Carnegie Mellon University
18-732: Secure Software Systems
Spring 2014
Homework 4

John Filleau
Norman Wu
Deanna Zhu

Stack 1.dfy
The `valid()`: checks 1. the array not being null; 2.topOfStack being within bounds; 3. when stack full/empty, the topOfStack should in the last/first position of stack.
For `constructor()`: requires capacity > 0 as stack should have space. ensures that the stack is valid after creation also the actual length of stack should match capacity.
For `isEmpty()`: requires objArr != null; to eliminate dereference null pointer error.
For `isFull()`: same with isEmpty();
For `push()`: requires valid(); requires topOfStack < objArr.Length-2; ensures valid(); mainly because push can not be done when stack is full.
For `pop()`: requires valid(); requires !isEmpty(); ensures valid(); mainly because pop can not be done when stack is empty.

Stack2.dfy

There were many errors that I tried to address in the array length being no less than 1 besides the valid check in Stack 1. In the constructor, I added a “requires” precondition so the capacity would always be greater than 0. Similarly, I used valid, isFull, and isEmpty in preconditions to make sure topOfStack and objArr.Length for push or pop wouldn’t produce an error. I used valid in pre and post conditions for isEmpty and isFull to fix the assertion errors for assert(s.isEmpty()) and assert(s.isFull()). Then I added postconditions for push and pop to ensure that the elements that are pushed or popped are the correct ones (objArr[topofStack] == x), which fixed errors with the assert (assert(two == 2)).

Traffic Engineering

Class Variables:

`state: State = RR | GR | GRW | RG | RGW`

The current state of the system. The first letter indicates the light color at A, second letter is the light color at B, and optional W indicates whether there are any cars waiting at a red light.

`Wa: nat and Wb: nat`

Current car queue counters for each side of the bridge, A and B, respectively. “W” indicates that cars are “W”aiting.

`La: Color and Lb: Color`

Current light colors for each side of the bridge, A and B, respectively. “L” indicates “L”ights.

`car: NextCar`

The most recently received car, as a NextCar it is one of A, B, T, or N.

`waitTimer: nat`

Indicates how long a particular side of the bridge has had a car waiting at a RED light. When this is 5, the lights need to change. Resets to 0 when there are no cars waiting at a red light.

valid() Function:

```
var state: State;

//car counters at each side of the bridge
var Wa: nat, Wb: nat;

//previous car arrival
var car: NextCar;

//how long a queue has been waiting at the red light
var waitTimer: nat;

//light colors at each end of the bridge
var La: Color, Lb: Color;

function valid() : bool
//verifies all requirements R-B1 thru B-B8
  reads this;
{
     (validStateImplications())
  && (bothLightsNeverGreen())
}
```

```
function validStateImplications() : bool
  reads this;
{
     (state == RR || state == GR || state == GRW
        || state == RG || state == RGW)
  && (state == RR   ==> validRRConditions())
  && (state == GR   ==> validGRConditions())
  && (state == GRW  ==> validGRWConditions())
  && (state == RG   ==> validRGConditions())
  && (state == RGW  ==> validRGWConditions())
}

function validRRConditions() : bool
  reads this;
{
     (car == N)
  && (La == Lb == RED)
  && (Wa == Wb == 0)
  && (waitTimer == 0)
}

function validGRConditions() : bool
  reads this;
{
     (car == A || car == N)
  && (La == GREEN && Lb == RED)
  && (Wb == 0)
  && (waitTimer == 0)
}

function validGRWConditions() : bool
  reads this;
{
     (La == GREEN && Lb == RED)
  && (0 < Wb)
  && (0 < waitTimer <= 5)
}

function validRGConditions() : bool
  reads this;
{
     (car == B || car == N)
```

```
          && (La == RED && Lb == GREEN)
          && (Wa == 0)
          && (waitTimer == 0)
        }

        function validRGWConditions() : bool
          reads this;
        {
             (La == RED && Lb == GREEN)
          && (0 < Wa)
          && (0 < waitTimer <= 5)
        }

        function bothLightsNeverGreen() : bool
          reads this;
        {
          !(La == GREEN && Lb == GREEN)
        }
```

High Level Discussion:

The method for writing the valid function is straightforward - the eight bullet point requirements are written in terms of class variables, and we ensure our valid() function has traceability to every requirement. Unfortunately, the valid() function cannot verify requirements related to updating variables - these must be placed as ensures clauses of the update() method.