EECS4313 week 4

Jerry Wu

2024-01-29

Contents

1 Structural coverage **3**
1.1 Abstract . . . 3
1.2 Basic example . . . 3
1.2.1 Test requirement & test criterion . . . 4
1.2.2 Criteria based on structures . . . 4
1.3 Measuring test sets . . . 5
1.4 Coverage . . . 5
1.4.1 Coverage level . . . 5
1.4.2 Subsumption . . . 5
1.4.3 Moral of the story . . . 6
1.5 Control flow graph definitions and terminology . . . 6
1.5.1 Subgraphs . . . 6
1.5.2 Path . . . 6
1.5.3 Subpath . . . 7
1.5.4 Test path . . . 7
1.5.5 Paths & semantics . . . 7
1.6 Reachability . . . 7

Section 1

Structural coverage

One of the biggest testing goals is maximum code coverage, and is still worked towards to this day.

1.1 Abstract

If we can have a test suite where every test case covers every statement, then we have 100% coverage, however this does not mean we have the best test suite, but the likelihood of finding bugs increases as we cover more lines of code.

"If you don't try, how can you win?" - H.V. Pham 2024

1.2 Basic example

- $floor(x)$ is some $max(n \in \mathbb{Z} \implies n \leq x)$, i.e. $floor(2.3) = 2$

```
1 public class Floor{
2     public static int floor(int x, int y){
3         if(x > y)
4             return x/y;
5         else
6           return y/x;
7     }
8 }
```

- What would we test here?

1.2.1 Test requirement & test criterion

"We need to try all the flavours" - H.V. Pham 2024

- **Test requirement**: A specific element of a software artifact that a test must satisfy or cover
  - Ice cream cone flavours: vanilla, chocolate, mint
  - Example: Test one chocolate cone
  - TR denotes a set of test requirements
- A test criterion is a rule or collection of rules that define the test requirements
  - Coverae is one recipe for generating TR in a systematic way
  - Flavour criterion [cover all flavours] (MORE GENERAL)
  - So, `TR={f=chocolate, f=vanilla, f=mint}`

1.2.2 Criteria based on structures

We have four ways to model software in testing:

- Graph (control flow graph)
- Logical expression (or, and, not, etc.)
- Input domain characterization:
  - $A : \{0, 1, n > 1\}$
  - $B : \{600, 700, 800\}$
  - $C : \{swe, cs, isa, infs\}$
- Syntactic structure

```
1 if x > y z = x - y
2 else z = 2 * x
```

1.3 Measuring test sets

- To test an ice cream shop, how do we know how good a test set is? We can choose the best test set based on widest coverage.
  - Test set 1: 1 chocolate cone
  - Test set 2: 2 chocolate, 1 vanilla
  - **Test set 3: 1 chocolate, 1 vanilla, 1 mint** - still not the best one even if we try one of each. We can still try more of each one, just don't get diabetes.

1.4 Coverage

Given a set of test requirements TR for a coverage criterion C, T satisfies C iff $\forall tr \in TR, \exists t \in T$ that satisfies tr

1.4.1 Coverage level

- Given TR and a test set T, the coverage level is the **ratio of number of test requirements satisfied by T to the size of TR**
  - $TR = f = chocolate, f = vanilla, f = mint$
  - $T_1 = chocolate(3), vanilla(1)$
  - Coverage level $= \frac{2}{3} \approx 66.7\%$
- Coverage levels help us evaluate the goodness of a test set, especially in the presence of infeasible test requirements

1.4.2 Subsumption

- Criteria subsumption: A test criterion C_1 subsumes C_2 iff **every** set of test cases that satisfies criterion C_1 also satisfies C_2
- Must be true $\forall T$, i.e. $\forall t \in C_1$, t satisfies C_2.
- A common example is that edge coverage subsumes node coverage, since if we visit every edge between two nodes, we also visit every node.
- Subsumption is a rough guide for comparing criteria, although it is hard to use in practice

1.4.3 Moral of the story

Maximum coverage can only go so far for detecting bugs. But just because we have high code coverage doesn't mean we can find every bug. At the end of the day, finding as many bugs as possible is the true goal.

1.5 Control flow graph definitions and terminology

- N: **Set of all nodes** $\{A, B, C, D, E\}$
- N_0: **Set of initial nodes** $\{A\}$ - doesn't usually happen in code
- N_f: **Set of final nodes** $\{E\}$ - possible in code since there can be multiple terminating conditions, return statements, etc.
- $E \subseteq N \times N$: **Edges**, e.g. (A, B) and (C, E), etc. C is the predecessor and E is the successor in (C, E).

1.5.1 Subgraphs

Leg G' be a subgraph if G; then

- $\forall v \in G'$, v must be a subset $N_\subseteq$ of N
- $\forall v_0 \in G'$ are $N_0 \cap N$
- The edges of G' are $E \cap (N_\subseteq \times N_\subseteq)$

1.5.2 Path

Recall that a path in a graph is a sequence of nodes from a graph G whose adjacent pairs all belong to the set of edges E of G.

- Path A: $\{2, 3, 5\}$, length is 2
- Path B: $\{1, 2, 3, 4, 5, 6\}$ length is 4

Paths have to be connected, and if the graph is directed, needs to go in the correct direction.

1.5.3 Subpath

It's just a smaller path within a bigger path. Self explanatory.

1.5.4 Test path

A test path is a path p (possibly of length 0) that starts at some $v_0 \in N_0$ and ends at some $v_f \in N_f$. In other words, it's just a simulation of how the program runs.

1.5.5 Paths & semantics

- Some paths in a control flow graph might not correspond to a program's semantics. For example:

```
1 if(false)
2     gamma();
3     return;
4 beta();
```

  In this case, β is never executed since it is a concrete false statement.

- In this course, we generally only talk about the syntax of the graph and not the semantics

1.6 Reachability

define $reach_G(x)$ as the subgraph that is syntactically reachable from x, where x is a single node, an edge, or a set of nodes and edges.