Assignment 0
Why3 Mechanics

15-414: Bug Catching: Automated Program Verification

Due 23:59pm, Friday, January 24, 2025
20 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope. Please carefully read the policies on collaboration and credit on the course web pages at http://www.cs.cmu.edu/~15414/assignments.html.

Working With Why3

Before you begin this assignment, you will need to install Why3 and the relevant provers. To do so, please follow the installation instructions on the course website (https://www.cs.cmu.edu/~15414/why3_install.html). To help you out with Why3, we've provided some useful commands below. We assume a simple Unix command line. If you use a Docker image, you will have to modify them as explained in the installation instructions.

- You can execute code in Why3 using

  ```
  why3 execute <filename>.mlw --use=<module> '<exp>'
  ```

  For example

  ```
  why3 execute mystery2.mlw --use=Mystery2 'g 17';
  ```

  will evaluate the expression `g 17` in the module `Mystery2` which must be contained in the file `mystery2.mlw`.

- You can check the sytax of the code and examine the verification condition with

  ```
  why3 prove <filename>.mlw
  ```

  Contrary to intuition, this command does not actually prove anything!

- To verify using the command line, run

  ```
  why3 prove -P <prover> <filename>.mlw
  ```

  for example, with `<prover> = alt-ergo`. This is useful for simple programs where more fine-grained control over the provers is unnecessary. However, your final submission should include proof sessions as created by the IDE.

- To open the Why3 IDE, run

  ```
  why3 ide <filename>.mlw
  ```

  – When you attempt to prove the goals in a file `<filename>.mlw` using the IDE, a folder called `<filename>` will be created, containing a *proof session*. Make sure that you always save the current proof session when you exit the IDE. To check your session after the fact, you can run the following two commands:

  ```
  why3 replay <filename>                # should say that session is okay
  why3 session info --stats <filename>  # prints a summary of the goals
  ```

  – Note that when using the IDE, **you should always refresh your session before retrying a proof**. This is true whether you are editing code directly in the IDE, in which case you should save your changes before refreshing, or if you are editing the code in an external editor.

What To Hand In

You should hand in the file `asst0.zip`, which you can generate by running `make`. This will include the raw `mystery2.mlw` file, as well as the proof sessions created by the IDE in the `mystery2/` directory.

If you completed the assignment successfully, you should see a screen like the following once you submit and the autograder finishes.

test_handin (__main__.ProofCheck) (0/0)

```
Successfully located all expected files and directories
```

test_replays (__main__.ProofCheck) (0/0)

```
Replay for mystery2:

received on stdout:
1/1 (replay OK)


received on stderr:
Progress: 0/1/0


return code: 0

Successfully replayed proofs
```

1 Mystery, Take 2 (20 pts)

In lecture we recognized that a mystery function computes Fibonacci numbers on nonnegative inputs. We then expressed this property with suitable pre- and post-conditions. In order to verify that the function is correct, we then needed loop invariants and also a loop variant to guarantee termination. In this problem we ask you to walk through the same process with a second mystery function g. Determine by experimentation what this function is supposed to compute, write logical contracts to express its correctness, and then verify the function in Why3. You should hand in a file `mystery2.mlw` with the completed module `Mystery2` included in `asst0.zip`.

```
module Mystery2

    use int.Int

    let g (n : int) : int =
      let ref a = -2 in
      let ref b = -1 in
      let ref c = 0 in
      while c < n do
        a <- a + 2;
        b <- a - c;
        c <- c + 1
      done;
      b

end
```