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

John Filleau
Norman Wu
Deanna Zhu

High Level Bridge-fix.dfy Approach

Where the previous valid function ensured and required that internal variables matched what expected behavior was, the actual class implementation fills in the gaps - tying these internal variables to **correct output**. First, the constructor was written to ensure the initial conditions given in the requirements document. Then, the update method was written to do in order: read input variables, update car counters, transition the state machine, set lights based on state machine, and then transition a car (if any) based on lights. Most importantly, the state machine was made to have no hidden boolean variables (AKA we prefer extra, seemingly redundant states over setting lights or transitioning the state machine based on extra boolean variables.)

Bridge-fix.dfy Variables

This is a list of the variables introduced in Bridge-fix.dfy and a short description of each.

`state: State`

State is one of RR, GR, RG, GRW, and RGW - indicating the colors of the lights on sides A and B, respectively, and additionally whether there are cars waiting at the red light (indicated by the presence of a W). This variable reflects the system's current state.

`nextState: State`

When transitioning states, we use this temporary holding variable so we can compare the current state and the next state. The main purpose is that a state transition from GRW to RGW or vice-versa requires us to set `waitTimer` to 0, and comparing these two state variables allows that if required.

`Wa: nat, Wb: nat`

These are the respective counters for side A and side B of the bridge. When a car arrives at each side they are incremented, and when the light on a side is GREEN, they are decremented if any cars exist to move thru the lights.

`car: NextCar`

This variable simply mirrors the input to `update()` so that we can apply the `verify()` function to its pre and post conditions.

`waitTimer: nat`

This is the timer that prevents any light from being RED more than five turns while a car is waiting. Incremented whenever `nextState` is RGW or GRW. Set to 0 whenever the state changes.

`La: Color, Lb: Color`

These variables mirror the light color outputs. They exist entirely so that light color can be checked in the `verify()` function.

Bridge-fix.dfy Functions

This is a list of the functions introduced in Bridge-fix.dfy and a short description of each.

`valid()`
: Provides a weak contract that ensures internal state is where it “should” be. Unfortunately cannot ensure that variables updated correctly, as the `old()` method cannot be invoked in functions. Checks two subfunctions: `validStateImplications()` and `bothLightsNeverGreen()`.

`validStateImplications()`
: Provides mapping to the 5 sub functions the check valid state conditions using the implies operator (==>) so that if we are in state X, check the function `validXConditions()`.

`validRRConditions()`
: Checks that there are no cars and all lights are RED.

`validGRConditions()`
: Checks that the lights are GREEN/RED, respectively, and no cars are waiting at side B.

`validGRWConditions()`
: Checks that the lights are GREEN/RED, respectively, and cars are waiting at side B.

`validRGConditions()`
: Checks that the lights are RED/GREEN, respectively, and no cars are waiting at side A.

`validRGWConditions()`
: Checks that the lights are RED/GREEN, respectively, and cars are waiting at side A.

`bothLightsNeverGreen()`
: Checks that both lights are not green at the same time.

Bridge-fix.dfy Methods

Our implementation did not add any methods, instead preferring a larger `update()` method.

Bridge-fix.dfy Changes

Our `valid()` function had no changes between the 4a and 4b handins.