Prof. O. Nierstrasz
Jorge Ressia, Jan Kurš

Serie 5 - Liveness and Guarded Methods

Exercise 1

Answer the following questions:

a) What is a guarded method and when should it be preferred over balking?

b) Why must you re-establish the class invariant before calling wait()?

c) What is, in your opinion, the best strategy to deal with an InterruptedException? Justify your answer!

d) How can you detect deadlock? How can you avoid it?

e) Why it is generally a good idea to avoid deadlocks from the beginning instead of relying on deadlock detection techniques?

f) Why is progress a liveness rather than a safety issue?

g) Why should you usually prefer notifyAll() to notify()?

Exercise 2

In a FSP model of a process a deadlocked state is a state with no outgoing transitions, a terminal state so to say. A process in such a state can engage in no further actions. In FSP, this deadlocked state is represented by the local process STOP. By performing a breadth-first search of the LTS graph (in CHECK PROGRESS), the LTSA tool guarantees that a sample trace is the shortest trace to the deadlock state.

Consider the maze depicted in Figure 1. Write a description of the maze as FSP process which, using deadlock analysis, finds the shortest path out of the maze starting at any square in the maze.

You may check your solution by inspecting the *trace to deadlock* from specific squares.

Hint: at each numbered square in the maze, a directional action can be used to indicate an allowed path to another square.

Exercise 3

Consider the FSP model for the dining philosphers:

```
PHIL = ( sitdown
-> right.get -> left.get -> eat
-> left.put -> right.put -> arise -> PHIL ).

FORK = ( get -> put -> FORK).

||DINERS(N=5) =
forall [i:0..N-1]( phil[i]:PHIL
|| {phil[i].left,phil[((i-1)+N)%N].right}::FORK ).
```

Modify the FSP specification to remove the deadlock.

Exercise 4

Recall your FSP process for the Russian Roulette exercise from serie 3. When you check the safety property using the LTSA tool, you should get the following output:

```
Trace to DEADLOCK:
charlton.pickup
charlton.bang
```

Modify the process to eliminate the deadlock. *Hint: The gun is also clickable without a bullet.*

Exercise 5

1. Implement your Russian Roulette process from serie 3 as a Java program. Make sure to use guarded methods: the player should block if there is no gun on the table.
2. Modify your program according to your solution from Ex. 4 to eliminate deadlock

Exercise 6

Recall the Dining Savages from serie 4, exercise 4. Implement the behaviour of the pot (which you have modelled as FSP process before) in Java.

```
const M = 5
SAVAGE = (getserving -> SAVAGE).
COOK   = (fillpot -> COOK).
POT              = SERVINGS[0],
SERVINGS[i:0..M] = (when (i==0) fillpot     -> SERVINGS[M]
                   |when (i>0 )  getserving -> SERVINGS[i-1]
                   ).
||SAVAGES = (SAVAGE || COOK || POT).
```

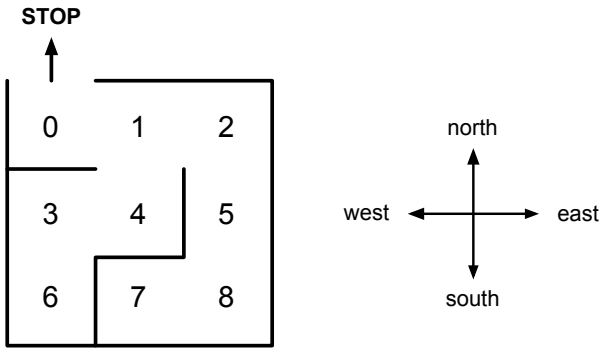

Figure 1: A maze.