Prof. Dr. Andreas Podelski
Dominik Klumpp

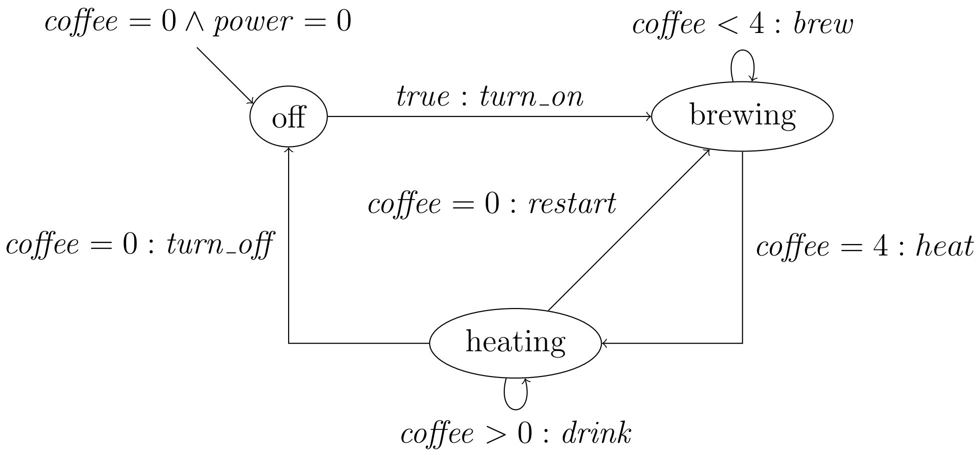
Hand in until December 11th, 2019
15:59 via the post boxes
Discussion: December 16th, 2019

Tutorial for Cyber-Physical Systems - Discrete Models
Exercise Sheet 8

Exercise 1: Coffee Machine and Transition System — 8 Points

The goal of this task is to provide some intuition on when the system described by a program graph satisfies given properties, by looking at the transition system.

The following program graph describes a simple coffee machine:



The effect of the operations is given by:

$$\begin{aligned}
\mathit{Effect}(turn\_on, \eta) &= \eta[power := 1]\\
\mathit{Effect}(turn\_off, \eta) &= \eta[power := 0]\\
\mathit{Effect}(brew, \eta) &= \eta[coffee := coffee + 1]\\
\mathit{Effect}(drink, \eta) &= \eta[coffee := coffee - 1]\\
\mathit{Effect}(restart, \eta) &= \eta\\
\mathit{Effect}(heat, \eta) &= \eta
\end{aligned}$$

(a) Give the program from which the above program graph is derived. Use the Guarded Command Language (GCL) for the program. Mark the lines of the program that correspond to the locations off, brewing, and heating.

(b) Draw the (reachable part of the) transition system corresponding to the program graph. Choose 5 transitions of the transition system, and justify their existence using the respective SOS-rule.

Use the SOS-rules to argue why the following transitions are *not* part of the transition system:

$$\langle \mathsf{off}, \{coffee \mapsto 0, power \mapsto 0\}\rangle \xrightarrow{heat} \langle \mathsf{heating}, \{coffee \mapsto 0, power \mapsto 0\}\rangle$$
$$\langle \mathsf{brewing}, \{coffee \mapsto 4, power \mapsto 1\}\rangle \xrightarrow{brew} \langle \mathsf{brewing}, \{coffee \mapsto 5, power \mapsto 1\}\rangle$$

(c) Use the transition system to check which of the following properties are true for every execution of the coffee machine.

   (i) If the machine is turned off ($power = 0$), it contains no coffee ($coffee = 0$).

   (ii) If there are two cups of coffee ($coffee = 2$), there are either three or four cups of coffee in the next step ($coffee = 3$, $coffee = 4$).

   (iii) There are always at most four cups of coffee ($coffee \leq 4$).

   (iv) The coffee machine will be turned off (i.e., in location *off*) infinitely often.

   (v) If there is no coffee ($coffee = 0$), there will be coffee after at most three steps.

Exercise 2: Arbiter with 3-way Synchronization 7 Points

The goal of this exercise is to gain an understanding how the different parallel composition operators behave.

In the lecture we considered a system for mutual exclusion with an arbiter. The system was composed of two transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ as well a transition system *Arbiter*, and we considered the parallel composition $(\mathcal{T}_1 \mid\mid\mid \mathcal{T}_2) \parallel_{Syn} Arbiter$ where $Syn = \{\texttt{enter}, \texttt{release}\}$. In this exercise, we will consider alternative ways to compose these components.

(a) Draw the parallel composition $(\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2) \parallel_{Syn} Arbiter$, where Syn is as above. How many component systems can synchronize on a single transition (i.e., change their state together in one step) in this system? Does the system ensure mutual exclusion?

(b) Draw the parallel composition $\mathcal{T}_1 \mid\mid\mid \mathcal{T}_2 \mid\mid\mid Arbiter$. How many component systems can synchronize on a single transition in this system? Does the system ensure mutual exclusion?

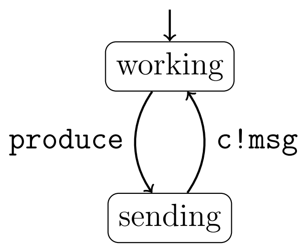
(c) Is the parallel composition $\mathcal{T}_1 \parallel \mathcal{T}_2 \parallel Arbiter$ allowed? Why/why not?

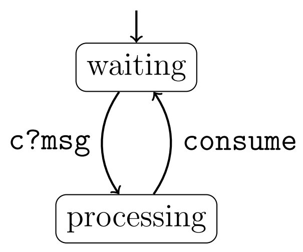
Exercise 3: Producer/Consumer 5 Points

The goal of this exercise is to understand how channel systems can be implemented.

On last week's exercise sheet, we saw that asynchronous channels can be implemented by a shared variable in a program graph. An alternative way to implement asynchronous channels is via handshaking (synchronous communication).

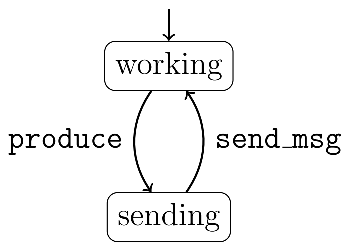
Consider as an example the two transition systems below that communicate over a channel c. They represent a simple producer/consumer system, where one component continuously produces data (here always represented by `msg`) and the other component consumes this data and processes it.

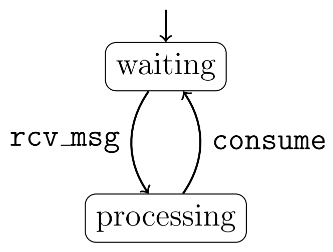




Producer *Consumer*

When we use handshaking instead, we need a third component *Channel*. We replace the actions `c?msg` and `c!msg` by action labels `send_msg` and `rcv_msg`, yielding the transition systems below:





Producer′ *Consumer′*

(a) Draw the transition system for the missing component *Channel*. Assume that the channel is reliable, but has a limited capacity of 2 messages. When the channel is full, all further sent messages are lost.

Specify how the components *Producer′*, *Consumer′* and *Channel* are composed, i.e. which parallel composition operators must be used.

(b) Now assume the channel c is unreliable: It may lose a message at any time. Furthermore, it still has a limited capacity of 2 messages and will lose all further messages. Draw the transition system for this variant of the component *Channel*. Specify how the components *Producer′*, *Consumer′* and *Channel* are composed.

(c) Above we assumed that a full channel loses all further messages sent by the sender (here *Producer′*). An alternative behaviour is that the sender blocks, and cannot send the message until there is space in the channel. How do the transition systems for the two variants of *Channel* have to change to implement this behaviour instead?