Prof. Dr. Andreas Podelski
Dominik Klumpp
Frank Schüssele

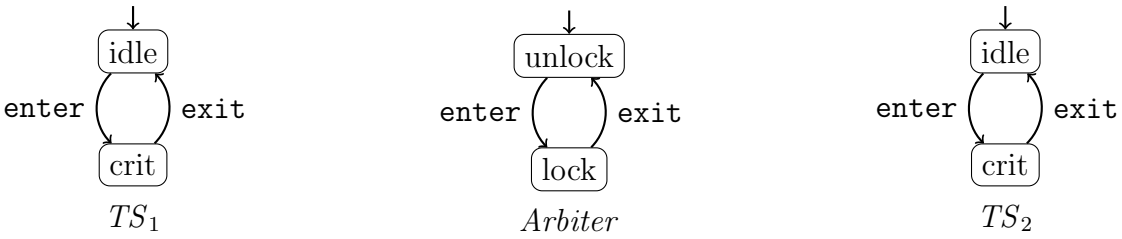
Hand in until December 9th, 2020
23:59 via ILIAS
Discussion: December 14th/15th, 2020

Tutorial for Cyber-Physical Systems - Discrete Models
Exercise Sheet 5

Exercise 1: Mutual Exclusion without Request — 6 Points

The goal of this exercise is to help you understand in detail the SOS-rules for parallel compositions.

The transition systems below describe a mutual-exclusion protocol with an arbiter. In contrast to the system discussed in the lecture, we omit the *request* action.



(a) Draw the transition system for the pure interleaving $TS_1 \,|||\, TS_2$. There must be no synchronization between the two transition systems.

For every transition in the interleaving, justify why it must exist using one of the two SOS-rules for pure interleaving.

Example: The interleaving must contain the transition $\langle \mathsf{idle}, \mathsf{idle} \rangle \xrightarrow{\texttt{enter}} \langle \mathsf{crit}, \mathsf{idle} \rangle$ due to the SOS-rule

$$\frac{\mathsf{idle} \xrightarrow{\texttt{enter}}_1 \mathsf{crit}}{\langle \mathsf{idle}, \mathsf{idle} \rangle \xrightarrow{\texttt{enter}} \langle \mathsf{crit}, \mathsf{idle} \rangle}$$

where $\to_1$ is the transition relation for TS_1. This is an instance of the first of the two SOS-rules,

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle}$$

where we set $s_1 = \mathsf{idle}$, $\alpha = \texttt{enter}$, $s_1' = \mathsf{crit}$ and $s_2 = \mathsf{idle}$.

(b) Draw the transition system for the parallel composition $(TS_1 \,|||\, TS_2) \parallel \mathit{Arbiter}$ of the transition system from (a) with the arbiter. The transition systems must synchronize (*"handshake"*) on the actions $\{\texttt{enter}, \texttt{exit}\}$.

For every transition in the composition, justify why it must exist using one of the three SOS-rules for the synchronization operator.

Exercise 2: Parallelism - Interleaving 6 Points

The goal of this exercise is to construct the interleaving of two program graphs and the corresponding transition system.

Consider the following parallel system consisting of two processes. The programs for each process are given in a low-level programming language.

Algorithm 1:	**Algorithm 2:**
$r_1 := x + 1;$	$r_2 := 3 * x;$
$x := r_1;$	$x := r_2;$

(a) Draw the program graphs P_1 and P_2 for each process.

(b) Draw the interleaving of the program graphs $P_1 \mid\mid\mid P_2$.

(c) Draw the reachable part of the transition system $\mathcal{T}_{P_1 \mid\mid\mid P_2}$. We assume that the initial value of x is 1 and the initial values of r_1 and r_2 are both 0. Use it to determine which values can finally be stored in x.

It is not strictly needed for solving the exercise above, but, for those who are interested in the motivation for the two exercises:

The above program can be seen as a model for an assembler program, namely the assembler program to which the program you have seen in the lecture (two processes consisting of the single statements $x := x + 1$ and $x := 3 * x$) is compiled.

This serves to explain that one cannot assume that an update statement in a high-level language is an atomic action.

In the exercise we just used a simplified version of the assembler code by merging the first two statements of the processes to reduce the size of the program graph.

Algorithm 1:	**Algorithm 2:**
`LOAD` x;	`LOAD` x;
`ADDI` 1;	`MULTI` 3;
`STORE` x;	`STORE` x;

Semantics of the assembler commands:

- `LOAD i` : Load the content of address i of the memory into an accumulator register ACC.
- `ADDI i` : Adds i to the content of register ACC and stores the result in ACC.
- `MULTI i` : Multiplies the content of register ACC with i and stores the result in ACC.
- `STORE i` : Store the content of register ACC at memory address i.

Note: A variable is here represented by a memory address which is a typical abstraction.