Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Leonel Merino

Series 03 — 04.10.2017 – v1.0
Safety & Synchronization

Exercise 1 (2 Points)

Answer the following questions:

a. What are safety properties? How are they modeled in FSP?

b. Is the busy-wait mutex protocol fair? Deadlock-free? Justify your answer.

c. Can you ensure safety in concurrent programs without using locks?

d. The Java language designers decided to implement concurrency based on monitors. What is the main reason behind this decision? What other options except monitors could have been chosen? (Hint: Consider slide 28 of lecture 1)

Exercise 2 (1 point)

Consider the following process definitions. Are T1 and T2 equivalent? Why?

```
R = (a->c->R).
S = (b->c->S).
||T1 = (R || S).
T2 = (a->b->c->T2|b->a->c->T2).
```

Exercise 3 (3 points)

A miniature portable FM radio has three controls. An on/off switch turns the device on and off. Tuning is controlled by two buttons, scan and reset, which operate as follows. When the radio is turned on or reset is pressed, the radio is tuned to the top frequency of the FM band. When scan is pressed, the radio scans towards the bottom of the band. It stops scanning when it locks on to a station or it reaches bottom (end). If the radio is currently tuned to a station and scan is pressed then it starts to scan from the frequency of that station towards bottom. Similarly, when reset is pressed the receiver tunes to top.

Using the alphabet $\{\mathrm{on}, \mathrm{off}, \mathrm{scan}, \mathrm{reset}, \mathrm{lock}, \mathrm{end}\}$ model the FM radio as an FSP process called RADIO and generate the corresponding LTS using the LTSA tool.

Exercise 4 (4 points)

Consider the following problem specification:

a. We have a stack that has two slots.

b. The stack operations *pop* and *push* should be atomic. In other words, the stack is locked while being *pop*ped or *push*ed.

c. The input space is 0..1 (You can only *push* 0s and 1s).

d. There is one producer process that *push*es 0s and 1s repeatedly.

e. There is one consumer that *pop*s the top of the stack.

f. Throw an error for *push*ing to a full stack or *pop*ping from an empty stack.

Write an FSP description for this problem and verify your model using LTSA.