Exercise 1. Traffic Lights in Promela and Spin (15 Marks)

a. (3 Marks) Model a single traffic light in Promela, using a single process. The traffic light has three lights (green, yellow, red), which you should model using three Boolean variables as shown in the listing below. Complete the process body so that the light cycles as follows:

i. Only the green light is on. (Note: The variables are already initialized accordingly.)
ii. Only the yellow light is on.
iii. Only the red light is on.
iv. Only the red and the yellow lights are on. (And then repeat as in (a).)

```promela
bool greenOn = true;
bool yellowOn = false;
bool redOn = false;
active proctype TrafficLight () {
    ...
}
```

b. (2 Marks) Specify the following invariants using a separate process with an assert statement for each property as shown in the lecture slides (see the slide titled “Assertions”).

i. “The green and yellow light must never be on at the same time.”
ii. “Never must all lights be off at the same time.”

Does your model satisfy these properties? Use Spin to verify whether the assertions hold. If any of the two properties are violated, change the model so that the model satisfies them. (Hint: You can use the atomic construct to model that sequences of statements that change the values of the Boolean variables are to be executed as an atomic sequence.) Submit the process(es) with the assertion and the fixed model, if corrections were necessary.

c. (3 Marks) Specify the following properties in LTL and check them using Spin. Your traffic light process should satisfy all the properties. If it does not, fix your model and submit the fixed version.

i. “The traffic light must always eventually turn green again.”

ii. “The traffic light must turn red infinitely often.”

iii. “Always if the traffic light is red, it must eventually turn green again.”

d. (7 Marks) Create a model of two traffic lights that guard an intersection. You can define a second traffic light process similar to the first and let it operate on three other Boolean variables for its lights as shown in the listing below. Note that here the initial state is such that the first light is showing green and the second is showing red.

```c
bool greenOn = true;
bool yellowOn = false;
bool redOn = false;
bool greenOn2 = false;
bool yellowOn2 = false;
bool redOn2 = true;

mtype = {release, block}
chan c = [0] of {bit, mtype}

active proctype Semaphor() {
    do
        :: c?0, release; c?1, block; c?1, release; c?0, block;
    od
}

active proctype TrafficLight() {
    ...
}

active proctype TrafficLight2() {
    ...
}
```

Make use of the semaphore process `Semaphor` and the channel `c` to synchronize the two traffic lights so that the properties formulated below hold. Specify the properties in LTL and verify your model with Spin.

i. “Always one of the lights must be red.”

ii. “Both lights must not be green at the same time.”

iii. “Always eventually the first traffic light will be green and always eventually the second traffic light will be green.”
Exercise 2. Crossing example in Promela and Spin (4 Marks)

Figure 1: Automaton of the controller from Assignment 1.

Have a look at the example of the train crossing in Assignment 1 (sketch, automaton of the train and the barrier). Additionally, the LTS of the controller is given in Fig. 1. The listing below describes the crossing example in Promela. Run this example in iSpin and observe that it ends in a deadlock.

a. Compare the Controller process in the Promela model and the LTS shown above. Explain why this Promela model ends in a deadlock.

b. Alter this Promela model so that no deadlock is reached.

c. Formulate the following property in LTL and use Spin to verify the model: “Always when the train is near the crossing, eventually the barriers will be closed.” (download CrossingExampleGotoVariant.pml from the website.)
```plaintext
mtype = {trainsApproaching, trainEnteredCrossing, trainLeftCrossing, closeBarrier, openBarrier};
chan sensors2controller = [0] of {mtype};
chan controller2barrier = [0] of {mtype};

active proctype Train()
    FAR:
        if :: sensors2controller!trainsApproaching; goto NEAR
    fi;

    NEAR:
        if :: sensors2controller!trainEnteredCrossing; goto EXIT
    fi;

    EXIT:
        if :: sensors2controller!trainLeftCrossing; goto FAR
    fi;

active proctype Barrier()
    OPEN:
        if :: controller2barrier?closeBarrier; goto CLOSED
    fi;

    CLOSED:
        if :: controller2barrier?openBarrier; goto OPEN
    fi;

active proctype Controller()
    CROSSINGISOPEN:
        if :: sensors2controller!trainsApproaching; goto CLOSECROSSING
    fi;

    CLOSECROSSING:
        if :: controller2barrier!closeBarrier; goto CROSSINGISCLOSED
    fi;

    CROSSINGISCLOSED:
        if :: sensors2controller!trainLeftCrossing; goto OPENCROSSING
    fi;

    OPENCROSSING:
        if :: controller2barrier!openBarrier; goto CROSSINGISOPEN
    fi;
```

Leibniz Universität Hannover, Software Engineering Group