Control Flow

Arnav Gupta

April 5, 2024

Contents

1 Program Graphs

Given a program written in an imperative programing language, its program graph is a directed graph in which

- nodes are statement fragments (or complete statements)
- edges represent flow of control

1.1 Control Flow Graphs

Models all executions of a method by describing control structures:

- nodes are statements or sequences of statements (basic blocks)
- edges are transfers of control
  - an edge (s_1, s_2) indicates that s_1 may be followed by s_2 in an execution

Basic block: sequence of statements such that if the first statement is executed all statements will be (no branches)

- intermediate statements not shown if there is ≤ 1 exiting edge and ≤ 1 entering edge
- has one entry point and one exit point

Sometimes annotated with branch predicates and defs.

1.1.1 The `if` Statement

From source code to CFG there is an issue about branching:

- in a CFG, nodes corresponding to branching should not contain any assignemnts
- `return` nodes must be distinct

Short circuiting conditions must be shown with separate branches for each condition.

1.1.2 `do`, `while`, and `for` Loops

Can have:

- dummy nodes for condition checking or initializing loops
- nodes to implicitly increment loop

`do` loops will branch at last step, `while` loops branch at first

1.1.3 `case` Structure

Cases without breaks fall through to the next case

1.1.4 `try-catch` Exceptions

Both `throw` and catching each type of exception must be shown separately.

2 Coverage

2.1 Statement Coverage

Has the following characteristics:

- achieved when all statements in a method have been executing at least once.
  - faults cannot be discovered if the parts containing them are not executed
- equivalent to covering all nodes in a CFG
- executing a statement is a weak guarantee of correctness, but easy to achieve

$$\text{Statement Coverage} = \frac{\#\text{ of executed statements}}{\text{total }\#\text{ of statements}}$$

Most used in industry with coverage target at 80-90%.

Problems include:

- predicate may be tested for only one value
- loop bodies may be iterated only once
- not all branches (cases) are necessarily covered

2.2 Segment Coverage

Counts segments (basic blocks) rather than statements

- can produce drastically different numbers since it does not account for # of statements per segment

2.3 Branch Coverage

Has the following characteristics:

- achieved when every branch from a node is executed at least once

- at least one true and one false evaluation for each predicate
- can be achieved with $D + 1$ paths in a CFG with D 2-way branching nodes and no loops
  - less if there are loops

$$\text{Branch Coverage} = \frac{\#\text{ of executed branches}}{\text{total }\#\text{ of branches}}$$

Problems include:

- short-circuit evaluation means that many predicates might not be evaluated
- compound predicate treated as a single statement
  - if n clauses, 2^n combinations but only 2 tested
- only a subset of all entry-exit paths is tested

2.4 Condition Coverage

Has the following characteristics:

- condition coverage reports the true or false outcome of each condition
- measures the conditions independetly of each other
- can fail to consider short-circuit

$$\text{Condition Coverage} = \frac{\#\text{ of conditions that are both T and F}}{\text{total }\#\text{ of conditions}}$$

2.5 Condition/Decision Coverage

Also called branch and condition coverage, it is computed by considering both branch and condition coverage measures.

2.6 Multple Condition Coverage

All combinations of condition constituents in decisions are considered when calculating coverage.

- also implies condition and branch coverage

2.7 Modified Condition/Decision Coverage (MC/DC)

Key idea: test important combinations of conditions and limiting testing costs

- extend branch and decision coverage with the requirement that each condition should affect the decision outcome independetly
- in other words, each condition should be evaluated one time to true and one time to false, and this with affecting the decision's outcome

2.8 Path Coverage

Has the following characteristics:

- test case for each possible path, though some are infeasible and number could be infinite
- key is to determine critical paths

2.9 Loop Coverage

Loops are highly fault-prone

- every loop involves a decision to traverse the loop or not
- can do boundary value analysis on the index variable

Has the following characteristics:

- minimal coverage should execute the loop body 0, 1, and 2+ times
- single loop should have more extensive coverage with setting loop control variable to:
  - min - 1, min, min + 1
  - typical
  - max - 1, max, max + 1
- for nested loop, start at the innermost loop
  - set all outer loops to min value
  - set all other loops to typical values
  - test cases for a single loop for the innermost loop

- move up in nested loop level
- if outermost loop done, do cases for single loop for all loops in the nest simultanously