Serie 5 - Liveness and Guarded Methods

Exercise 1

Answer the following questions:

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

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

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

4. How can you detect deadlock? How can you avoid it?

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

6. Why is progress a liveness rather than a safety issue?

7. Why should you usually prefer notifyAll() to notify()?

Exercise 2

Consider the FSP model for the dining philosophers:

\[
\text{PHIL} = ( \text{sitdown} \\
\rightarrow \text{right.get} \rightarrow \text{left.get} \rightarrow \text{eat} \\
\rightarrow \text{left.put} \rightarrow \text{right.put} \rightarrow \text{arise} \rightarrow \text{PHIL} ).
\]

\[
\text{FORK} = ( \text{get} \rightarrow \text{put} \rightarrow \text{FORK}).
\]

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

Modify the FSP specification, in a different manner than seen in the lecture, to remove the deadlock.

Exercise 3

Recall the Dining Savages from Serie 4, Exercise 2. Implement the behaviour of the pot (which you have modelled as FSP process before) in Java.

\[
\text{const } M = 5 \\
\text{SAVAGE} = (\text{getserving} \rightarrow \text{SAVAGE}). \\
\text{COOK} = (\text{fillpot} \rightarrow \text{COOK}). \\
\text{POT} = \text{SERVINGS}[0], \\
\text{SERVINGS}[i:0..M] = (\text{when } (i==0) \text{ fillpot} \rightarrow \text{SERVINGS}[M] \\
\quad | \text{when } (i>0) \text{ getserving} \rightarrow \text{SERVINGS}[i-1] ) . \\
||\text{SAVAGES} = (\text{SAVAGE} || \text{COOK} || \text{POT}) .
\]
Exercise 4

Russian roulette is the practice of placing one bullet in a gun, spinning the cylinder and closing it into the gun without looking, aiming the revolver at one’s own head in a suicidal fashion, and pulling the trigger.

In Figure 1 we show the Finite State Process (FSP) description of the Labeled Transition System (LTS) graph for a simulation of the Russian Roulette game. To simplify the number of possible states, the gun in our version has only two slots for holding bullets. Initially the gun is loaded with one bullet. There are two players labeled *Michael* and *Charlton* respectively. They share the table resource where the loaded gun is placed at the start of the game. Both *Michael* and *Charlton* compete for the gun. They are too impatient to take turns. To play the game, a player can take the gun from the table, press the trigger and if he is lucky, he survives to play again. If he is unlucky, he shoots himself.

- Implement Russian Roulette process as a Java program. Make sure to use guarded methods: the player should block if there is no gun on the table.
- Find the liveness violation. Modify your program to eliminate the problem.