Software Testing
(Graph based Testing)

विद्याधनं सर्वधन प्रधानम्

Subhasis Bhattacharjee
Computer Science and Engineering,
Indian Institute of Technology, Jammu

November 8, 2025

Outline I

Graph Theory: Overview & Definitions

Graph Theory

Definition

A graph is a tuple G = (V, E) where

- V is a set of nodes vertices.
- $E \subset V \times V$ is a set of edges.

Some Notions

- Graphs can be directed or undirected.
- Graphs can be finite or infinte.
- Finite (infinite) graphs have a finite (infinite) number of vertices.
- The degree of a vertex is the number of edges that are connected to it.
- Edges connected to a vertex are said to be incident on the vertex.

Extracting Graphs for Software Testing

Graphs in Testing: Coverage Criteria

Originating Graphs from Different Artifacts

Graphs can come from different software artifacts:

- Control flow graphs
- Data flow graphs
- Call graphs

Designs modelled as finite state machines and state-charts.

- Use case diagrams
- Activity diagrams

Graphs in Software Engineering / Testing

Characteristics of Graphs from Code

Special kind of vertices:

- Graphs usually have designated special vertices like **initial** and **final** vertices.
  - Initial node indicates the beginning of a Code / Computation / Section / Block / Property.
  - Final vertices indicate end of a Code / Computation / Section / Block / Property.

Number of Special vertices

- Typically there is only **one initial vertex**,
- But, there could be **several final vertices**.

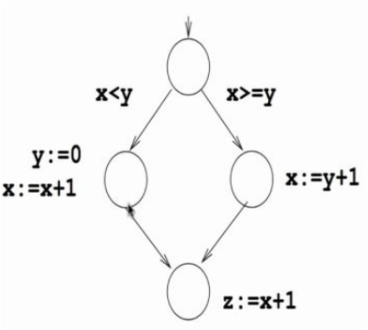
Labels & annotations:

- Graphs usually have labels associated with vertices and/or edges.
- Labels or annotations could be details about the software artifact that the graphs are modelling.

Graphs in Testing - Examples

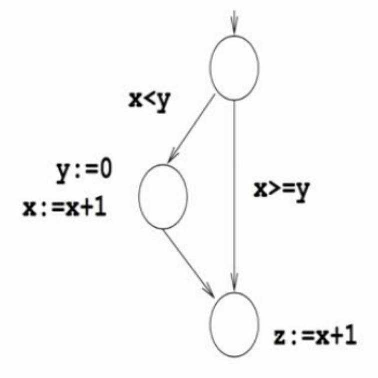
Graphs in Testing: CFG Example 1

```
if (x < y) {
  y = 0;
  x = x + 1;
} else {
  x = y + 1;
}
z = x + 1;
```



Graphs in Testing: CFG Example 2

```
if (x < y) {
  y = 0;
  x = x + 1;
}
z = x + 1;
```



Preliminary Terminologies in Graphs for Testing

Paths in Graphs

Definition (Path Related)

A path p is a sequence of vertices $v_1, v_2, \cdots, v_n$ such that $(v_i, v_{i+1}) \in E$ for $1 < i < n - 1$.

- Length of a path is the number of edges that occur in it.
  - A single vertex path has length 0.
- Sub-path of a path is a sub-sequence of vertices that occur in the path.

Paths and Reachability

Reachability - vertex, edge and sub-graph

- A vertex v is reachable in a graph G if there is a path from one of the initial vertices of the graph to v.
- An edge e = (u,v) is reachable in a graph G if there is a path from one of the initial vertices to the vertex u and then to v through the edge e.
- A sub-graph G' of a graph G is reachable if one of the vertices of G' is reachable from an initial node in G.

Reachability - Algorithms

- Depth First Search (DFS) and Breadth First Search (BFS) can be used for reachability in graphs.
- Many other reachability problems can be solved by modifying DFS/BFS algorithms.

Test Paths

Definition (Test Path Related)

A test path is a path that starts in an initial vertex and ends in a final vertex.

- Test paths represent execution of test cases.
- **Feasible paths**: Some test paths can be executed by many test cases.
- **Infeasible paths**: Some test paths cannot be executed by any test case.

Visiting and Touring

Definition (Visits & Tours)

- A test path p **visits** a vertex v, if v occurs in the path p.
- A test path p **visits** an edge e, if e occurs in the path p.
- A test path p **tours** a path q, if q is a sub-path of p.

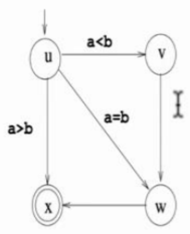
Observation

Since each vertex /edge is a sub-path, we can also say that a test path tours an edge or a vertex.

Path, Visit, Tour - On the Graph

Let the path is u, w, x, u, v in the graph.

- It visits the vertices u, w, x and v
- It visits the edges (u, w), (w, x), (x, u) and (u, v).
- It also tours the path w, x, u.



Test-Path in Graphs for Testing

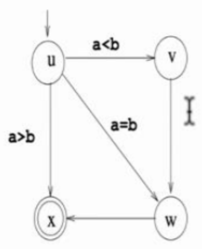
Tests and Test Paths

Notation for Test Paths

- When a test case t executes a path, we call it the test path executed by t, denoted by path(t).
- Similarly, the set of test paths executed by a set of test cases T is denoted by path(T).

Example

- Test case input: $(a = 0,\ b = 1)$,
  Test path: u, v, x, x
- Test case input: $(a = 1,\ b = 1)$,
  Test path: u, w, x
- Test case input: $(a = 3,\ b = 1)$,
  Test path: u, x



Reachability and Test Paths

Relation between Reachability and Test Paths

- A particular vertex edge /sub-graph can be reached from an initial vertex if there is a test case whose corresponding path can be executed to reach the vertex/edge/sub-graph respectively.
- Test paths that are infeasible will correspond to unreachable vertices/edges/sub-graphs.
- Several different test cases can execute the same path.

Graph Coverage Criteria

Some Terms

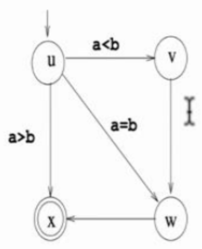
- **Test Requirement** describes properties of test paths.
- **Test Criterion** are rules that define test requirements.
- **Satisfaction**: Given a set TR of test requirements for a criterion C, a set of tests T satisfies C on a graph:
  - iff for every test requirement in $t \in TR$, there is a test path in T that meets the test requirement t.

Graph Coverage Criteria - Example 1

Set of Test Cases:

The set of test cases satisfies branch coverage at the node v in the graph.

- Test case input:(a = 1, b = 1) Test path: u, w, x
- Test case input:(a = 3, b = 1) Test path: u, x



Coverage Criteria on Graphs for Testing

Graph coverage criteria: Overview

Two different coverage criteria on Graphs

Graphs representing the software artifacts.

- Structural Coverage criteria.
- Data-flow Coverage criteria.

1. **Structural Coverage Criteria**:
    - Defined on a graph just in terms of nodes and edges
    - Representing only the structure of the graph or code
2. **Data Flow Coverage Criteria**:
    - Graph structure are annotated with references to variables / data
    - Definition, uses, modifications of variable / data
    - Testing for various kinds of referencing of data.

Structural Graph Coverage Criteria

Structural Coverage Criteria

Graphs as structures

Looking at graphs as structures representing the software artifacts (code, design elements and requirements)

- Node/vertex coverage.
- Edge coverage.
- Edge pair coverage.
- Path coverage:
  - Complete path coverage.
  - Prime path coverage.
  - Complete round trip coverage.
  - Simple round trip coverage.

Node Coverage

Node coverage - Definition

Test set T satisfies node coverage on graph G iff for every reachable node $v \in G$, there is some path p in path(T) such that p visits v.

- Requires that the test cases visit each node in the graph once.
- TR contains each reachable node in G.

Edge Coverage

Edge Coverage - Definition

Test set T satisfies edge coverage on graph G iff for every reachable edge $(u, v) \in G$, there is some path p in path(T) such that p visits u and v in order.

- Alternative Statement: TR contains each reachable edge in G path of length $= 1$, inclusive, in G.

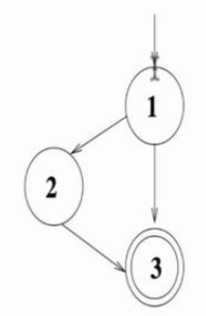
Observations

- Edge coverage is stronger than node coverage.
- Path length can be extended up to 1
  - Path length $= 0$ (It implies a node)
  - Path length $= 1$ (It implies an edge)
- Path length up to 1: It allows edge coverage for graphs with one node and no edges.
- Allowing length up to 1 allows edge coverage to subsume node coverage.

Node and Edge Coverage: Example 1

Example 1

- Node coverage: TR = {1, 2, 3},
  test path = [1, 2, 3].
- Edge coverage: TR = {(1, 2), (1, 3), (2, 3)},
  test paths = {[1, 2, 3], [1, 3]}



Edge-Pair Coverage

Covering Multiple Edges: Edge-Pair Coverage

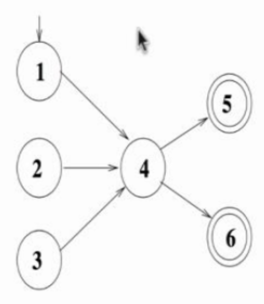
Edge-Pair Coverage (EPC) - Definition

TR contains each reachable path of length up to 2, inclusive, in G.

- Paths of length up to 2 correspond to pairs of edges.
- **length up to 2**: ensures edge-pair coverage holds for graphs with less than 2 edges.

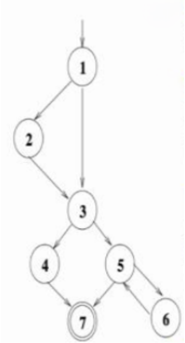
Covering Multiple Edges: Edge-Pair Coverage - Example 1

Example



- TR = {[1, 4, 5], [2, 4, 5], [3, 4, 5], [1, 4, 6], [2, 4, 6], [3, 4, 6]}.
- Test paths are the same.

Coverage Criteria - Example 2



- Node coverage:
  - TR = {1, 2, 3, 4, 5, 6, 7},
  - Test paths: {[1, 2, 3, 4, 7], [1, 3, 5, 6, 5, 7]}.
- Edge coverage:
  - TR = {(1, 2), (1, 3), (2, 3), (3, 4), (4, 7), (3, 5), (5, 6), (6, 5), (5, 7)}
  - Test paths: {[1, 2, 3, 4, 7], [1, 3, 5, 6, 5, 7]}.
- Edge-pair coverage:
  - TR = {[1, 2, 3], [1, 3, 4], [1, 3, 5], [2, 3, 4], [2, 3, 5], [3, 4, 7], $\cdots$ },
  - Test paths: {[1, 2, 3, 4, 7], [1, 2, 3, 5, 7], [1, 3, 4, 7], [1, 3, 5, 6, 5, 6, 5, 7]}.
- Complete path coverage:
  - TR = {[1, 2, 3, 4, 7], [1, 2, 3, 5, 7], [1, 2, 3, 5, 6, 5, 7], $\cdots$ }.

Covering Multiple Edges

Covering Multiple Edges - Two Variations

Basic Idea

An extension of edge-pair coverage is to consider all paths.

- **Complete path coverage**: TR contains all paths in G.
  - This can be an infeasible test requirement.
  - If a graph contains a loop, it has an infinite number of paths and hence complete path coverage is infeasible.
  - It may not be a useful test requirement.
- **Specified path coverage**: TR contains a set S of paths, where S is specified by the user/tester.

Specified Path Coverage

Basic Idea

Specified path coverage in presence of loops:

- Loops have boundary conditions and repeated executions.
- Effective test cases will not execute the loop for every iteration.
  - Execute the loop at its boundary conditions— skip the loop execution.
  - Execute the loop once for normal iterations.
- The notion of **prime paths** came into existence for working with loops.

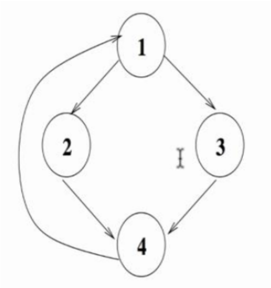
Prime Paths Coverage

Prime Paths in Graphs

Definitions

- A path from node n_i to n_j is **simple** if no node appears more than once, except possibly the first and last node.
  - No internal loops.
  - A loop is a simple path.
- A **prime path** is a simple path that does not appear as a proper subpath of any other simple path.

Simple Paths and Prime Paths: Example 1



- Simple paths: [1, 2, 4, 1], [1, 3, 4, 1], [2, 4, 1, 2], [2, 4, 1, 3], [3, 4, 1, 2], [3, 4, 1, 3], [4, 1, 2, 4], [4, 1, 3, 4], [1, 2, 4], [1, 3, 4], [2, 4, 1], 3, 4, 1], [4, 1, 2], [4, 1, 3], [1, 2], [1, 3], [2, 4], [3, 4], [4, 1], [1], [2], [3], [4]
- Prime paths: [2, 4, 1, 2], [2, 4, 1, 3], [1, 3, 4, 1], [1, 2, 4, 1], [3, 4, 1, 2], [4, 1, 3, 4] [4, 1, 2, 4], [3, 4, 1, 3]

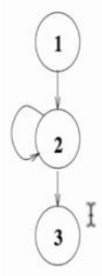
Prime Path Coverage

Prime Path Coverage - Basic Idea

TR contains each prime path in G.

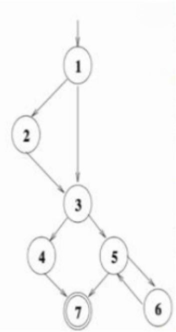
- Ensures that loops are skipped as well as executed.
- By touring all paths of length 0 and 1, it subsumes node and edge coverage.
- May or may not subsume edge-pair coverage.

Prime Path Coverage vs. Edge-Pair Coverage - Example 1



- In graphs where there are self loops, edge-pair coverage requires the self loop to be visited.
- Example:
  - TR for edge-pair coverage will be {[1, 2, 3], [1, 2, 2], [2, 2, 3], [2, 2, 2]}.
  - Some of these are prime paths/simple paths.
  - TR for prime path coverage for the above example is {[1, 2, 3], [2, 2]}.

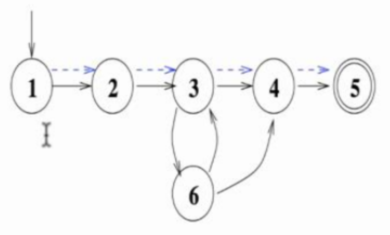
Prime Path coverage - Cover Loops - Example 2



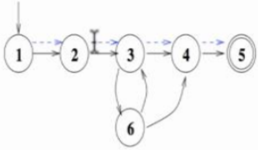
- There are nine prime paths.
- They correspond to

1, 3, 5, 7 : Skipping the loop,
1, 3, 5, 6 : Executing the loop once, and
6, 5, 6 : Executing the loop more than once.

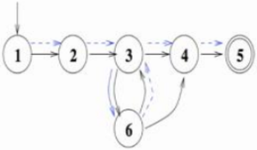
Prime Path coverage - Example 3 - P1



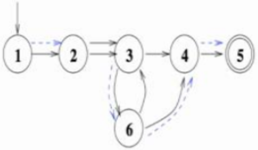
Prime Path coverage - Example 3 - P2



Touring the prime path without side trips and detours

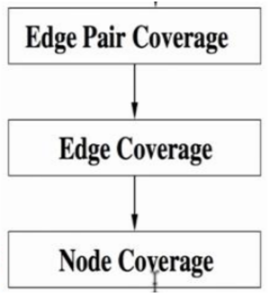


Touring the prime path with a side trip



Touring the prime path with a detour

Structural Coverage Criteria Subsumption



Data flow in graphs

Graphs with Data

- Graph models of programs can be tested adequately by including values of variables (data values) as a part of the model
- Data values are created at some point in the program and used later
- They can be created (definition) and used several times (use).
- Define coverage criteria that track a definition(s) of a variable with respect to its use(s).

Definition-Use or **Def-Use** Model

Def of a variable

A **def** is a location where a variable's value is stored into memory.

- It could be through an variable declaration, an assignment statement

Use of a variable

A **use** is a location where a value of a variable is is accessed.

- It could be assigned to another variable, be a part of an if, while or other conditions etc

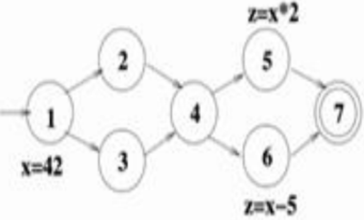
du-pairs or **def-use** pairs

A variable value is carried from its def to use(s). This is called du-pairs or def-use pairs.
A du-pair is a pair of locations (l_i, l_j), such that a variable v is defined at l_i and used at l_j.

Associating Data in graphs

- Let V be the set of variables that are associated with the program artifact being modelled as a graph.
- The subset of V that each node n (edge e) defines is called def(n) (def (e)).
- Graphs from programs don't have defs on edges
- The subset of V that each node n (edge e) uses is called as use(n) (use(e)). I

Associating Data in graphs: Example 1



- Defs:
  - $def(1) = \{x\}$,
  - $def(5) = def(6) = \{z\}$.
- Uses:
  - $use(5) = \{x\}$,
  - $use(6) = \{x\}$.

Data defs and uses

- A def of a variable may or may not reach a particular use.
  - A def of a variable v at location l_i will not reach use of v at location l_j if there is no path from l_i to l_j.
  - The value of v could be changed by another def before it reaches an use.
- A path from l_i to l_j is **def-clear** with respect to variable v, if v is not given another value on any of the nodes or edges in the path.
- If there is a def-clear path from l_i to l_j with respect to v, the def of v at l_i reaches the use at l_j.
- All the path definitions above are parameterized with G respect to a variable v.

Def-use paths or **du-path**

- A **du-path** with respect to a variable v is a simple path that is from a def of v to a use of v.
  - du-paths are parameterized by v.
  - They need to be simple paths.
  - There may be intervening uses on the path.
- du(l_i, l_j, v): The set of du-paths from l_i to l_j for variable v.
- du(mi,v): The set of du-paths that start at l_i for variable v.

Data Flow Graph Coverage Criteria

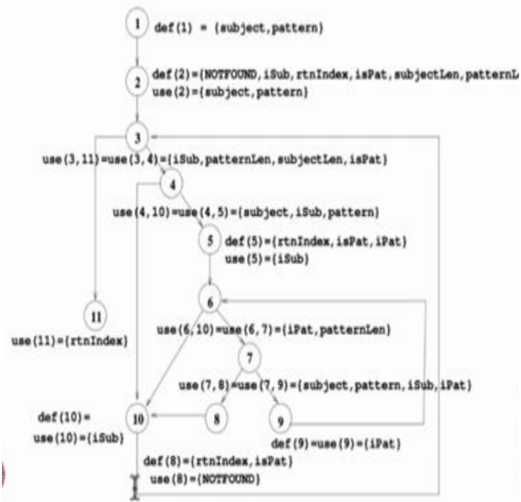
Data flow coverage criteria

- Data flow coverage criteria will be defined as sets of du-paths.
- du-paths will be grouped to define the data flow coverage criteria.
- Data flow coverage criteria defined using du-paths will check for definitions of variables reaching their uses in different ways.
- Grouping of du-paths:
  - Consider all du-paths with respect to a given variable defined in a given node.
  - The **def-path set** du(l_i, v) is set of du-paths with respect to variable v that start at node l_i.
  - For large programs, the number of def-path sets can be large.

Grouping du-paths

- Grouping du-paths as per definitions and uses allows definitions to flow to uses.
- A **def-pair set**, du(l_i, l_j, v) is the set of du-paths with respect to variable v that start at node l_i and end at node l_j.
- A def-pair set for a def at node l_i is the union of all the def-path sets for that def. du(n;,v) $= \cup_{l_j}$ du(l_i, l_j, v).

Pattern Matching example



Def-paths and def-pairs: Example

- In the pattern matching example, there is a definition of iSub at node 10.
- The following is the du-path set with respect to iSub at node 10:
  - du(10, iSub) = {[10, 3, 4], [10, 3, 4, 5], [10, 3, 4, 5, 6, 7, 8], [10, 3, 4, 5, 6, 7, 9], [10, 3, 4, 5, 6, 10], [10, 3, 4, 5, 6, 7, 8, 10], [10, 3, 4, 10], [10, 3, 11]}
- The above def-path set can be split into the following def-pair sets:
  - du(10, 4, 1Sub) = {[10, 3, 4]}.
  - du(10, 5, iSub) = {[10, 3, 4, 5]}.
  - du(10, 8, 15ub) = {[10, 3, 4, 5, 6, 7, 8]}.
  - du(10, 9, iSub) = {[10, 3, 4, 5, 6, 7, 9]}.
  - du(10, 10, iSub) = {[10, 3, 4, 5, 6, 10], [10, 3, 4, 5, 6, 7, 8, 10], [10, 3, 4, 10]}.
  - du(10, 11, iSub) = {[10, 3, 11]}.

Towards defining data-flow coverage criteria

- Like structural coverage criteria, we need to consider tours and side-trips for data flow coverage criteria too.
- This is to make the coverage criteria feasible.
- A test path pis said to **du tour** a sub-path d with respect to v if p tours d and the portion of p to which d corresponds is def-clear with respect to v.
- We can allow **def-clear side trips** with respect to v while touring a du-path, if needed.

Data flow criteria

There are three common data flow criteria:

- TR: Each def reaches *at least* one use.
- TR: Each def reaches *all possible* uses.
- TR: Each. def reaches all possible uses through *all possible* du-paths.

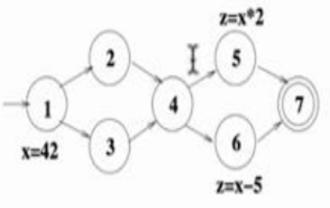
Best Effort Touring

To get test paths to satisfy these criteria, we can assume best effort touring, i.e., side trips are allowed as long as they are def-clear.

Data flow criteria

- **All-Defs Coverage**: For each def-path set S = du(l_i, v), TR contains at least one path d in S.
- **All-Uses Coverage**: For each def-pair set S = du(l_i, l_j, v), TR contains at least one path d in S.
  - Since a def-pair set du(l_i, l_j, v) represents all def-clear simple paths from a def of v at l_i to a use of v at l_i, all-uses requires us to tour at least one path for every def-use pair.
- **All-du-Paths Coverage**: For each def-pair set S = du(l_i, l_j, v), TR contains every path d in S.

Data flow coverage criteria: Example 1



- All defs for X: Test path is [1,2,4,6,7]
- All uses for X: Test paths are [1,2,4,5,7] and [1,2,4,6,7].
- All du-paths for X: Test paths are [1,2,4,5,7), 1,3,4,5,7],[1,2,4,6,7] and [1,3,4,6,7).

Data flow coverage criteria subsumption

Subsumption results for data flow coverage criteria are based on three assumptions:

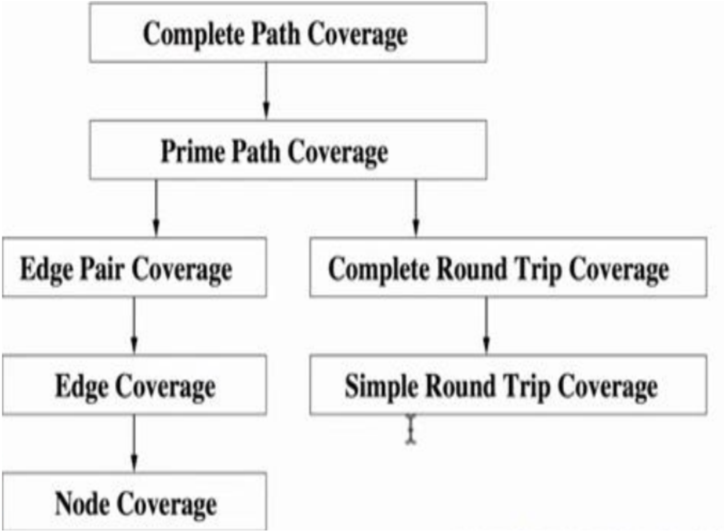
- Every use is preceded by a def.
- Every def reaches at least one use.
- For every node with multiple out-going edges, at least one variable is used on each out edge, and the same variables are used on each out edge.

Results based on above Assumptions:

- If we satisfy all-uses criteria, due to our assumption, we would have ensured that ebery def was used.
- If we satisfy all-du paths criteria, we would have ensured that every def reaches every possible use.
- Hence, all uses is also satisfied.

Graph data flow coverage criteria: subsumption

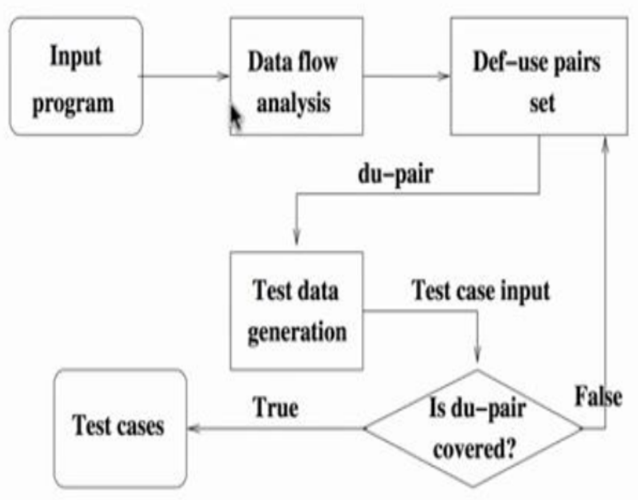
Graph coverage criteria subsumption



Graph criteria: subsumption - Conclusions

- Each edge of the graph is based on the satisfaction of some predicate. So, each edge has at least one use.
  - Hence, all uses will guarantee that each edge is executed at least once.
  - So, all-uses subsumes edge coverage.
- Each du-path is a simple path, so prime path coverage subsumes all-du-paths coverage.

Data flow testing: Process



Thank You.
End of Presentation.

Presentation Version

Date	Ver	**Remarks**
2024-06-14	v1.0	First Version.
2024-11-08	v1.1	Corrected.