary: x = 0 AND y >= 7 AND y = 14. This predicate defines a single boundary equation (x = 0), but it alternates closure, placing it in one of two domains depending on whether y < 7 or y > 14. Treat compound predicates with respect since they're more complicated than they look.

1. **Functional homogeneity of bugs :** Whatever the issue is, it will not affect the border predicate's functional form. If the predicate is ax >= b, for example, the bug will be in the value of **a or b**, but it will not modify the predicate to **ax >= b**, for example.
2. **Linear vector space :** Most domain testing articles assume linear boundaries, which is a reasonable assumption given that most boundary predicates are linear in practice.
3. **Loop free software :** Loops are difficult to test in domains. The problem with loops is that each iteration can produce a different predicate expression (after interpretation), which could result in a change in the domain boundary.

Q.38 Define following terms :
i. Linear and Non linear domain boundaries
ii. Complete domain boundaries
iii. Orthogonal Boundaries iv. Closure Consistency
v. Convex vi. Simply Connected

Ans. : i. Linear and Non linear domain boundaries :

- Linear inequalities or equations define nice domain boundaries.
- The impact on testing arises from the fact that a straight line requires only two points, a plane requires three points, and an n-dimensional hyper plane requires n+1 points in general.
- More than 99.99 percent of all border predicates are linear or can be linearized using simple variable transformations in practice.

ii) Complete domain boundaries :

In all dimensions, from plus to minus infinity.

- The Fig. Q.38.1 depicts some unfinished limits. There are gaps between boundaries A and E.
- Such boundaries can arise because the path that hypothetically corresponds to them is unachievable, because inputs are constrained in such a way that such values aren't possible, because compound predicates define a single

boundary or because redundant predicates convert such boundary values to a null set.

- The advantage of complete borders is that no matter how many domains they bound, only one set of tests is required to confirm the boundary.
- If the border is cut up and has holes in it, each segment of the boundary must be evaluated for each domain it encompasses.

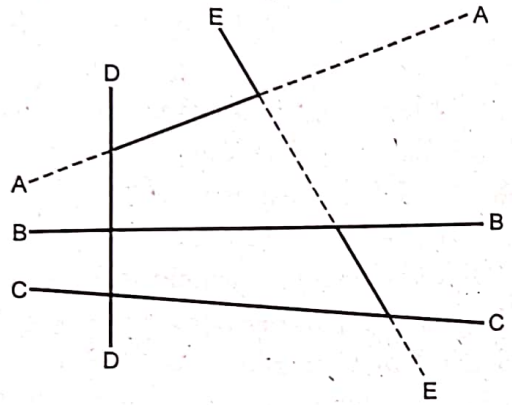


Fig. Q.38.1 Incomplete domain boundaries

iii) Orthogonal boundaries

- If every inequality in V is perpendicular to every inequality in U, two boundary sets U and V are said to be orthogonal.
- When two boundary sets are orthogonal, they can be separately tested.
- We have six borders in U and four in V in the good domain. A number of tests proportional to 6 + 4 = 10 (O(n) can confirm the boundary qualities. We must now test the intersections if we tilt the boundaries to get Fig. Q.38.2. We've gone from a linear to a quadratic number of cases: from O(n) to O(n) (n^2).

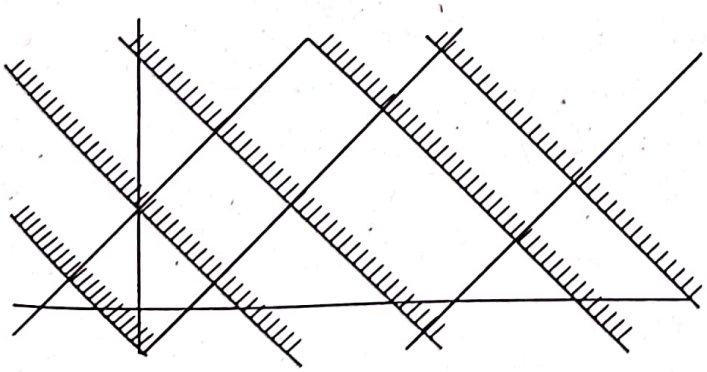


Fig. Q.38.2 Tilted boundaries

iv. Closure consistency :

- Another desirable domain attribute is that boundary closures are consistent and systematic, as seen in Fig. Q.38.3.
- The shaded areas on the border denote that the boundary belongs to the domain in which the shading is located - for example, the boundary lines belong to the right - hand domains.
- A consistent closure is one in which the closures follow a basic pattern, such as applying the same relational operator for all borders of a collection of parallel boundaries.

v. Convex :

- A geometric figure is convex if we can connect two arbitrary points on any two separate boundaries with a line and all points on that line lie within the shape.
- Clean domains are convex, whereas nice domains aren't.
- When we encounter sentences like ".. except if...," "However...," ".. but not...," ".. ". It's the buts in the specification that kill us in programming.

vi. Simply Connected :

- Nice domains are simply connected; that is, they are all in one piece rather than scattered around the internet with other domains.
- Convexity is a stronger condition than simple connectivity; if a domain is convex, it is simply connected, but not the other way around.

  Consider domain limits defined by a (Boolean) form ABC compound predicate. Assume that the input space is separated into two domains, one of which is defined by ABC and the other by its negation.

  Let's say we define legitimate numbers as those that fall between 10 and 17, inclusive. The unconnected domain of invalid numbers is made up of numbers less than 10 and bigger than 17.
- Simple connectivity may be impossible, especially in default cases.

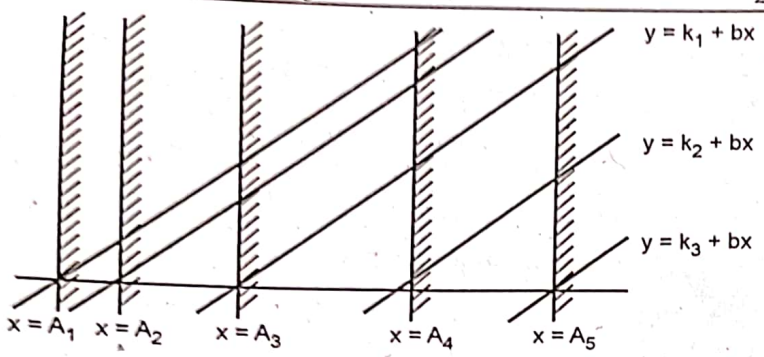


Fig. Q.38.3 Linear, Non-orthogonal domain boundaries

Q.39 Explain testing one dimensional domain.

☞ [JNTU : March-22, Marks 5]

Ans. : The closure may be incorrect (i.e., assigned to the incorrect domain), the boundary (in this case, a point) may be pushed one way or the other, a boundary may be absent or an extra boundary may exist.

1. For a one - dimensional open domain boundary, Fig. Q.39.1 depicts probable domain bugs.
2. In Fig. Q.39.1 a, the border was considered to be open for A. A closure error, which turns > to >= or to =, is the bug we're looking for (Fig. Q.39.1(b)). Because processing for that point will proceed to domain A rather than B, one test (marked x) on the border point finds this problem.
3. We've had a boundary movement to the left in Fig. Q.39.1(c). Because the bug forces the point from the B domain, where it should be, to A processing, the test point we used for closure detects this bug. We can't tell the difference between a shift n and a closure mistake, but we do know there's a problem.
4. Fig. Q.39.1(d) depicts a change in the other direction. Because the boundary shift has no effect on the fact that the test point will be processed in B, the on point is meaningless. We need a point near the boundary but within A to detect this movement. Because the border is open, the off point is in A by definition (Open off inside).
5. Because what should have been handled in A is now processed in B, the identical open off point is sufficient to detect a missing boundary.

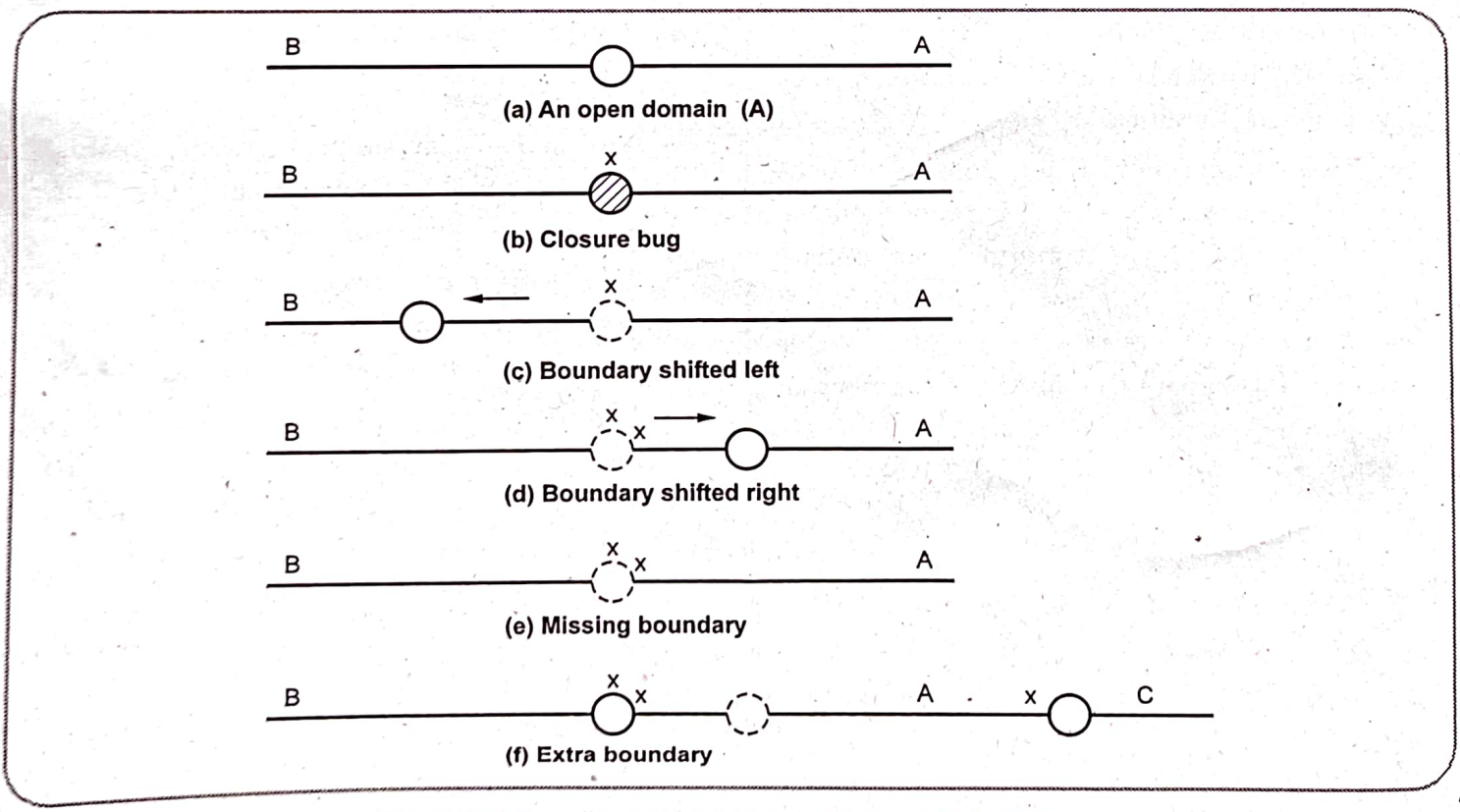


Fig. Q.39.1 One dimensional domain bugs, open boundaries

6. We must examine two domain borders in order to find an extra boundary. An extra border indicates that A has been split in two in this situation. The two off points we chose earlier (one for each boundary) are sufficient. The on test point at C would do it if point C had been a closed boundary.
7. Refer to Fig. Q.39.2 for closed domains. In the case of the open border, the closure bug is detected by a test point on the boundary. The remaining examples are similar to the open boundary case, with the exception that the technique now necessitates off points just outside the domain.

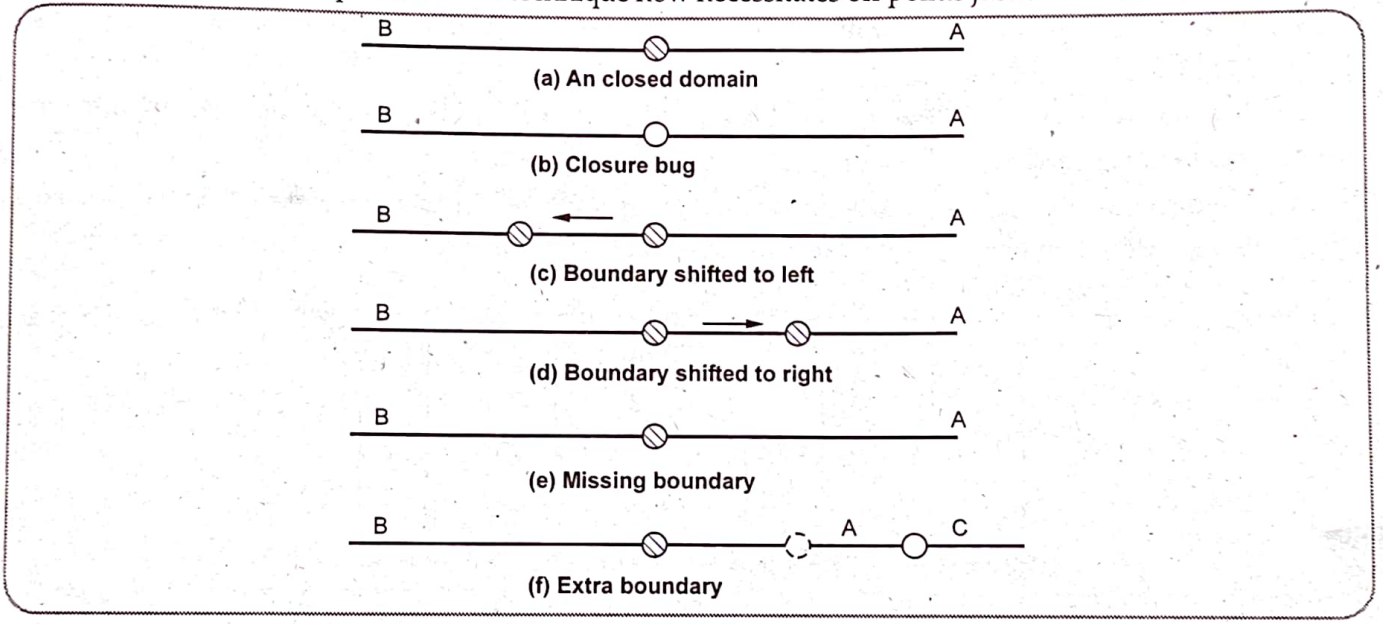


Fig. Q.39.2 One dimensional domain bugs, closed boundaries

Q.40 Explain procedure for testing.

Ans. : Conceptually, the technique is straightforward. For two dimensions and a few domains, it is possible to do by hand, but for more than two variables, it is almost impossible.

1 Determine the input variables.

2 Locate variables in domain - defining predicates, such as control flow predicates.

3 In terms of input variables, interpret all domain predicates.

4 There are at most 2^p choices of TRUE-FALSE values for p binary predicates and thus at most 2^p domains. Find the set of all domains that aren't null. In the predicates, the outcome is a Boolean expression consisting of a series of AND terms connected by OR's. ABC + DEF + GHI....., for example, where the capital letters signify predicates. Each product term is a set of linear inequalities that define a domain or a subset of a multiply connected domain.

5 Using any of the linear programming methods solve these inequalities to identify all of the domain's extreme points.

Q.41 What is program slicing ? Explain dynamic program slicing. ☞ **[JNTU : Part B, Dec.-19, Marks 5, Sept.-21, Marks 8]**

Ans. : Slicing, also known as program slicing, is a software testing approach that uses a slice or a series of program statements to test certain test circumstances or situations that may change a value at a specific point of interest.

Instead of all statements that may have impacted the value of a variable at a program point for any arbitrary execution of the program, a dynamic slice comprises all statements that actually affect the value of a variable at a program point for a particular execution of the program.

Q.42 Discuss with example the equal - span range / domain compatibility bugs. ☞ **[JNTU : Part B, May-19, Marks 5]**

Ans. : Range / Domain closure compatibility :

- Assume that the range of the caller and the range of the called domain are the same - for example, 0 to 17.
- The four ways in which the caller's range closure and the called's domain closure can agree are shown in Fig. Q.42.1.
- The thick line denotes closure, whereas the thin line denotes openness. Fig. Q.42.1 depicts domains that are closed on top (17) and bottom (0), open top and closed bottom, closed top and open bottom and open top and bottom.

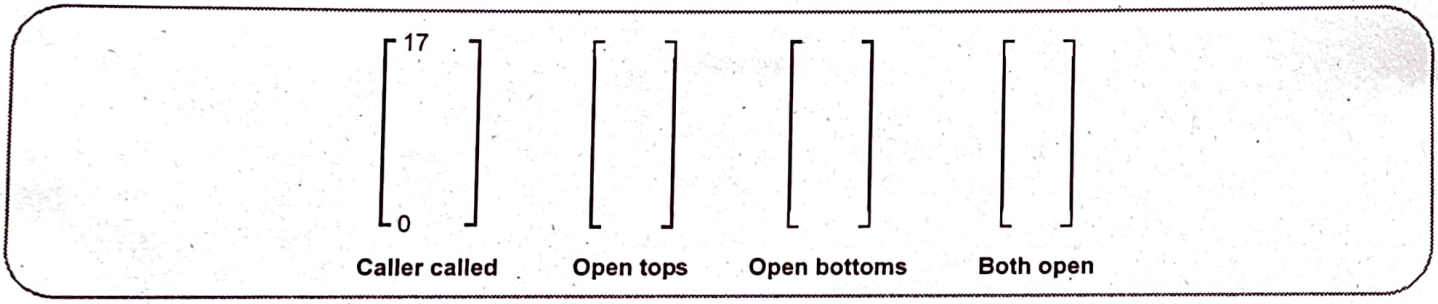


Fig. Q.42.1 Range / Domain closure compatibility

- Equal-span range / Domain compatibility bugs : The caller and the called can argue regarding closure in twelve distinct ways, as shown in Fig. Q.42.2. They aren't all necessarily bugs. The four scenarios (marked with a "?") in which a caller boundary is open but the called is closed are most likely not buggy. It indicates that the caller will not provide such values, but that the called may accept them.

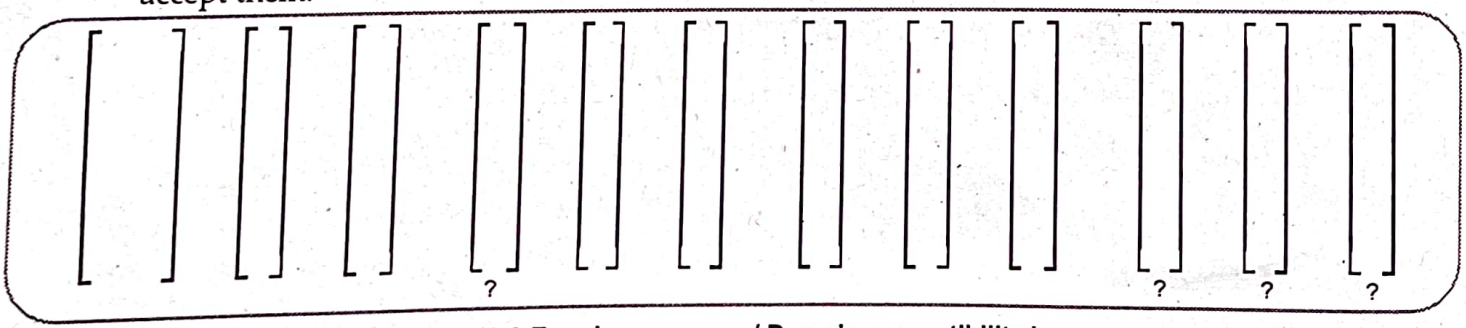


Fig. Q.42.2 Equal-span range / Domain compatibility bugs

Fill in the Blanks for Mid Term Exam

Q.1 _____ machines have the ability to store instructions in many ways.

Q.2 _____ is similar to maintenance testing.

Q.3 The _____ analysis of arrays is employed.

Q.4 _____ nodes include things like if then else and do while.

Q.5 _____ are the three anomaly states.

Q.6 _____ are two examples of transaction flow problems.

Q.7 For functional testing, the _____ graph is employed.

Q.8 The values of the border in a _____ pertain to the same domain.

Q.9 _____ refers to the number of planes in which the domain occurs.

Q.10 A _____ is a location in the epsilon neighbourhood where points are both in and out of domain.

Q.11 _____ are those that range from +infinity to –infinity.

Q.12 The domain is considered to be _____ when the union of specified domains is incomplete.

Q.13 The interface test investigates whether the caller unit test, _____ and called _____ are correct.

Q.14 In the case of good and ugly domains, F(x)>=k represents a _____.

Q.15 _____ is an approach for partitioning a program's input space into domains where the values in each domain are equivalent.

Q.16 Inserting elements into a stack is done with the _____ operation.

Q.17 The parallel set of pathways is denoted as _____.

Multiple Choice Questions for Mid Term Exam

Q.1 In the case of transactional flow instrumentation, the following statement is true :

[a] Dispatchers are needed

[b] Pay off is counters

[c] Counters are useful

[d] Counters are not useful

Q.2 Sensitization in transaction flows can be accomplished in a variety of ways _____.

[a] using of patches [b] use break points

[c] counters [d] processing queue

Q.3 Data objects' states can't be _____.

[a] created [b] killed

[c] used [d] modified

Q.4 There is no clear relationship between processes and decisions in _____.

[a] data flow diagram

[b] transaction flow diagram

[c] both a)-and b)

[d] none

Q.5 Each variable's definition is included in _____.

[a] ad strategy [b] apu strategy

[c] au strategy [d] adup strategy

Q.6 The following is the normal sequence with respect to the life time of a variable :

[a] kk [b] dd

[c] dk [d] uu

Q.7 a = 0; b = a; the sentence denotes the order of events.

[a] du [b] uu

[c] ud [d] dd

Q.8 The correct transaction flow sequence is _____.

[a] input, validate, acknowledge, record transaction

[b] input, acknowledge, record transaction

[c] acknowledge, input, validate record, transaction

[d] record transaction,input,validate,acknowledge

Q.9 When the proper equation is x+y>=5, instead of x+y>=10. is an illustration of _____.

[a] closure bug [b] titled bug

[c] missing boundary [d] shifted boundary

Q.10 When two distinct domains collide, they are said to be _____.

[a] ambiguos [b] overspecified

[c] contradictory [d] unambiguous

Q.11 Domain boundaries that differ by a constant are referred to as _____.

[a] system boundaries [b] irregular boundaries

[c] orthogonal boundaries [d] convex boundaries

Q.12 nice domains should not be _____.

[a] linear [b] simply connected

[c] concave [d] complete

Q.13 In certain cases, domains are more valuable in _____.

[a] interface testing [b] system testing

[c] unit testing [d] acceptance testing

Q.14 span is a term used to describe _____.

[a] largest value [b] range of values

[c] smallest value [d] middle value

Q.15 If all nice domains are complete, the span of the number space _____

[a] + infinity to – infinity in some dimensions

[b] + infinity to – infinity in all dimensions

[c] – infinity to +infinity in all dimensions

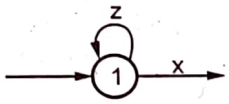
[d] – infinity to +infinity in some dimensions

Q.16 aa* = _____.

[a] a*a [b] a+

[c] a* [d] both a and b

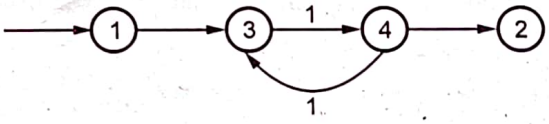
Q.17 The following loop's path expression would be :



[a] z*x [b] zx

[c] zx* [d] z+x

Q.18 The graph's comparable link weight would be



[a] 1 [b] 2

[c] 3 [d] 4

Q.19 The operation used to remove components from the stack is _____.

[a] POP [b] Remove

[c] Delete [d] None

Q.20 Concatenation is a method of expressing two consecutive path segments.

[a] Path product [b] Path sum

[c] Path [d] Path Expression

Answer Keys for Fill in the Blanks :

Q.1	Von Neuman	Q.2	Debugging
Q.3	dynamic	Q.4	Predicate
Q.5	ku, dk and dd	Q.6	Births and mergers
Q.7	transaction flow	Q.8	closed boundary
Q.9	Domain dimensionality	Q.10	boundary point
Q.11	Complete boundaries	Q.12	ambiguos
Q.13	integration test, unit test	Q.14	systematic boundary
Q.15	Partition testing	Q.16	PUSH
Q.17	Path Sum		

Answer Keys for Multiple Choice Questions :

Q.1	d	Q.2	d	Q.3	d
Q.4	b	Q.5	a	Q.6	d
Q.7	a	Q.8	a	Q.9	a
Q.10	c	Q.11	a	Q.12	c
Q.13	a	Q.14	b	Q.15	c
Q.16	d	Q.17	a	Q.18	c
Q.19	a	Q.20	a		

END...

UNIT - III

3 Paths, Path Products and Regular Expressions

3.1 Path Products and Path Expression, Reduction Procedure, Applications

3.2 Regular Expressions and Flow Anomaly Detection

Q.1 Explain path expression with examples.

☞ [JNTU : Part B, May-19, Marks 5, March-22, Marks 7]

OR Define following terms:

i. Path products
ii. Path expression
iii. Path sums

Ans. : i) Path products :

Flow graphs are typically used to represent solely control flow connections.

- A name is the simplest weight we may provide to a link.
- We then convert the graphical flow graph into an equivalent algebraic like expressions, which specifies the set of all feasible paths from entry to exit for the flow graph, using link names as weights.
- Each graph link can be given a name.
- Lower case italic letters will be used to indicate the link name.
- Tracing a path or path segment across a flow graph involves going through a series of link names.
- By concatenating the link names, the name of the path or path segment that corresponds to those links is naturally represented.
- For example, if we traverse links a, b, c, and d along a path, the path segment is called abcd.
- A path product is another name for this path name. The following are some examples :

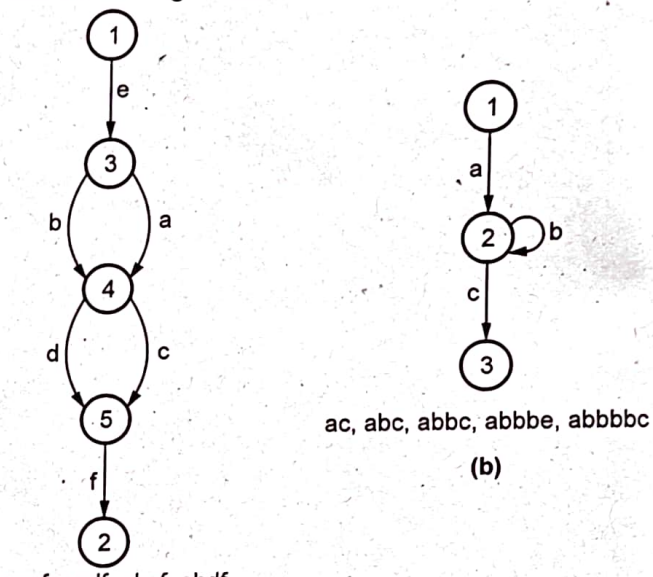


Fig. Q.1.1 Examples of path

The concatenation or path product of the segment names is a useful way to represent the name of a path that consists of two consecutive path segments.

For example, if X and Y are defined as X = abcde, Y = fghij,then the path corresponding to X followed by Y is denoted by

XY = abcdefghij

- Similarly,

YX = fghijabcde
aX = aabcde
Xa = abcdea
XaX = abcdeaabcde

ii) Path expression :

- When applied to structure (e.g., a control flow graph of an implementation), logic-based testing is

structural testing; when applied to a specification, it is functional testing.

- The truth values of control flow predicates are the subject of logic-based testing.
- A predicate is implemented as a process with a truth-functional value as the result.
- Logic-based testing is limited to binary predicates for our purposes.
- To weight the truth values of the predicates (e.g., A and) while converting path expressions to Boolean algebra
- Example : Consider a network with two nodes and a set of pathways connecting them.
- Use an upper case letter to denote that set of pathways, such as X,Y. The members of the path set can be stated as follows in Fig. Q.1.1 (b) :
- The same set of pathways can also be represented by :

  ac + abc + abbc + abbbc + abbbbc +.....
- The + sign is believed to signify "or" between the two nodes of interest, allowing us to take paths like ac, abc, abbc, and so on.
- A "Path Expression" is any expression that represents a set of paths between two nodes and consists of path names and "OR".

iii) Path sums :

Path names that were part of the same group of paths were indicated by the "+" sign.

- Paths in parallel between nodes are denoted by "PATH SUM."
- In Fig. Q.1.1 (a), links a and b are parallel pathways designated by a + b. Similarly, links c and d, represented by c + d, are parallel pathways between the following two nodes.
- The set of all pathways between nodes 1 and 2 is symbolized as eacf + eadf + ebcf + ebdf and can be thought of as a set of parallel paths.
- If X and Y are two sets of pathways that connect the same two nodes, then X + Y represents the UNION of those paths.

Example: Fig. Q.1.2 given below :

X + Y + d denote first set of parallel paths and the second set by U + V + W + h + i + j.

The set of all paths in this flowgraph is :

f (X + Y + d) g (U + V + W + h + i + j) k

- RULE 1 :

  A(BC) = AB(C) = ABC

  where A, B, C are path names, set of path names or path expressions
- It is commutative and associative
- RULE 2 : X+Y=Y+X
- RULE 3 : (X+Y)+Z=X+(Y+Z)=X+Y+Z

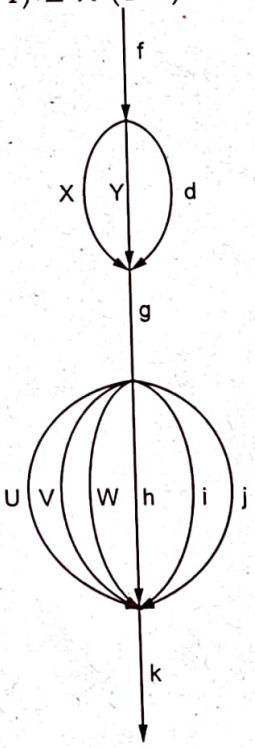


Fig. Q.1.2 Examples of path sums

Q.2 Write the procedure to count minimum number of paths in a graph. ☞ **[JNTU : Part B, Dec.-19, Marks 5, Sept.-21, March-22, Marks 8]**

Ans. :

- Because there may be unreachable paths resulting from correlated and dependent predicates, determining the actual number of alternative paths is an inherently difficult job.
- If we know both the maximum and least number of feasible paths, we will have a solid idea of how to finish our testing.
- It's pointless to inquire about "the average number of pathways."

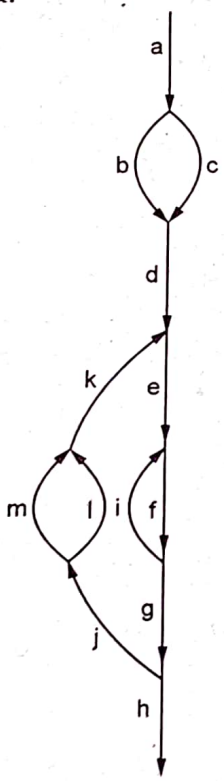
- Each link should be labeled with a link weight that corresponds to the number of paths it represents.
- The concatenation or path product of the segment names is a useful way to represent the name of a path that consists of two consecutive path segments.
- Also, write the maximum number of times each loop can be repeated. If the answer is unlimited, we should probably stop analyzing because the greatest number of pathways is infinite.
- Parallel linkages, serial links and loops are the three examples of interest.

Case	Path expression	Weight expression
Parallels	A + B	$W_A + W_B$
Series	AB	$W_A . W_B$
Loop	A^n	$\sum_{j=0}^{n} W_A j$

This is a standard algebra problem. The number of pathways in each set determines the weight.

Example :

The program that follows is reasonably well-structured.



a (b + c) d{e (f) * fgj (m + l) k} * e (fi) * fgh

Fig. Q.2.1

Each link represents a single link and has a weight of "1" to begin with. Let's say the outer loop is repeated four times, while the inner loop is repeated zero or three times its path expression, as follows : expression for the path :

a (b+c) d {d(fi) * fgj (m+l) k}*e(fi) * fgh*

A : To read the flow graph, replace the link name with the maximum number of trips through that link (1) and note the number of times for looping.

B : Outside the loop, join the initial pair of parallel loops, as well as the pair in the outer loop.

C : To clear the clutter, multiply the items and remove nodes.

Case	Path expression	Weight expression
Parallels	A + B	$W_A + W_B$
Series	AB	$W_A W_B$
Loop	A^n	$\sum_{j=0}^{n} W_A j$

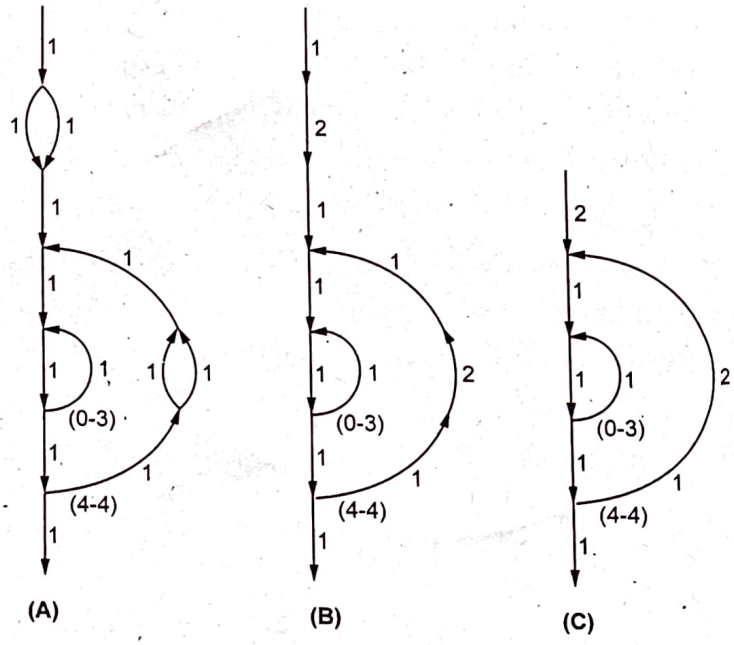


Fig. Q.2.2

Q.3 Write eight steps in a reduction procedure.

☞ [JNTU : Part A, Dec.-18, Marks 2]

Ans. : The following are the steps in the Reduction Algorithm :

1. Multiply the path expressions of all serial links to create a single link.
2. Add the path expressions of all parallel links together.
3. Replace any self-loops (from any node to itself) with a link of the form X*, where X is the path expression of the link in that loop.
4. Choose any node other than the initial or final node to be removed. Replace it with a collection of equivalent links whose path expressions correspond to all possible combinations of the node's set of inlinks and set of outlinks.
5. Multiply the path expressions of any remaining serial linkages to combine them.
6. Add the path expressions of all parallel links together.
7. Remove any self-loops in the same manner as in step 3.
8. Is there a single link between the entering node and the exit node in the graph? If we answered yes, the path expression for that connection is the same as the path expression for the original flowgraph; otherwise, go back to step 4.

- There are many alternative ways to generate the set of all paths between two nodes without altering the content of that set in a flowgraph; that is, there are many distinct ways to generate the set of all paths between two nodes without influencing the content of that set.
- The path expression's appearance is largely determined by the order in which nodes are deleted.

Q.4 Explain a reduction procedure Algorithm with example

Ans. : Refer Q.3.

Example :

Let's examine what happens if we apply this approach to the graph below, where we remove multiple nodes in order.

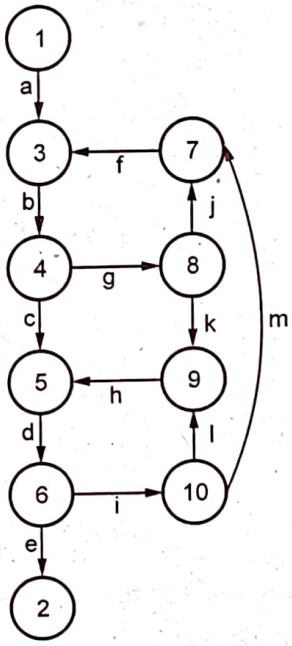


Fig. Q.4.1 Example flowgraph for demonstrating reduction procedure

- Step 4 is used to remove node 10 and step 5 is used to combine the nodes to produce.

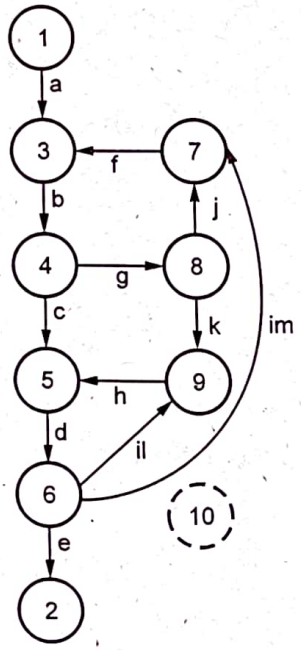


Fig. Q.4.2

- On yield, apply steps 4 and 5 to node 9 to remove it.

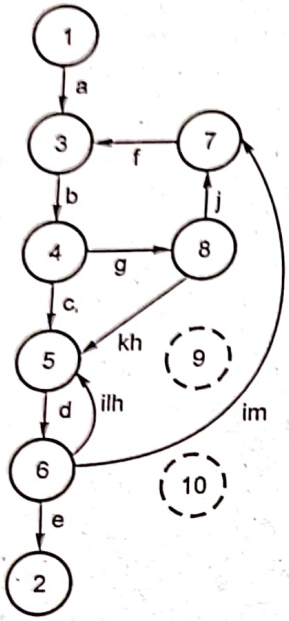


Fig. Q.4.3

- Steps 4 and 5 are used to remove node 7.

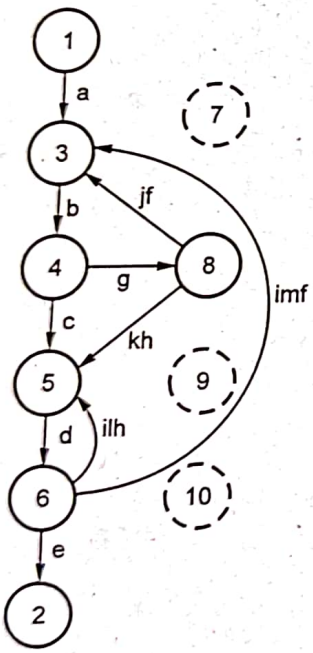


Fig. Q.4.4

- Using steps 4 and 5, remove node 8 to obtain :

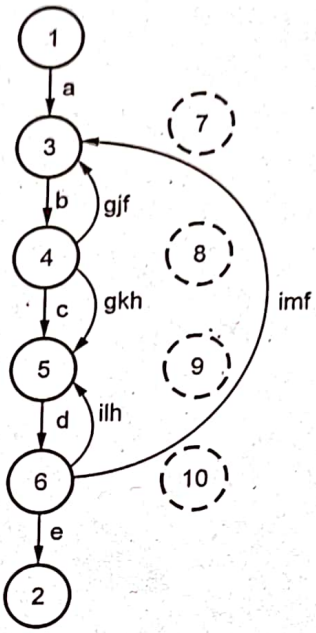


Fig. Q.4.5

- PARALLEL TERM (STEP 6): The removal of node 8 above resulted in a pair of parallel linkages between nodes 4 and 5. Combine them to make a path expression for an equivalent link with the path expression c + gkh.

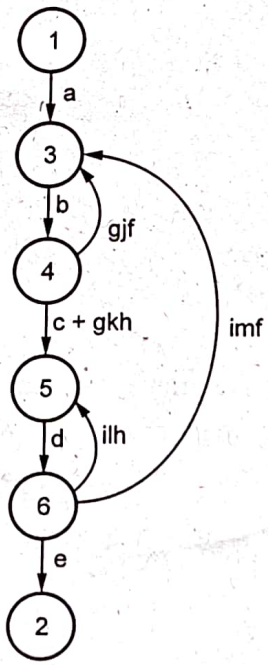


Fig. Q.4.6

- LOOP TERM (STEP 7): By removing node 4, a loop term is created. The graph has now been replaced by the following, simpler version :

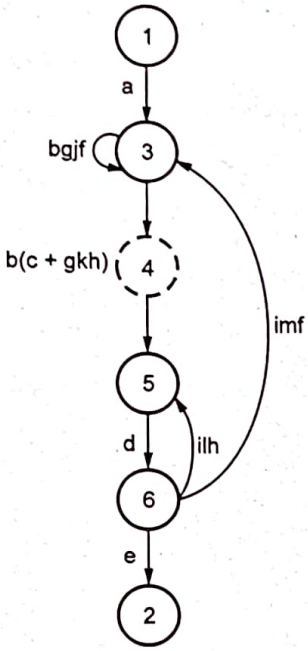


Fig. Q.4.7

- Apply the loop-removal step as follows to complete the process :

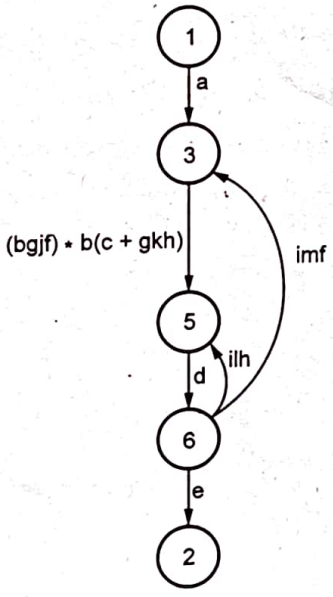


Fig. Q.4.8

- When node 5 is removed, the following results :

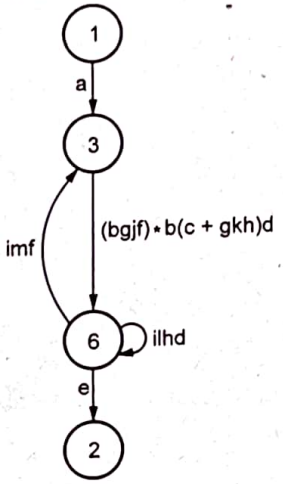


Fig. Q.4.9

- Remove the loop at node 6 to get the following :

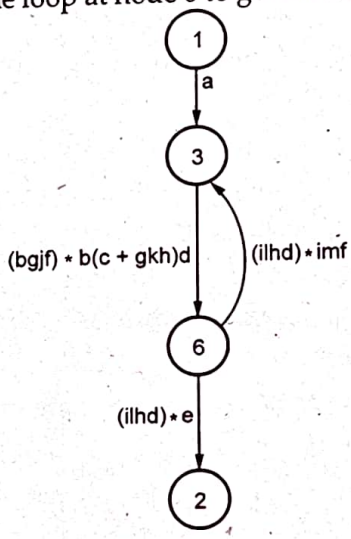


Fig. Q.4.10

- Remove node 3 to yield :

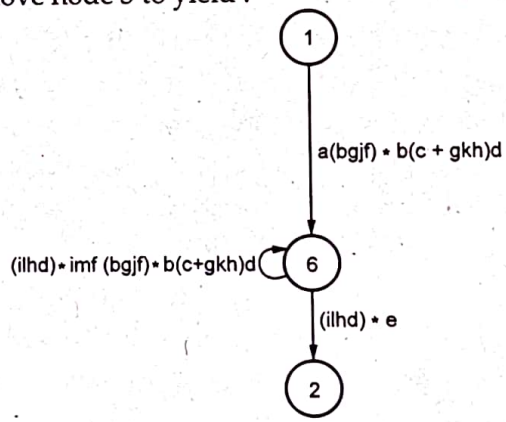


Fig. Q.4.11

The following expression is obtained by removing the loop and then node 6 :

a(bgjf)*b(c+gkh)d((ilhd)*imf(bjgf)*b(c+gkh)d)*(ilhd)*e

Q.5 Give applications of reduction procedure.

Ans. :

- The goal of the node removal technique is to present a single, very generalized concept : the path expression, as well as a method for obtaining it.
- This is the pattern that every application follows :

1. Create a path expression from the program or graph.
2. Identify a property of interest and generate a set of "arithmetic" rules to describe it.
3. For the property of interest, replace the link names with the link weights. The path expression has now been transformed into an expression in some algebra, such as regular expressions, Boolean algebra, or ordinary algebra. Over the set of all paths, this algebraic formula summarizes the attribute of interest.
4. To answer the question, simplify or evaluate the resulting "algebraic" statement.

Q.6 How many paths in a flow graph ?

Ans. :

- The question is not straightforward. Here are some possible questions :

1. What is the most number of alternative paths that can be taken ?
2. What is the smallest number of pathways that can be taken ?
3. How many distinct paths do you think there are ?
4. How many paths do you think there are on average ?

- Determining the real number of different paths is an intrinsically tough problem because correlated and dependent predicates might result in unattainable paths.
- We can get a decent notion of how thorough our testing is if we know both of these figures (maximum and least number of possible paths).
- It's pointless to inquire about "the average number of pathways."

Q.7 Compare structured and unstructured flow graphs and illustrate with an example.

☞ [JNTU : Part B, Dec.-19, Marks 5]

Ans. : Structured flow graphs :

- Structured code can be defined in a variety of ways that do not rely on ad hoc restrictions, such as the avoidance of GOTOs.
- A structured flowgraph is one that can be reduced to a single link by applying the operations in Fig. Q.7.1 over and over again. (Refer Fig. Q.7.1 on next page)

Properties :

- There are no cross-term transformations or GOTOs.
- There is no way to enter or exit a loop in the middle.

Example :

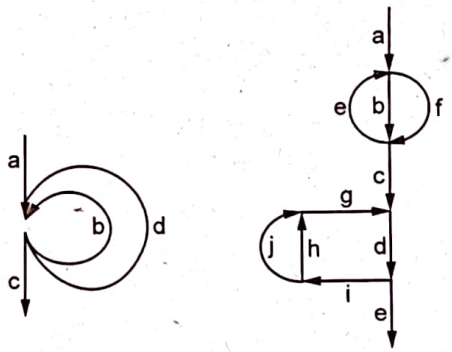


Fig. Q.7.2 Example of structured graph

Unstructured flow graphs :

The node-by-node reduction process can be used as a structured code test as well. Flow graphs that don't have one or more of the sub graphs listed in Fig. Q.7.3 are organized.

a) Jumping into loops

b) Jumping out of loops

c) Branching into decisions

d) Branching out of decisions

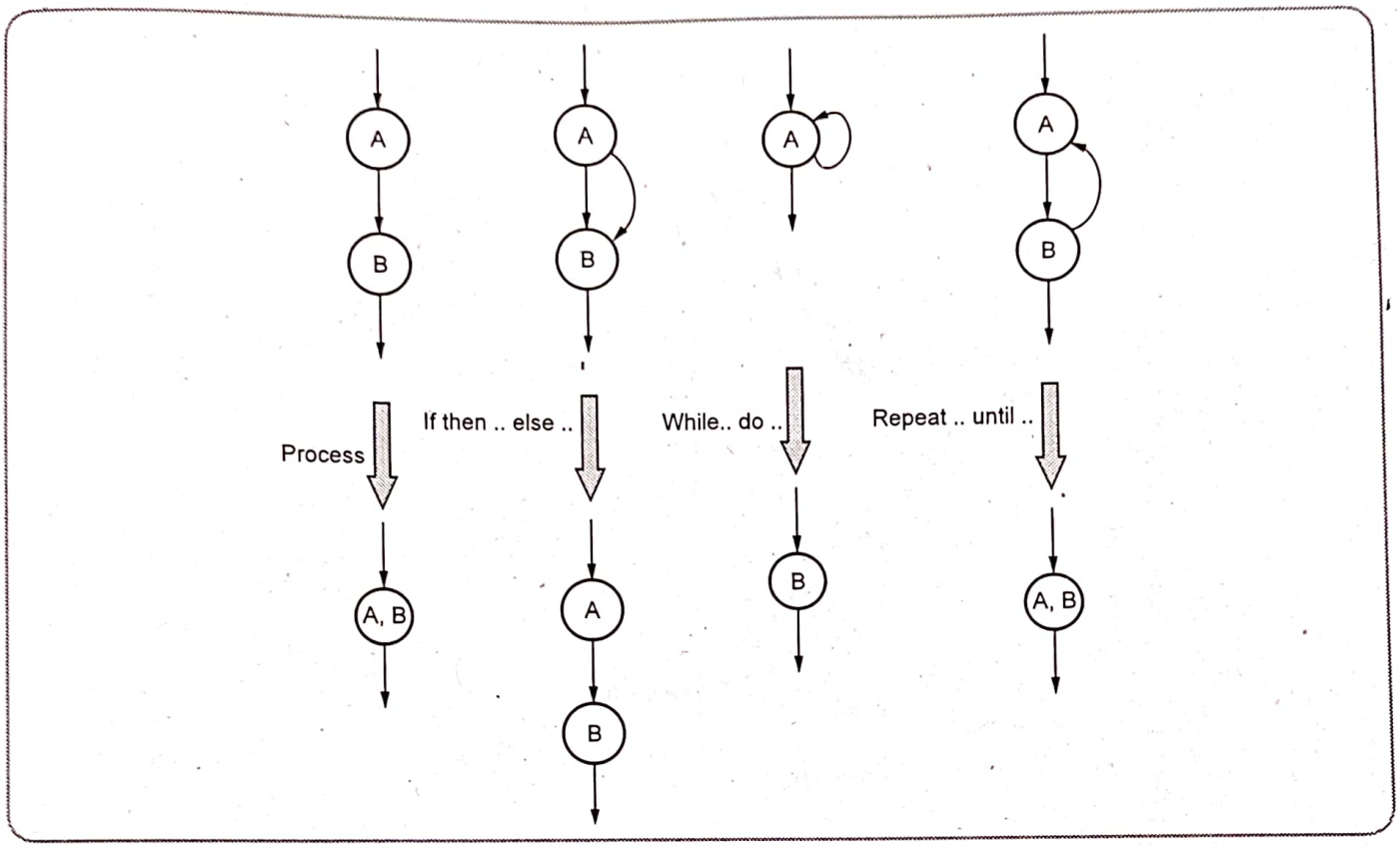
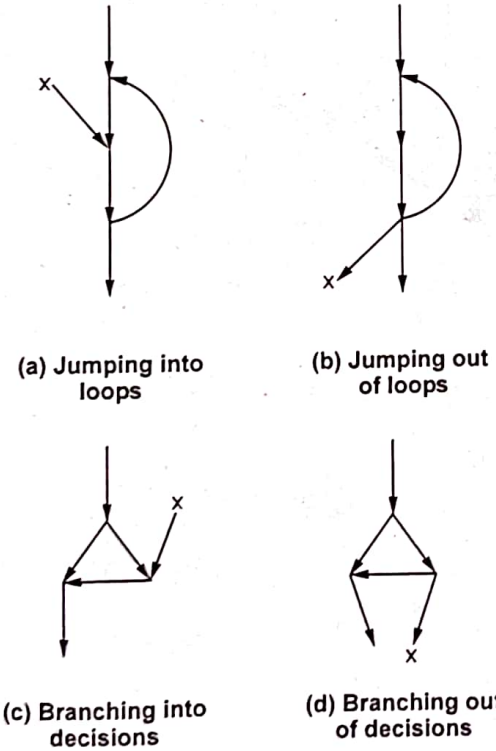


Fig. Q.7.1 Structural flow graph transformation



(a) Jumping into loops

(b) Jumping out of loops

(c) Branching into decisions

(d) Branching out of decisions

Fig. Q.8.3 Unstructured Flow Graphs

Q.8 Explain regular expressions.

☞ [JNTU : Sept.-21, Marks 5]

Ans. :

1. Assign an uppercase letter to each decision that indicates the truth value of the predicate. The YES or TRUE branch is labeled with a letter (for example, A), whereas the NO or FALSE branch is over scored with the same letter (say).
2. A path's true value is the sum of its component labels. "AND" is represented by concatenation or products. Figure's straight through path, for example, has a truth value of ABC because it passes through nodes 3, 6, 7, 8, 10, 11, 12, and 2. The value of the path via nodes 3, 6, 7, 9 and 2 is.
3. When two or more pathways intersect at a node, the fact is indicated using the plus sign (+), which stands for "OR."

The truth-functional values for several of the nodes can be stated in terms of segments from prior nodes using this standard. To identify the point, use the node name.

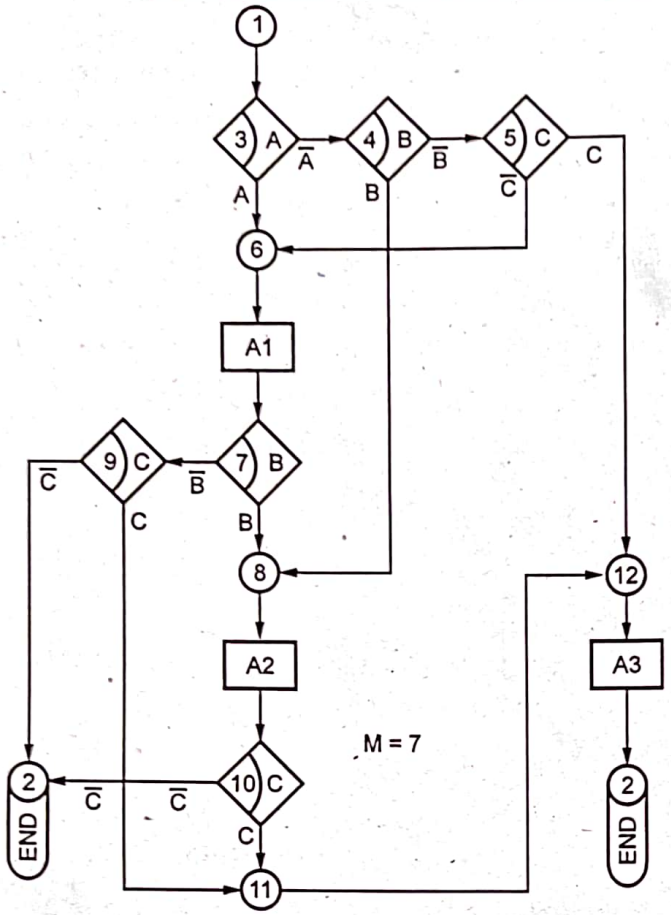


Fig. Q.8.1 A Troublesome program

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

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

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

$$N12 = N11 + \bar{A}\bar{B}C$$

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

- In Boolean algebra, there are only two numbers : zero (0) and one (1). "Always true" is one, whereas "always false" is zero.

Rules of Boolean algebra :

- There are three operators in Boolean algebra : X (AND), + (OR), and (NOT)
- X stands for AND, also called as multiplication. A sentence like AB (A X B) indicates "both A and B are true." In ordinary algebra, this symbol is generally omitted.
- + indicates that something is either true or false. "A + B" indicates "either A or B is true, or both are true."
- The letter 'A' stands for 'not'. Negation and complementation are also possible. This might be read as "not A" or "A bar." Under the bar, the entire expression is negated.

The following are the laws of Boolean algebra :

1.	$\frac{A}{A} + \frac{A}{A} = \frac{A}{A}$	If something is true, saying it twice doesn't make it truer, ditto for falsehoods.
2.	$A + 1 = 1$	If something is always true, then "either A or true or both" must also be universally true.
3.	$A + 0 = A$	
4.	$A + B = B + A$	Commutative law.
5.	$A + \bar{A} = 1$	If either A is true or not-A is true, then the statement is always true.
6.	$\frac{A}{A}\frac{A}{A} = \frac{A}{A}$	
7.	$A \times 1 = A$	
8.	$A \times 0 = 0$	
9.	$AB = BA$	
10.	$A\bar{A} = 0$	A statement can't be simultaneously true and false.
11.	$\bar{\bar{A}} = A$	"You ain't not going" means you are. How about, "I ain't not never going to get this nohow ?"
12.	$\bar{0} = 1$	
13.	$\bar{1} = 0$	
14.	$\overline{A + B} = \bar{A}\bar{B}$	Called "De Morgan's theorem or law."

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

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

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

18. $(A + B) + C = A + (B + C)$ So is addition.

19. $A + \bar{A}B = A + B$ Absorptive law.

20. $A + AB = A$

Q.9 Explain sum of product form and product of sum form. ☞ **[JNTU : Part A, April 18, Marks 2]**

Ans. :

1. A **product term** is the result of combining numerous literals. **(e.g., ABC, DE).**
2. An arbitrary Boolean statement that has been multiplied to produce the sum of products. (e.g., ABC + DEF + GH) is **called as product of sum.**

Q.10 Define literals and prime implicant.

Ans. : Literals :

Literals are individual letters in a Boolean algebra expression. (e.g. A,B).

Prime implicant :

The sum of product form is returned as a result of simplifications (using the criteria above), and each product term in this simplified version is referred to as a prime implicant.

For example, ABC + ABDEF reduce by rule 20 to AB + DEF; that is, AB and DEF are prime implicants.

Q.11 Briefly explain about regular expressions and flow-anomaly detection. **[JNTU : Part B, Dec.-18, Marks 5; Part A, April-18, May-19, Marks 2]**

Ans. :

- The generic flow-anomaly detection problem (note : not just data-flow anomalies, but any flow anomaly) entails searching for a given sequence of possibilities while examining all potential paths through a process.
- Assume the operations are SET and RESET, denoted by s and r, and we want to know whether there is a SET followed by a SET or a RESET followed by a RESET (an ss or an rr sequence).

Here are some more examples of applications :

1. A file can be opened, closed, read, or written (o, c, r, r, r, r, r, r, r, r (w)). The sequence is meaningless if the file is read or written to after it has been closed. As a result, cr and cw are outliers. Similarly, we may have an issue if the file is read before it is written, such as right after it is opened. As a result, or is also unusual. Furthermore, while oo and cc are not true bugs, they are a waste of effort and should be investigated as well.
2. Rewind (d), fast-forward (f), read (r), write (w), stop (p), and skip are all functions of a tape transport (k). There are rules about how to use the transportation; for example, we can't move from rewind to fast-forward without pausing, or from rewind or fast-forward to read or write without pausing. The sequences df, dr, dw, fd, and fr are out of the ordinary. Is there any way where the flowgraph leads to strange sequences ? If so, in what order and under what conditions ?
3. The dd, dk, kk, and ku sequences must be detected in order to detect the data-flow anomalies.

- The method : Each link in the graph should be anotated with the relevant operator or the null operator 1.
- Keep things as simple as possible by remembering that a + a = a and 12 = 1.
- We now have a regular expression that represents all potential operator sequences in the graph. We can now go through that regular expression for the sequences we are looking for.

Example : Assume A, B, and C are nonempty sets of character sequences with at least one character in the smallest string. Let T be a two-character character string. T will occur in AB2C if T is a substring of (that is, if T appears within) ABnC. (Theorem of HUANG)

As an example, let

A = pp
B = srr
C = rp
T = ss

The theorem states that ss will appear in pp(srr)nrp if it appears in pp(srr)^2rp.

However, let

A = p + pp + ps
B = psr + ps(r + ps)
C = rp
T = P^4

Is it obvious that there is a p4 sequence in ABnC ? The theorem states that we have only to look at

$(p + pp + ps)[psr + ps(r + ps)]^2rp$

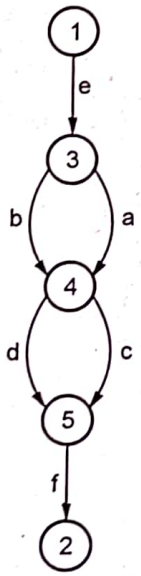
There is no p4 sequence when we multiply out the formula and simplify it. In case we are wondering, the following observation is an informal demonstration that looping twice is a good idea. Because data-flow anomalies are represented by two character sequences, the above theory dictates that finding such anomalies requires looping twice.

Q.12 Write short notes on following:

☞ [JNTU : Part B, April-18, Marks 10]

a) Distributive laws b) Absorption rule c) Loops

Ans. : a) Distributive laws :



eacf, eadf, ebcf, ebdf

Fig. Q.12.1 Example

The product and sum operations are distributive and the ordinary rules of multiplication apply; that is

RULE 4 : A(B+C)=AB+AC and (B+C)D=BD+CD

Applying these rules to Fig. Q.12.1 yields

e(a+b)(c+d)f=e(ac+ad+bc+bd)f = eacf+eadf+ebcf+ebdf

b) Absorption rule :

If X and Y denote the same set of paths, then the union of these sets is unchanged; consequently,

RULE 5: X+X=X (Absorption Rule)

If a set consists of paths names and a member of that set is added to it, the "new" name, which is already in that set of names, contributes nothing and can be ignored.

For example,

if X = a+aa+abc+abcd+def then

X+a = X+aa = X+abc = X+abcd = X+def = X

It follows that any arbitrary sum of identical path expressions reduces to the same path expression.

c) Loops

Loops can be understood as an infinite set of parallel paths. Say that the loop consists of a single link b. then the set of all paths through that loop point is $b^0+b^1+b^2+b^3+b^4+b^5+.....$

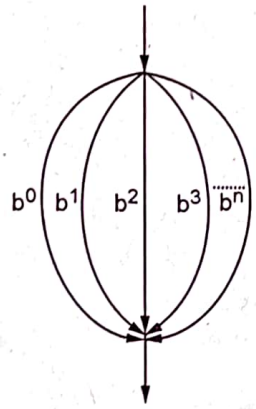


Fig. Q.12.2 Examples of path loops

Q.13 Describe push / pop and get / return models in path testing. ☞ [JNTU : Part B, Dec.-19, Marks 5]

Ans. : This model can be used to answer a variety of problems that may arise throughout the debugging process. It can also assist in determining which test scenarios to create.

The question is :

What is the net effect of a pair of complementary operations such as PUSH (the stack) and POP (the stack) when considering all potential pathways through the routine ? POP or PUSH ? How many times have you done this ? What are the circumstances ?

GET / RETURN a resource block.

OPEN / CLOSE a file.

START / STOP a device or process.

Example 1 (push / pop):

Here is the Push / Pop Arithmetic :

Case	Path expression	Weight expression
Parallels	A + B	$W_A + W_B$
Series	AB	$W_A W_B$
Loop	A^*	W_A^*

- The number 1 is used to signal that a link contains nothing of interest (neither PUSH nor POP).
- The letters "H" and "P" stand for PUSH and POP, respectively. Commutative, associative, and distributive operations are used.

PUSH / POP multiplication table

X	H PUSH	P POP	1 NONE
H	H^2	1	H
P	1	P^2	P
1	H	P	1

PUSH / POP addition table

*	H PUSH	P POP	1 NONE
H	H	P + H	H + 1
P	P + H	P	P + 1
1	H + 1	P + 1	1

Consider the following flow graph :

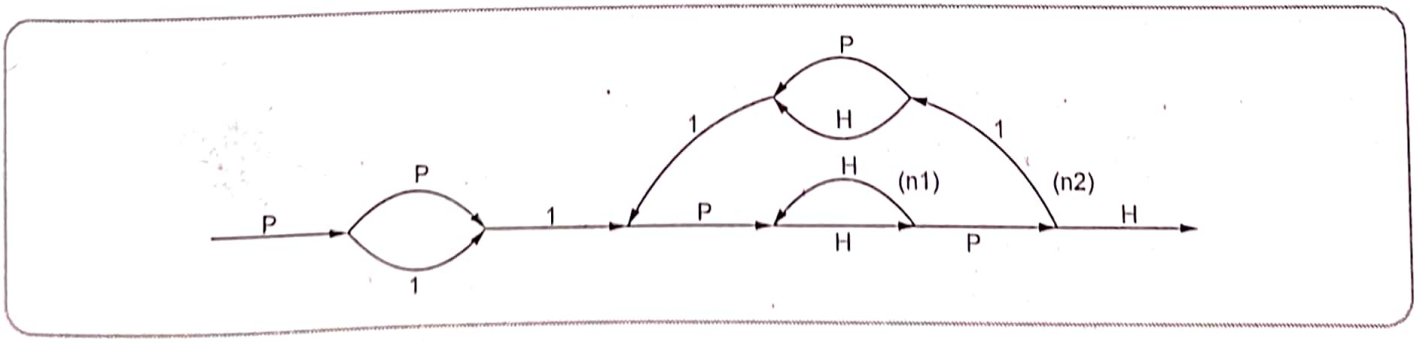


Fig. Q.13.1

Q.13 Describe push / pop and get / return models in path testing. [JNTU : Part B, Dec.-19, Marks 5]

Ans. : This model can be used to answer a variety of problems that may arise throughout the debugging process. It can also assist in determining which test scenarios to create.

The question is :

What is the net effect of a pair of complementary operations such as PUSH (the stack) and POP (the stack) when considering all potential pathways through the routine ? POP or PUSH ? How many times have you done this ? What are the circumstances ?

GET / RETURN a resource block.

OPEN / CLOSE a file.

START / STOP a device or process.

Example 1 (push / pop):

Here is the Push / Pop Arithmetic :

Case	Path expression	Weight expression
Parallels	A + B	$W_A + W_B$
Series	AB	$W_A W_B$
Loop	A^*	W_A^*

- The number 1 is used to signal that a link contains nothing of interest (neither PUSH nor POP).
- The letters "H" and "P" stand for PUSH and POP, respectively. Commutative, associative, and distributive operations are used.

PUSH / POP multiplication table

X	H PUSH	P POP	1 NONE
H	H^2	1	H
P	1	P^2	P
1	H	P	1

PUSH / POP addition table

*	H PUSH	P POP	1 NONE
H	H	P + H	H + 1
P	P + H	P	P + 1
1	H + 1	P + 1	1

Consider the following flow graph :

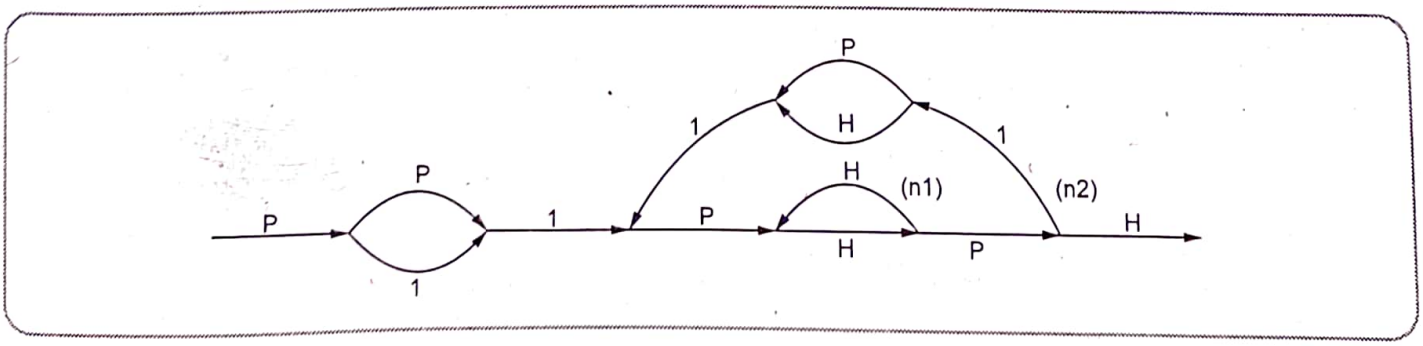


Fig. Q.13.1

$P(P + 1)1\{P(HH)^{n1}HP1(P + H)1\}^{n2}P(HH)^{n1}HPH$

Simplifying by using the arithmetic tables,

$$= (P^2 + P)\{P(HH)^{n1}(P + H)\}^{n1}(HH)^{n1}$$

$$= (P^2 + P)\{H^{2n1}(P^2 + 1)\{^{n2}H^{2n1}\}$$

Table Q.13.1 below illustrates several values for the two looping terms M_1 and M_2, where M_1 is the number of times the inner loop will be taken and M_2 is the number of times the outer loop will be taken.

M_1	M_2	PUSH / POP
0	0	$P + P^2$
0	1	$P + P^2 + P^3 + P^4$
0	2	$\sum_{1}^{6} P^i$
0	3	$\sum_{1}^{8} P^i$
1	0	$1 + H$
1	1	$\sum_{0}^{3} H^i$
1	2	$\sum_{0}^{5} H^i$
1	3	$\sum_{0}^{7} H^i$
2	0	$H^2 + H^3$
2	1	$\sum_{4}^{7} H^i$
2	2	$\sum_{6}^{11} H^i$
2	3	$\sum_{8}^{15} H^i$

Table Q.13.1 Result of the PUSH / POP graph analysis

- These expressions specify that the stack will only be popped if the inner loop is not executed.
- If the inner loop is iterated once, the stack will be left alone, but it can also be pushed.
- The stack will only be pushed for all other inner loop values.

Example 2 (GET / RETURN) :

For GET / RETURN a buffer block or resource, or, in fact, for any pair of complementary operations in which the total number of operations in either direction is cumulative, the same arithmetic tables are used as in the preceding example.

The following are the arithmetic tables for GET / RETURN :

Multiplication Table

X	G	R	1
G	G^2	1	G
R	1	R^2	R
1	G	R	1

Addition Table

*	G	R	1
G	G	G+R	G+1
R	G+R	R	R+1
1	G+1	R+1	1

"G" denotes GET and "R" denotes RETURN.

Consider the following flowgraph :

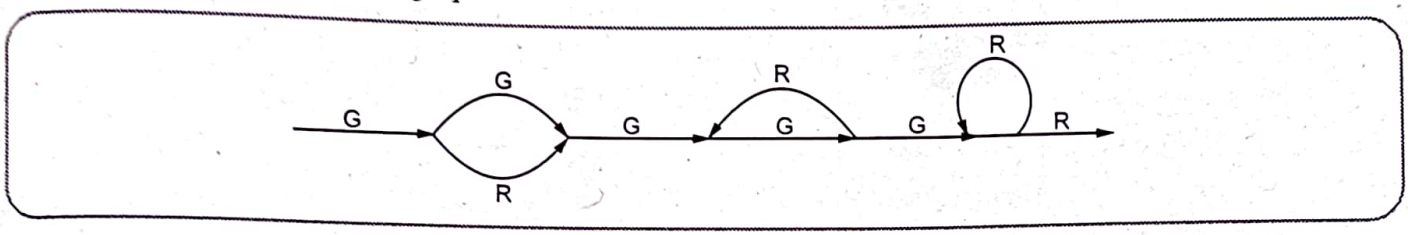


Fig. Q.13.2

$$G(G+R)G(GR)^*GGR^*R = G(G+R)G^3R^*R$$
$$= (G+R)G^3R^*$$
$$= (G^4+G^2)R^*$$

- When the routine ends, this expression indicates the conditions under which the resources will be balanced.
- If the first decision is to choose the top branch, the second loop must be taken four times.
- If the first decision is to choose the lower branch, the second loop must be taken twice.
- The process will not balance for any other values. As a result, the first loop does not need to be instrumented in order to test this behavior because it should have no impact.

3.3 Logic based Testing : Overview, Decision Tables, Path Expressions, KV Charts, Specifications

Q.14 Explain overview of logic based testing in detail.

OR What is logic based testing ? ☞ **[JNTU : Part A, May-19, Marks 2]**

Ans. :

- One of the most commonly used words in programmers' vocabularies is "logic," but it is also one of the least employed methodologies.
- As arithmetic is to mathematics, Boolean algebra is to logic. Many test and design methodologies, as well as tools that include those techniques, are unavailable to the tester or programmer without it.
- For decades, logic has been the principal tool used by hardware logic designers.
- Many hardware logic testing approaches can be converted to software logic testing. Hardware testing methods and accompanying theory are fertile ground for software testing methods since hardware testing automation is 10 to 15 years ahead of software testing automation.
- As programming and testing methodologies evolved, errors moved closer to the front end of the process, to requirements and their specifications. These bugs make up 8% to 30% of the total, and because they're the first in and last out, they're the most expensive.

- The problem with standards is that they are difficult to communicate.
- The most fundamental logic system is Boolean algebra (also known as the sentential calculus).
- For formal specifications, higher-order logic systems are required.
- Tools can and do contain a lot of logical analysis. However, these tools include methods for simplifying, transforming, and checking specifications, which are mostly based on Boolean algebra.

Q.15 What do you mean by knowledge based system ?

Ans. :

- The knowledge-based system (also known as an expert system or "artificial intelligence" system) has become the programming framework of choice for many formerly difficult applications.
- Knowledge-based systems use a database to store information from a knowledge domain such as medicine, law or civil engineering. After then, the data can be accessed and interacted with to find solutions to problems in that domain.
- Incorporating the expert's knowledge into a set of rules is one way to construct knowledge-based systems. After that, the user can provide data and ask questions based on it.
- The user's data is evaluated by the rule base, which results in tentative or definitive findings as well as requests for more data. The inference engine is the program that does the processing.
- An understanding of formal logic is required to comprehend knowledge-based systems and related validation issues.

Q.16 Write a short note on : Decision Tables

Ans. : Many programs functional requirements can be stated using decision tables, which serve as a valuable foundation for program and test design.

- A restricted - entry decision table is shown in table Q.16.1. It is divided into four sections : the condition stub, condition entry, action stub, and action entry.
- Each table column is a rule that specifies the conditions under which the actions listed in the action stub will occur.
- A condition stub is a list of condition names.

Condition Entry

	RULE 1	RULE 2	RULE 3	RULE 4
CONDITION 1	YES	YES	NO	NO
CONDITION 2	YES	1	NO	1
CONDITION 3	NO	YES	NO	1
CONDITION 4	NO	YES	NO	YES
ACTION 1	YES	YES	NO	NO
ACTION 2	NO	NO	YES	NO
ACTION 3	NO	NO	NO	YES

condition stub

action stub

ACTION ENTRY

Table Q.16.1 Example 1

More example of decision table given below in Table Q.16.2

Printer troubleshooter

		Rules							
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 unrecognised	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				

Table Q.16.2 Example 2

- A rule indicates whether or not a condition must be met in order for the rule to be fulfilled. "YES" indicates that the condition must be met, "NO" indicates that it must not be met, and "I" indicates that the condition has no bearing on the rule or is irrelevant to it.

  If the rule is met, the action stub names the actions that the routine will take or commence.

- The action will take place if the action input is "YES," and it will not take place if the action entry is "NO.".

Table Q.16.1 is translated in Table Q.16.3.

If conditions 1 and 2 are met and conditions 3 and 4 are not met (rule 1), action 1 will be taken. If conditions 1, 3, and 4 are met, action 2 will be taken (rule 2).

- Predicate is sometimes known as "condition."
- The terms "condition" and "satisfied" or "meet" are used in the Decision - table. Let's use the terms "predicate" and "TRUE / FALSE."
- The above translations are now :

  1. If predicates 1 and 2 are true and predicates 3 and 4 are false (rule 1), or if predicates 1, 3, and 4 are true, action 1 will be executed (rule 2).
  2. If all of the predicates are false, action 2 will be done (rule 3).
  3. If condition 1 is false and predicate 4 is true, action 3 will be taken (rule 4).

- In addition to the specified rules, we need a default Rule that indicates what should happen if all other rules fail. Table Q.16.3 shows the default rules for Table in Q.16.1.

	Rule 5	Rule 6	Rule 7	Rule 8
CONDITION 1	1	NO	YES	YES
CONDITION 2	1	YES	1	NO
CONDITION 3	YES	1	NO	NO
CONDITION 4	NO	NO	YES	1
DEFAULT ACTION	YES	YES	YES	YES

Table Q.16.3 The default rules of Table in Table Q.16.1

Q.17 Describe the mean processing time of a routine with example. ☞ **[JNTU : Part B, Dec.-18, Marks 5]**

Ans. : Find the mean processing time for the routine as a whole given the execution time of all statements or instructions for every link in a flowgraph and the probability for each direction for all decisions.

Every link in the model has two weights : the processing time for that link, represented by T and the probability of that link, denoted by P.

The following are the math rules for computing the average time :

Case	Path expression	Weight expression
Parallel	A + B	$T_{A+B} = (P_A T_A + P_B T_B) / (P_A + P_B)$ $P_{A+B} = P_A + P_B$
Series	AB	$T_{AB} = T_A + T_B$ $P_{AB} = P_A P_B$
Loop	A^n	$T_A = T_A + T_L P_L / (1 - P_L)$ $P_A = P_A / (1 - P_L)$

example :

1. Begin by looking at the original flow graph, which has been labeled with probabilities and processing times.

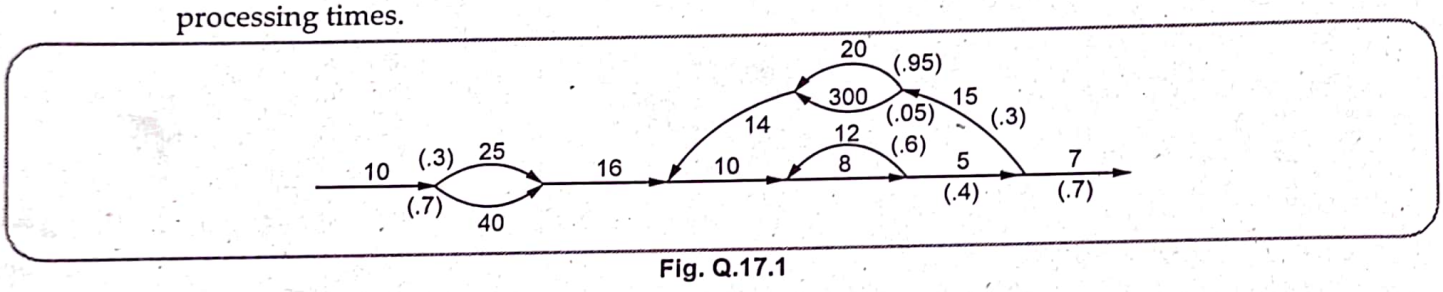


Fig. Q.17.1

2. Connect the outer loop's parallel links. Because there are no further links leaving the initial node, the result is simply the mean of the processing times for the linkages. Combine the two links at the start of the flow graph as well.

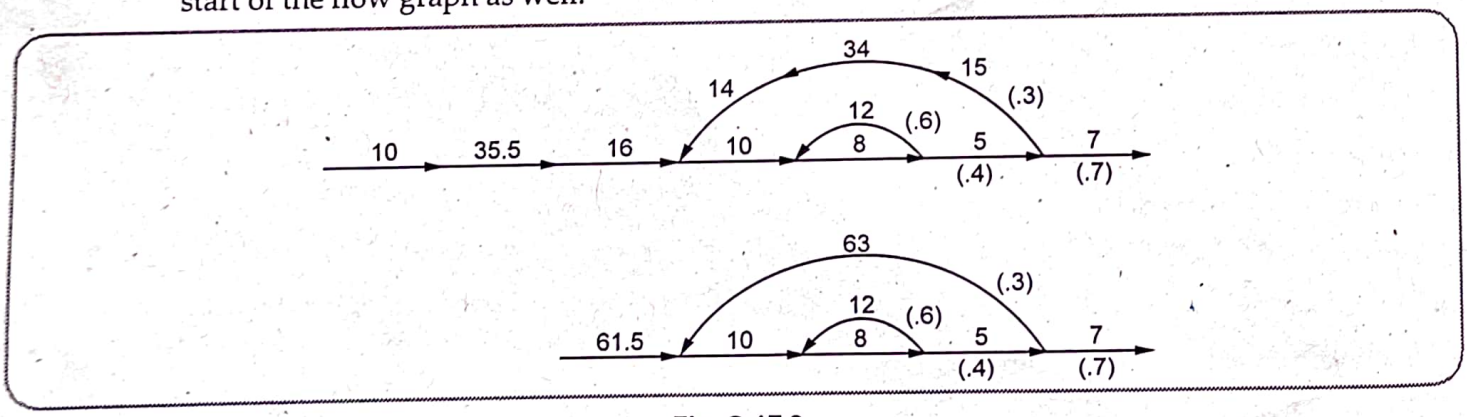


Fig. Q.17.2

3. Use as many serial links as possible.

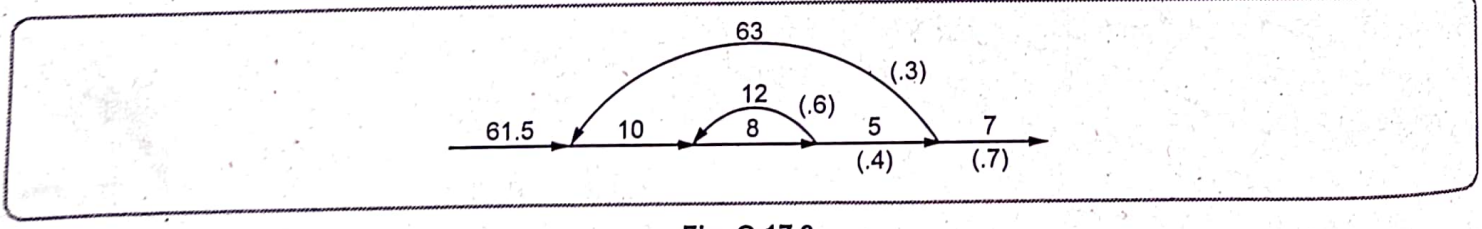


Fig. Q.17.3

4. To eliminate a node and build the inner self - loop, use the cross - term step.
5. Finally, using the following arithmetic rules, we can calculate the mean processing time:

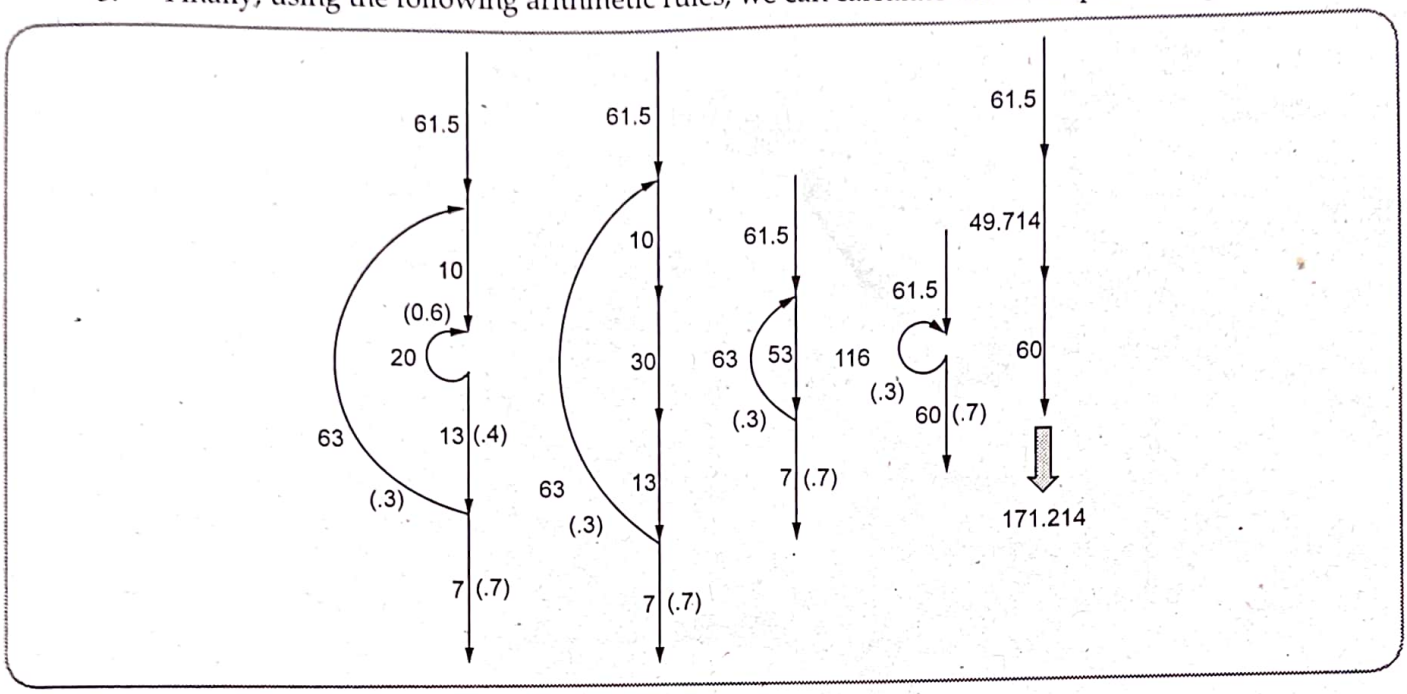


Fig. Q.17.4

Q.18 What is KV - Chart ? Draw KV-chart for Single, two and three variables.

OR What is KV - Chart ? Draw KV-chart for 4 variables.

☞ [JNTU : Part B, Dec.-19, Marks 5, March-22, Marks 7]

Ans. : We could become mired down in the algebra and make as many errors in constructing test cases as there are flaws in the process we're testing if we had to deal with expressions in four, five, or six variables.

Karnaugh-Veitch chart graphics trivialises boolean algebraic manipulations.

These graphs become clumsy after six variables and may not be effective.

Single Variable :

- Fig. Q.18.1 depicts all of a single variable's boolean functions and their KV chart representation.
- The graphs depict all potential truth values for the variable A.
- A "1" indicates that the value of the variable is "1" or TRUE. A "0" indicates that the value of the variable is 0 or FALSE.
- For that value of the variable, the item in the box (0 or 1) shows whether the function that the chart represents is true or false.
- We don't normally put 0 entries in, instead specifying simply the conditions under which the function is true.

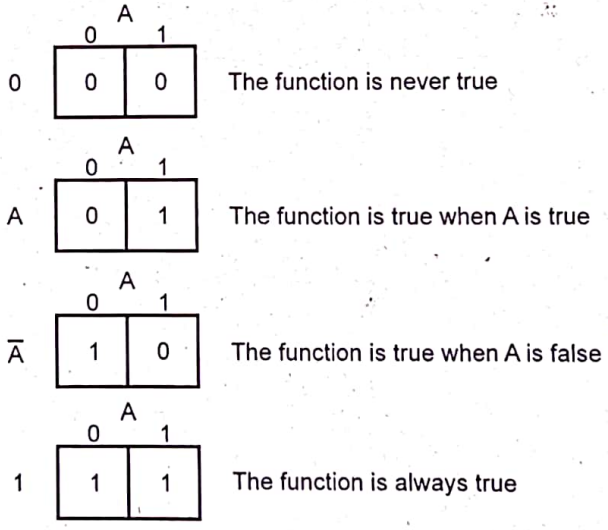


Fig Q.18.1 KV for single variable

Two variables :

- Fig. Q.18.2 depicts eight of the sixteen potential two-variable functions.

- Each box represents a combination of the variables' values for that box's row and column.
- A pair can be horizontally or vertically contiguous, but not diagonally.
- The expression excludes any variable that varies in either the horizontal or vertical direction.
- The B variable changes from 0 to 1 down the column in the fifth chart and because the A variable's value for the column is 1, the chart is identical to a simple A.

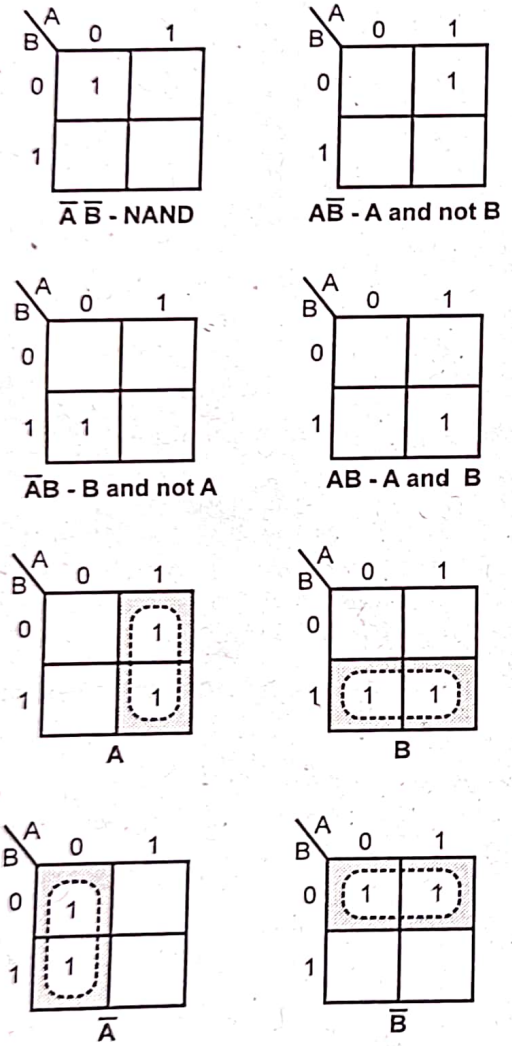


Fig. Q.18.2 KV for two variables

KV charts for three variables.

- Each box, as previously, represents a three-variable elementary term, with a bar appearing or not depending on whether the row-column heading for that box is 0 or 1.
- A three-variable chart can have 1, 2, 4, and 8 box groupings.
- The ideas will be illustrated by a few examples :

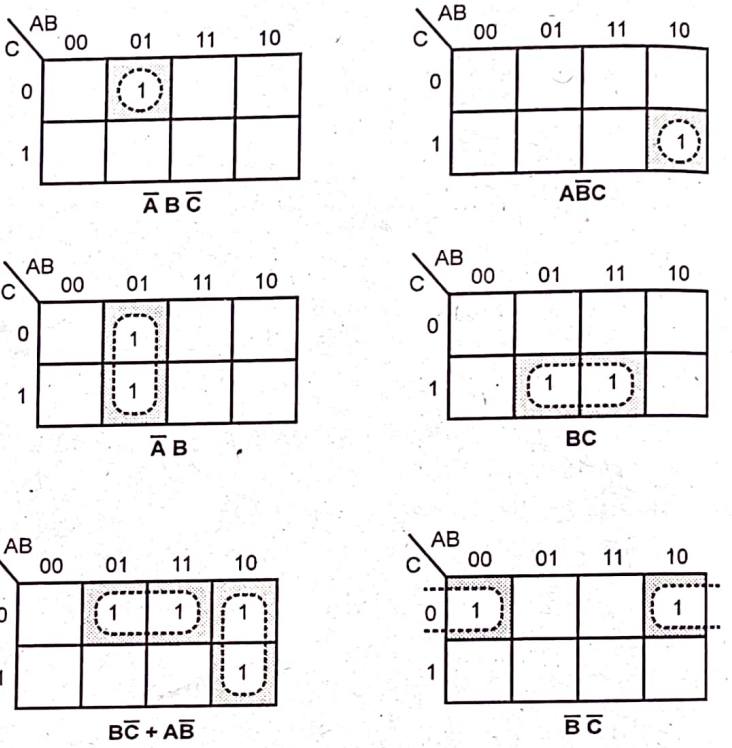


Fig. Q.18.3 KV for three variables

KV - chart for 4 variables :

The illustration depicts a changeable KV Chart as well as other alternative adjacencies. Adjacencies can now have 1,2,4;8, and 16 boxes, resulting in terms with 4,3,2,1 and 0 in them, accordingly.

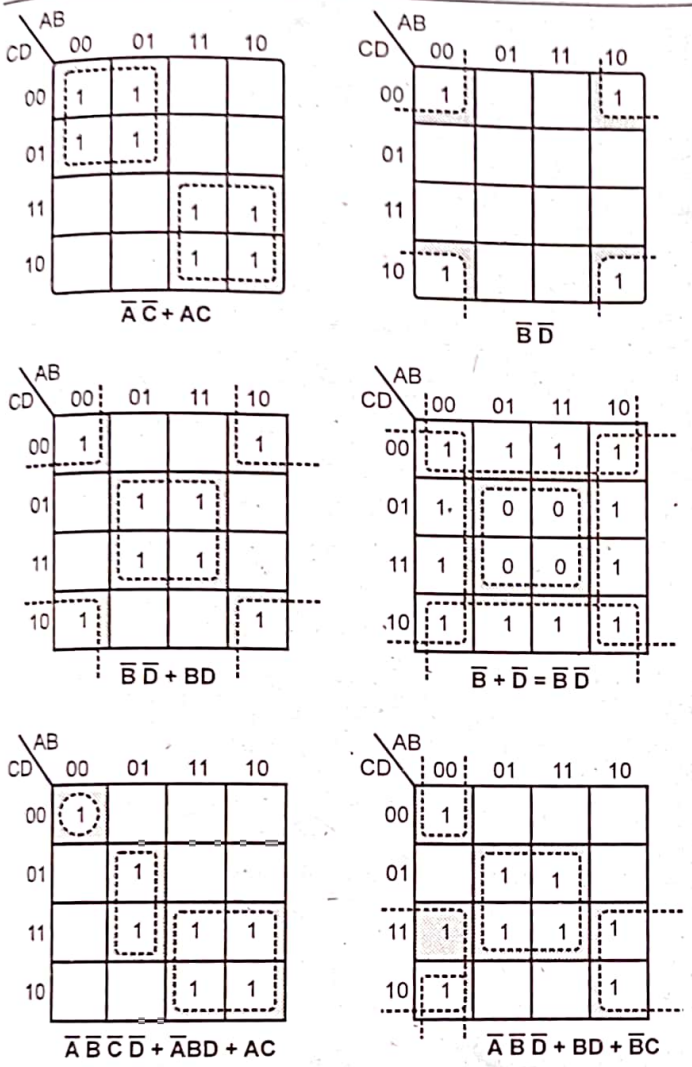


Fig. Q.18.4

Q.19 Reduce the following functions using K-Maps
F(A,B,C,D) = P(4,5,6,7,8,12,13)+d(1,15)

[JNTU : Part B, April-18, Marks 10]

Ans. :

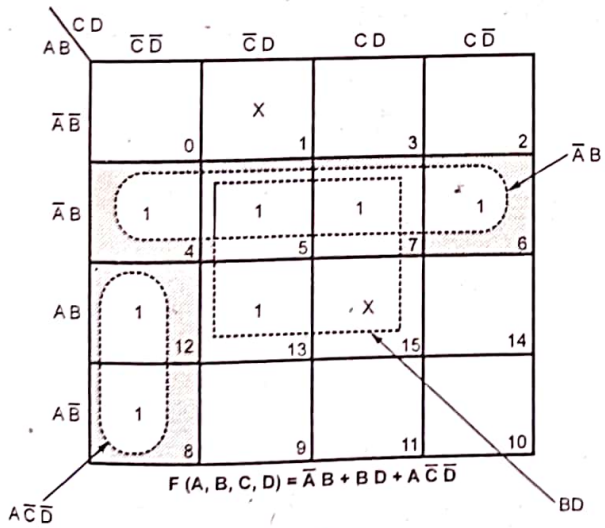


Fig. Q.19.1

Q.20 Write the procedure for specification validation.

[JNTU : Part B, Dec.-18, Marks 5]

Ans. : The technique for validating specifications is simple:

1. Rewrite the specification using wording that is consistent.
2. Determine the predicates that the cases are based on. Appropriate letters, such as A, B, and C, should be used to name them.
3. Rewrite the specs in English, using only the logical connectives AND, OR, and NOT, regardless of how awkward it appears.
4. Convert the modified specifications into a set of Boolean expressions that are equivalent.
5. If any default actions or circumstances are given, identify them.
6. Check for consistency by entering the Boolean expression in the KV chart. There will be no overlaps if the specifications are consistent, except in circumstances when several actions are required.
7. Check for consistency by using the default cases.
8. The specification is complete if all of the boxes are checked.
9. If the specification is unclear or inconsistent, translate the KV chart's appropriate boxes into English and request a clarification, explanation, or revision.
10. If the default cases were not clearly indicated, translate them back into English and get confirmation.

Fill in the Blanks for Mid Term Exam

Q.1 The simplest type of reasoning is _____.

Q.2 _____ gathers knowledge from knowledge repository or domain such as Engineering law into Database.

Q.3 _____ is a table that contains a collection of conditions as well as the actions that go with them.

Q.4 The table translator will look for _____ of source table and fill in the default rules.

Q.5 When a decision table acts as a _____, it is used for test case design.

Q.6 Entries that have no bearing on the decision table are referred to as _____.

Q.7 The procedure that produces a truth value is referred to as _____.

Q.8 ABCD+ABC+AB+A can be simplified to __.

Q.9 _____ is the one that covers the minimum and maximum terms.

Q.10 In order to assess _____, KV charts are employed.

Q.11 Don't care situations are most common in _____.

Q.12 _____ are the components of a decision table.

Q.13 Expanding tables of _____ can help ensure the consistency and completeness of the decision table.

Q.14 AB+AC = _____.

Q.15 The decision table will be translated into _____.

Multiple Choice Questions for Mid Term Exam

Q.1 The drawback of the decision table is that :

[a] Insufficient program logic

[b] No clarity

[c] Implicit relation to specification

[d] Low level maintainability

Q.2 If there are k predicates, the maximum number of possible predicate cases is _____.

[a] 2k [b] 2K

[c] 2k-1 [d] 2K-1

Q.3 The proportional calculus is a type of calculus in which there are more than two truth values is known as _____.

[a] Multiway predicate [b] Multivalued logic

[c] Multiway truth [d] None

Q.4 "FORTRAN 3 WAY IF" is _____.

[a] multiway predicate [b] multivalued logic

[c] both a and b [d] none

Q.5 According to absorptive law, $A+A\bar{B}$ is _____.

[a] B [b] A+B'

[c] A+B [d] A'+B

Q.6 ABC+BCD+CDE+EFG is in the form of _____.

[a] sum of products [b] product of additions

[c] reduction of sums [d] product of sums

Q.7 ABCD+BCD+CD+AB can be reduced to _____.

[a] CD+AB [b] 1

[c] CD [d] AB

Q.8 To reduce Boolean expression, the initial step is to _____.

[a] replace identical terms

[b] remove parenthesis

[c] the term containing a variable and its compliment is removed

[d] none

Q.9 In KV charts, the order of grouping should be _____.

[a] octets,islands,pairs,quads

[b] octets, quads, pairs, islands

[c] quads, pairs, islands, octets

[d] islands, pairs, quads, octets

Q.10 The number of interchanges for each combination of predicate truth values for n predicates would be _____.

[a] n(n-1)/2 [b] 2n

[c] n^2 [d] n(n-1)

Q.11 The list of conditions' names is referred to as _____.

[a] condition stub [b] condition entry

[c] action entry [d] action stub

Q.12 When all other rules have failed to meet the requirements, the following rules are,

[a] Conditional rules [b] Action rules

[c] Action entries [d] Default rules

Q.13 If a table has two unimportant cases, the extension will result in _____.

[a] 4 columns or cases. [b] 8 columns or cases.

[c] 16 columns or cases. [d] 2 columns or cases.

Q.14 Which of the following statements about predicate values is correct ?

[a] They are not restricted to binary truth values

[b] They are restricted to binary truth values in logic based testing

[c] Both a and b

[d] None

Q.15 Boolean Algebra expression laws AB = _____.

[a] A' . B' [b] A + B

[c] A' + B' [d] (A+B)'

Answer Keys for Fill In the Blanks :

Q.1	Boolean Algebra	Q.2	Knowledge based systems
Q.3	Decision table	Q.4	consistency and completeness
Q.5	specification	Q.6	immaterial entries
Q.7	predicate	Q.8	A
Q.9	Implicant	Q.10	specifications
Q.11	Digital Electronics	Q.12	Condition stub, condition entry, action stub, action entry
Q.13	Immaterial cases	Q.14	A (B + C)
Q.15	source code		

Answer Keys for Multiple Choice Questions :

Q.1	d	Q.2	a	Q.3	b
Q.4	a	Q.5	c	Q.6	a
Q.7	a	Q.8	b	Q.9	b
Q.10	a	Q.11	a	Q.12	d
Q.13	a	Q.14	c	Q.15	b

END...

UNIT - IV

4 State, State Graphs and Transition Testing

4.1 State Graphs, Good and Bad State Graphs

4.2 State Testing, Testability Tips

Q.1 Define the following terms :
1. Finite state 2. State graph
3. Finite state machine

Ans. : Finite state :

- The finite state machine, like Boolean algebra, is crucial to software engineering.
- State testing methodologies rely on finite state machine models for software structure, software behavior, or software behavior specifications.
- Finite state machines can also be implemented as table-driven software, making them a versatile design tool.

State graph :

"A mix of conditions or traits relating for the time being to a person or object," says the dictionary.

For example, a moving automobile with its engine operating can have the transmission states listed below.

Reverse gear
Neutral gear
First gear
Second gear
Third gear
Fourth gear

State graph - Example

Software that recognizes the character sequence "ZCZC," for example, could be in one of the following stages.

Z has been detected.

ZC has been detected.

ZCZ has been detected.

ZCZC has been detected.

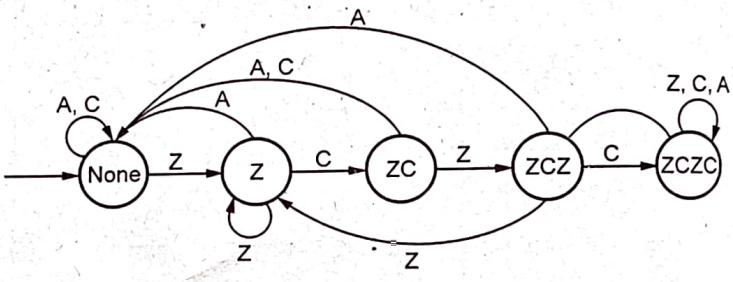


Fig. Q.1.1

Nodes are the nodes that represent the states. States can be designated by numbers, words or anything else that is convenient.

Input and transitions :

1. Inputs are applied to whatever is being modeled. The state changes as a result of those inputs, and is said to have transitioned.
2. Links that connect the states are used to indicate transitions.
3. The link is identified with the inputs that produce the transition; the inputs are link weights.
4. For each input, there is only one out link from each state.

5. Instead of displaying a number of parallel linkages when several inputs in a state produce a transition to the same future state, we can condense the notation by listing the multiple inputs as "input1, input2, input3....."

Finite state machine :

- A finite state machine is a non-physical device that may be represented by a state graph with a finite number of states and transitions between them.
- Any link can be paired with n outputs.
- Letters or words signify outputs, which are separated from inputs by a slash, as in "input/output."
- Output, as always, refers to anything observable that is of interest and is not limited to explicit device outputs.
- Link weights are also outputs.
- If all of the inputs to a transition result in the same output, write it down as: **"input1, input2, input3... ..... /output"**

Q.2 Explain state tables with example.

Ans. : Large state graphs are cluttered and difficult to understand.

The state graph is more easily represented as a table (the state table or state transition table) that lists the states, inputs, transitions and outputs.

The following conventions are followed :

- Each state is represented by a row in the table.
- Each column represents a different input condition.

The next state (the transition) and, if applicable, the output are specified in the box at the intersection of a row and a column.

Example :

	Inputs		
STATE	Z	C	A
NONE	Z	NONE	NONE
Z	Z	ZC	NONE
ZC	ZCZ	NONE	NONE
ZCZ	Z	ZCZC	NONE
ZCZC	ZCZC	ZCZC	ZCZC

1. State graphs indicate sequence rather than time.
2. A system could be in one state for milliseconds and another for years, and the state graph would be the same since it has no concept of time.
3. Although, using temporal Petri Nets, the finite state machines model can be expanded to include time as well as sequence.

Software development :

There is rarely a direct correlation between program behavior and the behavior of a state graph-based process.

The state graph depicts the overall behavior, which includes transportation, software, the executive, status returns, interrupts and so on.

There isn't a straightforward relationship between lines of code and states. The state table is the foundation.

Q.3 Explain good state graph with suitable example.

☞ **[JNTU : Part B, Dec.-19, Marks 5]**

OR Define good state and bad state graphs.

☞ **[JNTU : Part A, April-18, Marks 2, Sept.-21, Marks 7]**

Ans. : The types of state graphs that are likely to be used in the context of a software test design influence whether a state graph is excellent or bad.

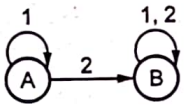
The characteristics of a good state graph :

- The product of the possibilities of the elements that make up the stat equals the total number of states.
- There is precisely one transition stated to exact one, perhaps the same state, for each state and input.
- One output action is provided for each transition. It doesn't matter if the output is little or useful.
- Every state has a set of inputs that will cause the system to return to that state.
- A state graph should, in general, have at least two separate input codes.
- A state graph should, in general, have at least two separate input codes.

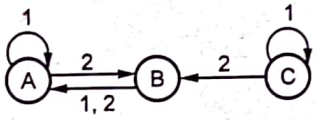
The outputs of two state graphs with the same state, inputs and transitions might be different. However, from the standpoint of state-testing, they may be equivalent.

Unreachable states can be found in Bad State graphs. It is impossible to reach any state from any other state, and it is also impossible to reach the start state from any other state.

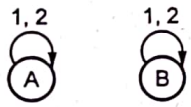
The diagrams that follow are examples of incorrect state graphs.



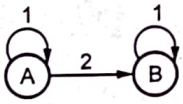
(a) State B can never be left. The initial state can never be entered again.



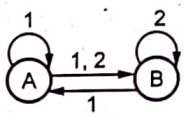
(b) State C cannot be entered



(c) State A and B are not reachable.



(d) No transition is specified for an input of 2 when in state B.



(e) Two transitions are specified for an input of 1 in state A.

Fig. Q.3.1 Improper state graphs

Q.4 Explain state bugs- in detail.

☞ [JNTU : Sept.-21, Marks 8, March-22, Marks 7]

Ans. : 1. Number of states :

The number of states we select to recognize or model is the number of states in a state graph.

The state is recorded either directly or indirectly as a set of values for variables in the database.

The state might, for example, be made up of the value of a counter with possible values ranging from 0 to 9, as well as the setting of two bit flags, resulting in a total of 2 * 2 * 10 = 40 states.

The following formula can be used to calculate the number of states:

Identify all of the state's constituent factors.

- List all of the possible values for each factor.
- The number of states equals the product of all the components' allowed values.

Before we do anything else, before we think of one test case, talk about how many states we think there are against how many states the programmer thinks there are.

It's pointless to write tests to examine the system's behavior in different states if there's no agreement on how many states there are.

Impossible States: The gap between the programmer's and tester's state counts is sometimes due to a disagreement about what constitutes "impossible states."

When an impossible state appears to have occurred, a robust piece of software will recognize it and activate an illogical condition handler.

2. Equivalence states :

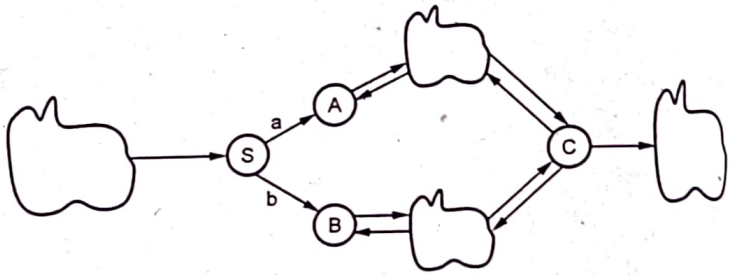


Fig. Q.4.1

- If every sequence of inputs starting from one state produces exactly the same sequence of outputs when begun from the other state, the two states are equivalent. This concept can also be used to a collection of states.

Merging of equivalent states :

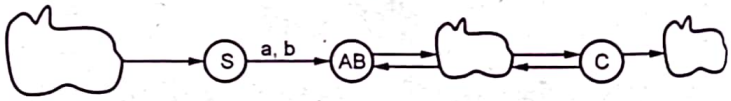


Fig. Q.4.2

3. Recognizing equivalent states :

The following approaches can be used to identify equivalent states :

1. In terms of input/output/next state, the rows corresponding to the two states are identical, but the name of the next state may differ.

2. There are two sets of rows with identical state graphs in terms of transitions and outputs, except for the state names. Both sets can be combined.

4. Transition bugs :

Unspecified and contradictory transitions :

- A transition must be defined for each input-state combination.
- If the transition is impossible, a mechanism must be in place to prevent the input from reaching that state.
- For each input and state combination, only one transition must be given.
- There can't be any inconsistencies or ambiguities in software.
- Ambiguities are impossible because the software will respond to each input in some way.
- Even though the state does not change, this is a transition to the same state by definition.

5. Unreachable states :

- Unreachable states are similar to unreachable code.
- No input sequence can reach this state.
- It's not impossible to have an inaccessible state, just as it's not impossible to have unreachable code.
- There may be transitions from one unreachable state to another; this is usually because the state becomes unreachable due to an inappropriate transition.
- Unreachable states can take one of two forms :

There is a flaw in the program; some transitions are missing.

o Transitions are present, but we are unaware of them.

Dead states :

- A dead state is one that cannot be exited once entered.
- This isn't necessarily a bug, but it's a suspicious.

Output errors :

The states, transitions, and inputs may all be valid, and no dead or inaccessible states may exist, but the transition's output may be erroneous.

Independent of states and transitions, output activities must be confirmed.

Testing by state :

- If a routine is specified as a state graph, it has been thoroughly checked for accuracy.
- It is still necessary to implement program code, tables, or a combination of both.

One or more of the following symptoms may indicate the presence of a bug :

- The number of states is incorrect.
- For a given state-input combination, incorrect transitions.
- For a specific transition, the output is incorrect.
- Inadvertently formed equivalent pairs of states or sets of states.
- Splitting a state or a set of states into equivalent duplicates.
- States or groups of states that have ceased to exist.
- States or groups of states that are no longer accessible.

Q.5 Explain software implementation of state graphs. .

☞ **[JNTU : Part B, Dec.-18, Marks 5, March-22, Marks 8]**

Ans. :

- A state graph is a form of diagram used to represent the behavior of a system in computer science and related subjects. State Graphs necessitate that the system being described has a finite number of states, which is sometimes the case and other times is a valid abstraction.
- A state diagram is a graphical depiction of a program's states. inputs, outputs, and transitions
- In a state network, nodes represent states.
- States are numbered or identified using words.
- The total number of states can be calculated by counting the number of options available.

Software implementation :

There are 4 tables involved :

1. A table or process that encodes the input value into a compact list (INPUT_CODE_TABLE)

2. A Table that specifies the next state for every combination of state and input code(TRANSITION_TABLE).
3. A table or case statement that specifies the output or output code. if, any, associated with every state combination(OUTPUT_TABLE)
4. A table that stores the present state of every device or process that uses the same state table – eg one entry per tape transport(DEVICE_TABLE).

The routine operates as follows, where # means concatenation :

```
BEGIN
    PRESENT_ STATE:=DEVICE_TABLE(DEVICE_NAME)
    ACCEPT INPUT_VALUE
    INPUT_CODE:=INPUT_CODE#PRESENT STTAE
    NEW_STATE:=TRANSITION_TABLE(POINTER)
    OUTPUT_CODE:=OUTPUT_TABLE(POINTER)
    CALL OUTPUT_HANDLER(OUTPUT_CODE)
    DEVICE_TABLE(DEVICE_NAME):=NEW_STATE
END
```

If a routine is specified as a state graph, it has been thoroughly checked for accuracy. It is still necessary to implement program code, tables, or a combination of both.

Q.6 What is impossible state ?

☞ [JNTU : Part A, Dec.-18, Marks 2]

Ans. :

- Certain combinations of circumstances may appear to be unattainable at times.
- The mismatch between the programmer's and tester's state counts is frequently due to differing viewpoints on "impossible states."
- A reliable piece of software will not disregard impossible states; instead, it will recognize them and call an illogical condition handler when they exist.

Q.7 Explain bugs in state graphs.

Ans. : When evaluating a state graph implementation, look for the following common defects (incorrect sequences of events, transitions, or actions) :

- Transition isn't stated (nothing happens with an event)
- Inappropriate transition (the resultant state is incorrect)
- An unspecified or erroneous event has occurred.
- Incorrect or unspecified action (wrong things happen as a result of a transition)
- Unspecified, erroneous, or corrupt status
- Unreachable state (A state that can't be reached by any input sequence)
- State of death (A state that once entered cannot be exited)
- Path of least resistance (an event is accepted when it should not be)
- Door with a trap (the implementation accepts undefined events)

Q.8 Write testers comments about state graphs.

☞ [JNTU : Part B, April-18, Marks 5]

Ans. : The state graph test cases are a series of input events. Exhaustive test case design to perform every path at least once is difficult and impractical in state graphs. As a result, the following concepts or guidelines can be used to construct test cases using the state graph model :

- When each state of the state graph is exercised at least once during testing, all states are covered. Because behavior flaws are only discovered by chance, this is frequently insufficient coverage. Even though all states are covered, if there is a defect in a transition between a specific state pair, it may go unnoticed.
- All-events coverage : Each state graph event is covered by the test suite (and is part of at least one test case).
- All-actions coverage : Each action is conducted at least once.
- All-transitions coverage : When a test executes every transition in the model at least once, it is said to have achieved all-transitions coverage. This immediately includes coverage in all 50 states. It is not necessary to follow any certain order to cover all transitions.

- Coverage of all transitions : Coverage of all transitions is achieved when the test runs through the model at least once. This immediately includes coverage in all 50 states. It is not necessary to run any specific sequence in order to cover all transitions; all transitions must be executed once. Even at this coverage level, a defect that appears only when a specified sequence of transitions is run gets missed. The coverage can be extended by mandating All n-transition coverage, which means that the test suite must contain all conceivable transition sequences of n or more transitions.
- All n-transition sequences :
- All transitions equal all 1-transition sequences
- All n-transition sequences implies (subsumes) all (n-1)-transition sequences
- All n-transition sequences implies (subsumes) all (n-1)-transition sequences
- All round-trip paths are tested at least once: every sequence of transitions that starts and ends in the same state is tested at least once.

Q.9 What are the roles of state graphs in software testing ?

Ans. : Before beginning the actual implementation phase, a state graph is executed or simulated with event sequences as test cases, providing a framework for model testing. State graphs define system specifications and provide facilities for comparing system implementations to the specifications. State testing is largely a functional testing technique that works well in the early stages of development. We may create test cases using state graphs by following the steps below :

- The elements of the state graph (states, events, actions, transitions, and guards) must be explicitly mapped to the elements of the implementation (e.g., classes, objects, attributes, messages, methods, expressions)
- Ascertain that the present state of the state graph underpinning the implementation can be verified, either by the runtime environment or by the implementation itself (built-in tests with, e.g., assertions and class invariants)

Q.10 What is state transition testing.

Ans. : State Transition testing is a black box testing technique in which changes in the input conditions or the system's' state' activates outputs. To put it another way, tests are created to carry out both valid and invalid state changes.

When to use ?

- When a series of events occurs, as well as the criteria that apply to those events
- When the right treatment of a single incident is contingent on previous events and circumstances
- It's employed in real-time systems that have several states and transitions.

Deriving test cases :

- Recognize the various states and transitions and identify which are legitimate and which are invalid.
- Defining an event sequence that leads to a test ending state that is authorized.
- Each state that has been visited and each transition that has been traversed should be recorded.
- Steps 2 and 3 should be performed until all states and transitions have been explored.
- Actual input values and output values must be generated for test cases to have appropriate coverage.

Advantages :

- Allows testers to become familiar with the software design and create successful tests.
- It also allows testers to account for unanticipated or invalid situations.

Example :

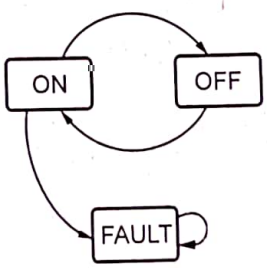


Fig. Q.10.1

The tests are developed from the above state and transition, and the scenarios that need to be evaluated are listed below.

Tests	Test 1	Test 2	Test 3
Start State	Off	On	On
Input	Switch ON	Switch Off	Switch off
Output	Light ON	Light Off	Fault
Finish State	ON	OFF	On

Q.11 Write a short note on static testing.

Ans. : Static testing is a type of software testing in which the software is tested without having to run the code. It is divided into two sections, as follows :

1. **Review -** A technique for identifying and correcting flaws or ambiguities in documents like as requirements, designs, and test cases.

Types of reviews :

A simple diagram can show the many types of reviews :

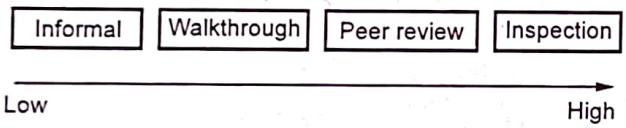


Fig. Q.11.1

2. **Static analysis -** Developers' code is examined (typically using tools) for structural flaws that could lead to faults.

Static analysis - By tools :

The sorts of flaws discovered by the tools during static analysis are as follows :

- An undefined variable is one that has no value.
- Module and component interfaces are inconsistent.
- Declared variables that are never used
- Unreachable code (sometimes known as "dead code") is a type of code that cannot be accessed.
- Standards violations in programming
- Vulnerabilities in security
- Inconsistencies in syntax

Q.12 What is Statistical Testing (ST) and how does it work ?

Ans. : Statistical testing use statistical approaches to verify the program's reliability. Statistical testing examines how defective programs can affect the system's operation.

What is the best way to do ST ?

Software is evaluated using test data that simulates the working environment statistically.

Failures are gathered and examined.

An estimate of the program's failure rate is calculated using the computed data.

By constructing an algebraic function, a statistical approach for assessing the potential paths is computed.

Because the goal of statistical testing isn't to detect problems, it's a bootless activity.

State Table can be used to identify erroneous system transitions. All valid states are displayed on the left side of a state table, and the events that cause them are shown on the top. When the corresponding event occurs, each cell represents the state to which the system will transition.

If we enter a proper password while in state S1 :

For example, we will be moved to state S6 (Access Granted). If we give input the erroneous password on the first try, we will be directed to state S3 or 2nd Try. We can also determine all other states. This technique highlights two invalid states, which is what happens when we are already signed into the application and open a new instance of flight reservation, entering valid or invalid credentials for the same agent.

Q.13 What are some situations in which state testing may prove useful ? Explain.

☞ [JNTU : Part B, Dec.-18, Marks 5]

OR Explain state testing in detail.

☞ [JNTU : Part B, Dec.-19, Marks 5]

OR What are the principles of state testing. Discuss advantages and disadvantages.

☞ [JNTU : Part B, May-19, Marks 5]

Ans. : The usage of finite state machine models for software structure, software behavior, or specifications of software behavior is the foundation of state testing methodologies.

Principles of State Testing

- The state testing strategy is similar to the flow graph path testing strategy.
- It's unrealistic to walk through every path in a state graph, just as it's impractical to go through every potential path in a flow graph.
- The concept of coverage is the same as that of flow graphs.
- Despite the fact that more state testing is done as a single case in a grand tour, it is impractical to do so for a variety of reasons.
- Bugs will prevent us from completing the grand tour during the early stages of testing.
- Later, when it comes to maintenance, the testing objectives are clear, and just a few states and transitions need to be tested.
- It's a waste of time to go on a large tour.
- In a long test sequence, there isn't much history, and so much has transpired that verification is tough.

Starting point of state testing :

- Define a set of covering input sequences that, when started from the beginning, return to the initial state.
- Define the expected next state, the expected transition, and the expected output code for each step in each input sequence.
- A set of tests, then, consists of three sets of sequences :
  1. Input sequences
  2. Corresponding transitions or next-state names
  3. Output sequences

The fundamental benefit of this testing technique is that it provides a visual or tabular depiction of system activity, allowing the tester to quickly cover and comprehend system behavior. The biggest downside of this testing method is that it cannot be relied upon 100 % of the time.

Limitations and Extensions

- In a state graph model, state transition coverage does not guarantee thorough testing.
- How to create covers for state graphs by defining a hierarchy of paths and methods for merging paths.
- The simplest is a "0 switch," which equates to individually checking each transition.
- The next level involves testing transition sequences made up of two "1 switch" transitions.
- When there are n states, the maximum length switch is "n-1 switch."

Situations where state testing comes in handy :

- Any processing in which the output is determined by the occurrence of one or more sequences of events, such as the detection of defined input sequences, sequential format validation, parsing, and other circumstances in which the order of inputs is critical.
- The majority of protocols exist between systems, between humans and machines, and between system components.
- Device drivers with intricate retry and recovery techniques, such as those for tapes and discs, if the action is dependent on the state.
- Whenever one or more state transition tables are directly and openly implemented as a feature.

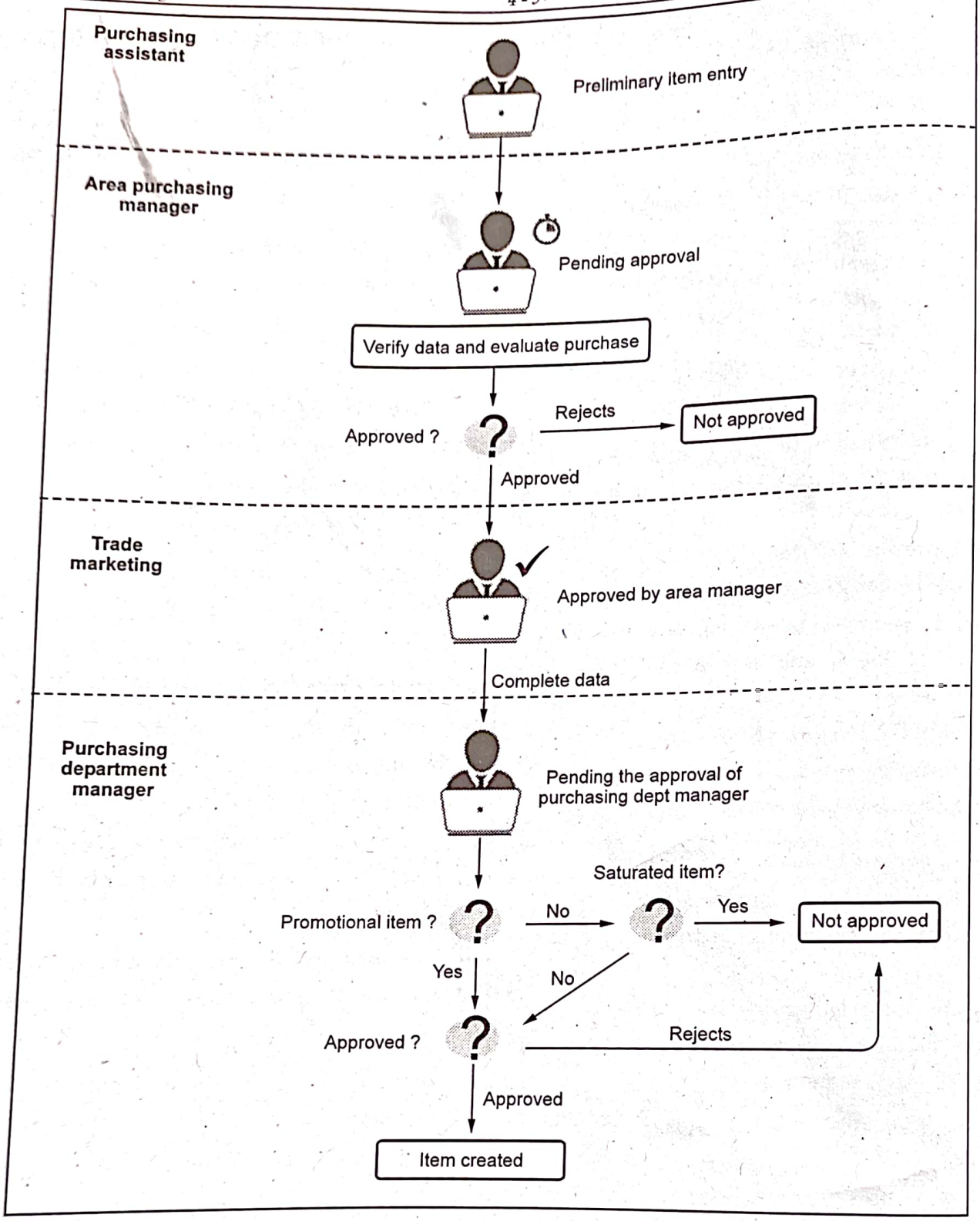


Fig. Q.14.1

Q.14 Write the guidelines to design state machines.

☞ [JNTU : Part B, Dec.-19, Marks 5]

Ans. :

- State machines are models of a system's or an element of a system's behavior. They're depicted as graphs, with nodes representing states and arrows representing transitions between them.
- State machines can be used to create test cases that cover a system based on a set of criteria. The following are some of the coverage criteria :

State coverage : When the test cases cover all of the states, this requirement is met.

Transition coverage : The test cases cover all transitions.

Transition pair coverage : All combinations of input and output transitions are covered for each state.

Now let's look at a more complex scenario that is more representative of what we may encounter when evaluating a system.

- We'll demonstrate in this example that we're not limited to a single representation of state machines, and that this technique can be used manually.
- Assume we're evaluating a system for managing inventory items. The goal is to create test cases for item creation that take into account the complete item's life cycle as well as the many persons that must execute various actions and validations on the data provided to the system.
- This is an excellent example of a scenario in which we might use a state machine to represent the behavior of the objects and build test cases from it. The things may be in many states, and the intended behavior for each action or system function will be determined by the item's state.

A partial state machine for the concept, item, is shown in the Fig. Q.14.1. The task's actor may be seen to the left of the picture, and in this case, the guards are represented as decision points in the figure. (See Fig. Q.14.1 on previous page)

The states determine how items behave in different situations. The transitions are system features that cause the item's behavior, or state, to change.

Q.15 Explain testability tips in detail.

☞ **[JNTU : March-22, Marks 7]**

Ans. : Testability tips :

We need to create explicit state diagrams in order to design testability. Furthermore, if the State Graph is created with only two states, testability is simple.

Additionally, programmers need put out considerable effort in determining which types of behaviors should be considered when generating state graphs and which behaviors should be discarded. If programmers ignore this and create a state graph that is easy to work with, the purpose of using state graphs as the foundation for model based testing is defeated, because model based testing requires identifying relevant states, inputs, and transitions while ignoring irrelevant states, inputs, and transitions with a specific rationale. A state graph is a type of functional testing that is more complex. A state graph is a more advanced kind of functional testing.

State graph concepts assist us in creating a state graph model from specifications in order to do model-based testing. It is critical that a state graph be good. To guarantee that the state graph we arrived at is good, it will be checked to see if the state graph modeled from specifications is valid, comprehensive, and consistent enough for testing to be implemented. Exhaustive testing is not possible with state graph-based testing. Furthermore, designing state graphs for complicated and larger systems is quite difficult. As a result, state graphs are not suitable candidates for developing exhaustive testing, but they should be taken into account when designing test cases for sophisticated and larger systems.

Software testability is a non-functional criteria that indicates how simple it is to test the software. It should be included in software to ensure that test cases and test scripts are completely executed. "The extent to which a software system or its component enables the establishment of test criteria and the execution of tests to determine whether those criteria have been met or not," according to the definition.

"Low Software Testability" degrades software slowly in many situations because it may not be detected immediately; testers, unaware of this, may recommend a variety of solutions to address the problem, such as extending work hours, assigning more resources, expensive automation tools, risk-based testing, the need for better estimation and planning, and so on, without fully understanding the problem. This aggravates the situation because the true issue will go unnoticed.

Because every SDLC includes requirements collecting, analysis, design, coding, testing, implementation, and maintenance, software testability is a requirement for software development. Only if the application being developed is considerably testable can complete execution of the test scripts be guaranteed. When adequate test coverage is used, the majority of faults will be discovered and addressed before the product goes live in the market, resulting in fewer issues being reported by end users.

Q.16 What is Petri net ?

☞ **[JNTU : Part A Dec.-18, Marks 2]**

Ans. : Petri nets make use of operations that can contain and distinguish between all of the variations, as well as tools for explicitly describing tokens that move through stages in the process. Petri nets have a well-developed theory (MURA89, PETE81). Petri nets have been used to solve difficulties in hardware testing, protocol testing, network testing, and other fields, but they are still in their infancy when it comes to generic software testing. We also don't have enough experience with Petri nets in software testing to say whether they're a good model for general software testing.

Fill in the Blanks for Mid Term Exam

Q.1 A tabular representation of a state graph is referred to as _____.

Q.2 In the start state of the finite state machine, computing begins with _____.

Q.3 _____ is the process of turning characters to numbers.

Q.4 The collection of different input values is referred to as _____.

Q.5 Impossible states increase the number of states, which raises the _____.

Q.6 Two states are known as _____ if every series of inputs from one state creates exactly the same sequence of outputs for other states.

Q.7 The bugs that appear as a result of being in a dead condition are called as _____.

Q.8 Under all conditions, FSM behaves in the _____.

Q.9 In terms of _____, there is a distinction between FSMs and combinational machines.

Q.10 Combinational program implementation is based on _____.

Q.11 Finite state machines are tested in all conceivable states using _____.

Q.12 _____ modifies the machine's state..

Q.13 A _____ is the condition that describes the system's behavior.

Q.14 In state tables, each input condition of the state graph is defined in _____.

Q.15 The feature of _____ is that "for each unique combination of state and input, only one transition need be stated."

Multiple Choice Questions for Mid Term Exam

Q.1 The state graph's elements do not include _____.

[a] states [b] input/output

[c] transitions [d] State machines

Q.2 The first step in implementing the state graph is to _____.

[a] implementation and operation

[b] input encoding

[c] output encoding

[d] state codes

Q.3 Bugs in state graphs are caused by _____.

[a] no. of states [b] impossible states

[c] equivalent states [d] all of the above

Q.4 The state in which no input sequence may be reached is referred to as _____.

[a] dead state [b] reachable state

[c] unreachable state [d] ambiguous State

Q.5 The outputs of a state graph are independent on _____.

[a] transitions [b] inputs

[c] both a and b [d] none

Q.6 The bugs that aren't caused by state graphs are _____.

[a] state bugs [b] output errors

[c] encoding bugs [d] none

Q.7 The benefit of state testing is that it does not contain _____.

[a] less expensive

[b] cannot catch deep bugs

[c] provides rewards during rest design

[d] is used to detect specified input

Q.8 The state graph depicts _____.

[a] time [b] sequence

[c] states [d] none

Q.9 A situation in which the "engine is not working yet the vehicle is running" is an illustration of _____.

[a] reachable state [b] impossible state

[c] equivalent state [d] contradictory state

Q.10 The steps for determining the number of states do not include _____.

[a] identifying all component factors

[b] identifying all allowable factors

[c] no. of states = product of no of allowable values of all factors

[d] No. of states = sum of all allowable values of all factors

Q.11 Is the below state graph good ?

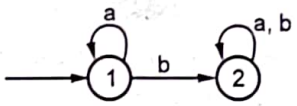


Fig. 1

[a] Yes [b] No

[c] Cannot be determined [d] Both a and b

Q.12 In the situation of _____, a contradictory and ambiguous state occurs.

[a] Transaction bugs [b] Encoding bugs

[c] Output errors [d] State bugs

Q.13 Which of the following is not an inessential criterion for finite state behaviour ?

[a] It can be obtained by parallel program in dataflow machine

[b] It is obtained from decision table or tree.

[c] It is obtained from flow graph.

[d] Expression of exit =1

Q.14 In the case of a good state graph, _____.

[a] bugs are easy to find

[b] sequence of inputs does not lead to initial state

[c] bugs are more

[d] both a and b

Q.15 In a graph, the outputs and inputs are divided by _____.

[a] *(asterisk) [b] /(slash)

[c] -(eiphen) [d] none

Answer Keys for Fill in the Blanks :

Q.1	state table	Q.2	input string
Q.3	Encoding	Q.4	input alphabet
Q.5	test cases	Q.6	equivalent sets
Q.7	transition bugs	Q.8	encoding
Q.9	quality	Q.10	decision table
Q.11	switches and flags	Q.12	Transition function
Q.13	state	Q.14	columns.
Q.15	good state graphs		

Answer Keys for Multiple Choice Questions :

Q.1	d	Q.2	a	Q.3	d
Q.4	c	Q.5	c	Q.6	d
Q.7	b	Q.8	b	Q.9	b
Q.10	d	Q.11	b	Q.12	a
Q.13	c	Q.14	a	Q.15	b

END...

UNIT - V

5 Graph Matrices and Application

5.1 Motivational Overview, Matrix of Graph, Relations, Power of a Matrix,

Q.1 Define problem with pictorial graphs and tool building.

OR Explain motivational overview of graph matrices.

☞ [JNTU : March-22, Marks 7]

Ans. : Problem with pictorial graphs :

- Graphs were first introduced as a way to abstract software structure.
- We trace graphs across a graph to determine a set of covering paths, a set of values that would sensitize paths, the logic function that regulates the flow, the processing time of the routine, the equations that define the domain, or whether a state is reachable or not whenever a graph is used as a model.
- It is not a simple path to follow, and it is prone to error. We may occasionally miss a link or cover certain links twice.
- To solve this challenge, one option is to represent the graph as a matrix and utilize matrix operations that are analogous to path tracing. These approaches are more deliberate and mechanical, and they are more trustworthy since they do not rely on our ability to visualize a path.

Tool building :

- If we are making test tools or just curious about how they function, we will be implementing or investigating analysis routines based on these methods sooner or later.
- Because it's difficult to design algorithms over visual graphs, the attributes or graph matrices are essential for tool development.

Q.2 How can the graph be represented in matrix form ?

☞ [JNTU : Part A, April-18, Marks 2]

Ans. : For each node in the graph, a graph matrix is a square array with one row and one column. Each row-column combination represents a relationship between the row's node and the column's node. If there is a link between the nodes, the relation could be as basic as the link name.

Some of the things to be observed :

- The number of nodes equals the size of the matrix.
- There is a location for each and every direct connection or link between any two nodes.
- The link weight of the connection that connects the two nodes in that direction is the entry at a row and column junction.
- A node i to j link does not imply a node j to node i connection.
- If there are many links between two nodes, the entry is a sum; the "+" symbol, as expected, denotes parallel connectivity.

Some graphs and their matrics :

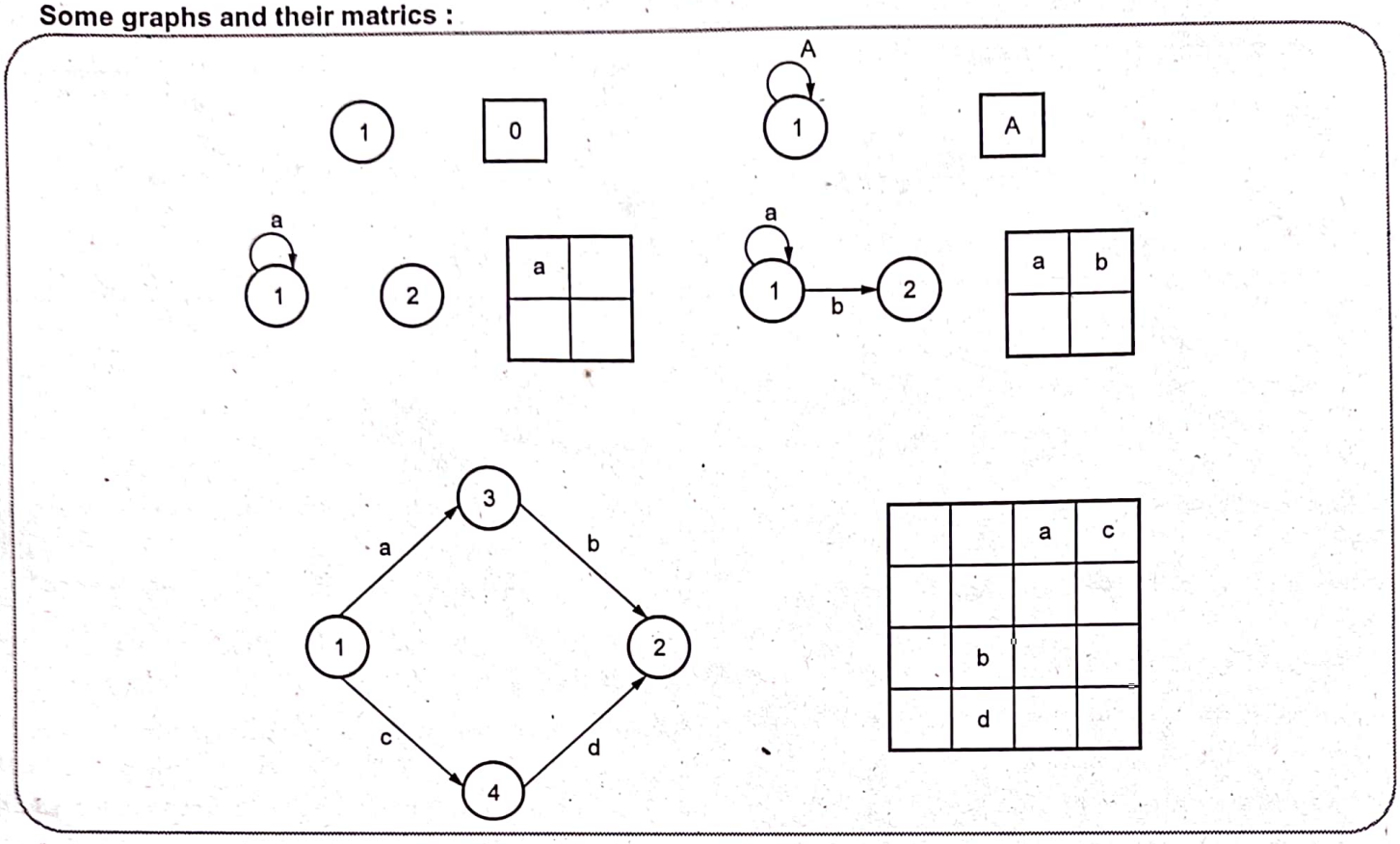


Fig. Q.2.1

5.2 Node Reduction Algorithm, Building Tools

Q.3 How can the graph be represented in matrix form, give suitable example ?

Ans. : Refer Q.2.

Let's have a look at an example.

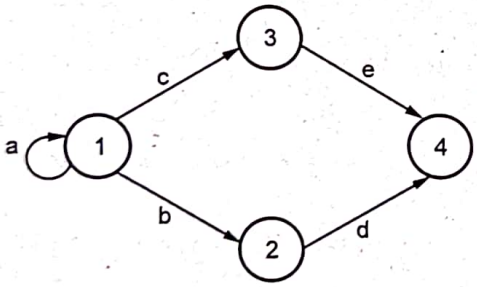


Fig. Q.3.1 Graph

Let's make a graph matrix out of this control flow graph. Because there are four nodes in the graph, the graph matrix will be 4 × 4. The following Table Q.3.1 shows how the matrix entries will be filled :

- Because there is an edge from node 1 to node 1, (1, 1) will be filled with 'a'.
- Because there is an edge from node 1 to node 2, (1, 2) will be filled with 'b'.
- It's worth noting that (2, 1) will not be filled since the edge is unidirectional rather than bidirectional; however, (1, 3) will be filled with 'c' because edge c runs from node 1 to node 3.
- Because there is an edge from node 2 to node 4, (2, 4) will be filled with 'd'.
- Because there is an edge from node 3 to node 4, (3, 4) will be filled with 'e'.
- The resulting graph matrix is as follows :

	1	2	3	4
1	a	b	c	
2				d
3				e
4				

Table Q.3.1 Graph matrix

Q.4 What do you mean by a simple weight ? Explain connection matrix in detail with suitable example.

Ans. : The simplest weight we can use is to note whether or not there is a link. Let "1" denote a connection and "0" denote the absence of one.

The following are the arithmetic rules :

$$1+1 = 1$$
$$1*1 = 1$$
$$1+0 = 1$$
$$1*0 = 0$$
$$0+0 = 0$$
$$0*0 = 0$$

A connection matrix is a matrix with weights defined in this way.

- To obtain the connection matrix, replace each entry with 1 if there is a link and 0 if there isn't.
- To keep things simple, we don't put down 0 entries.

		a	c
	b		
	d		

		1	1
	1		
	1		

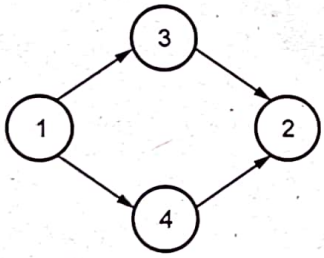


Fig. Q.4.1

- The out links of the node corresponding to that row are represented by each row of a matrix.
- Each column represents the number of inbound links for that node.
- A node with more than one nonzero element in its row is referred to as a branch.
- A junction is a node that has more than one nonzero column entry. A diagonal entry is known as a self loop.
- As an example, if we describe the above control flow graph as a connection matrix, we get :

	1	2	3	4
1	1	1	1	
2				1
3				1
4				

Table Q.4.1 Connection matrix of Table Q.3.1

- As can be seen, the edges' weights are simply replaced by 1 and the cells that were previously empty are left vacant, i.e., signifying 0.
- The cyclomatic complexity of the control graph is determined using a connection matrix.
- Although there are three alternative approaches for determining cyclomatic complexity, this method is as effective.

Q.5 Explain cyclomatic complexity with example.

Ans. : By removing 1 from the total number of entries in each row and omitting rows with no entries, we can calculate the equivalent number of decisions for each row. The graph's cyclomatic complexity is calculated by adding these numbers and then adding 1 to the total.

$$2-1 = 1$$
$$1+1 = 2 \text{ (cyclomatic complexity)}$$
$$1-1 = 0$$
$$1-1 = 0$$

- The cyclomatic complexity of the control graph is determined using a connection matrix.
- Although there are three alternative approaches for determining cyclomatic complexity, this method is as effective.
- The steps for calculating cyclomatic complexity are as follows :

1. Count the number of 1s in each row and record it at the conclusion.
2. For each row, subtract 1 from the total (Ignore the row if its count is 0)
3. Add each row's count from the previous step.
4. Increase the total by one.

5. The cyclomatic complexity of the control flow graph is the result of step 4's final total.

- Let's use these steps to calculate the cyclomatic complexity of the graph 1.

	1	2	3	4	
1	1	1	1		3 – 1 = 2
2				1	1 – 1 = 0
3				1	1 – 1 = 0
4					Ignore
Cyclomatic complexity = 2 + 0 + 0 + 1 = 3					

- Other approaches can be used to verify this result for cyclomatic complexity :

Method - 1 :

Cyclomatic complexity

$= e - n + 2 * P$

Since here,

$e = 5$

$n = 4$

and, $P = 1$

Therefore, cyclomatic complexity,

$= 5 - 4 + 2 * 1$

$= 3$

Method - 2 :

Cyclomatic complexity

$= d + P$

Here,

$d = 2$

and, $P = 1$

Therefore, cyclomatic complexity,

$= 2 + 1$

$= 3$

Method - 3 :

Cyclomatic complexity

= Number of regions in the graph

Region 1 is bounded by the edges b, c, d, and e;

Region 2 is bounded by the edge a (in loop); and

Region 3 is outside the graph.

As a result of cyclomatic complexity,

$= 1 + 1 + 1$

$= 3$

Q.6 Define transpose of a matrix.

Ans. : The transpose of a matrix is one in which the rows and columns have been swapped. A superscript letter "T," as in A^T, is used to indicate it. When C = A^T, $c_{ij} = a_{ji}$. The intersection of two matrices of the same size, indicated by A#B, is a matrix created by multiplying the entries element by element. C = A#B, for example, indicates $c_{ij} = a_{ij} \# b_{ij}$. Normally, Boolean AND or set intersection is used for multiplication. Similarly, the element-by-element addition operation, such as a Boolean OR or set union, is defined as the union of two matrices.

Q.7 What are properties of relations ? Explain.

☞ **[JNTU : Part B, Dec.-18, Marks 5]**

Ans. : A property that exists between two (typically) items of interest is referred to as a relation.

- Here's an example, with a and b representing objects and R indicating that a has the relation R to b :

1. "Node a is linked to node b," or aRb, where "R" stands for "is linked to."
2. "a > = b" or "aRb," with "R" standing for "more than or equal."
3. "a is a subset of b," where "is a subset of" is the relation.
4. "To get from node a to node b, it takes 20 microseconds of processing time." The number 20 represents the relationship.
5. "node a defines data item X, which is used by node b." The relationship between nodes a and b is that they are connected by a du chain.

- A graph is made up of nodes, which are abstract objects and R, which is a relation between them. A link from a to b is denoted as aRb, which means that a has the relation R to b. We can connect one or more attributes with some relations in addition to the fact that they exist. These are referred to as link weights. The weight of a connection can be numerical, logical, illogical, objective, subjective or anything else. Furthermore, the number and type of link weights that can be associated with a relation are limitless.
- The simplest relation is "is related to," which is represented by an un-weighted link. Connection matrices are graphs defined over "is connected to." A relation matrix is used for more general relationships.

Properties of relations:

1. **Transitive relations :** If aRb and bRc imply aRc, then the relation R is transitive. The majority of test relations are transitive. Is related to, is more than or equal to, is less than or equal to, is a relative of, is quicker than, is slower than, takes longer than, is a subset of, contains, shadows, and is the boss of are examples of transitive relations. Is familiar with, is a friend of, is a neighbor of, is lied to, and has a du chain between are examples of intransitive relations.
2. **Reflexive relations :** If for every a, aRa, a relation R is reflexive. At each node, a reflexive relation is equivalent to a self-loop. Equals, is familiar with (save potentially for amnesiacs), and is a relative of are examples of reflexive relations. Not equals, is a buddy of, is on top of, and is under are examples of irreflexive relationships.
3. **Symmetric relations :** If aRb entails bRa for every a and b, then the relation R is symmetric. A symmetric relation means that if there is a connection from a to b, there is likewise a link from b to a; this also means that arrows can be removed and the pair of links replaced with a single undirected link. Because we must employ arrows to denote the direction of a relation in a graph with non-symmetric relations, we call it a directed graph. An undirected graph is a graph that is based on a symmetric relation. An undirected graph's matrix is symmetric ($a_{ij} = a_{ji}$ for every i j).
   - Asymmetric relations include the following : is the boss of, is greater than, controls, dominates and can be reached from.
4. **Antisymmetric relations :** If for every a and b, if aRb and bRa, then a = b or they are the same elements, then the relation R is antisymmetric.
   - Antisymmetric relationships include : is more than or equal to, is a subset of, and time. Non-antisymmetric relations include those that are connected to, can be reached from, are greater than, are a relative of, and are a friend of.
5. **Equivalence relations :** A relation that satisfies the reflexive, transitive, and symmetric qualities is called an equivalence relation. The most well-known example of an equivalence relation is numerical equality. We say that a set of items constitute an equivalence class over an equivalence relation if they satisfy it. The significance of equivalence classes and relations is that each member of the equivalence class is equivalent to any other member of that class with regard to the relation. We can partition the input space into equivalence classes using partition-testing methodologies like domain testing and path testing. If we can accomplish that, testing any member of the equivalence class is as good as testing all of them. When we say that testing one set of input values for each member of a branch-covering set of routes is sufficient, we're implying that the set of all input values for each path (e.g., the path's domain) is an equivalence class with respect to the relation that defines branch-testing paths.
6. **Partial ordering relations :** The reflexive, transitive and antisymmetric qualities are all satisfied by a partial ordering relation. Partial ordered graphs

contain several significant properties : They are loop-free, have at least one maximum and minimum element and the resulting graph is also partly ordered if all the arrows are reversed. The relation xRa does not hold for any other element x for a maximum element a. A minimum element a, on the other hand, is one for which the relation aRx does not hold for any other element x. Partial ordering can be seen in trees. Partial ordering is important because, whereas rigid ordering (as in numbers) is uncommon with graphs, partial ordering is prevalent. Partially ordered loop-free graphs. Call trees, most data structures and an integration plan are all instances of useful partly ordered graphs. Furthermore, while the basic control-flow or data-flow graph is not always ordered.

Q.8 Explain the powers of a matrix in detail.

OR Discuss the algorithm for finding set of all paths.

Ans. :

- Each element in the matrix of the graph expresses a relationship between the nodes that correspond to that entry.
- Under the premise that the relation is transitive, squaring the matrix generates a new matrix that describes the relation between each pair of nodes via one intermediate node.
- All path segments with two links are represented by the square of the matrix and
- All path segments with three links are represented by the third power.
- Let A be a matrix with a_{ij} entries. The collection of all potential paths between any node i and any other node j (perhaps i itself) via all possible intermediary nodes is given by,
- As intimidating as this term may look, it only expresses the following :

1. Think about how each node interacts with its neighbors.
2. Extend that relationship by treating each neighbor as a node in the middle.
3. Consider each neighbor's neighbor as an intermediate node to extend the network even farther.
4. Keep going until we have found the longest non-repeating path possible.
5. Repeat steps 3 to 5 for each pair of nodes in the graph.

1. Matrix powers and products :

- The square of a matrix whose elements are a_{ij} is generated by substituting each entry with :
- n
- $a_{ij} = \sum_{k=1}^{n} a_{ik}\, a_{kj}$
- k = 1

In general, the product of two matrices A and B with entries a_{ik} and b_{kj}, respectively, is a new matrix C with entries c_{ij}, where :

- n
- $c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$
- k = 1

$A^2A = AA^2$

that is, if the underlying relation arithmetic is associative, matrix multiplication is associative (for most interesting relations). As a result, A^4 can be obtained in the following ways :

A^2A^2, $(A^2)^2$, A^3A, AA^3

Because multiplication is not always commutative, we must remember to put the left-hand matrix's contribution in front of the right-hand matrix's part to avoid mistakenly reversing the order. The terminology used in the loop is crucial. This is a list of terms that appear on the main diagonal (the one that slants down to the right).

2. The set of all paths : Using the right arithmetic rules for such weights, we want to get the set of all paths between all nodes or, equivalently, a property (defined by link weights) over the set of all paths from every node to every other node using matrix operations. Matrix operations can be used to express the set of all paths between all nodes. It's determined by an infinite succession of matrix powers.

- This is a beautiful but practically worthless phrase. Let I be a n by n matrix, with n being the number of nodes in the matrix. Let's say i's entries are all multiplicative identity elements on the primary diagonal. This can be the number "1" for link names. It is the multiplicative identity for those weights for other types of weights. The following product can be renamed :

$A(I + A + A^2 + A^3 + A^4 \dots A^\infty)$

$A + A = A$, $(A + 1)^2 = A^2 + A + A + IA^2 + A + I$. Furthermore, for any n that is finite,

$(A + I)^n = I + A + A^2 + A^3 \dots A^n$

As a result, the original infinite sum can be substituted with

$$\sum_{i=1}^{\infty} A^i = A(A + I)^\infty$$

- This is an improvement because the prior statement included both infinite products and infinite sums, whereas now we only have to deal with one infinite product. Whether or not there exist loops, the preceding holds true. If we limit our attention for the time being to paths with length n – 1, where n is the number of nodes, the set of all possible paths is given by

$$\sum_{i=1}^{n-1} A^i = A(A + I)^{n-2}$$

- With n nodes, no path can surpass n – 1 node without incorporating some path segment that is already incorporated in another path or path segment. It's a little easier to find the set of all such paths because all of the intermediate products don't have to be done explicitly. The following algorithm works well :

1. As a binary number, write n – 2.
2. Make a series of (A + I) squares to get to $(A + I)^2$, $(A + I)^4$, $(A + I)^8$ and so on.
3. Only keep the binary powers of (A + 1) that correspond to a 1 in the binary representation of n– 2.
4. Take the product of the matrices we received in step 3 and the original matrix to get the set of all paths of length n – 1 or less.

Let's say there are 16 nodes in the network. The set of all paths with a length of less than or equal to 15 is what we're looking for. n – 2 (14) has a binary representation.

As a result, the set of pathways is determined by,

$$\sum_{i=1}^{15} A^i = A(A + I)^8(A + I)^4(A + I)^2$$

- This required one multiplication to obtain the square, squaring that to obtain the fourth power, squaring that again to obtain the eighth power and then three more multiplications to obtain the sum, for a total of six matrix multiplications without additions, as opposed to fourteen multiplications and additions if obtained directly.

3. Loops : Every loop pushes us to add up the matrix powers in an endless number of ways. We handled loops in a similar way to how we handled regular expressions. Every loop appears as a term in the diagonal of some power of the matrix, the power at which the loop eventually closes or, in other words, the loop's length. The impact of the loop can be retrieved by placing the path expression of the loop term starred before every element in the row of the node where the loop occurs, and then deleting the loop term.

4. Breaking loops and applications : Consider the matrix of a subgraph with a large number of links. Start by breaking the loop for the links on the primary diagonal if there are any. Consider the matrix's successive powers. A loop appears as an entry on the major diagonal at a certain power. Furthermore, the diagonal entry's regular expression over the link names supplies us with all of the points when we can

or must interrupt the loop. Another option is to use node-reduction, which will identify the loops and, as a result, the break points that are required. The equivalency partitioning method's divide-and-conquer, or rather partition-and-conquer, properties serve as a foundation for building tools. Because most algorithms need n^2 or n^3 arithmetic operations, where n is the number of nodes, they are computationally expensive. Even with rapid, low-cost equipment, keeping up with such growth laws is difficult. Breaking down large problems (hundreds of nodes) into smaller ones is the key to dealing with them. We can get processing on the order of n if we go far enough, which is good. The partition algorithm turns graphs into trees, which makes them much easier to work with.

Q.9 What is idempotent, idempotent generator, transitive closure ?

Ans. : Idempotent matrices are those in which $A^2 = A$. An idempotent generator is a matrix whose successive powers finally give an idempotent matrix that is, a matrix for which there is no solution a_k such that $A^{k+1} = A^k$

- The beauty of idempotent generator matrices is that by successively squaring, we can gain properties over all pathways. An idempotent generator is a graph matrix of the form (A + I) over a transitive relation; thus, everything of interest can be constructed using even simpler methods than the binary technique.
- For example, after we reach A^{n-1}, the relation "connected" remains unchanged because no connection may have more than n – 1 links and nodes cannot be disconnected once linked. As a result, if we needed to know which nodes in an n-node graph were connected to which, via whatever paths, we merely had to compute : A^P where p is the next power of 2 bigger than or equal to n, A^2, $A^2A^2 = A^4$ ... A^P. Because the relation "is connected to" is reflexive and transitive, we can do so. Because it is reflexive, each node possesses the equivalent of a self-loop, making the matrix an idempotent generator. If a relationship is transitive but not reflexive, it is said to be transitive.
- That is, while the relation established over A is not reflexive, the relation defined over A + I is. For powers bigger than n – 1, the length of the longest non repeating path through the graph, A + I is an idempotent generator, thus there's nothing new to learn. The transitive closure of the matrix is the n^{th} power of a matrix A + I over a transitive relation.

Q.10 List the different types of tools required for test planning. ☞ [JNTU : Part A, May-19, Marks 2]

Ans. : A product that supports one or more test activities, such as planning, requirements, building a build, test execution, defect recording, and test analysis, can be defined as a tool in the context of software testing.

Tool classifications

- Several parameters can be used to classify tools. They contain the following :
- The tool's purpose
- The activities that are supported by the instrument
- Its support for various types and levels of testing
- The type of licence (open source, freeware, commercial)
- The technology that was employed

Tool type	Used for	Used by
Test management tool	Managing tests, scheduling them, logging defects, tracking them and analyzing them.	Testers
Configuration management tool	For implementation, execution, tracking changes	Members of the team
Static analysis tools	Static evaluation	Developers
Test data preparation tools	Analysis and design, creation of test data	Testers

Test execution tools	Implementation, execution	Testers
Test comparators	Taking a look at the difference between what was expected and what happened	Members of the team
Coverage measurement tools	It provides structural protection	Developers
Performance testing tools	Performance monitoring and reaction time	Testers
Project planning and tracking tools	For the purpose of planning	Managers of projects
Incident management tools	To keep track of the tests	Testers

Implementation of tools - a step-by-step guide

- Carefully examine the problem to determine its strengths, flaws and opportunities.
- Budgetary constraints, time constraints and other criteria are stated.
- Evaluating the possibilities and whittling down the list to the ones that satisfy the criteria.
- Creating a proof of concept that encapsulates the benefits and drawbacks.
- Within a defined team, create a pilot project using the selected tool.
- Phased rollout of the tool across the business.

Q.11 Categorize various testing tools necessary for testing. ☞ **[JNTU : Part B, May-19, Marks 5]**

Ans. :

- The use of software testing tools is necessary for the improvement of an application or software.
- That is why there are so many tools on the market, some of which are free and others which are not.
- The main distinction between open-source and paid tools is that open-source tools have limited functionalities, whereas paid or commercial tools have no such restrictions. The choice of tools is based on the needs of the user, regardless of whether they are paid or free.
- Software testing tools can be classified based on their licencing (paid or commercial, open-source), technology used, testing method and other factors.
- We can increase the performance of our software, create a high-quality product, and reduce the time spent on manual testing with the help of testing tools.
- The following are the different types of software testing tools :
  - Tool for managing tests
  - Bug-tracking software
  - Tool for automating testing
  - Tool for evaluating peformance
  - Tool for cross-browser testing
  - Tool for integration testing
  - Tool for unit testing
  - Tool for mobile/Android testing
  - Tester for graphical user interfaces
  - Tool for security testing

Tool for managing tests

Test management solutions are used to keep track of all testing activity, do quick data analysis, manage manual and automation test cases and plan and sustain manual testing in a variety of scenarios.

Bug-tracking software

The defect tracking tool is used to keep track of issue fixes and ensure a high-quality product is delivered. This tool can assist us in identifying faults during the testing stage so that we can deliver data that is defect-free to the production server. End-users can use these tools to report bugs and issues directly on their applications using these technologies.

Tool for automating testing

This type of tool is used to increase product productivity while also improving accuracy. By writing certain test scripts in any language, we can cut the application's time and cost.

Tool for evaluating performance

Performance or load testing tools are used to verify the application's load, stability and scalability. We require load testing tools to get over this type of issue when n-number of users are using the application at the same time and the application crashes due to the massive load.

Tool for cross-browser testing

When we need to compare a web application across several web browser platforms, we use this type of tool. When we are working on a project, it is critical. We'll use these tools to guarantee that the app behaves consistently across numerous devices, browsers and platforms.

Tool for integration testing

This tool is used to test the interface between modules, detect important defects that have occurred as a result of the various modules and ensure that all of the modules are functioning according to the client's needs.

Tool for unit testing

This testing tool is used to assist in improving the quality of their code, as well as reducing the time it takes to code and the overall cost of the product.

Tool for mobile / Android testing

This type of tool can be used to test any type of mobile application. Some of the tools are free to use, while others require a licence. Each tool has its own set of features and functions.

Tester for graphical user interfaces

The GUI testing tool is used to test the application's user interface since a good GUI (graphical user interface) is always helpful in catching the user's attention. These kinds of tools will assist in identifying flaws in the application's design and improving it.

Tool for security testing

The security testing tool is used to assure the software's security and to look for security flaws. If there is a security flaw in the product, it could be addressed early on. When the software has encoded a security code that is not accessible by unauthorized users, we require this type of technology.

Q.12 What are the using of win-runner ?

☞ [JNTU : Part B, May-19, Marks 5]

Ans. :

- WinRunner is a functional testing tool that can be used to work on a collection of tests in conjunction with HP QuickTest Professional and as a supporting element for the quality assurance process during the test phase of the software development life cycle. This testing software is used as part of the process of improving product quality.
- Capturing the functional requirement / test requirement, validating the actual results versus the expected outcomes, and reproducing the user operations / functional activities conducted on the software product are all part of the testing process. During this phase, the tool can run the entire testing process involuntarily while discovering problems in the product design provided by the business / client employees. The Test Script Language (TSL) is used by the WinRunner Automation tool, which is similar to the C ming language in terms of receiving user activities as process input and enabling exceptional freedom for modification and constraint.
- The recording technique is a useful tool for creating strong functional test items. In order to do so, the tool often records the software application's functional flow by emulating user activities throughout the recording process. It also enables the testing expert to straight-up rewrite and update the scripts produced to fit the most functionality stated in the functional requirement specification documentation.
- The method continues by allowing testers to create checkpoints in order to facilitate comparison testing between the functionality allocated as expected results and the functionality assigned as actual

results. The test criteria, user interface features, images / logo, and navigation flow in the form of URLs are all examples of checkpoints.

- This testing procedure may include a number of validations, such as the application's cosmetic look, online communication interfaces, middleware communication elements and database validation to assess functions. It is well-known as a data-driven automated testing tool. Another tool, the Virtual Object Wizard, enables testing specialists to learn the WinRunner in identifying features, recording and replaying the objects given to the application functionality.

Q.13 Write a partition algorithm.

☞ [JNTU : March-22, Marks 7]

Ans. :

- Take any graph and place it over a transitive relation. There could be loops in the graph.
- We'd like to split the graph by grouping nodes so that each loop is contained within one of two groups.
- This type of graph is only partially sorted.
- There are a variety of algorithms that can be employed to do this.
- We might want to encapsulate the loops in a function so that the final graph is loop-free at the top level.
- If we know where to break the loops in several graphs with loops, we can easily examine them.
- While we can spot loops, programming a tool to do so is far more difficult unless the technology is bases on a soild algorithm.
- This is a simple procedure to follow.

find out the below matrix : $(A + I)^n \# (A + I)^{nT}$. The nodes are grouped into strongly connected groupings of nodes, which are somewhat sorted.

- In addition, every such set has an equivalence class, which means that any node in it can represent the set. We can hide the loop in a conceptual subroutine just as readily as we can in a real subroutine. Analyze the partially ordered graph produced by the partitioning algorithm, treating each loop-connected node set as a subroutine. Break the loop for each such component and repeat the operation.
- By selecting a row (or column) and searching the matrix for identical rows, we can identify equivalent nodes. As we continue, mark the nodes that match the pattern and remove that row. Then, starting from the beginning, add another row and a new design. All of the rows were eventually clustered. The following are the equivalent node sets as a result of the algorithm :

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

Example :

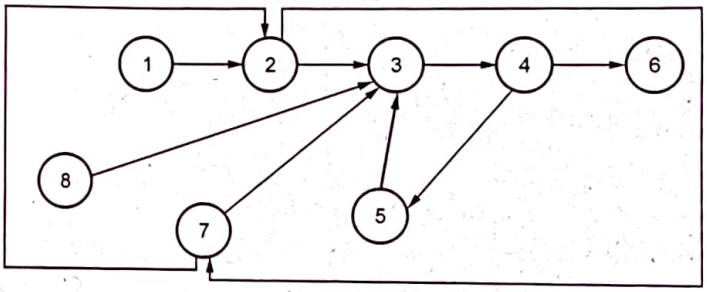


Fig. Q.13.1

a) Relation matrix

	1	2	3	4	5	6	7	8
1	1	1						
2		1	1				1	
3			1	1				
4				1	1	1		
5			1		1			
6						1		
7		1	1				1	
8			1					1

Diagonal elements represents loop - self loop

b) Transitive closure matrix

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	
2		1	1	1	1	1	1	
3			1	1	1	1		
4			1	1	1	1		
5			1	1	1	1		
6						1		
7		1	1	1	1	1	1	
8			1	1	1	1		1

c) Interaction with transpose

	1	2	3	4	5	6	7	8
1	1							
2		1					1	
3			1	1	1			
4			1	1	1	1		
5			1	1	1			
6						1		
7		1					1	
8								1

A = [1]

B = [2, 7]

C = [3, 4, 5]

D = [6]

E = [8]

Resultant graph is :

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

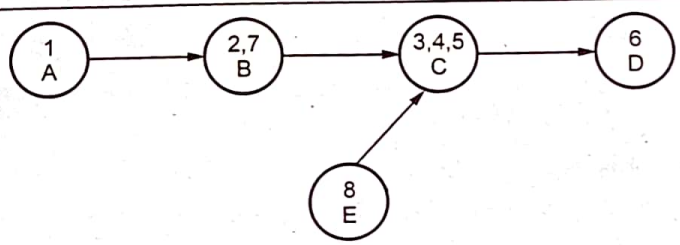


Fig. Q.13.2

Q.14 Explain about node reduction algorithm.

☞ [JNTU : Part B, May-19, Marks 5, March-22, Marks 7]

OR Elaborate node reduction algorithm with an example. ☞ [JNTU : Part B, Dec.-19, Marks 5]

Ans. :

- The matrix powers usually tell us more than we want to know in most graphs.
- In the context of testing, we're usually interested in creating a relationship between two nodes, usually the entry and exit nodes.
- In a debugging situation, we are unlikely to want to know the path expression between every node and every other node.
- The matrix reduction approach has the advantage of being more methodical than the graphical node by node elimination procedure.

1. Remove a node; replace it with equivalent links that bypass the node and add them to the links that they parallel.
2. Combine and simplify the parallel phrases as much as possible.
3. Pay attention to loop terms and update the out links of any node with a self loop to account for the loop's effect.
4. As a result, the size of the matrix has been lowered by one. Continue until the two nodes of interest are the only ones left.

Some matrix properties

If we numbered the nodes of a graph from 1 to n, we would not expect the graph's or the it represents to behave differently if the nodes were numbered otherwise. The numbering of nodes is completely random and has no bearing on anything. Interchanging the rows and columns of the appropriate matrix is equal to renumbering the nodes

of a network. Let us assume that we wished to rename nodes i and j to j and i respectively. On the graph, we would achieve this by erasing and rewriting the names. To modify the names of the nodes in the matrix, we must alter both the rows and columns that correspond to them.

Example : Step 1 :

	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		h

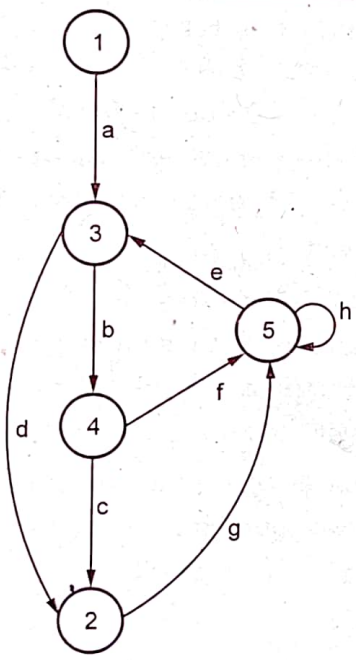


Fig. Q.14.1

- Delete a node and replace it with a set of equivalence links. e.g. - Start with node 5

Step 2 : Self loop is denoted by '*' and outgoing links are multiplied (Remove self loop)

	1	2	3	4	5
1			a		
2					
3		d		b	
4		c		t	
5		h*g	h*e		

Step 3 :

Eliminate node 5

	1	2	3	4	5
1			a		
2					
3		d		b	
4		c+fh*g	fh*e		
5					

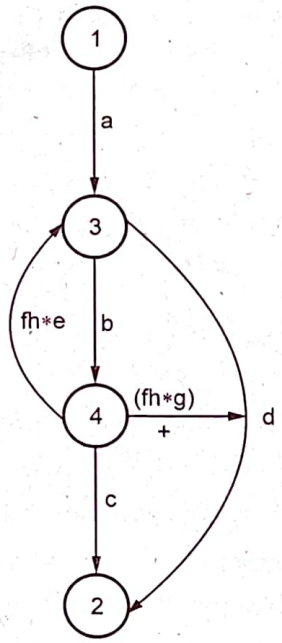


Fig. 7

Fig. Q.14.2

Final step : Now delete 4

Parallel link is added and serial links are multiplied.

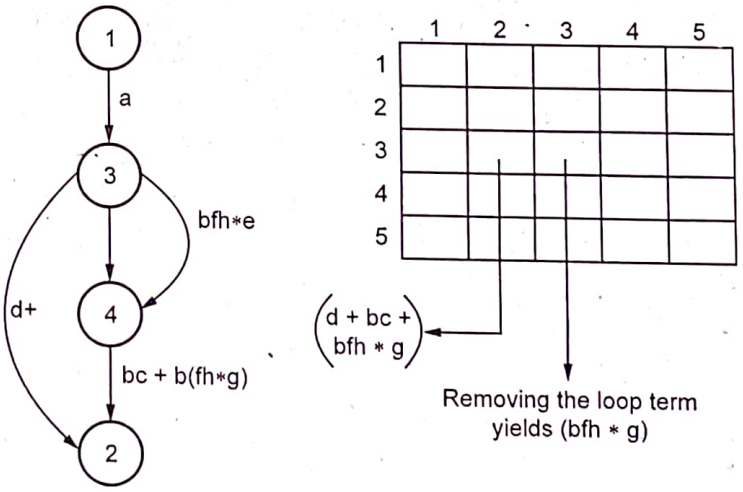


Fig. Q.14.3

		a
	(bfh * e) * X (d + bc + bfh * g)	

Result is

a(bfh * e) * (d + bc + bfh * g)

Q.15 What are graph matrices and their applications ? Explain in detail. ☞ **[JNTU : Part B, April-18, Marks 10]**

Ans. : A graph matrix is a type of data structure that can be used to aid in the creation of a path testing automation tool. Because their qualities are vital for building a test tool, graph matrices are extremely useful in learning software testing principles and theory.

Applications :

1. General - Obtaining the path expression is frequently the most complex and time-consuming. Most applications have simpler arithmetic rules.
2. Maximum number of paths : On the opposite page is the matrix that corresponds to the graph on page 261. The following steps are depicted. Remember that the inner loop between nodes 8 and 9 was supposed to be repeated three times, while the outer loop between nodes 5 and 10 was supposed to be repeated exactly four times. This will have an impact on how the diagonal loop terms are treated.
3. The probability of getting there :

   A graph matrix is a data structure that can help with the development of a path testing automation tool. Graph matrices are highly beneficial in understanding software testing concepts and theory since there properties are essential for designing a test tool.

Q.16 What are the matrix operations in tool building.

OR Discuss about matrix representation software.

☞ **[JNTU : Part B, Dec.-18, Marks 5]**

OR Explain cross-term reduction and node term reduction optimization.

OR Discuss the linked list representation.

Ans. : A matrix representation software :

We draw graphs or display them on screens as visual objects; we use matrices to prove theorems and design graph algorithms; and we represent graphs in a computer as linked lists when we're developing tools. Because graph matrices are typically sparse (i.e., the rows and columns are primarily empty), we employ linked lists.

1. Node degree and graph density

- The number of outlinks a node has is its out-degree. The number of inlinks a node has determines its in-degree. The sum of a node's out-degree and in-degree is its degree. For a typical graph constructed over software, the average degree of a node (the mean over all nodes) is between 3 and 4. A simple branch has a degree of 3 while a simple junction has a degree of 3. A loop's degree is only 4 if it's entirely contained in one phrase. A flowgraph with a mean node degree of 5 or 6, for example, would be extremely active.

2. What's wrong with arrays ?

- For short graphs with simple weights, we can express the matrix as a two-dimensional array, but this is inconvenient for bigger graphs because

1. **Space :** Space grows as n 2 for the matrix representation, but only as kn for a linked list, where k is a small integer like 3 or 4.
2. **Weights :** Most weights are complex, with multiple components. For each such weight, an extra weight matrix would be required.
3. **Weights of variable length :** If the weights are regular expressions or algebraic expressions (as we need for a timing analyzer), we'll need a two-dimensional string array with mostly null entries.
4. **Processing time :** While actions on null entries are quick, accessing and discarding such entries takes time. We have to spend a lot of time analyzing combinations of inputs that we know will result in null results because of the matrix representation.

- The matrix representation is handy for prototyping methods that haven't been tested before. Direct

matrix representations make it much easier to implement algorithms, especially if we have matrix manipulation subroutines or library functions. Matrices are reasonable up to roughly 20 – 30 nodes, which is sufficient for prototyping most tools.

3. Linked-list representation

- Although graphical representations of flow graphs are useful, the intricacies of a 's control flow are frequently uncomfortable.
- Each node in a linked list representation has a name, and each link in the flow graph has an entry on the list. Only the data related to the control flow is displayed.

B. Matrix operations

1. **Simultaneous reduction :** This is the most straightforward process. Following sorting, parallel linkages are neighboring entries with the same pair of node names. Consider the following scenario :

   node 17,21;x

   ,44;y

   ,44;z

   ,44;w

   From node 17 to node 44, we have three parallel links. Using the y, z, and w pointers, we construct a new link that is the sum of the weight expressions :

   node17,21;x

   ,44;y (where y = y + z + w).

2. **Loop reduction :** Loop reduction is nearly as simple as loop creation. A self-link is identified as a loop word. The loop's effect must be applied to the entire node's outlinks. To find the loop, search the link list for the node (s). Except for another loop, apply the loop computation to every outlink. Get rid of the loop. Repeat for all loops centered on that node. Repeat for each node. For instance, node 5's loop might be removed.

List entry	Content	Content after
5	node5,2; g →	node5,2;h*g
	,3;e →	,3;h*e
	,5;h →	
	4,	4,
	5,	

3. Cross-term reduction : Choose a node to be reduced. Every inlink to the node must be combined with every outlink from that node in the cross-term step. The outlinks are linked to the node that we have chosen. The back pointers are used to obtain the inlinks. The nodes of the inlinks will be associated with the new links generated by removing the node. Let's say the node that needed to be eliminated was node 4.

List entry	Content before	
2	node2,exit	node2,exit
	3,	3,
	4,	4,
	5,	5,
3	node3,2;d	node3,2;d
	,4;b	,2;bc
		,5;bf
	1,	1,
	5,	5,
4	node4,2;c	
	,5;f	
	3,	
5.	node5,2;h*g	node5,2;h*g
	,3,h*e	,3;h*e
	4,	

- If we do careful bookkeeping and maintain our pointers correct, we can remove several nodes in one pass as implemented. The node removal links are saved in a separate list, which is subsequently sorted and finally merged into the master list.

- Multiplication, addition, and other operations : The addition of two matrices is simple. Simply merge the lists and combine parallel entries if the lists are kept ordered.
- Multiplication is more difficult, yet it is also simple. We must defeat the outlinks of the node against the inlinks of the list. It's possible to do it in place, but it's much easier to make a new list. The list will be in sorted order once more, and we will utilize parallel combination to add and condense it.
- Transposition is accomplished by reversing the pointer directions, which results in an incorrectly sorted list. The transposition is obtained by sorting that list. Sorting, merging and combining parallels can be used to do all other matrix operations.

C. Node-reduction optimization

- The best sequence for node reduction is to start with the lowest-degree nodes. The goal is to make the lists as brief as feasible while yet completing them as rapidly as possible. When nodes of degree 3 (one in and two out or two in and one out) are eliminated, the total number of links is reduced by one. All higher-degree nodes increase the link count, but a degree-4 node keeps the link count constant. Although it is not guaranteed, selecting the lowest-degree node available for reduction almost eliminates the possibility of infinite list growth. This method is effective because list length dominates processing rather than the number of nodes in the list.
- The difference in processing time between not optimizing the node-reduction order and optimizing it was roughly 50:1 for big graphs with 500 or more nodes and an average degree of 6 or 7.

Fill in the Blanks for Mid Term Exam

Q.1 In a matrix, every _____ in the graph is represented as a single column.

Q.2 The corresponding node is _____ if the element of the primary diagonal is '1'.

Q.3 A relation that satisfies the reflexive, transitive, and antisymmetric qualities is _____.

Q.4 For the partition algorithm, the matrix that must be recognized is_____.

Q.5 The matrix is represented in software as _____.

Q.6 In the node reduction optimization process, the node of _____ degree should be deleted first.

Q.7 The automated regression testing tools are _____.

Q.8 In matrix operations, the set of all pathways between nodes is denoted as _____.

Q.9 The matrix is the one in which $A^2 = A$ is _____.

Q.10 A node's number of outlinks is referred to as _____.

Q.11 The _____ algorithm is used to determine the path from one node to another.

Q.12 The binary weights representation of a matrix is called as _____.

Q.13 Two types of matrices are _____.

Q.14 Formula _____ in state table can be used to calculate the amount of decisions made at a specific node.

Q.15 The _____ of a matrix is calculated by summing all of the decision values and then multiplying by one.

Multiple Choice Questions for Mid Term Exam

Q.1 The issues with graphical representation do not include the _____.

[a] tracking is not easy

[b] paths may be missed

[c] paths may be marked twice

[d] generating test cases is not difficult

Q.2 For binary weight, the false arithmetic rule is _____.

[a] 1+1=1 [b] 1*1=1

[c] 1÷0=0 [d] 1+0=1

Q.3 If a matrix column has two or more 1s, the matrix is said to be _____.

[a] branch node [b] junction node

[c] loop [d] both (a) and (b)

Q.4 "If aRb and bRa, then a=b," says the relationship _____.

[a] symmetric [b] antisymmetric

[c] reflexive [d] transitive

Q.5 The equivalence connection does not have to be satisfied _____.

[a] reflexive [b] symmetric

[c] antisymmetric [d] transitive

Q.6 The tools for creating graph matrices do not include _____.

[a] matrix operations

[b] matrix representation

[c] node reduction optimization

[d] none

Q.7 It is not necessary to satisfy partial ordering relations.

[a] reflexive [b] symmetric

[c] antisymmetric [d] transitive

Q.8 When nodes of degree 3 are deleted from a node reduction optimization, total linkages are _____.

[a] reduced 1 [b] not changed

[c] increased [d] reduced by 2

Q.9 The use of an array to represent a matrix is convenient because _____.

[a] it occupies large space

[b] it takes long processing times

[c] it takes more null values for string array

[d] all the above

Q.10 By replacing each item with _____, the square of the matrix whose entries are as.

[a] $\sum_{k=1}^{n} a_{ik} a_{kj}$ [b] $\sum_{k=1}^{n} a_{ik} a_{ji}$

[c] $\sum_{k=1}^{n} a_{ii} a_{jk}$ [d] none

Q.11 The number of nodes in the state graph equals

[a] size of matrix

[b] no. of rows × No of columns

[c] both (a) and (b)

[d] none

Q.12 In graph matrices, if a link between nodes does not exist, it is represented by _____.

[a] 0 [b] 1

[c] ∅ [d] null

Q.13 For the given matrix, the cyclomatic is _____.

$$\begin{matrix} & & 1 & & \\ & 1 & & 1 & \\ 1 & & & & 1 \\ 1 & & & & 1 \\ 1 & 1 & & & 1 \end{matrix}$$

[a] 0 [b] 2

[c] 3 [d] 5

Q.14 When compared to graphs, the advantages of the node reduction approach in matrices is _____.

[a] more methodical in graphs

[b] error free

[c] not necessary redrawing of graphs

[d] all of the above

Answer Keys for Fill in the Blanks :

Q.1	node	Q.2	loop node
Q.3	partial ordering relation	Q.4	$(P+1)^n \# (P+1)^{nl}$
Q.5	linked lists	Q.6	lowest or least
Q.7	Winrunner and Jmeter	Q.8	$\sum_{i=1}^{n} (A^i) = A + A^2 + A^3 + \ldots + A^n$
Q.9	Idempotent	Q.10	out degree
Q.11	matrix determinant	Q.12	connection matrix
Q.13	relational matrix and connection matrix	Q.14	n–1
Q.15	cyclomatic complexity		

Answer Keys for Multiple Choice Questions :

Q.1	d	Q.2	c	Q.3	b
Q.4	b	Q.5	a	Q.6	b
Q.7	d	Q.8	a	Q.9	d
Q.10	a	Q.11	c	Q.12	a
Q.13	d	Q.14	d		

END...

February - 2017 (Set 1)

Software Testing Methodologies [116ER]

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 20 Min.] [Max. Marks : 10

Answer All Questions. All Questions Carry Equal Marks.

I. **Choose the correct alternatives :**

1. *The following phase purpose of testing is to show that the software works :* **[Ans. : b]**
   a) PHASE 0 b) PHASE 1 c) PHASE 2 d) PHASE 3
2. *Which of the following believes that bugs are nice, tame and logical ? Only weak bugs have a logic to them and are amenable to exposure.* **[Ans. : a]**
   a) Benign bug hypothesis b) Bug locality hypothesis
   c) Control bug dominance d) Code / Data separation
3. *Which of the following is a bug where the outputs are misleading or redundant ? The bug imports the system performance.* **[Ans. : d]**
   a) Mild b) Annoying c) Disturbing d) Moderate
4. *The following includes paths left out, unreachable code, improper nesting of loops, loop-back or loop termination :* **[Ans. : b]**
   a) Logic bugs b) control and sequence bugs c) processing bugs d) data-flow bugs
5. *Static or dynamic data can service as :* **[Ans. : c]**
   a) Parameter b) Control c) Both A and B d) None of the above
6. *The following is a graphical representation of a programs' control structure :* **[Ans. : a]**
   a) Control flow graph b) Flowgraph c) Chart d) Flowchart
7. *PDL stands for* **[Ans. : a]**
   a) Program Design Language b) Project Design Language
   c) Program Design Lab d) Project Design Lab
8. *The following executes all statements in a program atleast once under some test* **[Ans. : b]**
   a) path testing b) statement testing c) branch testing d) none
9. *If the variables' value changes as a result of the processing, the variable is* **[Ans. : d]**
   a) process dependent b) process independent c) dependent d) independent
10. *The following occurs when the buggy predicate appears to work correctly because the specific value chosen for an assignment statement work with both correct and incorrect predicate.* **[Ans. : b]**
    a) Testing blindness b) Assignment blindness c) Equality blindness d) Self blindness

II. **Fill in the Blanks**

11. *In _____ (phase), the purpose of testing is not to prove anything, but to reduce the perceived risk of not working to an acceptable value.* **[Ans. : phase 3]**
12. *In _____ the program or system is treated as black box, it is subjected to inputs and outputs are verified.* **[Ans. : functional testing]**

13. _____ is a process by which components are aggregated to create larger components. [Ans. : Integration]

14. A _____ anamoly occurs when using an uninitialized variable, attempting to use a variable before it exists. [Ans. : data flow]

15. _____ are used to communicate with the world, includes devices, actuators, sensors, input terminals. [Ans. : The external interfaces]

16. A _____ is a program point at which the control flow can diverge. [Ans. : decision]

17. A program's _____ resembles a control flowgraph. [Ans. : flow chart]

18. _____ loops fall between single and nested loop with respect to the test cases. [Ans. : Concatenated]

19. TCB stands for _____. [Ans. : Transaction Control Block]

20. An object is _____ or undefined when it is released or made unavailable or when its contents are no longer known with certitude. [Ans. : killed]

February - 2017 (Set 2)

Software Testing Methodologies [116ER]

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 20 Min.] [Max. Marks : 10

I. Choose the correct alternatives :

1. The following includes paths left out, unreachable code, improper nesting of loops, loop-back or loop termination : [Ans. : b]

   a) Logic bugs b) Control and sequence bugs c) Processing bugs d) Data-flow bugs

2. Static or dynamic data can service as : [Ans. : c]

   a) Parameter b) Control c) Both A and B d) None of the above

3. The following is a graphical representation of a programs' control structure : [Ans. : a]

   a) Control flow graph b) Flowgraph c) Chart d) Flowchart

4. PDL stands for [Ans. : a]

   a) Program Design Language b) Project Design Language c) Program Design Lab d) Project Design Lab

5. The following executes all statements in a program atleast once under some test [Ans. : b]

   a) path testing b) statement testing c) branch testing d) none

6. If the variables' value changes as a result of the processing, the variable is [Ans. : d]

   a) process dependent b) process independent c) dependent d) independent

7. The following occurs when the buggy predicate appears to work correctly because the specific value chosen for an assignment statement work with both correct and incorrect predicate. [Ans. : b]

   a) Testing blindness b) Assignment blindness c) Equality blindness d) Self blindness

8. The following phase purpose of testing is to show that the software works : [Ans. : b]

   a) PHASE 0 b) PHASE 1 c) PHASE 2 d) PHASE 3

9. Which of the following believes that bugs are nice, tame and logical ? Only weak bugs have a logic to them and are amenable to exposure. [Ans. : a]

a) Benign bug hypothesis b) Bug locality hypothesis

c) Control bug dominance d) Code / Data separation

10. Which of the following is a bug where the outputs are misleading or redundant ? The bug imports the system performance. [Ans. : d]

a) Mild b) Annoying c) Disturbing d) Moderate

II **Fill in the Blanks**

11. A _____ anamoly occurs when using an uninitialized variable, attempting to use a variable before it exists. [Ans. : data flow]

12. _____ are used to communicate with the world, includes devices, actuators, sensors, input terminals. [Ans. : The external interfaces]

13. A _____ is a program point at which the control flow can diverge. [Ans. : decision]

14. A program's _____ resembles a control flowgraph. [Ans. : flow chart]

15. _____ loops fall between single and nested loop with respect to the test cases. [Ans. : Concatenated]

16. TCB stands for _____. [Ans. : Transaction Control Block]

17. An object is _____ or undefined when it is released or made unavailable or when its contents are no longer known with certitude. [Ans. : killed]

18. In _____ (phase), the purpose of testing is not to prove anything, but to reduce the perceived risk of not working to an acceptable value. [Ans. : phase 3]

19. In _____ the program or system is treated as black box, it is subjected to inputs and outputs are verified. [Ans. : functional testing]

20. _____ is a process by which components are aggregated to create larger components. [Ans. : Integration]

February - 2017 (Set 3)

Software Testing Methodologies [116ER]

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 20 Min.] [Max. Marks : 10

I. **Choose the correct alternatives :**

1. The following is a graphical representation of a programs' control structure : [Ans. : a]

a) Control flow graph b) Flowgraph c) Chart d) Flowchart

2. PDL stands for [Ans. : a]

a) Program Design Language b) Project Design Language c) Program Design Lab d) Project Design Lab

3. The following executes all statements in a program atleast once under some test [Ans. : b]

a) path testing b) statement testing c) branch testing d) none

4. *If the variables' value changes as a result of the processing, the variable is* **[Ans. : d]**

a) process dependent b) process independent c) dependent d) independent

5. *The following occurs when the buggy predicate appears to work correctly because the specific value chosen for an assignment statement work with both correct and incorrect predicate.* **[Ans. : b]**

a) Testing blindness b) Assignment blindness c) Equality blindness d) Self blindness

6. *The following phase purpose of testing is to show that the software works :* **[Ans. : b]**

a) PHASE 0 b) PHASE 1 c) PHASE 2 d) PHASE 3

7. *Which of the following believes that bugs are nice, tame and logical ? Only weak bugs have a logic to them and are amenable to exposure.* **[Ans. : a]**

a) Benign bug hypothesis b) Bug locality hypothesis

c) Control bug dominance d) Code / Data separation

8. *Which of the following is a bug where the outputs are misleading or redundant ? The bug imports the system performance.* **[Ans. : d]**

a) Mild b) Annoying c) Disturbing d) Moderate

9. *The following includes paths left out, unreachable code, improper nesting of loops, loop-back or loop termination :* **[Ans. : b]**

a) Logic bugs b) Control and sequence bugs c) Processing bugs d) Data-flow bugs

10. *Static or dynamic data can service as :* **[Ans. : c]**

a) Parameter b) Control c) Both A and B d) None of the above

II. Fill in the Blanks

11. *A _____ is a program point at which the control flow can diverge.* **[Ans. : decision]**

12. *A program's _____ resembles a control flowgraph.* **[Ans. : flow chart]**

13. *_____ loops fall between single and nested loop with respect to the test cases.* **[Ans. : Concatenated]**

14. *TCB stands for _____.* **[Ans. : Transaction Control Block]**

15. *An object is _____ or undefined when it is released or made unavailable or when its contents are no longer known with certitude.* **[Ans. : killed]**

16. *In _____ (phase), the purpose of testing is not to prove anything, but to reduce the perceived risk of not working to an acceptable value.* **[Ans. : phase 3]**

17. *In _____ the program or system is treated as black box, it is subjected to inputs and outputs are verified.* **[Ans. : functional testing]**

18. *_____ is a process by which components are aggregated to create larger components.* **[Ans. : Integration]**

19. *A _____ anamoly occurs when using an uninitialized variable, attempting to use a variable before it exists.* **[Ans. : data flow]**

20. *_____ are used to communicate with the world, includes devices, actuators, sensors, input terminals.* **[Ans. : The external interfaces]**

February - 2017 (Set 4)

Software Testing Methodologies [116ER]

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 20 Min.] [Max. Marks : 10

I. Choose the correct alternatives :

1. *The following executes all statements in a program atleast once under some test* [Ans. : b]
   a) path testing b) statement testing c) branch testing d) none
2. *If the variables' value changes as a result of the processing, the variable is* [Ans. : d]
   a) process dependent b) process independent c) dependent d) independent
3. *The following occurs when the buggy predicate appears to work correctly because the specific value chosen for an assignment statement work with both correct and incorrect predicate.* [Ans. : b]
   a) Testing blindness b) Assignment blindness c) Equality blindness d) Self blindness
4. *The following phase purpose of testing is to show that the software works :* [Ans. : b]
   a) PHASE 0 b) PHASE 1 c) PHASE 2 d) PHASE 3
5. *Which of the following believes that bugs are nice, tame and logical ? Only weak bugs have a logic to them and are amenable to exposure.* [Ans. : a]
   a) Benign bug hypothesis b) Bug locality hypothesis
   c) Control bug dominance d) Code / Data separation
6. *Which of the following is a bug where the outputs are misleading or redundant ? The bug imports the system performance.* [Ans. : d]
   a) Mild b) Annoying c) Disturbing d) Moderate
7. *The following includes paths left out, unreachable code, improper nesting of loops, loop-back or loop termination :* [Ans. : b]
   a) Logic bugs b) Control and sequence bugs c) Processing bugs d) Data-flow bugs
8. *Static or dynamic data can service as :* [Ans. : c]
   a) Parameter b) Control c) Both A and B d) None of the above
9. *The following is a graphical representation of a programs' control structure :* [Ans. : a]
   a) Control flow graph b) Flowgraph c) Chart d) Flowchart
10. *PDL stands for* [Ans. : a]
    a) Program Design Language b) Project Design Language
    c) Program Design Lab d) Project Design Lab

II. Fill in the Blanks

11. _____ *loops fall between single and nested loop with respect to the test cases.* [Ans. : Concatenated]
12. *TCB stands for* _____. [Ans. : Transaction Control Block]

13. *An object is _____ or undefined when it is released or made unavailable or when its contents are no longer known with certitude.* **[Ans. : killed]**

14. *In _____ (phase), the purpose of testing is not to prove anything, but to reduce the perceived risk of not working to an acceptable value.* **[Ans. : phase 3]**

15. *In _____ the program or system is treated as black box, it is subjected to inputs and outputs are verified.* **[Ans. : functional testing]**

16. *_____ is a process by which components are aggregated to create larger components.* **[Ans. : Integration]**

17. *A _____ anamoly occurs when using an uninitialized variable, attempting to use a variable before it exists.* **[Ans. : data flow]**

18. *_____ are used to communicate with the world, includes devices, actuators, sensors, input terminals.* **[Ans. : The external interfaces]**

19. *A _____ is a program point at which the control flow can diverge.* **[Ans. : decision]**

20. *A program's _____ resembles a control flowgraph.* **[Ans. : flow chart]**

April - 2018

Software Testing Methodologies [R15] (126VR)

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 3 Hours] **[Max. Marks : 75**

Note : This question paper contains two parts A and B. Part A is compulsory which carries 25 marks. Answer all questions in part A, part B consists of 5 units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

Part - A **(25 Marks)**

1. a) *What is meant by testing ? Why we need it.* **(Refer Q.1 of Chapter - 1)** [2]

   b) *Define a model for software testing.* **(Refer Q.8 of Chapter - 1)** [3]

   c) *Explain various loops. Give example for each.* [2]

Ans. : Single loop, nested loops, concatenated loop, horrible loops.

   d) *Write the applications of data flow testing.* **(Refer Q.13 of Chapter - 2)** [3]

   e) *In what a nice domain differs from and ugly domains.* **(Refer Q.30 of Chapter - 2)** [2]

   f) *Define domain testing with example.* **(Refer Q.22 of Chapter - 2)** [3]

   g) *Explain regular expressions.* **(Refer Q.8 of Chapter - 3)** [2]

   h) *Explain sum of product form and product of sum form.* **(Refer Q.9 of Chapter - 3)** [3]

   i) *Define good state and bad state graphs.* **(Refer Q.3 of Chapter - 4)** [2]

   j) *How can the graph be represented in matrix form ?* **(Refer Q.2 of Chapter - 4)** [3]

Part - B **(50 Marks)**

2. *State and explain various dichotomies in software testing.* **(Refer Q.6 of Chapter - 1)** [10]

OR

3. a) *What is meant by program's control flow ? How is it useful for path testing ?* **(Refer Q.23 of Chapter - 1)**

b) *Discuss various flow graph elements with their notations.* **(Refer Q.21 of Chapter - 1)** [5 + 5]

4. a) *What is meant by transaction flow testing. Discuss its significance.* **(Refer Q.4 of Chapter - 2)**

b) *Compare data flow and path flow testing strategies.* **(Refer Q.16 of Chapter - 2)** [5 + 5]

OR

5. a) *Explain data-flow testing with an example. Explain its generalizations and limitations.* **(Refer Q.19 of Chapter - 2)**

b) *Explain the terms dicing. Data-flow and debugging.* **(Refer Q.20 of Chapter - 2)** [5 + 5]

6. a) *State and explain various restrictions at domain testing processes.* **(Refer Q.25 of Chapter - 2)**

b) *With a neat diagram, explain the schematic representation of domain testing.* **(Refer Q.31 of Chapter - 2)** [5 + 5]

OR

7. *Discuss the domains and interface testing in detail.* **(Refer Q.23 of Chapter - 2)** [10]

8. *Write short notes on following : a) Distributive laws b) Absorption rule c) Loops d) Identity elements.* **(Refer Q.12 of Chapter - 3)** [10]

OR

9. *Reduce the following functions using K-maps*

$F(A,B,C,D) = P(4,5,6,7,8,12,13) + d(1,15)$ **(Refer Q.19 of Chapter - 3)** [10]

10. a) *Write testers comments about state graphs.* **(Refer Q.8 of Chapter - 4)**

b) *Explain about good state and bad state graphs.* **(Refer Q.3 of Chapter - 4)** [5+5]

OR

11. *What are graph matrices and their applications ? Explain in detail.* **(Refer Q.15 of Chapter - 5)** [10]

December - 2018

Software Testing Methodologies [R15] (126VR)

Solved Paper
III[rd] Year, B.Tech., Sem - II (CSE/IT)

Time : 3 Hours] [Max. Marks : 75

Part - A **(25 Marks)**

1. a) *What are feature bugs ?* **(Refer Q.10 of Chapter - 1)** [2]

b) *Distinguish between builder and buyer.* **(Refer Q.6 of Chapter - 1)** [3]

c) *What is petri net ?* **(Refer Q.16 of Chapter - 4)** [2]

d) *Explain about data flow anomaly graph with example.* **(Refer Q.10 of Chapter - 2)** [3]

e) *What is domain span ?* **(Refer Q.29 of Chapter - 2)** [2]

f) *Discuss about domain dimensionality.* **(Refer Q.29 of Chapter - 2)** [3]

g) *Define silicon compilers.* [2]

Ans. : Silicon compilers : Software that converts a chips electronic architecture into the logic gate layout, including the actual masking from one transistor to the next. The netlist or a high-level description are the sources of the compilation.

h) *Write eight steps in a reduction procedure.* **(Refer Q.3 of Chapter - 3)** [3]

i) *What is impossible state ?* **(Refer Q.6 of Chapter - 4)** [2]

j) *Distinguish between manual testing and automated testing.* **(Refer Q.35 of Chapter - 1)** [3]

Part - B (50 Marks)

2. a) *What are structural bugs ? Explain.* **(Refer Q.11 of Chapter - 1)**
   b) *Describe notational evolution of control flow graph with example.* **(Refer Q.22 of Chapter - 1)** [5 + 5]

OR

3. a) *Explain heuristics procedures for sensitizing paths.* **(Refer Q.33 of Chapter - 1)**
   b) *Is testing is everything ? Explain.* **(Refer Q.1 of Chapter - 1)** [5 + 5]
4. a) *Discuss about the data flow model.* **(Refer Q.18 of Chapter - 2)**
   b) *Explain transaction-flow graph implementation with example.* **(Refer Q.8 of Chapter - 2)** [5 + 5]

OR

5. a) *What are the structural test strategies based on the program's control flowgraph ? Explain.* **(Refer Q.16 of Chapter - 2)**
   b) *Discuss about complication in transaction-flow testing.* **(Refer Q.6 of Chapter - 2)** [5 + 5]
6. a) *Explain about testing one - dimensional domains.* **(Refer Q.39 of Chapter - 2)**
   b) *Write about restrictions of domain testing.* **(Refer Q.25 of Chapter - 2)** [5 + 5]

OR

7. *Define domain testing. Explain about nice domains in detail.* **(Refer Q.24 of Chapter - 2)** [10]
8. a) *Describe the mean processing time of a routine with example.* **(Refer Q.17 of Chapter - 3)**
   b) *Write rules of boolean algebra.* **(Refer Q.8 of Chapter - 3)** [5 + 5]

OR

9. a) *Briefly explain about regular expressions and flow-anomaly detection.* **(Refer Q.11 of Chapter - 3)**
   b) *Write the procedure for specification validation.* **(Refer Q.20 of Chapter - 3)** [5 + 5]
10. a) *What are some situations in which state testing may prove useful ? Explain.* **(Refer Q.13 of Chapter - 4)**
    b) *What are properties of relations ? Explain.* **(Refer Q.7 of Chapter - 4)** [5 + 5]

OR

11. a) *Explain software implementation of state graphs.* **(Refer Q.5 of Chapter - 4)**
    b) *Discuss about matrix representation software.* **(Refer Q.16 of Chapter - 5)** [5 + 5]

May - 2019

Software Testing Methodologies [R15] (126VR)

Solved Paper

III[rd] Year, B.Tech., Sem - II (CSE/IT)

Time : 3 Hours] **[Max. Marks : 75**

Part - A **(25 Marks)**

1. a) *List the goals of software testing.* **(Refer Q.3 of Chapter - 1)** [2]
   b) *What is path sensitization ?* **(Refer Q.33 of Chapter - 1)** [3]
   c) *Explain various loops with an example.* [2]

Ans. : When it comes to path testing, there are only three types of loops :

Nested Loops : The number of tests to be run on nested loops is proportional to the number of tests run on single loops. We can't always afford to test all possible iterations values for nested loops. Here's a strategy for getting rid of some of these values :

1. Begin at the most inner loop. Set the minimum values for all of the outer loops.
2. For the innermost loop, test the minimum, minimum+1, typical, maximum-1, and maximum, while keeping the outside loops at their minimal iteration parameter values. For out of range and excluded values, expand the tests as needed.
3. GOTO step 5 if we have completed the last loop; otherwise, move one loop out and set it up as in step 2 with all other loops set to normal values.
4. Continue in this manner outward until all loops are covered.
5. Complete all cases for all loops in the nest at the same time.

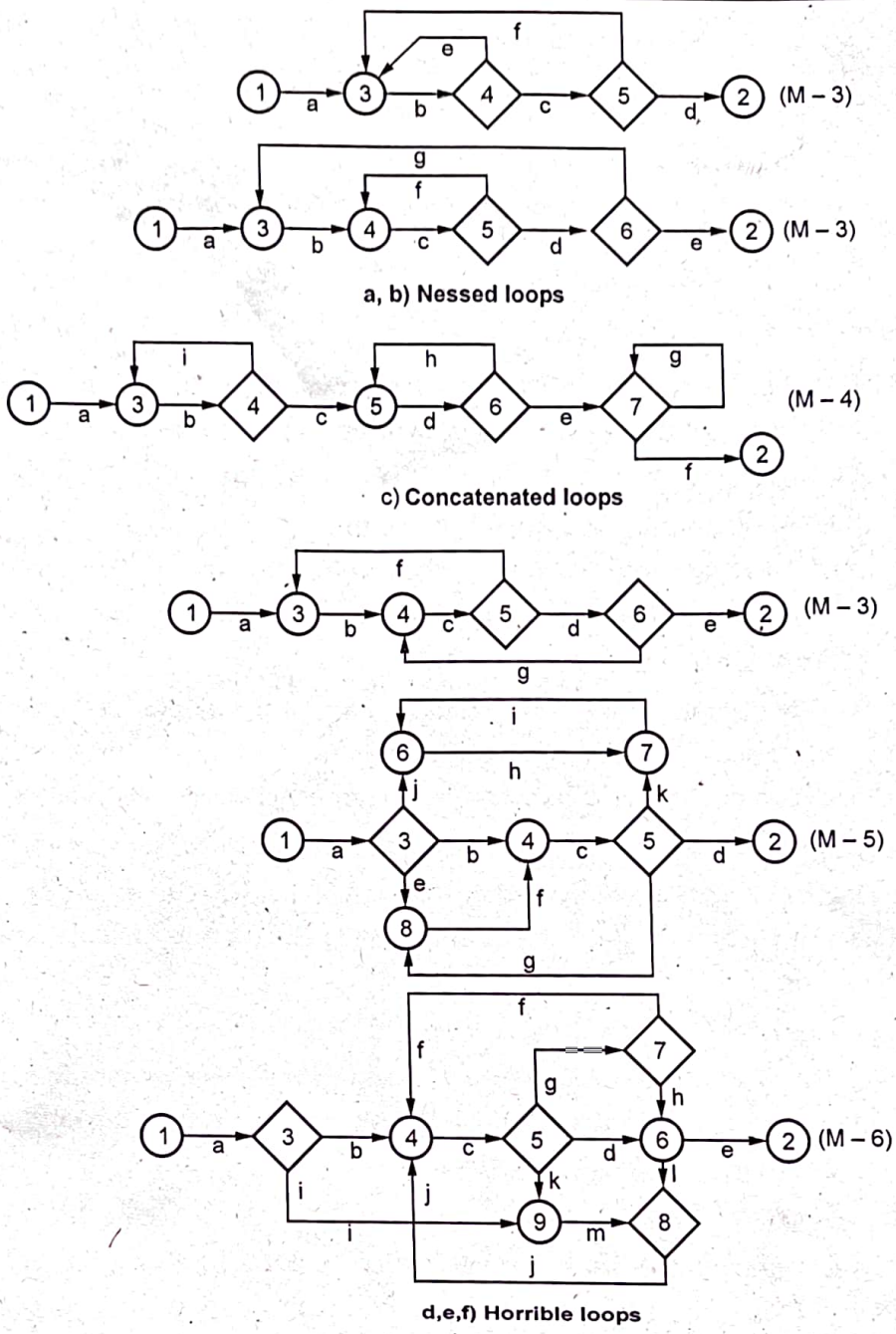


Fig. 1 Example of loop types

Concatenated loops : When it comes to test cases, concatenated loops sit between single and nested loops. If it's possible to reach one after quitting the other while still on a path from entrance to exit, two loops are concatenated. If the loops can't all be on the same path, they're not concatenated and can be addressed separately.

Horrible loops are made up of nested loops, code that jumps in and out of loops, intersecting loops, hidden loops, and cross-connected loops. Another reason such structures should be avoided is that they make iteration value selection for test cases an amazing and unpleasant chore.

d) *What is meant by testing ? Write about any two application of data flow testing.* **(Refer Q.13 of Chapter - 2)** **[3]**

e) *Define nice and ugly domains.* **(Refer Q.30 of Chapter - 2)** **[2]**

f) *Define domain testing with example.* **(Refer Q.22 of Chapter - 2)** **[3]**

g) *What is regular expression ?* **(Refer Q.8 of Chapter - 3)** **[2]**

h) *What is logic based testing ?* **(Refer Q.14 of Chapter - 3)** **[3]**

i) *What are testability tips ?* **(Refer Q.15 of Chapter - 4)** **[2]**

j) *List the different types of tools required for test planning.* **(Refer Q.10 of Chapter - 5)** **[3]**

Part - B **(50 Marks)**

2. a) *Discuss about myths related software testing and its facts.*

Ans. : Some of the most popular software testing myths are listed below :

Myth 1 : Testing is too expensive reality there's an old adage that says you should pay less for testing during software development and more for maintenance or correction later. In many ways, early testing saves both time and money; yet, cutting costs without testing may result in a software application with an incorrect design, leaving the product unusable.

Myth 2 : Testing takes a long time to complete.

Reality : Testing is never a time-consuming activity during the SDLC phases. Diagnosing and correcting mistakes discovered via proper testing, on the other hand, is a time-consuming but worthwhile exercise.

Myth 3 : Only fully developed products are tested.

Reality : Only fully developed products are tested. Testing is, without a doubt, dependent on the source code, but analysing requirements and building test cases are not. However, as a development life cycle model, an iterative or incremental method may lessen the testing's reliance on completely produced software.

Myth 4 : Complete testing is possible

Reality : Complete testing is possible It's possible that the team has tested all of the options, but thorough testing is never achievable. Some scenarios may never be run by the test team or the customer throughout the software development life cycle, but they may be run once the project is launched.

Myth 5 : A bug-free software is one that has been thoroughly tested.

Reality : This is a widespread misconception held by clients, project managers, and the management team. Even if a software product has been thoroughly tested by a tester with exceptional testing skills, no one can guarantee that it is bug-free

Myth 6 : Testers are to Blame for Missed Defects

Reality : Testers are to Blame for Missed Defects This misconception is about altering constraints like as time, cost, and requirements. However, the test strategy may result in the testing team missing bugs.

b) *Explain about life cycle of bug.* **[5 + 5]**

Ans. : Bug Life Cycle or Defect Life Cycle In software testing, the life cycle refers to the exact set of phases that a defect or bug passes through throughout the course of its existence. The goal of the defect life cycle is to make the defect fixing process more systematic and efficient by conveniently coordinating and communicating the current status of the problem to various assignees.

From project to project, the number of states that a fault passes through varies. The lifespan diagram below depicts all conceivable states.

New : When a new fault is first documented and publicised, it is referred to as "new." It is given the status of NEW.

Assigned : Once the tester has filed the bug, the tester's lead accepts it and assigns it to the programming team.

Open : The developer begins researching and repairing the flaw.

Fixed : A developer can mark an issue as "Fixed" once he or she has made a necessary code modification and verified it.

Pending retest : Once the flaw is repaired, the developer offers the tester a specific code to retest the code. The status issued is "pending retest" because the software testing is still waiting from the testers' end.

Retest : At this level, the tester retests the code to see if the defect has been repaired by the developer and changes the status to "Re-test."

Verified : The tester re-tests the bug after the developer has solved it. If no bugs are found in the software, the problem is fixed, and the status is changed to "confirmed."

Reopen : The tester updates the status to "reopened" if the bug continues after the developer has solved it. The bug goes through the life cycle once more.

Closed : When a bug is no longer present, the tester marks it as "Closed."

Duplicate : The status is changed to "duplicate" if the defect is repeated again or if the defect relates to the same bug idea.

Rejected : If the developer believes the problem isn't genuine, the defect is marked as "rejected."

Deferred : If the current bug is not a top priority and is expected to be fixed in the next release, it is granted the status "Deferred."

Not a bug : A bug's status is set to "Not a bug" if it has no impact on the application's operation.

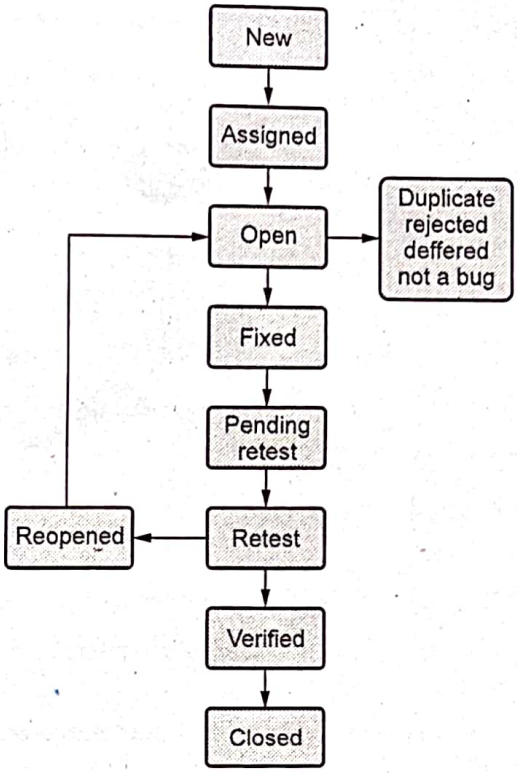


Fig. 2 Bug life cycle.

OR

3. a) *What is meant by integration testing and what are the goals of it.*

Ans. : After unit testing, integration testing is the next step in the software testing process. Units or individual components of the software are tested in a group during this testing. The integration testing level focuses on exposing faults that occur during the interaction of integrated components or units.

Unit testing employs modules for testing purposes, and integration testing combines and tests these modules. The software is made up of a variety of software modules that have been coded by various programmers or coders. The purpose of integration testing is to ensure that all components are communicating correctly.

When all of the components or modules are working independently, integration testing is used to check the data flow between the dependent modules.

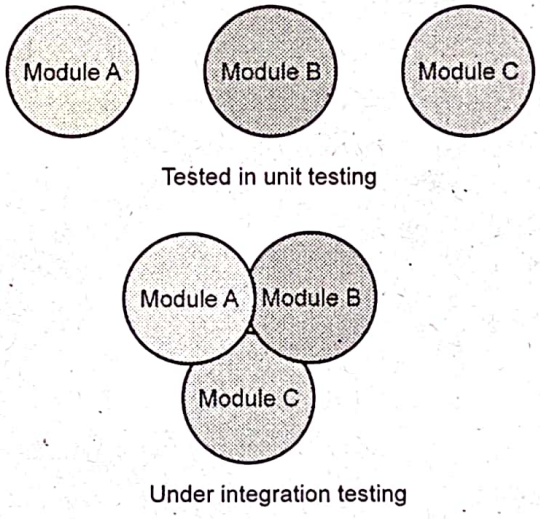


Fig. 3

b) *What are control and sequence bugs ? How they can be caught ?* **(Refer Q.11 of Chapter - 1)** **[5 + 5]**

4. *Discuss in detail data - flow testing strategies.* **(Refer Q.17 of Chapter - 2)** **[10]**

OR

5. a) *Compare data flow and path flow testing strategies.* **(Refer Q.16 of Chapter - 2)**

b) *Distinguish between control flow and transaction flow.* **(Refer Q.5 of Chapter - 2)** **[5 + 5]**

6. a) *Discuss with example the equal - span range/ domain compatibility bugs.* **(Refer Q.42 of Chapter - 2)**

b) *Discuss*

i) Non linear domain boundaries **(Refer Q.38 (i) of Chapter - 2)**

ii) Complete domain boundaries. **(Refer Q.38 (ii) of Chapter - 2)** **[5 + 5]**

OR

7. *State and explain various restrictions at domain testing processes.* **(Refer Q.25 of Chapter - 2)** **[10]**

8. *Explain regular expressions and flow anomaly detection.* **[10]**

OR

9. *Explain path expression with examples.* **(Refer Q.1 of Chapter - 3)** **[10]**

10. a) *Categorize various testing tools necessary for testing.* **(Refer Q.11 of Chapter - 1)**

b) *What are the using of win-runner ?* **(Refer Q.12 of Chapter - 5)** **[5 + 5]**

OR

11. a) *What are the principles of state testing ? Discuss advantages and disadvantages.* **(Refer Q.13 of Chapter - 4)**

b) *Explain about node reduction algorithm.* **(Refer Q.14 of Chapter - 5)** **[5 + 5]**

December - 2019

Software Testing Methodologies [R15] (126VR)

Solved Paper

IIIrd Year, B.Tech., Sem - II (CSE/IT)

Time : 3 Hours] **[Max. Marks : 75**

Part - A **(25 Marks)**

1. a) *What is software testing and write its purpose.* **(Refer Q.1 of Chapter - 1)** **[2]**

b) *Write the applications of path testing.* **(Refer Q.34 of Chapter - 1)** **[3]**

c) *What is data flow anomaly ?* **(Refer Q.9 of Chapter - 2)** **[2]**

d) *Compare data flow Vs transaction flow.* **(Refer Q.1 of Chapter - 2)** **[3]**

e) *What is loop free software ?* **(Refer Q.26 of Chapter - 2)** **[2]**

f) *Write about linear vector space.* **(Refer Q.27 of Chapter - 2)** **[3]**

g) *Define cross and parallel term in path testing.* **(Refer Q.28 of Chapter - 1)** **[2]**

h) *Write the limitations of path testing.* **(Refer Q.29 of Chapter - 1)** **[3]**

i) *What is state transition ?* **(Refer Q.10 of Chapter - 4)** **[2]**

j) *Describe encoding bugs and give examples.* **[3]**

Ans. : Encoding bugs are a type of bug that occurs when data is encoded incorrectly.

A bug is a coding fault in a computer program in computer technology. Debugging begins with the initial writing of the code and continues in phases when the code is coupled with other programming units to build a software product, such as an operating system or an application.

For example, if we use the Gmail program and click on the "Inbox" link, it will lead us to the "Draft" page. This is a defect since the developer used incorrect coding.

Part - B **(50 Marks)**

2. a) *What is bug ? Explain various kinds of bugs.* **(Refer Q.12 of Chapter - 1)**
   b) *Explain in detail about path sensitization.* **(Refer Q.33 of Chapter - 1)** **[5 + 5]**

OR

3. a) *List the elements of flow graph and explain each element with suitable diagram.* **(Refer Q.21 of Chapter - 1)**
   b) *What is path testing ? Give a note on path selection and predicates.* **(Refer Q.27 of Chapter - 1)** **[5 + 5]**
4. a) *What is transaction flow testing ? Explain with example.* **(Refer Q.2 of Chapter - 2)**
   b) *What is transaction instrumentation in transaction flow ? Explain with example.* **(Refer Q.3 of Chapter - 2)** **[5 + 5]**

OR

5. a) *What is program slicing ? Explain dynamic program slicing.* **(Refer Q.41 of Chapter - 2)**
   b) *Explain different data object states in data flow graphs.* **(Refer Q.20 of Chapter - 1)** **[5 + 5]**
6. a) *Explain predicates of domain testing with examples.* **(Refer Q.22 of Chapter - 2)**
   b) *Compare domain testing and interface testing.* **(Refer Q.23 of Chapter - 2)** **[5 + 5]**

OR

7. a) *Write about nice and ugly domains and give examples to each domain.* **(Refer Q.30 of Chapter - 2)**
   b) *Explain various bugs encountered at systematic and domain boundaries.* **[5 + 5]**

Ans. : The domain testing bug assumption is that processing is fine, but the domain definition is incorrect.

- Incorrectly implemented domains result in improper bounds, which may lead to incorrect control flow predicates.
- Domain errors can be caused by a variety of problems.

Boundary errors : Errors that occur at or near a domain's boundary. For instance, a boundary closure bug, a warped, skewed, or missing boundary, or an extra boundary

An interior point is a point in the domain that is shared by all points within an arbitrarily small distance.

A boundary point is one where there are points both in the domain and not in the domain within an epsilon neighbourhood.

A point that does not lie between any two other arbitrary but distinct points of a domain is called an extreme point. A point on the boundary is referred to as a on point.

An off point is a place near the domain boundary that is in the adjacent domain if the domain boundary is closed.

An off point is a place near the boundary but within the domain being tested if the domain is open.

Systematic Boundary :

Example :

$f_1(x) >= k_1$

or $f_1(x) >= g(1, c)$

$f_2(x) >= k_2$

or $f_2(x) >= g(2, c)$

$f_i(x) >= k_i$ or $f_i(x) >= g(1, c)$

Where f_i is an arbitrary function, x is the input vector, k_i and c are constants and g(i, c) is a decent function over I and c that yields a constant such as k + ic.

8. a) *Write the procedure to count minimum number of paths in a graph.* **(Refer Q.2 of Chapter - 3)**
   b) *Describe push/pop and get/return models in path testing.* **(Refer Q.14 of Chapter - 3)** [5 + 5]

OR

9. a) *Compare structured and unstructured flow graphs and illustrate with an example.* **(Refer Q.7 of Chapter - 3)**
   b) *What is KV-chart ? Draw KV-chart for 4 variables.* **(Refer Q.18 of Chapter - 3)** [5 + 5]

10. a) *Explain state testing in detail.* **(Refer Q.13 of Chapter - 4)**
    b) *Write the guidelines to design state machines.* **(Refer Q.14 of Chapter - 4)** [5 + 5]

OR

11. a) *Elaborate node reduction algorithm with an example.* **(Refer Q.14 of Chapter - 5)**
    b) *Explain good state graph with suitable example.* **(Refer Q.3 of Chapter - 4)** [5 + 5]

September - 2021

Software Testing Methodologies (R18)[156CW]

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 3 Hours] **[Max. Marks : 75**

Answer any five Questions.
All Questions Carry Equal Marks.

Q.1 a) *State and explain various dichotomies in software testing.* **(Refer Q.6 of Chapter - 1)** **[8 + 7]**
b) *Explain the life cycle of bug.*

Ans. : The defect is The Life Cycle, commonly known as the Bug Life Cycle, is a series of faults that it experiences during the course of its entire life. This begins when a tester discovers a new flaw and ends when a tester closes that fault, ensuring that it will not be repeated again.

Defect workflow

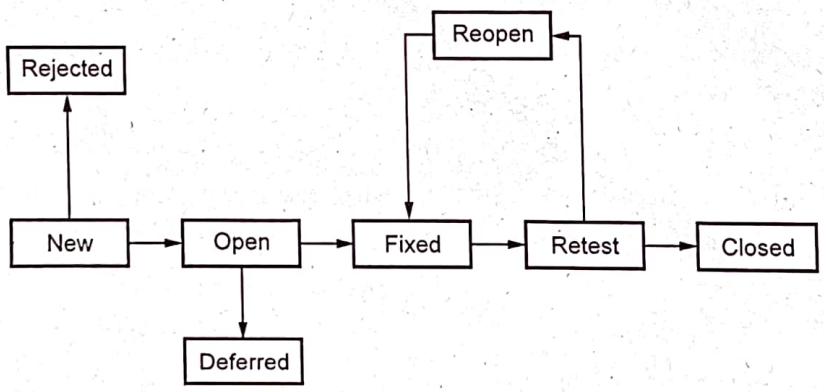


Fig. 1 : Defect workflow

With the use of a simple diagram like the one below, we can now grasp the actual operation of a Defect Life Cycle.

Defect states

#1) New : In the Defect Life Cycle, this is the first state of a defect. When a new flaw is discovered, it is classified as 'New,' and validations and testing are done on it later in the Defect Life Cycle.

#2) Assigned : A freshly created defect is assigned to the development team for further investigation. This is given to a developer by the project lead or the testing team manager.

#3) Open : Here, the developer begins the process of studying the fault and if necessary, repairing it. If the developer believes the defect is inappropriate, it may be moved to one of the four states listed below : Duplicate, Deferred, Rejected or Not a Bug, depending on the reason.

#4) Fixed : Once a developer has completed the work of repairing a defect by implementing the necessary changes, he can label the problem as "Fixed."

#5) Pending retest : After resolving the defect, the developer assigns it to the tester to retest it at their end, and the defect's state stays "Pending Retest" until the tester completes the retest.

#6) Retest : At this phase, the tester begins retesting the defect to see if the developer has accurately corrected the problem according to the requirements.

#7) Reopen : If a defect's issue continues, it will be assigned to a developer for testing again and the defect's status will be updated to 'Reopen.'

#8) Verified : If the tester finds no issues with the defect after it has been assigned to the developer for retesting and believes that the defect has been repaired correctly, the defect's status is set to 'Verified.'

#9) Closed : When the defect is no longer present, the tester changes the defect's state to "Closed."

Q.2 **a)** *What are control and sequence bugs ? How can they be caught ?* **(Refer Q.11 of Chapter - 1)**

b) *Explain heuristics procedure for sensitizing paths.* **(Refer Q.33 of Chapter - 1)** **[7 + 8]**

Q.3 **a)** *Compare data flow and path flow testing strategies.* **(Refer Q.7 of Chapter - 2)**

b) *Explain transaction flow graph implementation with example.* **(Refer Q.7 and Q8 of Chapter - 2)** **[7 + 8]**

Q.4 **a)** *What is program slicing ? Explain dynamic program slicing.* **(Refer Q.41 of Chapter - 2)**

b) *Explain predicates of domain testing of examples.* **(Refer Q.22 of Chapter - 2)** **[8 + 7]**

Q.5 **a)** *Write the role of path expression and path predicates in testing*

Ans. : Path expression : Refer Q.1 of Chapter - 1.

Path predicates :

PREDICATE : Predicate is the logical function that is assessed when making a decision. The value of the decision variable influences the choice's outcome. A > 0, x + y > = 90 and so on are some examples.

A Path Predicate is a predicate that is related with a path. "x is greater than zero," "x + y > = 90," and "w is either negative or equal to 10 is true" are examples of predicates whose truth values lead the routine to follow a particular path.

b) *Write the procedure to count the minimum number of paths in a graph.* **(Refer Q.2 of Chapter - 3)** **[7 + 8]**

Q.6 *Demonstrate through truth tables the validity of the following theorems of Boolean algebra :* **[5 + 5 + 5]**

a. Associative laws *b. Demorgan's theorem for 3 variables*

c. Distributive law

Ans. : For Boolean algebra table - Refer Q.8 of Chapter - 3.

a. Associative laws : This law states that : $(A + B) + C = A + (B + C)$ $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

A	B	C	A + B	(A + B) + C	B + C	A + (B + C)
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	0	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

b. Demorgan's theorem for 3 variables :

Inputs	Truth table outputs for each term					
B	A	A · B	$\overline{A \cdot B}$	A	B	A + B
0	0	0	**1**	1	1	**1**
0	1	0	**1**	0	1	**1**
1	0	0	**1**	1	0	**1**
1	1	1	0	0	0	0

c. Distributive law: $A(B + C) = A \cdot B + A \cdot C$

A	B	C	B + C	A · (B + C)	A · B	A · C	A · B + A · C
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Q.7 a) *Explain about good state and bad state graphs. How to handle bad state graphs.* **(Refer Q.3 of Chapter - 4)**

b) *Write short notes on :* **[7 + 8]**

a. Transition bugs **(Refer Q.4, Point 4 of Chapter - 4)**

b. Dead states **(Refer Q.4 of Chapter - 4)**

Q.8 a) *Explain the features of the Jmeter testing environment*

Ans. : Apache Jmeter has the following key features.

- GUI design and interface
- Result analysis and caches
- Highly extensible core
- 100 % Java scripted
- Pluggable samplers
- Multithreading framework
- Data analysis and visualization
- Dynamic input
- Compatible with HTTP, HTTPS, SOAP / REST, FTP, Database via JDBC, LDAP, Message-oriented middleware (MOM), POP3, IMAP and SMTP
- Scriptable Samplers (JSR223-compatible languages, BSF - compatible languages and BeanShell)

b) *How to record tests and set check points in win runners? Explain.* **[7 + 8]**

Ans. : There are two recording modes :

Context sensitive mode : This recording mode is utilised when the exact placement of the GUI controls (i.e. X and Y coordinates) or mouse movements are not required.

Analog mode : This recording option is employed when the mouse positions, as well as the positioning of the controls in the application, are crucial factors in the application's testing. This recording mode must be used to validate bitmaps, test signatures and so on.

The following is the technique for recording a test case :

Step 1 : Create a new document in Word : Select "New Test" from File -> New (or) File -> New (or) File -> New (or) File -> New
from the Welcome screen of WinRunner

Step 2 : Launch (run) the application that will be evaluated.

Step 3 : Begin capturing a test case.

To record in Context Sensitive mode, go to Create ->Record - Context Sensitive (or) press the "Record" button on the toolbar once.

Step 4 : Click on the title bar of the application to choose it for testing.

Step 5 : Complete all of the acts that need to be recorded.

Step 6 : Once all of the needed actions have been captured, the recording should be stopped.

Stop -> Create (or) To end the recording, click the "Stop" button on the toolbar. The script for the recoded activities is generated by WinRunner.

March - 2022

Software Testing Methodologies (R18) [156CW]

Solved Paper
III[rd] Year, B.Tech.,
Sem - II (CSE/IT)

Time : 3 Hours] [Max. Marks : 75

Q.1 a) *Discuss about implementation and application of path testing.* [8 + 7]

Ans. :

1. Integration, coverage and paths in called components :

- Unit testing, especially for new software, is the most common application.
- **In an idealistic bottom-up integration test procedure,** one component at a time is integrated. For lower-level components (subroutines), use stubs, test the interfaces and then replace the stubs with real subroutines.
- In actuality, integration takes place in blocks of related components. It is possible to avoid stubs. Paths within the subroutine must be considered.
- Predicate interpretation may force us to regard a subroutine as an in-line-code in order to achieve C1 or C2 coverage.
- The process of desensitization becomes more difficult.
- The selected path may be impassable because the processing of the called components may obstruct it.
- Path testing's flaws include the assumption that good testing can be done one level at a time without regard for what occurs at lower levels.
- Anticipate coverage issues and blinding.

2. Using path testing to test new code :

Carry out C1 + C2 coverage path tests.

- Using a mechanised test suite, use a technique similar to idealistic bottom-up integration testing.
- A defect could be indicated by a path that is blocked or unreachable.
- The path may be blocked if a bug occurs.

3. Incorporation of path testing into routine maintenance :

- Apply path testing to the updated component first, following a technique similar to idealistic bottom-up integration testing but without the use of stubs.
- Choose a path to C2 over the altered code.
- In the maintenance phase, newer and more effective techniques for providing coverage may develop.

4. Path testing in the context of rehosting :

- For rehosting outdated software, path testing with C1 and C2 coverage is a powerful tool.
- Software is rehosted because maintaining the application environment is no longer cost feasible.
- Use path testing in conjunction with structural test generators that are either automatic or semiautomatic.
- Path testing is a procedure used during the rehosting process.
- A translator is constructed and tested to translate from the old to the new environment. The purpose of the rehosting process is to identify faults in the translation software.
- For the old software, a complete C1 + C2 coverage path test suite has been constructed. The tests are carried out in the previous setting. The outcomes become the rehosted software's specs.
- A second translator may be required to adjust the exams and results to the new setting.

- Although the process is expensive, it avoids the hazards of altering the code.
- It can be optimised or enhanced for additional functionalities once it operates in a new environment (which were not possible in the old environment.)

b) *Describe instrumentation and sanitization in transaction flow testing.* **(Refer Q.2 of Chapter - 2)**

Q.2 a) *Explain about approximate number of paths with suitable example.* **(Refer Q.2 of Chapter - 3)**

b) *Discuss in detail about state bugs.* **(Refer Q.4 of Chapter - 4)** **[8 + 7]**

Q.3 a) *Describe software implementation of state graphs.* **(Refer Q.5 of Chapter - 4)**

b) *Explain about partition algorithm with example.* **(Refer Q.13 of Chapter - 5)** **[8 + 7]**

Q.4 a) *Briefly explain about consequences of bugs.* **(Refer Q.13 of Chapter - 1)**

b) *Discuss about three and four variables KV charts with example.* **(Refer Q.18 of Chapter - 3)** **[8 + 7]**

Q.5 a) *What are data flow anomalies ? Explain about dataflow anomaly state graph.*

Ans. : Data flow anomalies : Refer Q.11, point 5, Chapter - 1

Dataflow anomaly state graph : Refer Q.10 of Chapter - 2

b) *Write and explain testability tips of state testing.* **(Refer Q.15 of Chapter - 4)** **[8 + 7]**

Q.6 a) *Distinguish the following : (i) Modularity vs Efficiency :* **(Refer Q.6, point 4, Chapter - 1)**

(ii) Function vs Structure : **(Refer Q.6, point 2, Chapter - 1)**

b) *Describe motivational overview of graph matrices.* **(Refer Q.1, Chapter - 5)** **[8 + 7]**

Q.7 a) *Explain basic concepts of path products and path expressions.* **(Refer Q.1 of Chapter - 3)**

b) *Discuss about node reduction algorithm.* **(Refer Q.14 of Chapter - 5)** **[7 + 8]**

Q.8 *Explain the following : i) Testing one dimensional domains ii) Coding bugs iii) Identity elements* **[5 + 5 + 5]**

Ans. : i) Testing one dimensional domains : Refer Q.39 of Chapter - 2.

ii) Coding bugs : Refer Q.15 of Chapter - 1

iii) Identity elements : If for every a and b, if aRb and bRa, then a = b or they are the same elements, then the relation R is antisymmetric.

Examples :

1. In the binary operation :

   *: A*A->A, e is referred to as the identity of * if: a*e=e*a=a; this is referred to as the identity element of binary operations.

2. +: R*R->A, e is referred to as the identity of * if: a*e=e*a=a; that means a+e=e+a=a; and this is possible when e=0

   Therefore 0+a=a+0, Hence 0 is the identity element of addition.

3. a*e=e*a=a; this is possible when e=1. Hence 0 is the identity element of multiplication.
4. a*e=e*a=a, that means a-e=e-a=a, here no assumption for value of e. Hence subtraction has no identity element in relation R.
5. a*e=e*a=a, that means a/e=e/a=a, here no assumption for value of e. Hence division has no identity element in relation R.

END...