CSCI 2600 — Principles of Software

Homework 2: Reasoning About Loops and Dafny

Due: Tuesday, Feb. 18, 2025, 11:59:59 pm

Introduction

In this assignment you will prove correctness of loops using the techniques we discussed in class.

Submission Instructions

- Follow the directions in the version control handout for cloning your hw02 git repo.The URI that you will need to use to clone your personal repo for this homework would have the form of `https://submitty.cs.rpi.edu/git/s25/csci2600/hw02/RCSID` where `RCSID` is your RCS ID. Submit your answers in a single .PDF file named `hw2_answers.pdf` in the `answers/` directory of your repository.
- Submit your Dafny code as `problem1.dfy`, `problem2.dfy`, `problem3.dfy`, and `problem4.dfy` files in the `answers/` directory of your repository. Submit your answers in a single .PDF file named `hw2_answers.pdf` in the answers/ directory of your repository. **You MUST type up your answers. Handwritten solutions will not be accepted or graded, even if they are scanned into a PDF file.** We recommend using LaTeX. If you have never used LaTeX, take a look at this tutorial.
- Be sure to commit and push the files to Submitty. Follow the directions in the version control handout for adding and committing files.
- Important: You must click the **Grade My Repository** button for you answers to be graded. If you do not, they will not be graded and you will receive a zero for the homework.

Problems

All answers should be placed in `answers/hw2_answers.pdf` except for Dafny files which should be separate files with names as given in each problem description.

Problem 1 (2 pts.): Code Verification with Dafny

Below we give, in Dafny syntax, an outline of the `psp` method with the accompanying specification. You need to write Dafny code and any necessary annotations to implement `psp` and make sure that Dafny verifies it. You must not add or change anything in the specification that is given to you. You are not allowed to use loops for this problem.

```
method psp(x: int, y:int) returns (x': int, y': int)
  requires x > 0
  requires y > 0
  ensures x + y == 2 * x' + y'
  ensures (x' % 2 == 1) || (y' % 2 == 0)
{
  // Write a suitable implementation of psp.
  // You must not add or change anything in the specification!
}
```

Submission requirements

- **Do NOT** include method `Main` in your Dafny code that you submit on Submitty. If you used `Main` method for testing, make sure you comment it out before submitting on Submitty.
- The method must be named `psp` and have the header, the postcondition, and no precondition as given in the code above.
- Verify your code with Dafny before submitting.
- Submit your Dafny code with the implementation of `psp` as a file named `problem1.dfy` in the `answers/` directory.

Problem 2 (12 pts.): Sum of natural numbers

Below we give, in Java syntax, the `altSum` method which computes the alternating sum of the n natural numbers from 1 to n. For example, $altSum(0) = 0$, $altSum(1) = -1$, $altSum(2) == 1$, $altSum(3) == -2$, and $altSum(6) = 3$.

```
// Precondition: n >= 0
int altSum(int n)
{
        int i = 0;
        int t = 0;
        while (i < n)
        {
                i++;
                if (i % 2 == 0)
                {
                        t = t + i;
                }
                else
                {
                        t = t - i;
                }
        }
        return t;
}
// Postcondition: n % 2 = 0 ==> t = (n / 2) &&
//   && n % 2 = 1 ==> t = -1 * ((n + 1) / 2)
```

a) Find a suitable loop invariant. (2 pt.)

b) Show that the invariant holds before the loop (base case). (1 pt.)

c) Show by induction that if the invariant holds after k-th iteration, and execution takes a k+1-st iteration, the invariant still holds (inductive step). (3 pts.)

d) Show that the loop exit condition and the loop invariant imply the postcondition $n\%2 = 0 ==> t = (n/2) \wedge n\%2 = 1 ==> t = -1 * ((n + 1)/2)$. (2 pts.)

e) Find a suitable decrementing function. Show that the function is non-negative before loop starts, that it decreases at each iteration and that when it reaches 0 the loop is exited. (2 pts.)

f) Implement the sum of natural numbers in Dafny. (2 pts., autograded)

Submission requirements

- **Do NOT** include method `Main` in your Dafny code that you submit on Submitty. If you used `Main` method for testing, make sure you comment it out before submitting on Submitty.
- Method that implements the sum of natural numbers must be named `altSum` and have the following header: `method altSum(n: int) returns (t: int)`
- Make sure to include the precondition and the postcondition, as well as your invariant and the decrementing function.
- Verify your code with Dafny before submitting.
- Submit your Dafny code as a file named `problem2.dfy` in the `answers/` directory.

Problem 3 (18 pts.): Loopy square root

Below we give, in Dafny syntax, the square root method which should be computing the square root of a number.

```
method loopysqrt(n:int) returns (root:int)
    requires n >= 0
    ensures root * root >= n && (root - 1) * (root - 1) < n
    {
        root := 0;
        var a := n;
        while (a > 0)
            //decreases //FILL IN DECREMENTING FUNCTION HERE
            //invariant //FILL IN INVARIANT HERE
            {
                root := root + 1;
                a := a - (2 * root - 1);
            }
    }
```

a) Test this code by creating the `Main()` method and calling `loopysqrt()` with arguments like $4, 25, 30$, etc. to convince yourself that this algorithm appears to be working correctly. In your answer, describe your tests and the corresponding output. (2 pts.)

b) Yet, the code given above fails to verify with Dafny. One of the reasons for this is that it is actually incorrect. More specifically, this code may produce the result which does not comply with the specification. Write a test (or tests) that reveals the bug. In your answer, describe your test(s), the corresponding outputs, and the bug that you found. Also, indicate which part of the specification is violated. (2 pts.)

c) Now, make this code correct by changing the precondition. In your answer, describe the change and show the output of the re-run of the same tests you ran before. (2 pts.)

d) Does your Dafny code verify now? Why or why not? If it doesn't verify, does it mean that your code still has bugs in it? (2 pts.)

e) If your Dafny code doesn't verify, uncomment `invariant` and/or `decreases` annotations and supply the actual invariant and/or decrementing function. Make sure your code now verifies. In your answer, describe how you guessed the invariant and/or the decrementing function. Explain why your code was failing Dafny verification earlier but does verify now, despite the fact that you have not made any changes to your actual code (annotations are not part of the code). (2 pts.)

f) Submit your final Dafny code. (2 pts., autograded)

g) Use computational induction to prove by hand the total correctness of the final version of your Dafny code. (6 pts.)

Submission requirements

- **Do NOT** include method `Main` in your Dafny code that you submit on Submitty. If you used `Main` method for testing, make sure you comment it out before submitting on Submitty.
- Method that implements the loopy square root must be named `loopysqrt` and have the following header: `method loopysqrt(n:int) returns (root:int)`
- Make sure to include the precondition and the postcondition, as well as your invariant and the decrementing function.
- Verify your code with Dafny before submitting. Check that your postcondition is the strongest.
- Submit your Dafny code as a file named `problem3.dfy` in the `answers/` directory. You only need to submit the last version of your `loopysqrt()` (the one with your decrementing function and invariant).

Problem 4 (17 pts.): Progressive dot product

Below is the pseudocode for creating an array containing elements each of which is the dot product of input arrays up to element i:

```
Precondition: a ≠ null ∧ b ≠ null ∧ a.Length = b.Length ∧ a.Length > 0 ∧ b.Length > 0
int[] dot(int[] a, int[] b) {
  c ← new int[a.Length]
  i ← 0
  c[0] ← a[0] * b[0]
  while (i < a.Length)}
  {
    n ← a[i] * b[i]
    c[i] ← n + c[i - 1]
    i ← i + 1
  }
  return c
}
Postcondition: c ≠ null ∧ c.Length = a.Length ∧ c[0] = a[0] * b[0] ∧
  ∀ k, s.t. 0 < k < a.Length, c[k] = a[k] * b[k] + c[k - 1]
```

a) Find a suitable loop invariant. (2 pts.)

b) Show that the invariant holds before the loop (base case). (2 pts.)

c) Show by induction that if the invariant holds after k-th iteration, and execution takes a k+1-st iteration, the invariant still holds (inductive step). (5 pts.)

d) Show that the loop exit condition and the loop invariant imply the postcondition $c \neq null \wedge c.Length = a.Length \wedge c[0] = a[0] * b[0] \wedge \forall k, s.t. 0 < k < a.Length, c[k] = a[k] * b[k] + c[k-1]$. (3 pts.)

e) Find a suitable decrementing function. Show that the function is not negative before loop starts, that it decreases at each iteration and that when it reaches 0 the loop is exited. (3 pts.)

f) Implement the progressive dot product in Dafny. (2 pts., autograded)

g) Extra credit (3 pts.) Implement `dot` in Dafny using sequences instead of arrays.

Submission requirements

- **Do NOT** include method `Main` in your Dafny code that you submit on Submitty. If you used `Main` method for testing, make sure you comment it out before submitting on Submitty.
- Method that implements progressive dot product using arrays must be named `dot` and have the following header: `method dot(a:array?<int>, b:array?<int>) returns (c:array?<int>)`
- Method that implements progressive dot product using sequences instead of arrays must be named `dot` and have the following header: `method dot(a:seq<int>, b:seq<int>) returns (c:seq<int>)`
- Make sure to include the precondition and the postcondition, as well as your invariant and the decrementing function.
- Verify your code with Dafny before submitting.
- Submit your Dafny code that uses arrays as a file named `problem4.dfy` in the `answers/` directory. If you decide to do the extra credit part with sequences instead of arrays, submit it as a file named `problem4s.dfy` in the `answers/` directory.

Collaboration (0.5 pts.)

Please answer the following questions in a file named `hw2_collaboration.pdf` in your `answers/` directory.

The standard academic integrity policy applies to this homework.

State whether or not you collaborated with other students. If you did collaborate with other students, state their names and a brief description of how you collaborated.

Reflection (0.5 pts.)

Please answer the following questions in a file named `hw2_reflection.pdf` in your `answers/` directory. Answer briefly, but in enough detail to help you improve your own practice via introspection and to enable me to improve Principles of Software in the future.

- In retrospect, what could you have done better to reduce the time you spent solving this homework?
- What could we, the teaching staff, have done better to improve your learning experience in this homework?
- What do you know now that you did not know before beginning the homework?

  We will be awarding up to 1 extra credit point (at the discretion of the grader) for particularly insightful, constructive, and helpful reflection statements.

Submission

Push your repository containing the following files to Submitty:

- `answers/problem1.dfy`
- `answers/problem2.dfy`
- `answers/problem3.dfy`
- `answers/problem4.dfy`
- Optional `answers/problem4s.dfy`
- `answers/hw2_answers.pdf`
- `answers/hw2_collaboration.pdf`
- `answers/hw2_reflection.pdf`

Hints

- When trying to come up with a loop invariant for prewritten code, it often helps to trace through the execution of the code on paper. Choose a few different starting values of variables defined outside the block of code (such as method arguments), and write down the values of all the variables used in the loop for each iteration.
- Your Dafny code will be autograded by Submitty. If you are not getting full credit from the autograder, make sure to click "Show Details" in the autograding section to check the autograder output.

Errata

Check the Submitty Forum for possible updates or corrections.