Open in Colab

SWEN90006 Tutorial 1 solutions

Task 1: find_last

1. The input domain is $int[] \times int$. That is, a set of pairs, in which the first element is an array of integers and the second element is an integer.

2. The fault is that the line `i > 0` should be `i >= 0`. The faulty version does not inspect the element at `x[0]`.

3. Any input falls in the input domain will execute the faulty statement.
4. Any case with a last element, where the element is not in position 0. Or, the element `y` is not in the list.
5. Any case in which the element is not in the list. When the loop executes, the state is in error, but will not be revealed.

Task 2: last_zero

1. The input domain is $int[]$.
2. The loop should iterate backwards instead of forwards.
3. All inputs execute the fault.
4. Any list with no zeros, or any list with exactly one zero.
5. Any non-empty list with no zeros, or any list with exactly one zero.

Task 3: count_positive

1. The input domain is $int[]$.
2. `x[i] >= 0` should be `x[i] > 0`: it should count only strictly positive elements
3. An empty list (`x = []`).
4. Any list with no occurrence of 0.
5. None!

Discussion

The point of this exercise is three-fold:

1. To illustrate the difference between testing and debugging. In SWEN90006, we cover testing, but not debugging. As part of this exercise, we did debugging to determine where the fault is.

2. To clarify the terminology of faults, failures, and errors. Faults and failures are quite often confused, and importantly, the term "bugs" is used to mean both. In SWEN90006, the term "bugs" is disallowed due to its ambiguity.

3. To illustrate an important point about coverage-based testing (which we will cover later in the subject): just executing every statement of a program is a *necessary but not sufficient* condition for a good test suite. It is not only possible to execute a statement but not trigger an error, but to trigger an error that does NOT result in a failure.

   These cases are known as *coincidental correctness*, and are exceedingly common in real-world software.