S3: Testing

CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2022

Testing

Our textbook does more of this than we will discuss.

All important, but it is more on the "methodologies" side and less on the "mechanics" side.

So we will not say too much more than these brief set of slides.

Testing to support correctness

Software correctness is a goal we discussed at the very beginning of class.

We have certain principles support that this goal:

- strong static type systems
- testing, the topic of these notes
- formal reasoning for proving correctness properties, up next

Test coverage

An important question: When do we have enough tests?

"Coverage" is a concept that can help answer this question.

- Have we covered enough of the possible input values?
- Have we exercised all of the code?
- Have we exercised all parts of a Boolean condition?

  In `a && b`, do we have tests that make the condition fail because
  - `a` is false
  - `b` is false ?

Black-box testing

- The "black-box" approach to testing treats software as something whose internal structure or form cannot be observed.
- That is, in writing tests for a module we consider the interface to the module being tested, but not the implementations in the module.
- What kinds of data might we want to test with?
- Consider finding a maximal element of an `int list`.
- If the type of `maximum` is `int list -> int option` what are the "interesting" choices for test data?

  Let's do this as an in-class exercise.

Exercise S3: #1: Test data for `maximum: int list -> int option`

If the type of `maximum` is `int list -> int option` what are the "interesting" choices for test data?

-
-
-
-

This is example in 7.2 of Cornell book

Glass-box testing

- This lets us "look inside" and consider the implementation.
- Now, we might consider covering all the paths
- Consider `max3` where

```
let max3 x y z =
  if x>y then
    if x>z then x else z
  else
    if y>z then y else z
```

(from the textbook, chapter 6)

- What input would we need to exercise all paths through this function?

  Let's do this as an in-class exercise too.

Exercise S3: #2: Test data for `max3`

What input would we need to exercise all paths through this function?

```
let max3 x y z =
  if x>y then
    if x>z then x else z
  else
    if y>z then y else z
```

-
-
- ...

Types of tests

Another point of discussion about testing is the scope of the tests.

- unit tests - tests for individual components

  Do the components work "properly"?

- integration tests - testing for the composition

  Does the composition of the components work "properly"?

How to test

- Our simple framework used in the assignments.
- The OUnit system described in the textbook, section 3.3.
- Or simple `assert` expressions, see `sum.ml` in the sample programs directory.
- These can al be used in `test driven development`.

  This is a common approach in which we write the write tests first, let's try that with `sum.ml`

Knowing what to test

- Recall `product`, which was to multiply all the numbers in a list.
- We could use `assert` for that and write our tests.
- But how do we know what `product []` should be?
- We use the property:

  $\forall$ ℓ_1 `: int list`, ℓ_2 `: int list` .
  `product (` ℓ_1 `@` ℓ_2 `) = (product` ℓ_1 `) * (product` ℓ_2 `)`

Properties for a set?

- What tests might we want for sets in Hwk 04?
- What properties might have motivated these tests?
- What properties might we want to test? Let's do this as an exercise.

Exercise S3: #3: Properties for sets

- What properties should a set data type satisfy?
- These should be expressions over set operations like `insert` and `elem` similar to the property we had for `product` earlier.
- List a few:
  1.
  2.
  3.

Correctness specifications

- The properties are sometimes called "correctness specifications."
- It is not easy to know exactly what "correct" means for certain functions and programs.
- Thus we might write correctness specifications as properties that, if they do not specify everything we mean by "correct", at least give us something concrete to reason about.

Getting systems "right"

More on testing

- See CSCI 3081 and CSCI 5081
- "V & V" - verification and validation
  - Does it do the thing right?
  - Does it do the right thing?

Ethics in systems

- Ethics in AI is a big issue.
- Systems that make decisions based on data from biased humans/societies may reflect or amplify those biases.

Our interests in CSCI 2041

Our interests are in reasoning about programs, so we move on to the next topic:

S4: Reasoning about Correctness.