Lecture 13 - Testing III - Proving Program Correctness

Collin Roberts

October 26, 2023

Outline

1. Testing Versus Correctness Proofs
    1. Example of a Correctness Proof
    2. Correctness Proof Mini Example
    3. Correctness Proofs and Software Engineering
2. Who Should Perform Execution-Based Testing?
3. When Testing Stops

Testing Versus Correctness Proofs

Definition 1

A **correctness proof** *is a mathematical technique for demonstrating that a program is* **correct**.

Remarks

1. The text shows a technique which uses **flowcharts** to argue the correctness of a program. This technique is cute, but is not used in industry. So we will **not** spend time learning this technique.

Remarks

2. My lecture notes/slides show an example (directly stolen from CS 245) which uses the technique of **Hoare triples** (assertions inserted into the code, which assemble into a proof of program correctness). This technique is used in industry, but requires mathematical machinery (**Predicate logic**, a.k.a. **first-order logic**) which we do **not** have as a pre-requisite for CS 430. So we will not spend time learning this technique in detail either.

Remarks

3. It is enough for us to know that the Hoare triple technique can be carried out, with enough mathematical background, and patience.

Example of a Correctness Proof

Prove the total correctness of the program below, which computes a **factorial**.

$(\!|x \geq 0|\!)$

```
y = 1 ;
z = 0 ;
while (z != x) {
          z = z + 1 ;
          y = y * z ;
}
```

$(\!|y = x!|\!)$

Example of a Correctness Proof

At the `while` statement:

x	y	z	$z \neq x$
5	1	0	true
5	1	1	true
5	2	2	true
5	6	3	true
5	24	4	true
5	120	5	false

From the trace and the post-condition, a candidate **loop invariant** is $y = z!$

Example of a Correctness Proof

Here is the annotated program.

```
$(\!| x \geq 0 |\!)$
$(\!| 1 = 0! |\!)$                            assignment
y = 1 ;
$(\!| y = 0! |\!)$                            assignment
z = 0 ;
$(\!| y = z! |\!)$                            assignment
while (z != x) {
        $(\!| (y = z! \wedge z \neq x) |\!)$  partial-while
        $(\!| y(z+1) = (z+1)! |\!)$           implied (b)
        z = z + 1 ;
        $(\!| yz = z! |\!)$                   assignment
        y = y * z ;
        $(\!| y = z! |\!)$                    assignment
}
$(\!| (y = z! \wedge z = x) |\!)$             partial-while
$(\!| y = x! |\!)$                            implied (b)
```

Example of a Correctness Proof

Proof of implied (a): $\{x \geq 0\} \vdash 1 = 0!$.
This result is obvious, by definition of factorial.

Example of a Correctness Proof

Proof of implied (b):
$\{(y = z! \wedge z \neq x)\} \vdash y(z+1) = (z+1)!$.
This result is obvious.

Example of a Correctness Proof

Proof of implied (c):
$\{(y = z! \wedge z = x)\} \vdash y = x!$.
This result is also obvious.
This completes the proof of partial correctness.

Example of a Correctness Proof

Proof of Termination: The factorial code from earlier has a **loop guard** of $z \neq x$, which is equivalent to $x - z \neq 0$.

Example of a Correctness Proof

What happens to the value of $x - z$ during execution?

$(\!| x \geq 0 |\!)$

```
y = 1 ;
z = 0 ;
while (z != x) {
        z = z + 1 ;
        y = y * z ;
}
```

$(\!| y = x! |\!)$

At start of loop: $x - z = x \geq 0$ ✓

$x - z$ decreases by 1 ✓

$x - z$ unchanged

Example of a Correctness Proof

The value of $x - z$ will eventually reach 0.
The loop then exits and the program terminates. ✓
This completes the proof of total correctness.

Correctness Proof Mini Example

See the Example document.

Correctness Proof Mini Example

Moral: Even if a proof of a program's correctness has been found, the program must still be tested thoroughly.

Proposed reasons why correctness proving should not be a standard software engineering technique

1. S/W Engineers lack the mathematical training to write correctness proofs.
   Partial Refutation:
    1. This may have been true in the past.
    2. However many CS graduates today (including all from uWaterloo) do have the required mathematical background.

Proposed reasons why correctness proving should not be a standard software engineering technique

2. Correctness proving is too time consuming and hence too expensive.
   Partial Refutation:
   1. Costs can be assessed using a cost-benefit analysis, on a project-by-project basis.
   2. The benefit is weighted higher the more that correctness matters, e.g. where human lives depend on program correctness.

Proposed reasons why correctness proving should not be a standard software engineering technique

3. Correctness proving is too difficult.
   Partial Refutation:
   1. Some non-trivial S/W products have successfully been proven correct.
   2. There exists theorem-proving software to save manual work in some situations.
   3. However proving program correctness in general is an **undecidable** problem, so no theorem-prover can handle every possible situation.

Morals

1. Correctness proving is a useful tool, when human lives are at stake, or when the cost-benefit analysis justifies doing it for other reasons.
2. However correctness proving alone is not enough. Testing is still a crucial need for a S/W product.

Morals

3. Languages like Java and C++ support variations of an `assert` statement, which permits a programmer to embed assertions directly into the code. A switch then controls whether assertion checking is enabled (slower) or not (faster) at run time.
4. **Model checking** is a new technology that may eventually replace correctness proving. It is describe in Chapter 18 of the text, which unfortunately will be beyond the scope of CS 430.

Who Should Perform Execution-Based Testing?

1. Programmers should **not** have the ultimate responsibility to test their own code. **Reasons:**
    1. Fundamental conflict of motivations
        1. Coding is **constructive**.
        2. Testing's goal (exposing faults) is **destructive**.
        3. Programmers feel protective of their own code, hence they have an incentive not to expose faults in the code.
    2. The programmer may have misunderstood the specification.
        1. An SQA professional has a better chance to understand the specification correctly, and to test accordingly.

Who Should Perform Execution-Based Testing?

2. After the programmer completes and hands off the code artifact, SQA should perform **systematic testing**:

Definition 2

Systematic testing *is described by the following procedure:*

1. *Select test cases to exercise all parts of the specification.*
2. *For each test case, determine its expected output* **before execution starts***.*
3. *Execute the program on each test case, and* **record the actual results***.*
4. *Compare the actual results to the expected results.* **Document all differences***.*
5. *Correct faults (either in the specification or in the code or possibly both) which explain each difference, and repeat the execution.*
6. *Archive all test results electronically, for purposes of regression testing during future projects and post-delivery maintenance.*

Ambiguity about the term **desk checking** in the text

1. first mention (description of testing workflow): Here desk checking meant the testing that a programmer does during development. This is the meaning with which I was already familiar from my time in industry.
2. second mention (description of who should perform execution-based testing): Here desk checking means the checking of the design artifact that the programmer does before starting to code.

Who Should Perform Execution-Based Testing?

3. As outlined earlier, the SQA group must have managerial independence from the development team.

When Testing Stops

1. Only when the S/W product is decommissioned and removed from service, should testing stop.

Questions from the Class

1. Will we have to write correctness proofs like the one in the notes for this lecture? **Answer:** No.
    1. I will include a small example of the Hoare Triple technique for the next assignment, which can be done "with bare hands" (i.e. you will not need the machinery that the example uses).
    2. There will be no correctness proving on the Final Exam.