Homework 1 Commentary
COMP3151 T2 2021

Vincent Jackson

June 28, 2022

1 Question 1: No skip

This question had two parts:

The first part asked for you to interpret why the formula

$$\Diamond(x = 1 \,\mathcal{U}\, x = 3)$$

is true when applied to the program

```
byte x = 0;

active proctype P() {
  x = 1;
  x = 2;
  x = 3;
}
```

The answer is because the formula is true for any trace in which $x = 3$ holds at some point. To show the LTL formula $\varphi \,\mathcal{U}\, \psi$, it is enough to show that either ψ or $\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi)$ is true at the current timestep. To show $\Diamond\varphi$ requires that φ be true at some point in the future. Thus, to show $\Diamond(x = 1 \,\mathcal{U}\, x = 3)$ it is enough to simply show $\Diamond(x = 3)$.

Indeed, this question illustrates the more general logical property that

$$\Diamond(\varphi \,\mathcal{U}\, \psi) \Leftrightarrow \Diamond\psi$$

The second part asked you to find a better formula for identifying the property "the value of x becomes 3 immediately after being 1".

There are a number of solutions, but the most common was one of

$$\Diamond(x = 1 \wedge (x = 1 \,\mathcal{U}\, x = 3))$$
or
$$\Diamond(x = 1 \wedge \bigcirc(x = 3)).$$

Both correctly distinguish the behaviours we want them to, but note that if you try to use them with SPIN, only the first will work, as SPIN does not have the next operator in its LTL language (unless you compile it with a certain flag). This is for a good reason, as the use of next prevents one of the key method of simplification SPIN uses to model check programs, leading to much larger models that SPIN has to check.

2 Question 2: Dekker's Algorithm

Question 2 was about extending Dekker's algorithm to three processes. The aim was for you to obtain some understanding of how hard it is to write a correct mutual exclusion algorithm for more than two processes (and that extending Dekker's algorithm doesn't really work for this).

It was also about ensuring that you can use Promela and SPIN, and that you know how to write mutex and eventual entry properties for several processes.

To get full marks for this question, I expected you to provide some generalisation of Dekker's algorithm that SPIN would compile, obeyed the limited critical reference restriction, and that contained correct LTL for the mutex and eventual entry conditions.

There was some speculation as to why eventual entry was not satisfied in many of the provided extensions. Recall that eventual entry will not be true if it is possible to deadlock.

2.1 Limited Critical Reference

There was some confusion about the *limited critical reference* restriction. To give a motivating reason, the limited critical reference restriction is a style for writing concurrent programs in Promela so that what Promela does matches what happens if you write the same program in an actual machine language like C. Promela's model of atomic execution executes each statement atomically. This means that if you use more than one shared variable in the same statement, there can be a mismatch between what Promela does and how real code runs on an actual machine.

The limited critical reference restriction ensures these models match by only allowing one shared variable to be read from or written to in any statement. Thus the level of atomicity checked by your SPIN program better matches what can be feasibly implemented on an actual machine.