IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2012

BEng Honours Degree in Information Systems Engineering Part III
MEng Honours Degree in Information Systems Engineering Part III
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
MSc in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C528

CONCURRENT PROGRAMMING

Tuesday 1 May 2012, 10:00
Duration: 120 minutes

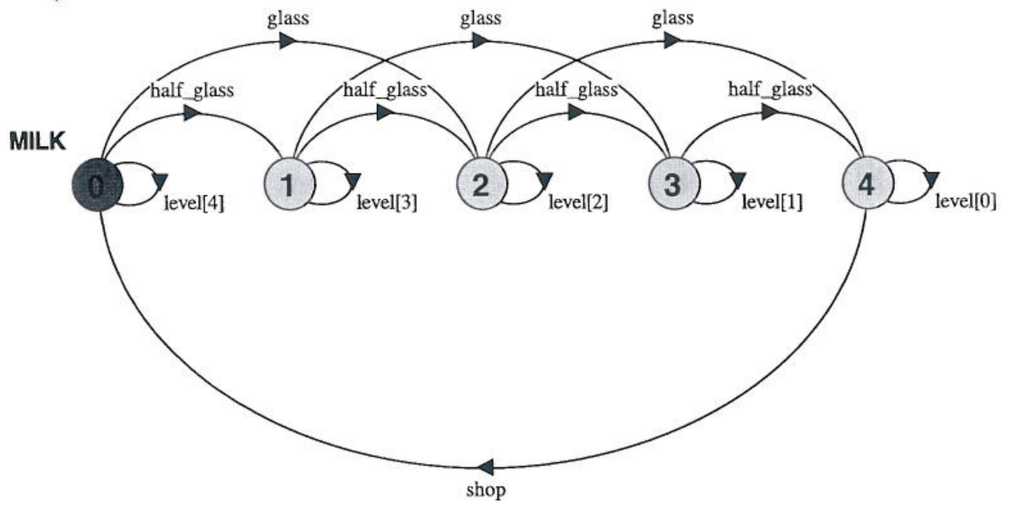
Answer THREE questions

Paper contains 4 questions
Calculators not required

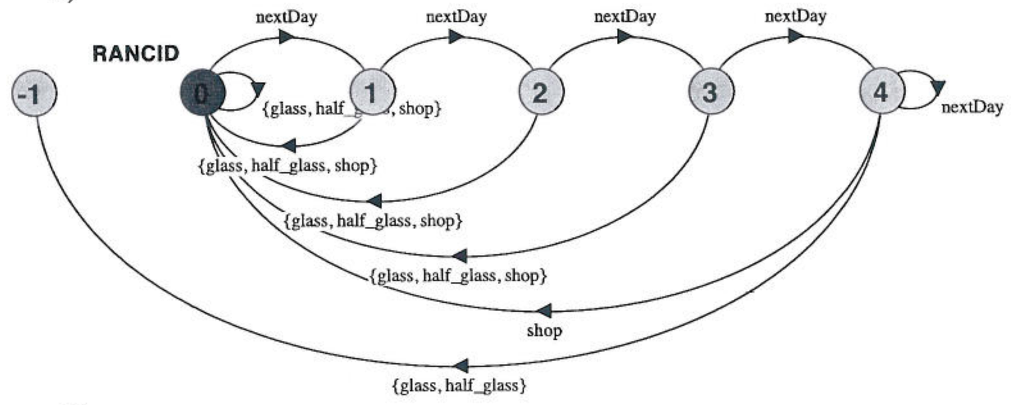
1a Explain the term *alphabet extension* and how it can be used in a simple FSP example.

1b For each of the following Labelled Transition Systems (*LTS*), give an equivalent *FSP* specification.

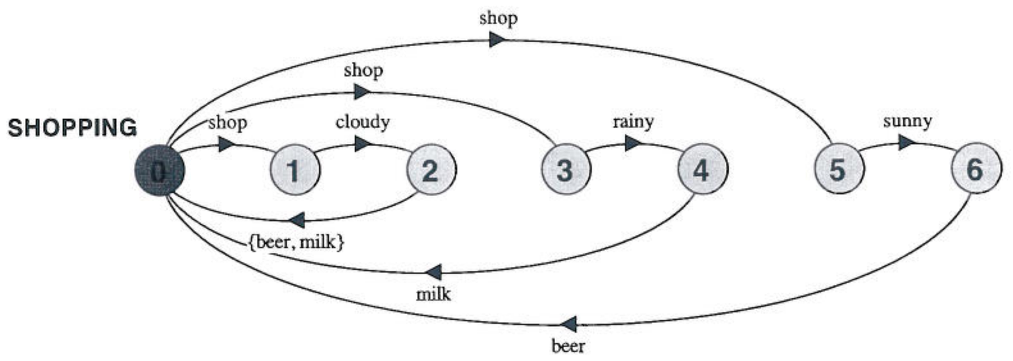
i)



ii)


RANCID
nextDay
nextDay
nextDay
nextDay
nextDay
{glass, half_glass, shop}
{glass, half_glass, shop}
{glass, half_glass, shop}
{glass, half_glass, shop}
shop
{glass, half_glass}

iii)



The two parts carry, respectively, 40% and 60% of the marks.

2a For each of the following *FSP* specifications, give an equivalent *LTS*.

i)
```
property FILE = (open ->  OPEN | close -> FILE),
         OPEN = (read -> OPEN |
                 close -> FILE |
                 write -> DIRTY),
        DIRTY = (read -> DIRTY |
                 write -> DIRTY |
                 flush -> OPEN).
```

ii)
```
const MAX = 4
SEQ_READ(N=2) = (open -> OPEN[N]),
OPEN[i:1..MAX] = (read -> OPEN[i-1]),
OPEN[0] = (close -> END).


||CONCAT = (a:SEQ_READ(2) || b:SEQ_READ(3))
                  /{open/a.open,
                    close/b.close,
                    continue/{a.close, b.open}}.
//draw LTS for CONCAT
```

iii)
```
SPACE_HOG = (new -> outOfMemory ->SPACE_HOG |
             new -> reference -> SPACE_HOG).

SPACED_OUT = STOP + {outOfMemory}.

||TOGETHER = (SPACE_HOG || SPACED_OUT).
//draw LTS for TOGETHER
```

2b Explain the difference between the `notify()` and `notifyAll()` methods of the `Thread` class in Java. Give some example code to illustrate the difference.

The three parts carry, respectively, 60% and 40% of the marks.

3a The interface to a buffer that stores characters is specified in Java as follows:

```
interface Buffer {

  public static int N = 8;    //capacity of buffer

  /* put puts a character into the buffer;
   *   blocks calling thread when the buffer is full (i.e. holds N characters)
   */
  public void put(char ch) throws InterruptedException;

  /* swap removes s characters from the buffer
   *          and adds s characters from ch into the buffer.
   *   Before swapping the buffer must have at least s characters
   *          and at least s free spaces,
   *          otherwise the calling thread will block.
   */
  public char swap(char[] ch, int s);

  /* get removes and returns a character from the buffer;
   *   blocks calling thread if the buffer is empty.
   */
  public char get() throws InterruptedException;

}
```

Ignoring the character values, specify the abstract behaviour of the buffer as an *FSP* process **BUFFER** with the alphabet **{put, get, swap[0..N]}**.

b Provide the Java program for a class that implements the **Buffer** interface. Note that your Java syntax need not be perfect.

c Specify progress in *FSP* to check that threads eventually get to swap N/2 elements. Give the specification for a congested system that models a situation in which the progress property would be violated in this system.

The three parts carry, respectively, 30%, 40% and 40% of the marks.

4 The following is an outline for a *Multiple Reader Single Writer* (MRSW) lock in Java:

```
public class MRSWLock {
  private int readers = 0;  // Number of Readers that have acquired the lock
  private boolean writer = false;   // If a Writer has acquired the lock
  private Object lock = new Object();

  public void acquireReaderLock()  throws InterruptedException
  { /*TODO*/ }
  public void releaseReaderLock()
  { /*TODO*/ }
  public void acquireWriterLock() throws InterruptedException
  { /*TODO*/ }
  public void releaseWriterLock()
  { /*TODO*/ }
}
```

A **MRSWLock** object is shared by Reader and Writer client threads. The **MRSWLock** is used to control access to a shared resource. The **MRSWLock** should behave as follows:

- Multiple Readers can acquire the lock concurrently. No Writer can acquire the lock while any Readers have acquired it.
- Only a single Writer can acquire the lock at any one time. No Reader can acquire the lock while any Writer has acquired it.

You may assume that a Reader calls **acquireReaderLock** only if it has not yet acquired the lock, only calls **releaseReaderLock** if it has already acquired the lock, and never calls the Writer methods. Similarly for Writers and the **write** methods.

4a The **acquireReaderLock** and **acquireWriterLock** are called by Readers and Writers respectively to acquire the lock. The methods should only return when the caller has successfully acquired the lock. Give an implementation of each these methods. (Hint: use the **lock** field for synchronisation and check/update the **readers** and **writer** fields in an appropriate way.)

4b Give implementations of the **releaseReaderLock** and **releaseWriterLock** methods, called by Readers and Writers to release their locks respectively.

4c Explain why it can be preferable to use an explicit, internalised **private** lock object, such as the **lock** field in the **MRSWLock** class above, rather than the implicit lock of each object (i.e. the lock of **this**). You may illustrate your reasoning through a brief code fragment.

The three parts carry, respectively, 50%, 30% and 20% of the marks.