Secure Software Systems *Lujo Bauer*

Assignment 4

Due: April 9 and 14

Submission will be via **Autolab** and must be received by **11:59 PM EST on Monday, April 9th, 2014 (part a)** and **11:59 PM EST on Monday, April 14 (part b)**. You may resubmit as many times as you like before the deadline; only the last submission will be graded. Please carefully read and follow the instructions on the following pages carefully to ensure you receive maximum credit for your submission. Assignment 4 will be done in groups of two to three members. Directions to register your group will be posted on piazza.

1 Assignment overview

In this assignment, you will use the Dafny verification language to explore verification of security properties.

1.1 What is Dafny?

Description taken from Koenig and Leino: "Getting Started with Dafny."

Dafny is a language that is designed to make it easy to write correct code. This means correct in the sense of not having any runtime errors, but also correct in actually doing what the programmer intended it to do. To accomplish this, Dafny relies on high-level annotations to reason about and prove correctness of code. The effect of a piece of code can be given abstractly, using a natural, high-level expression of the desired behavior, which is easier and less error prone to write. Dafny then generates a proof that the code matches the annotations (assuming they are correct, of course!). Dafny lifts the burden of writing bug-free code into that of writing bug-free annotations. This is often easier than writing the code, because annotations are shorter and more direct.

1.2 Using Dafny

One of the benefits of Dafny is that there is no installation; Dafny is available online for you to use. You can find it here: `http://rise4fun.com/dafny` To verify code in Dafny, write or paste it in the editor on the website and press the play button at the bottom of the page. The correctness of your program will be output at the bottom of the page.

NOTE: The web version of Dafny is currently experiencing a slightly weird program-cache-related bug. If you try to verify the default sample program provided at the `rise4fun` page and receive a bunch of errors about `_System.Real.RealToInt`, try inserting a space or newline somewhere in the code and pressing the play button again. This problem should be cleared up soon.

It is also possible to download a local, command-line version of Dafny at `https://dafny.codeplex.com/`. The binary runs on Windows. It's possible to run the binary on a Mac or Linux machine using Mono — see example instructions at `http://rjoshi.org/cs116/tools.html` (the "boogie" directory actually means wherever you have installed Dafny). You are welcome to try this (I have gotten it to run in Windows 7 and OSX 10.8.5), but we are not supporting it or providing additional instructions.

Dafny has its own syntax, which is similar to Java. To help you get started, there are links to Dafny resources below.

2 The assignment

Discuss each item in your writeup according to the directions given with that item (below). Instructions for submitting all your files and writeup are in Section 4 below.

Please note that different pieces of this assignment are due on different dates.

2.1 Verifying a stack – due April 9

The homework package includes `Stack1.dfy` and `Stack2.dfy`, two versions of a Stack class (partially) written in Dafny. Copy and paste this code into the Dafny editor. Your job is to add annotations (also known as "contracts") to methods and functions in each file, as necessary, until Dafny reports no errors. As part of your job, you may have also to add appropriate content to the `valid()` function of the `Stack` class.

a **Fix the Stack1 class (20 points).** Eliminate all the errors reported by Dafny within the Stack1 class. **NOTE:** All errors can be fixed by adding contracts to functions or methods of the Stack1 class, and/or by adding code within the valid() function. **Do NOT change the code within any other function or method, or you may not receive full credit.**
Turn in: A text file, named `Stack1-fix.dfy`, for which Dafny reports no errors. If you can't fix all the errors, turn in the best version that you have, and you will receive credit for the errors that you did fix. For maximum points, fix the errors as efficiently as possible; do not include stronger contracts than are necessary. In your writeup, **briefly** (1-2 sentences) describe each error that you fixed and how you fixed it. If one fix solved more than one error, explain it once and mention which errors it covered.

b **Fix the Stack2 class (20 points).** Eliminate all errors reported by Dafny within the Stack2 class. The Stack2 class is the same as the Stack1 class, except it contains an extra `testStack()` method that exercises the various methods that define the stack operation. You may want to start by including all of the same contracts you added to Stack1. **NOTE:** Again, all errors can be fixed by adding contracts to functions or methods of the Stack2 class, and/or by adding code within the valid() function. **Do NOT change the code within any other function or method, or you may not receive full credit.**
Turn in: A text file, named `Stack2-fix.dfy`, for which Dafny reports no errors. If you can't fix all the errors, turn in the best version that you have, and you will receive credit for the errors that you did fix. For maximum points, fix the errors as efficiently as possible; do not include stronger contracts than are necessary. In your writeup, **briefly** (1-2 sentences) describe each error that you fixed and how you fixed it. **Do NOT** repeat explanations for the errors you fixed in Stack1; only describe what you changed from your Stack1 solution to get rid of the additional errors found in Stack2.

2.2 Traffic engineering – part due April 9, part due April 14

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.

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

Using Dafny, write and verify software implementing the above-specified bridge controller. Your implementation must meet all of the properties described above.

The homework package includes `Bridge.dfy`, a skeleton program you will use to implement the controller. This file includes:

- Definitions of two datatypes used in the program, one for modeling car events and one for modeling traffic lights.
- A `Main` method used to test the BridgeController. This method defines a sequence of car events, and instantiates a BridgeController object. The main function passes the car events to the BridgeController's `update` method, one event at a time, and then prints out the resulting state of the universe.
- A skeleton BridgeController class containing a trivial `valid` function, an empty constructor, and an empty `update` method.

You should fill in the BridgeController class so that it meets the specification above, with Dafny reporting no errors. Your final version **must not** change the datatypes or the `Main` method, only the BridgeController class. (During testing you will probably want to test several different event sequences, or make other changes in `Main` as part of debugging; this is fine, but your final version must not depend on these changes.)

Also as part of the homework package, we have provided two example event sequences along with expected output.

Turn in by April 9 (10 points): Your writeup should include a brief description of your initial approach to defining the BridgeController class, as follows:

- The class variables your BridgeController will include, what they represent, and why you chose them.

- The `Valid` function you will enforce and how it captures the requirements of the spec.
- 1-2 sentences outlining your high-level approach to implementing the class.

The purpose of this is to ensure that you are on the right track, and not headed toward "verifying" incorrect invariants or verifying them trivially, so that you don't get an unfortunate surprise when we grade the final version of this task. (For example, it's easy to guarantee that both lights are never green at the same time if you simply never allow either light to turn green, but this will not earn you any points.)

Turn in by April 14 (50 points): A text file, named `Bridge-fix.dfy`, with your `BridgeController` implementation. You should only change things within the BridgeController class — assume that for grading, we will reset the `Main` function to the version provided in the homework package (possibly with a different event sequence.)

In your writeup, **briefly** (one paragraph) describe your high-level approach. Additionally, explain each variable, function or method you include in the `BridgeController` class, using 2-3 sentences to explain what that item is for and each associated contract. If anything has changed since you turned in your initial approach, explain what has changed and why. (**Note:** It's expected for things to have changed in small ways; if you find that you need to rethink your fundamental approach to the problem, it's probably a good idea to check in with the TAs.)

3 Resources

While working on this assignment, you may find the following resources useful.

- *Getting started with Dafny* tutorial. `http://rise4fun.com/Dafny/tutorial/`
- *Dafny Quick Reference*. `https://research.microsoft.com/en-us/projects/dafny/reference.aspx`
- *Getting started with Dafny: A Guide*. Partial overlap with the tutorial above. `https://research.microsoft.com/en-us/um/people/leino/papers/krml220.pdf`
- Dafny discussion boards. Dafny creator Rustan Leino answers questions regularly. `https://dafny.codeplex.com/discussions`
- Occasional Dafny items also appear on the general Boogie discussion boards. `https://boogie.codeplex.com/discussions`

4 Submission and grading

By April 9: Using Autolab for assignment 4a, turn in a gzip tarball containing exactly three files:

- Stack1-fix.dfy — your fix for Stack1.
- Stack2-fix.dfy — your fix for Stack2.
- hw4a.pdf – your writeupcovering Stack1, Stack2, and your initial approach to the BridgeController as directed above.

By April 14: Using Autolab for assignment 4b, turn in a gzip tarball containing exactly two files:

- Bridge-fix.dfy – your implementation of BridgeController.
- hw4b.pdf – your writeup for BridgeController, as directed above.

In each case, the gzip tarball will be renamed automatically by Autolab to your-Andrew-ID handin.tgz. To create the tarball, put all your files in a directory gXX, where XX is your group number, and run: tar -czvf handin.tgz gXX/ When the tarball is unzipped, it should result in a folder named gXX containing all your files. Parts of the assignment will be tested using an automated grading script. Therefore, you must conform to the naming conventions if you wish to receive credit.

Late submissions will incur a penalty of 20% per day (not including grace days).

5 Acknowledgements

Thanks to Jonathan Aldrich and Sanjit Seshia for ideas used in this assignment.