Dr. Wolfgang Emmerich
Dept. of Computer Science
University College London
Pearson Building Room 402

18-Dec-98

C340 Concurrency

Tutorial Session 4 -Answer Sheet

1.a

- Threads are more lightweight than processes. Multiple threads can be executed within a single process. Processes might be owned by different users. All threads that are executed within a process are owned by the same user.
- A critical section is a sequence of actions that must be executed by at most one process at a time.
- Critical regions are guaranteed to be executed by at most one process at a time.
- Safety properties assert that nothing bad will happen during the lifetime of a process.
- Liveness properties assert that eventually something good will happen during the lifetime of a project
- A process is in a deadlock if it is spinning waiting for a condition never to become true.
- A process is in a livelock if it is actively waiting for a condition never to become true.

1.b

```
range T=0..3

// Process LIGHTSTATE stores the state of a traffic light with
// the following coding: 0="red",
//                       1="red_and_amber",
//                       2="amber" and
//                       3="green"
LIGHTSTATE(N=3) = LIGHTSTATE[0],
LIGHTSTATE[i:T] = (set[u:T]->LIGHTSTATE[u]
                  |get[i]->LIGHTSTATE[i]).

// Process LIGHT establishes the sequence of action with which lights
// change. Another acceptable solution for subquestion b would be to
// merge processes LIGHTSTATE and LIGHT. For the formulation of the
// safety property, however, we need to be able to get access to the
// current state of the light.
LIGHT= (get[i:T]->
             (when (i==0) switch_red_and_amber -> set[1]->LIGHT // red
             |when (i==1) switch_green -> set[3] ->LIGHT        // red_and_amber
             |when (i==2) switch_red -> set[0]  ->LIGHT          // amber
             |when (i==3) switch_amber -> set[2] ->LIGHT         // green
             |suspend->resume->LIGHT))+{get[T]}.

// A traffic light is capable of telling its state and it changes
// in the sequence established by LIGHT. Because we do not want
// other processes to change the state of a traffic light, we do
// not export actions set[T]
||TRAFFICLIGHT = (LIGHT || LIGHTSTATE)\{set[T]}.

// Process START starts-up the traffic light by enabling the
// southern light to switch to green (via red_and_amber)
START = (south.switch_red_and_amber -> CTRL),

// We attempt to make the junction safe by rotating green lights in
// clockwise order. Before a light can change to red_and_amber we
// force it to wait for the light (in counter-clockwise order) to have
// switched to red.
CTRL = ( south.switch_red -> west.switch_red_and_amber ->
         west.switch_red -> north.switch_red_and_amber ->
         north.switch_red -> east.switch_red_and_amber ->
         east.switch_red -> south.switch_red_and_amber -> CTRL).
```

```
// A junction is then the parallel composition of four TRAFFICLIGHT
// processes with the START process.
||JUNCTION= (east:TRAFFICLIGHT || west:TRAFFICLIGHT||
            north:TRAFFICLIGHT || south: TRAFFICLIGHT || START)
```

1.c

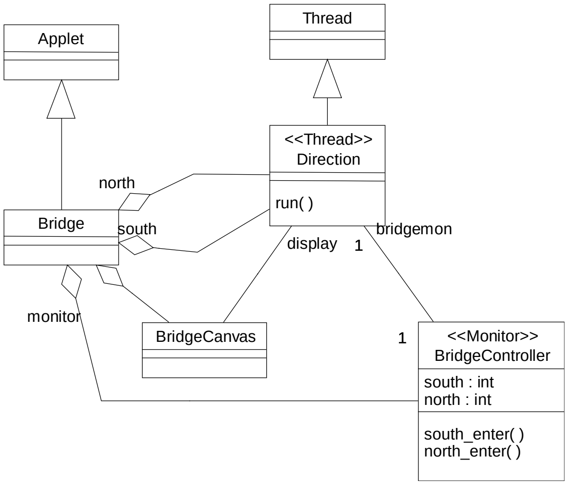
```
// The Safety Property ONLY_ONE_GREEN checks whether there are more
// than one lights green. One way of doing this is to add the states of
// all traffic lights together. As we want at most one light to be
// green, the sum has to be equal or smaller than three. In order to
// perform this safety check, we actually need to stop the traffic light
// system as wrong results would occur if it were continuing to work.
ONLY_ONE_GREEN =(stop->east.get[e:T]->south.get[s:T]->west.get[w:T]->north.get[n:T]->
                 ( when (e+s+w+n<=3) safe ->start -> ONLY_ONE_GREEN
                 | when (e+s+w+n>3) unsafe -> ERROR))/{stop/east.suspend,
           stop/west.suspend,
           stop/north.suspend,
           stop/south.suspend,
           start/east.resume,
           start/west.resume,
           start/north.resume,
           start/south.resume}.

// The Safety property check is then done by composing the processes
// JUNCTION and ONLY_ONE_GREE
||SAFE_JUNCTION = (JUNCTION || ONLY_ONE_GREEN).
```

2.a The Java Thread Life cycle is as follows:

- Started by start() which invokes run()
- Terminated when
  - run() returns or
  - explicitly terminated by stop()
- A started thread may be
  - running or
  - runnable (waiting to be scheduled)
- Thread gives up processor using yield()
- A thread may be suspended by suspend()
- If Suspended gets runnable by resume()
- sleep() suspends for a given time and then resumes

The Java virtual machine implements concurrency by implicitly switching between thread states running or runnable. Threads may also explicitly be deactivated using yield.

b.



c. In order to make an asynchronous call, create a new thread and then call the method in that new thread. While doing this, the main thread continues to operate and the newly created thread waits for m to return. Synchronise the two threads using semaphores or monitors when the result of m is needed.