Concurrency Theory WS 2017/2018
Chair for Software Modeling and Verification
RWTH Aachen

Prof. Dr. Ir. Dr. h. c. Joost-Pieter Katoen
apl. Prof. Dr. Thomas Noll
Christoph Matheja, Kevin Batz

Concurrency Theory WS 2019/2020

— Exercise 1 —

Hand in until October 17th before the exercise class.

Exercise 1 (20 Points)

Consider the following process definition:

$$B = a.\overline{a}.B \;+\; b.\overline{b}.B$$

Draw LTS(B) and write down all necessary derivation trees for doing so.

Exercise 2 (50 Points)

We extend CCS by the sequential composition $P;Q$ of two processes P and Q. Intuitively, the *sequential composition* $P;Q$ means that first P is executed until no further rule is applicable and then Q is executed. Please find below three possible extensions of Definition 2.4 suggested by students from previous years:

1. The sequential composition $P;Q$ is defined by the following two additional rules:

$$\frac{P \xrightarrow{\alpha} \mathsf{nil}}{P;Q \xrightarrow{\alpha} Q} \qquad \frac{P \xrightarrow{\alpha} P' \quad P' \neq \mathsf{nil}}{P;Q \xrightarrow{\alpha} P';Q}$$

2. The sequential composition $P;Q$ is defined by the following two additional rules:

$$\frac{\nexists \alpha \;\nexists P' \colon P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\tau} Q} \qquad \frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q}$$

3. We define $P;Q$ as the process

$$(\overline{a}.Q \;||\; P') \;\setminus\; \{a\}\;,$$

where P' denotes the process obtained from P by replacing every occurrence of nil by $a.\mathsf{nil}$.

For every $i \in \{1,2,3\}$ and two processes P and Q, denote by $\mathsf{LTS}_i\,(P;Q)$ the semantics of $P;Q$ according to the respective alternative described above. Prove or disprove for all $i,j \in \{1,2,3\}$ with $i < j$: For all processes P and Q, $\mathsf{LTS}_i\,(P;Q)$ and $\mathsf{LTS}_j\,(P;Q)$ are isomorphic[1].

[1] Two LTS are isomorphic if and only if they are identical up to the names of states.

Exercise 3 (30 Points)

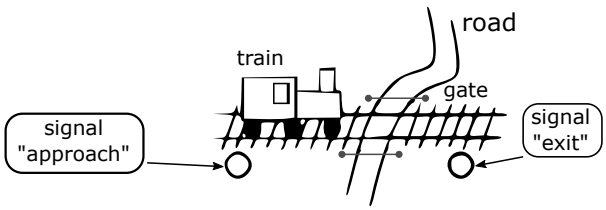

Figure 1: Railroad crossing

Model a railroad crossing with the three components train, gate, and controller, like the one shown in Figure 1, as a CCS. Please take the following considerations into account:

- The (relevant) state of the train is based on its location: The train is either *far away* from the road, it is *approaching* the road, or it is *crossing* the road.
- The gate can be *up* or *down*.
- The gate and the train do not communicate directly. The controller should make sure that the gate is *down* whenever the train is *crossing* the road.

Think about possible states that the systems can be in, on which actions systems should synchronize, and what pitfalls simple designs might hold. Be aware that the considerations above are not complete.