Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Mohammadreza Hazhirpasand

Series 03 — 02.10.2019 – v1.0
Safety & Synchronization

Exercise 1 (2 Points)

Answer the following questions:

a) What does a safety property do in FSP and how do you model it?

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

c) Can you ensure safety in concurrent programs without using any kind of 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 page number 40 of the intro lecture (01) PDF available online.

Exercise 2 (1 point)

Consider the following FSP 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. Frequency 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 frequency band. When scan is pressed, the radio scans towards the bottom of the band. It stops scanning when it locks on a station or it reaches the 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 downwards. Similarly, when reset is pressed the receiver tunes to the top.

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

Exercise 4 (4 points)

Implement a stack in LTS and obey the following requirements:

a) There exists one PUSH process which provides the push functionality.

b) There exists one POP process that provides the pop functionality.

c) There exists one LOCK process that provides the locking functionality.

d) There exists one STACK process that provides the stack functionality.

e) The stack has two slots to store values.

f) The stack operations *pop* and *push* must use a lock (each acquiring the lock before and releasing it after the push/pop operation).

g) The supported data values are only 0 and 1 (*i.e.*, you can only *push* 0s and 1s).

h) An error must be thrown for *push*ing to a full stack or *pop*ping from an empty stack.

You can verify your model using the LTSA tool.

Hint: Try to first model the processes `PUSH`, `POP`, and `LOCK` on paper. After that you can assemble the `STACK` process by reusing them properly. You should not try to build only one process for the whole stack, because the result would be super complex. Instead, you can use the "COMPOSE" feature of the FSP tool which does that for you automatically.