18-732, Spring 2018	*Lujo Bauer, Bryan Parno*
Assignment 3	
Version No.: 1.1	*Due: April 25, 2017, 5:00PM EDT*

This assignment consists of two (2) parts. In this assignment you will:

- **part 1:** familiarize yourself with formal logic and its use as a formal language for specifying security properties of programs, and
- **part 2:** use the Dafny language and verification toolset to explore verification of properties of programs.

1 Theoretical background (15 points)

1.1 The assignment (15 points)

This part of the assignment will be done **individually**.

You will read specific chapters from "The science of programming" by David Gries, and answer questions from these chapters. The book is accessible online through the CMU library at:

`http://vufind.library.cmu.edu/vufind/Record/1494791`

The chapters you have to read, and the exercises that you will answer in your writeup are:

Chapter	Exercises	Points (Total 15)	Description
Part 0			Motivation and high level overview
Chapter 1	Question 1: (a), (b), (e), (f), (h) Question 2: (d), (e), (f) Question 3: (b) - (e), (h)	2 (0.4 each) 1 2 (0.4 each)	Propositional logic – expressing properties in propositional logic
Section 4.1			Predicates in first order logic
Section 4.2	Question 4 Question 5 Question 6: (a) - (d),(f)	1 1 5 (1 each)	Quantifiers in first order logic – expressing properties in first order logic
Chapter 6	Question 6.2.1: (a), (b), (c)	3 (1 each)	Specification of programs, preconditions and postconditions
Chapters 2, 3, 5			Optional, but recommended

1.2 Additional resources

There are plenty of resources for propositional and first order logic (both online and in the library). Feel free to consult additional references that you might find helpful. However, keep in mind the following points:

- As mentioned in the preface of "The science of programming", the book focuses on specific parts of logic that are relevant to proving properties of programs. Often, in other resources, additional topics of logic are discussed, e.g., soundness and completeness. These topics are related to properties of the logic itself and are not directly relevant to us; thus, they might be confusing at first sight. You do not have to know or understand these extra topics in order to complete this assignment.

- The book introduces first order logic, but it uses notation that is not typical. First order logic has two quantifiers: the "for all" quantifier (universal) and the "there exists" quantifier (existential). These quantifiers are typically written using the symbols $\forall$ and $\exists$, respectively. The book does not use these quantifiers: it uses $\boldsymbol{A}$ instead of $\forall$, and $\boldsymbol{E}$ instead of $\exists$. Although in your answer, either use is fine (as long as it is consistent throughout), keep this notational difference in mind if you decide to read from different resources. Also, the book introduces the *counting quantifier* $\boldsymbol{N}$, which is not typical in the syntax of first order logic but it might be useful for some of the questions.

Some additional references (seminal papers) that you might find helpful:

1. R.W. Floyd, "Assigning Meanings to Programs", Pm. Am. Math. Soc. Symp. in Applied Math., Vol. 19, J.T. Schwartz, ed. American Mathematical Society, Providence, R.I., 1967, pp. 19-31.
2. C.A.R. Hoare, "An Axiomatic Basis for Computer Programming", Comm. ACM, Vol. 12, No. 10, Oct. 1969, pp. 576-580.
3. C.A.R. Hoare, "Proof of a program: FIND", Comm. ACM, Vol. 14, No. 1, Jan. 1971, pp. 39-45.

2 Dafny (85 points)

You will need to use Dafny for this assignment.

Installation We will provide a VM with Dafny preinstalled. This will be the official version that your assignment will be graded against. If you'd like to do some of your development on your own machine, we suggest using the Dafny plugin for Visual Studio Code (`https://code.visualstudio.com/`), which runs on all major platforms and will handle installing Dafny for you. Once you have VS Code installed, select the View menu and then Extensions. Search for "Dafny", and select the extension from "University of Applied Science (HSR) Rapperswil - Switzerland". If you prefer Emacs or Vim, you can find plugins for them from Dafny's GitHub page (`https://github.com/Microsoft/Dafny`).

When using the plugin, Dafny is always verifying in the background. If you don't receive any verification errors, then that means your program is in a verifying state. There is no need to manually invoke the verifier or to attempt to compile your program.

Documentation Start by completing the official Dafny guide `https://rise4fun.com/Dafny/tutorial/guide`. Then complete the following tutorials (linked at the bottom of the guide):

1. Termination: `https://rise4fun.com/Dafny/tutorial/Termination`
2. Sequences: `https://rise4fun.com/Dafny/tutorial/Sequences`
3. Collections: `https://rise4fun.com/Dafny/tutorial/Collections`
4. Lemmas: `https://rise4fun.com/Dafny/tutorial/Lemmas`

You can also refer to the reference manual: `https://github.com/Microsoft/dafny/blob/master/Docs/DafnyRef/out/DafnyRef.pdf`

Note that Dafny continues to evolve and the documentation does not always keep with it, so don't be surprised if there are a few discrepancies along the way.

2.1 Primality test (15 points)

Several cryptographic protocols have prime number generation as one of their steps. Several algorithms exist to efficiently generate very large prime numbers. The main component of these algorithms is the primality test procedure: an algorithm that, given a positive number, decides whether it is prime. In this problem you will implement a simple and inefficient version of the primality test.

Your task

The source code for this problem can be found in the `prime.dfy` file of the handout. You are given a specification of the `prime` predicate, that returns true if and only if the integer given as an argument is prime. You need to write the method `IsPrime`, which should return true if and only if the argument is prime.

Note: We recommend that you do not try to implement an "efficient" version of the primality test, as it would quite hard to convince Dafny that it satisfies the post-condition.

2.2 Traffic engineering (30 points)

U.S. intelligence agencies have learned that the government of Blackhattia intends to destabilize the United States by employing hackers to find and exploit vulnerabilities in the traffic engineering algorithms that manage traffic lights, dynamic road signs, and other transportation infrastructure, leading to chaos and crashes. Your group has been hired as consultants by the Department of Transportation to stop this from happening. Your first assignment is to develop and verify an algorithm for correctly managing traffic lights at either end of an infrastructure-critical one-lane bridge, according to the following specifications.

Bridge controller specification

- At each end of the bridge (end A and end B), there is a traffic light that can be either red or green. The two lights can never be green at the same time.
- At each end of the bridge, there is also a sensor that can detect the number of cars currently waiting to cross. These can be modeled with counters W_a and W_b for the two ends.
- At any time, a car traveling from A to B is either waiting at A or has passed across the bridge. During a given "clock tick," the following actions happen atomically: If necessary, the lights change, and one car from one end may cross, decrementing the appropriate counter W_a or W_b. Time spent traveling across the bridge is not modeled, and there is no need to keep track of cars once they cross the bridge.
- Once a car is waiting at one end to use the bridge, it continues waiting until it crosses.
- Both lights start out red, with no cars at either end.
- If there is at least one car waiting at A and no cars at B, the light at A turns green (and vice versa for end B).
- If both lights are red and cars arrive simultaneously at both ends, the light at A turns green first.
- If the light at A is green and there are cars waiting at B, no more than 5 cars may travel A to B from that time point. When either no more cars are waiting at A **or** 5 cars have crossed from A to B, the light at A turns red and the light at B turns green. If the light at A is green and no cars are waiting at B, then the light at A may stay green until a car arrives at B, from which time the 5-car limit is imposed. (And vice versa for end B). In short, this means that if any cars are waiting at one end of the bridge, and that end has a red light, that light will turn green no more than 5 clock ticks later.

Your task

The source code for this problem can be found in the `bridge.dfy` file of the handout. You are given an implementation of this bridge controller specification in Dafny. You need to specify the appropriate post- and pre-conditions for the various methods (20 points). In addition, you will also need to fill in the implementation of the predicate `Valid` (10 points).

2.3 Sum Range (40 points)

In this problem, we are interested in efficiently computing the sum of elements between two indices in an array. Given an array of integers a, and a range delimited by i (inclusive) and j (exclusive), we want to compute the value $\sum_{k=i}^{j-1} a[k]$.

We will first consider a naive implementation, and then we will move towards a more efficient program. Both implementations will be proven correct.

Your task

The source code for this problem can be found in the `sumrange.dfy` file of the handout. Your task consists of the following three parts.

Specification (10 points)

We first have to define what a sum is. We wish to define this as a recursive function `sum`, taking a sequence s of integers, and the range limits i and j as arguments, such that:

- if $i \geq j$, then: $sum(s, i, j) = 0$
- if $i < j$, then: $sum(s, i, j) = s[i] + sum(s, i + 1, j)$

Write the code of the function `sum` given in the module `Specification`. Make sure it covers all of the possible corner cases.

A Linear Loop (9 points)

Your first task will be to write a simple, inefficient algorithm to compute the sum. In particular, you will use a loop to iterate over all values between i and j. A skeleton called `query` is available in the module `Simple`. You need to write the code for this method, such that it proves the post-conditions provided, given the preconditions provided. Do not change either pre- or the post-conditions.

You may find it helpful to state and prove a lemma `sum_right` capturing the following property:

- if $i < j$, then: $sum(a, i, j) = sum(a, i, j - 1) + a[j - 1]$

Cumulative-Sum Array (21 points)

We now want to be able to answer multiple queries more efficiently. To this end, we consider the use of an auxiliary *cumulative-sum array*, written c. The i-th cell of c, i.e. $c[i]$ contains the sum $\sum_{k=0}^{i-1} a[k]$. c has length $n + 1$ when a has length n, and $c[0] = 0$.

We provide the predicate `is_cumulative_array_for(c, a)` specifying that c is a correct cumulative-sum array for a. You need to provide implementations satisfying the provided post-condition annotations for the following methods in the `CumulativeArray` module:

- `construct(a)` takes an array a, and returns the corresponding cumulative-sum array c.
- `query2(c,i,j,a)` assumes that c is a cumulative-sum array for a, and returns `sum(a, i, j)`.
- `update(c,a,i,v)` executes the update $a[i] = v$. You need to update c accordingly so that c remains a cumulative-sum array for a. You should touch as few cells of c as possible to compute the update to the cumulative-sum array.

Do not change either pre- or the post-conditions.

Hint: You may need to specify additional lemmas regarding the behaviour of `sum` to complete this part.

If you do add additional lemmas, please be sure they are verifying before you submit. Otherwise, you may lose additional points due to the way the autograder counts errors reported.

3 Submission and Grading

For Part 1, use **Canvas** to turn in your individual solution for Assignment 3. Submit your writeup as a PDF file (you can scan a handwritten solution if you choose). Your answers will be graded via GradeScope.

For Part 2, submit your solution to **Autolab** for Assignment 3. You will be working in groups of two. To create your group, use the "Group options" button under "assignment options" on **Autolab**. You should create groups in accordance to the pairing posted on Piazza.

Turn in a tar file containing exactly three files:

- `prime.dfy` — your modified file for the primality test problem.
- `bridge.dfy` — your modified file for the bridge controller problem.
- `sumrange.dfy` — your modified file for the sum-range problem.

To create the tar file, put all your files a directory and run:

```
tar czvf AndrewID-handin.tgz prime.dfy bridge.dfy sumrange.dfy
```

When the tar file is unzipped with `tar xvzf AndrewID-handin.tgz`, it should result in all your files being extracted to the current directory.

Your grade for this part will be directly correlated to the score reported by AutoLab.

4 Acknowledgments

Thanks to Jonathan Aldrich and Sanjit Seshia for ideas used in the bridge controller problem of this assignment, and Claude Marché and Arthur Charguéraud for ideas used in the sumrange problem of this assignment.