Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Leonel Merino

Solution

Series 02 — 27.09.2017 – v1.0a
Concurrency and Java

Exercise 1 (2 Points)

Answer the following questions (0.5 point each):

a. What states can a Java thread be in? **Answer:**

From a very simplified perspective, a thread can be either in the runnable *(thread is executing) or the* not runnable *(everything else) state. However, there exist six states in the Java runtime environment such as* NEW *(thread not yet started),* RUNNABLE, BLOCKED *(thread waiting for a monitor lock),* WAITING *(waiting indefinitely for another thread to perform a particular action),* TIMED_WAITING *(WAITING with a timeout, i.e. upper time limit for the wait) and* TERMINATED *(thread exited). For more details, please check the Java 9 API documentation on* https://docs.oracle.com/javase/9/docs/api/java/lang/Thread.State.html

b. How can you turn a Java class into a monitor? **Answer:**

It is as simple as declaring all public methods synchronized. *You will hear more on that in the fourth lecture titled "safety patterns".*

c. What is the *Runnable* interface good for? **Answer:**

An object that implements Runnable *can be set into the active state, i.e. you can create a running thread based on a* Runnable *object similarly as with the ordinary* Thread *class. This is very useful since Java up to release 7 does not support the concept of multiple inheritance, so inheriting from the* Thread *class (to implement concurrent code routines) makes it impossible to inherit from something else.*

d. Specify an FSP that repeatedly performs hello, but may stop at any time. **Answer:**

```
HELLO = (hello -> HELLO | hello -> STOP).
```

Exercise 2 (2 Points)

Consider the following Java implementation of a Singleton within a single-threaded application:

```
public class Singleton {
  private static Singleton instance = null;
  private Singleton() {}
  public static Singleton getInstance() {
    if(instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
}
```

a. What happens if the application is multithreaded? **Answer:**

There exists the possibility that many instances of the Singleton class are created, because multiple threads could concurrently check the condition `(instance == null)`*, succeed, and then continue with the instantiation of the class* Singleton *potentially overwriting previously assigned variable values.*

b. How to implement a thread-safe singleton in Java? **Answer:**

The `getInstance()` *method needs to be synchronized as shown below:*

```
public static synchronized Singleton getInstance() {
   if(instance == null) {
      instance = new Singleton();
   }
   return instance;
}
```

c. Suppose there arrive 1000 requests/second from different threads at this Singleton. Does your implementation introduce a bottleneck? If yes, how can you improve it? **Answer:**

Making the `getInstance()` *method synchronized will lock the entire object every time a thread calls this method and the lock won't be released before the active thread exits the method. This is not efficient. A more efficient and fine-grained solution is as follows ():*

```
public static Singleton getInstance() {
   if(instance == null) {
      synchronized (Singleton.class) {
         if(instance == null) {
            instance = new Singleton();
         }
      }
   }
   return instance;
}
```

Please be aware that this solution frequently causes much complexity in real-life scenarios (e.g. when the same object references get used in- and outside of the synchronized block) and could lead to concurrency bugs which are even harder to track than with the simpler generic approach.

Exercise 3 (3.5 Points)

Download LTSA from `http://www.doc.ic.ac.uk/~jnm/book/ltsa/download.html`. For each of the following processes shown in Figure 1, provide the Finite State Process (FSP) description of the corresponding Labeled Transition System (LTS) graph. You may verify the FSP descriptions by generating the state machines and using the "draw" functionality of the tool. **Answer:**

```
APPOINTMENT = (hello -> converse -> goodbye -> STOP).

HOLIDAY = (arrive->relax->leave->HOLIDAY).
```

```
SPEED = (on->DRIVING),
DRIVING = (speed->DRIVING | off->SPEED).

LEFTONCE = (ahead-> (left->STOP |right->LEFTONCE)).

TREBLE = (in[i:1..3] -> out[i*3] -> TREBLE).

FIVETICK (N=5) = FIVETICK[1],
FIVETICK[i:1..N] = ( when(i<N) tick -> FIVETICK[i+1]
                   | when(i==N) tick -> STOP).

PERSON = (
  workday -> sleep -> work -> PERSON
  | holiday -> sleep -> (
                          play -> PERSON
                          | shop -> PERSON
                        )
).
```

Exercise 4 (2.5 Points)

Consider the full Race5K FSP from the lecture.

a. How many states and how many possible traces does it have if the number of steps is 5 (as in the lecture)? **Answer:**

   36 states (cartesian product of individual state spaces) fostering 252 traces (see next question)

b. What is the number of states and traces in the general case (i.e. for n steps)? **Answer:**

   We assume 2 processes.

   - *Number of states:* $(n+1)^2$ *since* n *is the number of steps, so we have* $(n+1)$ *states per process*
   - *Number of traces (from n steps, respectively transitions):* $\frac{(2\cdot n)!}{n!\cdot n!}$ *is the number of possible paths in a* $n \times n$ *regular, directed grid graph from node* $(0,0)$ *to node* (n,n)
   - *Number of traces (from n states of each process):* $\frac{(2\cdot(n-1))!}{(n-1)!\cdot(n-1)!}$ *is the number of possible paths in a* $n \times n$ *regular, directed grid graph from node* $(0,0)$ *to node* (n,n)

   There also exist various detailed code snippets for finding all possible paths in such grids online.[1]

c. Check your solution using the LTSA tool.

[1] http://www.geeksforgeeks.org/count-possible-paths-top-left-bottom-right-nxm-matrix/

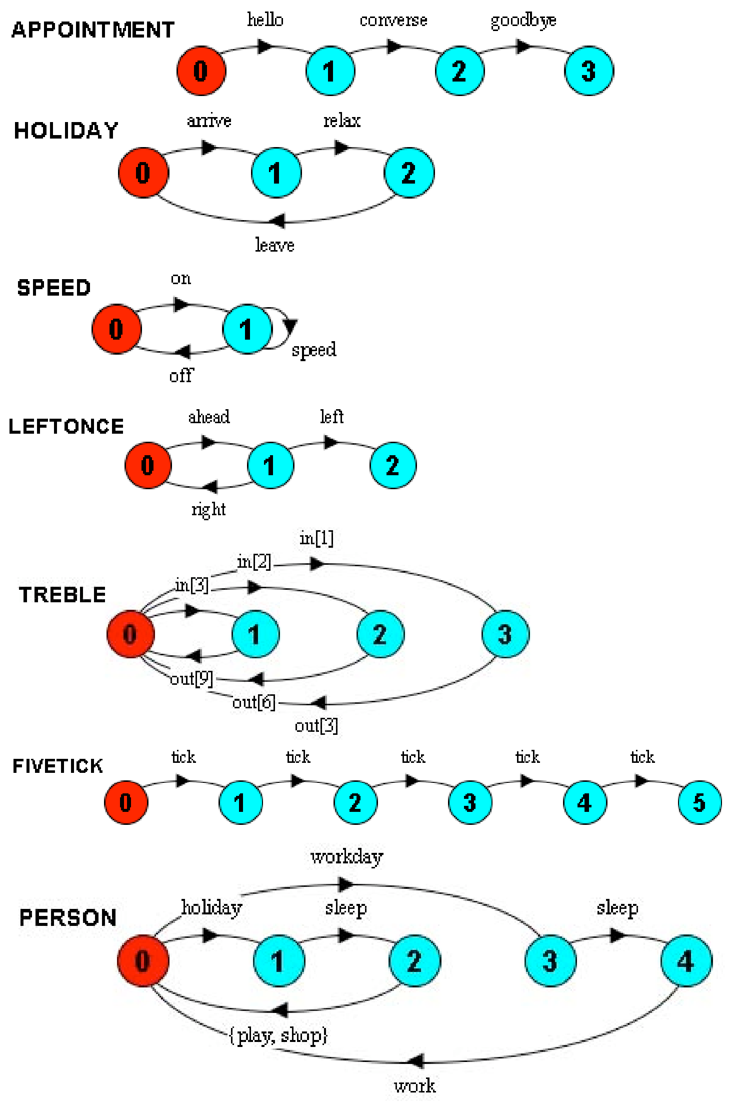

Figure 1: LTSA graphs.