Concurrent Programming (02158) - resubmission of assignment 2

Matej Majtan - s184457, Jun Chen - s202781

November 2020

1 Problem 1

In order to prevent race conditions that were caused by several threads reading and writing into the HashMap at the same time. The map is wrapped using the **Collections.synchronizedMap** method at creation time. When a car enters a tile it firstly checks whether the current position key exists in the map if not it creates a new element with a semaphore for the particular position by using **putIfAbsent** method. Then it calls **P** function for the corresponding semaphore to decrease the value by 1, and its value remains until the car leaves so that other cars waiting to enter can continue.

2 Problem 3

Semaphore **dirMutex** is used to protect variable **isDirDownward** from being read and written at the same time within function **enter**. In order to protect variable **isFirst** properly, the operation of changing value of variable **isFirst** is put in the critical region right after the declartion of local variavle **isFirstDownwards** which is protected by mutex **isFirstMutex**. Therefore only one car can have boolean variavle **isFirstDownwards** as true.

3 Problem 4

The changes made to the problem 4 mostly rely on the solution from problem 3. The main issue was the inconsistency between the *Java* implementation and the *Promela* code. These include the use of the same semaphores as in the *Java* case, which were fixed in the *Promela* implementation. Statements as the *counter* increment were protected by a semaphore (as in *Java*) and not by atomic statement in the *Promela* code and finally all changes from problem 3 were translated into the *Promela* script.