COMP 587
Spring 2020

Final Review without Answers

The following review, **in conjunction with the midterm, midterm review, assignments, and in-class handouts**, are intended to be comprehensive for the final exam. The questions below are representative of the sort of questions you'll see on the exam.

Property-Based Testing

1.) Consider the following code representing a list of integers in Java:

```
public class MyList {
  // creates a new, empty list
  public MyList() { ... }

  // returns a copy of this list
  public MyList copy() { ... }

  // returns the number of elements in this list
  public int getLength() { ... }

  // adds this element to the end of the list
  public void addElement(final int element) { ... }

  // gets the element at the given index (0-indexed)
  public int getElementAt(final int index) { ... }

  // adds all the elements of the given list to this list
  public void appendAll(final MyList other) { ... }
}
```

The property below checks that the list's length is non-negative:

```
@Property
public void list_has_non_negative_length(final MyList list) {
  assertTrue(list.getLength() >= 0);
}
```

Write at least **three** additional properties below which collectively involve getLength, addElement, getElementAt, and appendAll. You don't have to call all methods in each property. The specific syntax doesn't matter much; this isn't for any one particular library. You may put constraints on the input if you wish.

2.) Consider the following property, which has a precondition (`assume`):

```
@Property
public void add_prop(final int x, final int y) {
  assume(x > 0 && x < 2);
  assertEquals(y + 1, y + x);
}
```

When we attempt to run this property, our unit testing framework reports that the test was skipped. Why might this be happening?

3.) The properties below are intended to test `min`, with the usual intention for `min`:

```
@Property
public void result_is_an_input(final int x, final int y) {
  final int result = min(x, y);
  assertTrue(result == x || result == y);
}

@Property
public void result_is_lte_first(final int x, final int y) {
  assertTrue(min(x, y) <= x);
}
```

3.a.) The above properties are not sufficient for a correct `min` implementation. Write an incorrect definition of `min` below which will nonetheless satisfy the above two properties.

3.b.) Write an additional correct property of min, where the property would fail (in general) underneath your above definition.

Fully Automated Testing Tools and Techniques

4.) You decide to run AFL on your SUT. After 5 days, AFL finds no crashes, but it achieves 100% code coverage. How confident are you that the system is bug-free? Explain.

5.) In general, fully automated tools (e.g., AFL, concolic execution) can only find crash bugs, not arbitrary bugs. Why do these tools give up on other kinds of bugs?

6.) Why is concolic execution well-suited to test code with lots of `if` statements?

7.) Why is concolic execution poorly-suited to test code with lots of loops?

Verification

8.) Alice has verified that her SUT correctly implements its specification. Should Alice still bother with tests? Why or why not?

9.) Theoretically, can you ever be 100% sure a system works correctly? Explain.

10.) Consider the Dafny code below, which implements some basic operations on natural numbers:

```
datatype Nat = Zero | Succ(n: Nat)

predicate lessThan(n1: Nat, n2: Nat)
{
  (n1.Zero? && n2.Succ?) ||
  (n1.Succ? && n2.Succ? && lessThan(n1.n, n2.n))
}

function method add(n1: Nat, n2: Nat): Nat
{
  if (n1.Zero?) then n2 else Succ(add(n1.n, n2))
}
```

10.a.) Write a lemma that would prove that `lessThan` is transitive. You only need to worry about preconditions and/or postconditions.

10.b.) Write a lemma that would prove that `add` is commutative. You only need to worry about preconditions and/or postconditions.