Course Notes Set 10:
Structural Testing

Computer Science and Software Engineering
Auburn University

Structural Testing

- Structural testing is based on selecting paths through a code segment for execution.
- First, we assume that each code segment has a single point of entry and a single point of exit.
- A path is a sequence of instructions or statements from the entry point to the exit point.
  - There could be many paths from entry to exit for a given code segment.
- The more paths that are exercised, the more thoroughly tested the code segment is.

Path Arithmetic
(a.k.a., Path Products and Sums)

- Computation of the number of syntactic paths in a software component
- The total number of syntactic paths is denoted by P*.
- Path arithmetic may be done by inspecting the code or by using graph reduction operations on a flowgraph.
- Computing P* for a control structure:
  - Sequence - multiply
  - Decision - add
  - Iteration - exponential

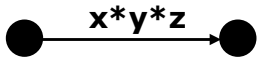
Path Arithmetic

Sequence

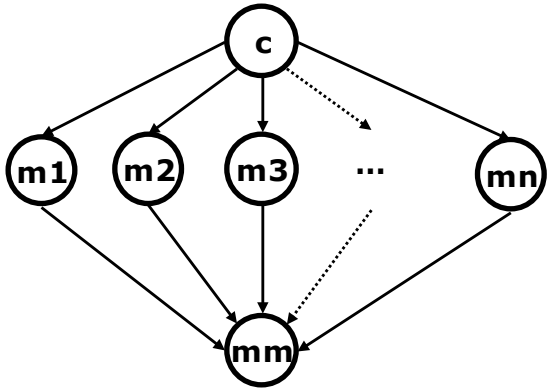
```
{
    m1 () ;
    m2 () ;
    m3 () ;
}
```

$P^* = P^*_{m1} \times P^*_{m2} \times P^*_{m3}$

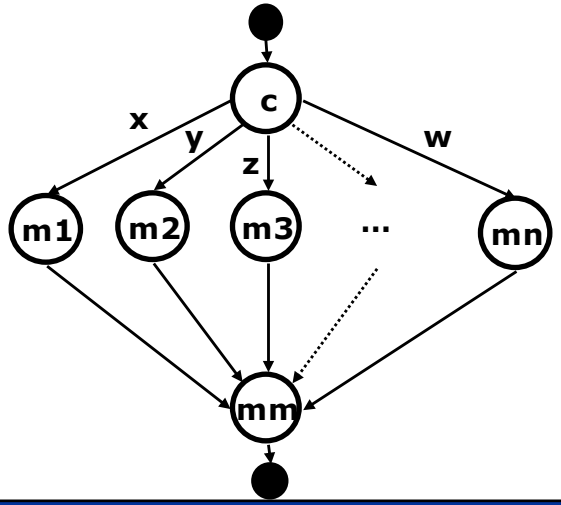
Reduces to:



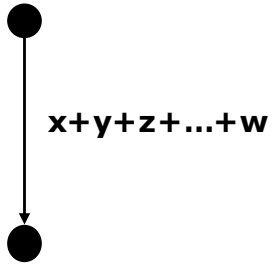
Decision



$P^* = P^*_{m1} + P^*_{m2} + \ldots + P^*_{mn}$



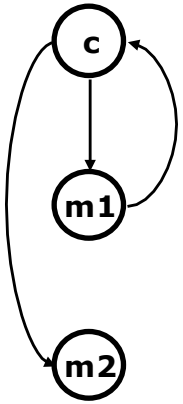
Reduces to:



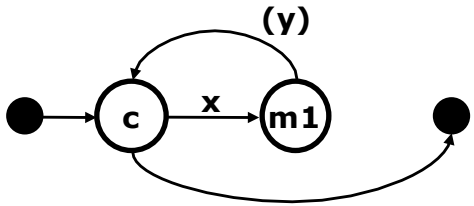
Path Arithmetic

Definite Iteration

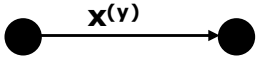
```
while (c) {
    m1();
}
m2();
```



$P^* = P^*_{m1}(\#\text{iterations})$

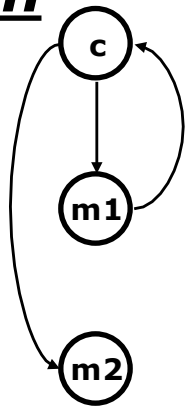


Reduces to:

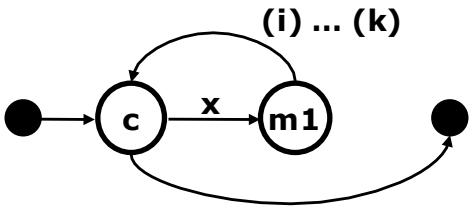


Indefinite Iteration

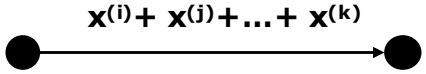
```
while (c) {
    m1();
}
m2();
```



$P^* = P^*_{m1}(\text{min } \#\text{iterations}) + \ldots + P^*_{m1}(\text{max } \#\text{iterations})$



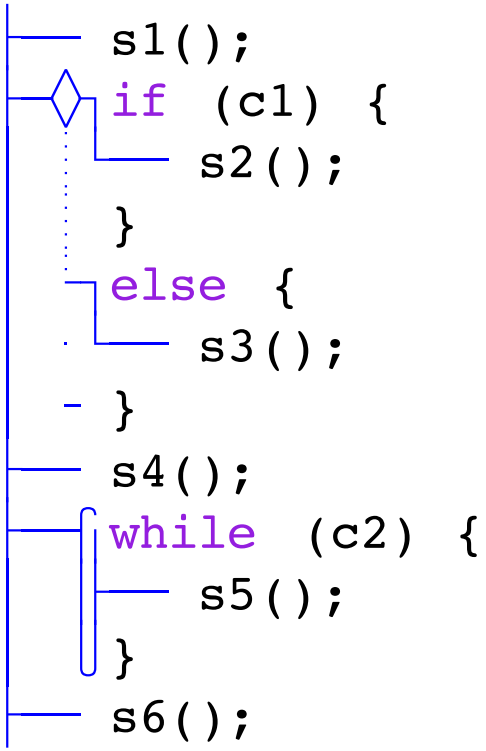
Reduces to:



Path Arithmetic - Example

```
s1();
if (c1) {
   s2();
}
else {
   s3();
}
s4();
while (c2) {
   s5();
}
s6();
```

Path Arithmetic - Example



DD-Paths

- Decision-Decision path
- Depicts control flow between decision points in a program.
- The formal definition is a bit obscure, but DD-paths are essentially either (1) single nodes or (2) maximal chains.
- Traversing all the DD-paths ensures that all nodes in the program graph have been visited.

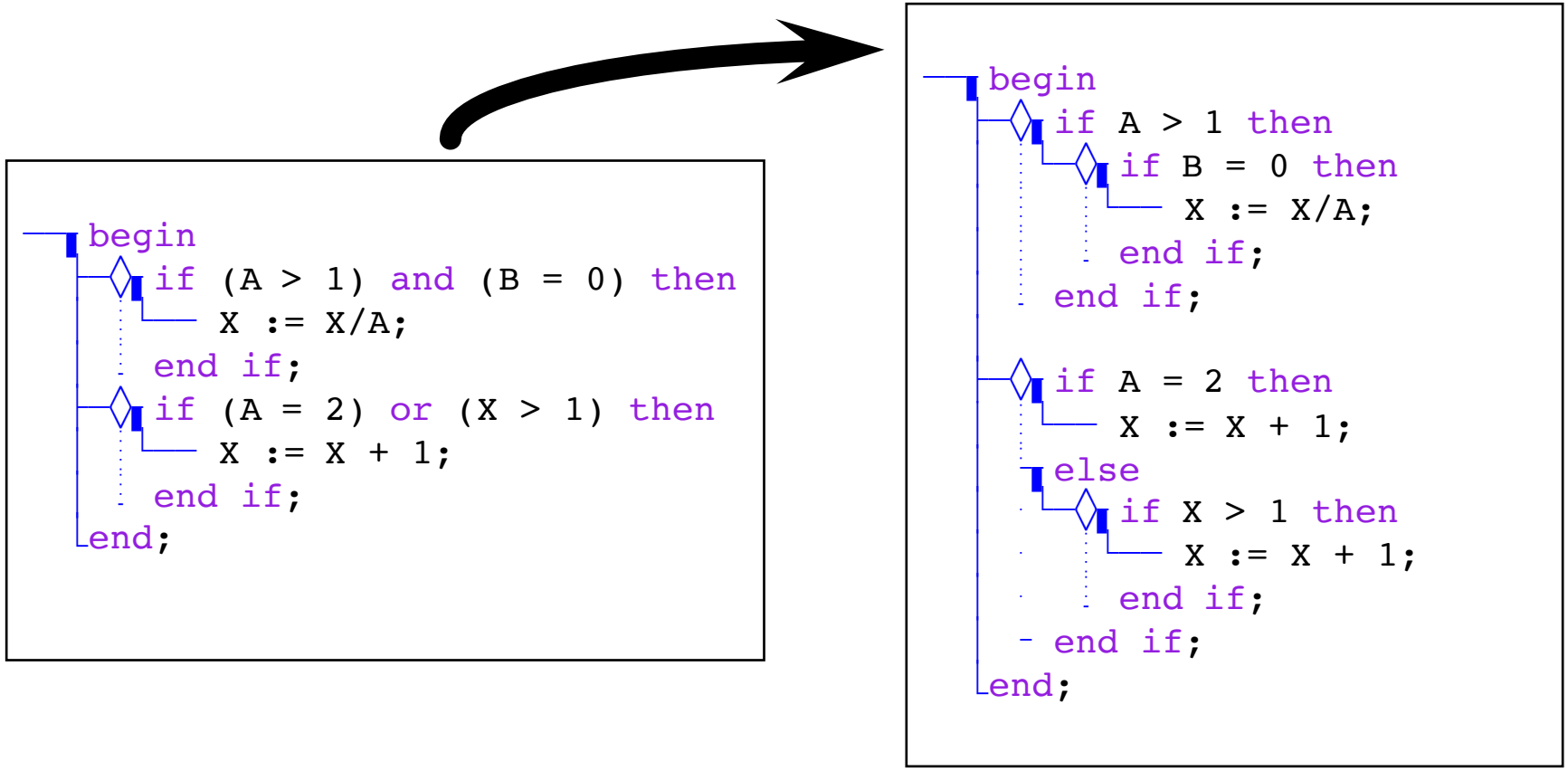
Basis Paths (IP)

- Linearly independent control flow paths through the program graph.
- McCabe applied the mathematical notion of a vector space to program execution.
- The cyclomatic number of the program graph gives the upper bound on the number of paths in the basis set.
- Traversing all the paths in a basis set of a program graph ensures that all nodes and all edges have been traversed.

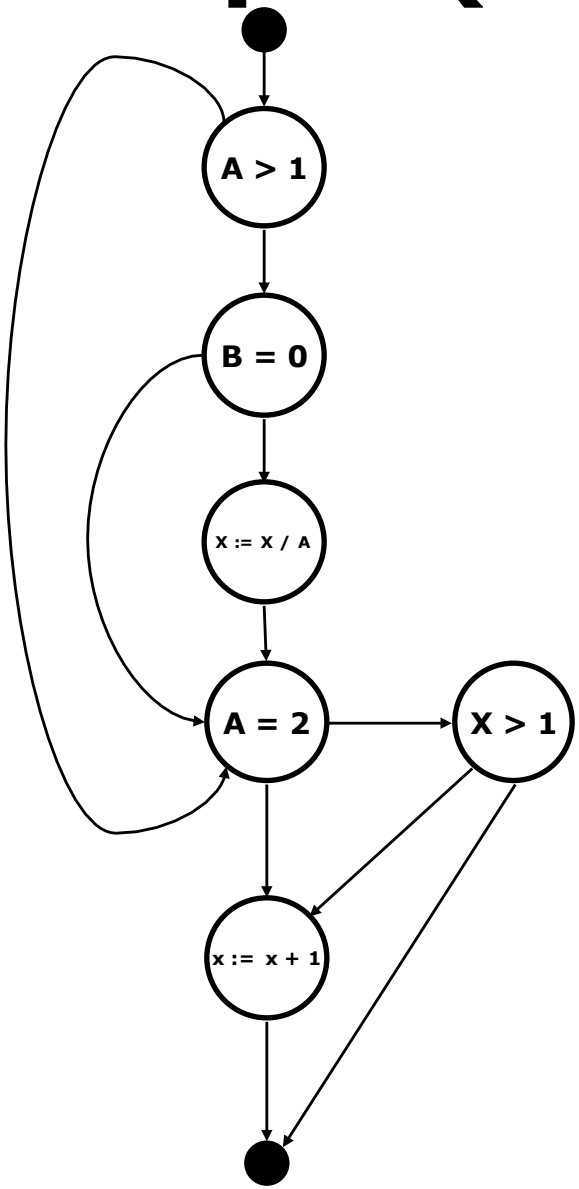
Example

```
 begin
1.     if (A > 1) and (B = 0) then
2.        X := X/A;
       end if;
3.     if (A = 2) or (X > 1) then
4.        X := X + 1;
       end if;
  end;
```

Example (cont.)



Example (cont.)



Levels of Test Coverage

- **Statement Coverage**
  - Sufficient number of test cases so that each statement is executed at least once
- **Branch Coverage**
  - Sufficient number of test cases so that each branch of each decision is executed at least once
- **Condition Coverage**
  - Sufficient number of test cases so that each condition (in each decision) takes on all possible outcomes at least once
- **Decision/Condition Coverage**
  - Sufficient number of test cases so that each condition and each decision takes on all outcomes at least once
- and on, and on, and on …
  - There are an infinite number of coverage levels [*Software Testing Techniques*, 2nd Edition, by Boris Beizer, Van Nostrand Reinhold, 1990]

Levels of Test Coverage

- **All Paths Coverage (P*)**
  - Sufficient number of test cases so that all syntactic paths are executed at least once
  - Usually infeasible, generally impossible
- **Independent Path/Basis Set Coverage**
  - Each path in the *basis set* is executed at least once
  - Independent Path - any path that introduces at least one new statement or condition outcome
  - Basis Set - A set of independent paths (not necessarily unique)
    - Can be constructed to insure decision/condition coverage

Cyclomatic Complexity and IP Coverage

- v(G)
  - Defines the maximum number of independent paths required for a basis set (assumes connected, but not strongly connected graph)
  - Determined by
    - The number of regions in the flow graph + 1
    - Number of conditions in the flow graph + 1
    - Number of edges - number of nodes + 2
- Constructing the Basis Set
  - Find the shortest path
  - Find the next path by adding as few edges as possible
  - Continue until all edges are covered

Steps in Basis Path Testing

- Using the design or code as a foundation, draw a corresponding flow graph
- Determine the cyclomatic complexity of the resultant flow graph
- Construct the basis set
- Prepare test cases that will force execution of each path in the basis set

[Adapted from *Software Engineering 4th Ed*, by Pressman, McGraw-Hill, 1997]

Test Case Generation

- Non-trivial
  - Traditionally done heuristically.
  - Some paths are not feasible.
  - Some paths must be executed as part of other paths.
- Symbolic execution
  - Derive a path predicate
    - A set of conditions “and'ed” that are required to be true in order to derive a path

Problems

- There are some fundamental problems when relying on cyclomatic complexity and basis path testing
  - Loops are not tested thoroughly
  - Data structures may not be exercised fully

Loops

- Looping to test for
  - Initialization errors
  - Index or incrementing errors
  - Bounding errors which occur at loop limits
- Classes of loops
  - Simple loop
  - Nested loop
  - Concatenated loop
  - Unstructured loop

Testing Simple Loops

- Skip the loop entirely
- Only 1 pass through the loop
- 2 passes through the loop
- n passes through the loop (n is "typical")
- m-1, m, m+1 passes through the loop (m is maximum number of loop iterations)

[Adapted from *Software Testing Techniques*, 2nd Edition, by Boris Beizer, Van Nostrand Reinhold, 1990]

Testing Other Loops

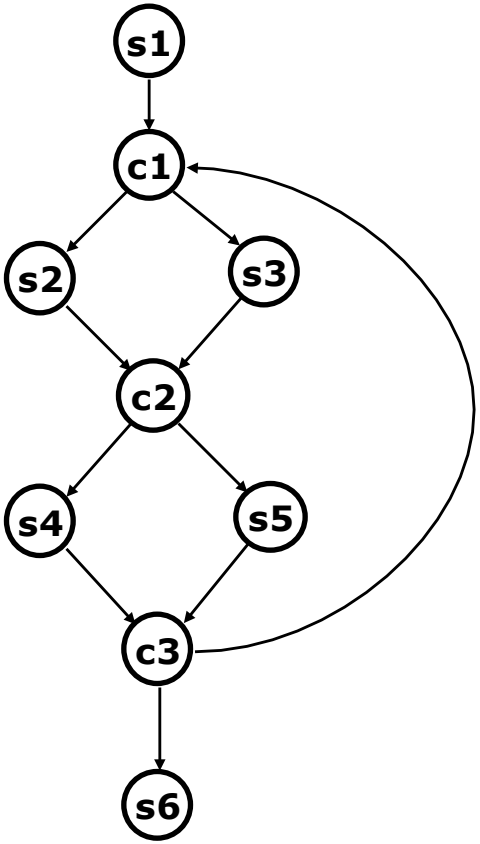
- Nested Loops
  - Start at the innermost loop, set all other loop to minimum (e.g., 1)
  - Conduct a simple loop test for the innermost loop while holding outer loops at minimum
  - Work outward, keeping outer loops at minimum and setting inner loops at typical number of iterations (when their testing is completed)
  - Continue until finished
- Concatenated Loops
  - If the loops are independent of each other, treat as simple loops
  - If the loops are dependent, then treat them as nested loops
- Unstructured Loops - REWRITE THEM!!

[Adapted from *Software Testing Techniques*, 2nd Edition, by Boris Beizer, Van Nostrand Reinhold, 1990]

Data Flow Testing

- A form of structural testing that is based not on control flow, but on how data flows through the program.
- Focuses on the points at which variables receive values and the points at which these values are used.
- Two major flavors: DU-paths and Program slices.

Why Consider Dataflow?



v(G) = 4, thus <= 4 basis paths

IP1: s1-c1-s3-c2-s4-c3-s6
IP2: s1-c1-s3-c2-s5-c3-s6
IP3: s1-c1-s3-c2-s5-c3-c1-
s3-c2-s5-c3-s6
IP4: s1-c1-s2-c2-s4-c3-s6

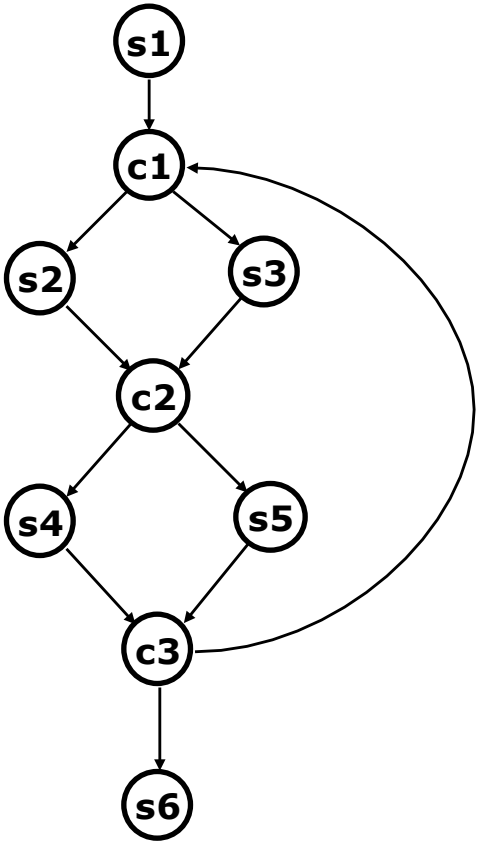
Exercising these paths ensures that all nodes and all edges are traversed at least once, but we could still be missing something.

Suppose: s2 -> "x = 0;" and
s5 -> "y = z/x;"

DU-Paths

- Node n is a defining node of variable v, written DEF(v,n), iff the value of v is defined at the statement fragment corresponding to node n.
- Node n is a usage node of the variable v, written USE(v,n), iff the value of v is used in the statement fragment corresponding to node n.
- A definition-use path (du-path) with respect to variable v is a path from node m to node n such that DEF(v,m), USE(v,n), and there exists no other node j on this path such that DEF(v,j).

DU-Paths



Since DEF(x,s2) and USE(x,s5) s2-c2-s5 is a du-path with respect to the variable x.

DU-paths provide yet more coverage levels to consider.

All-DU-Paths coverage: The test set covers all the du-paths for every variable in the program.

Program Slices

- A program slice is (informally) a set of statements that affect the value of a variable at a particular point in execution.
- Phrased in terms of a program graph, a program slice on the variable v at node n would be denoted S(v,n) and would define a set of nodes in the graph.
- Slices don't correspond directly to test cases, but can aid the testing process.
- Slices have application in software engineering beyond testing.

Why Perform Structural Testing?

- Not testing a piece of code leaves a residue of bugs in proportion to the size of the untested code and the probability of bugs.
- The high-probability paths are always thoroughly tested.
- Logic errors and fuzzy thinking are inversely proportional to the probability of a path's execution.
- The subjective probability of a path's execution as seen by its designer can be very far removed from its actual execution probability.
- The subjective evaluation of the importance of a code segment as judged by its programmer is biased by aesthetic sense, ego, and familiarity. (Elegant code might be heavily tested to demonstrate its elegance or to prove its concept, whereas straightforward code might be given only a cursory treatment.)
- Random errors (e.g. typos) can be throughout the code.

[Adapted from *Software Testing Techniques*, 2nd Edition, by Boris Beizer, Van Nostrand Reinhold, 1990]