UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2003

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
MSc in Computing for Industry
BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
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 C223=MC223

CONCURRENT PROGRAMMING

Wednesday 30 April 2003, 15:30
Duration: 120 minutes

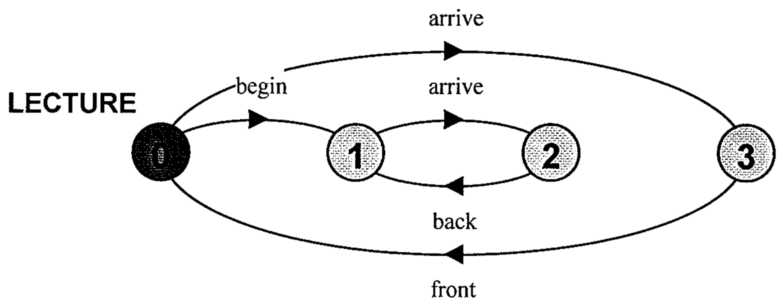
Answer THREE questions

Paper contains 4 questions
Calculators not required

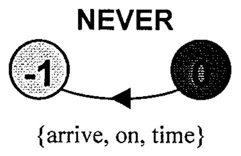
1a Explain how *non-deterministic choice* is expressed in the Finite State Processes (FSP) modelling notation. Briefly explain why it is useful in modelling systems.

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

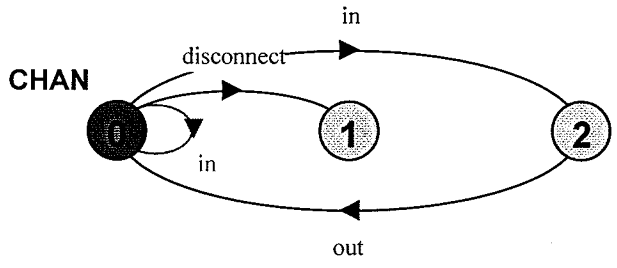
i)



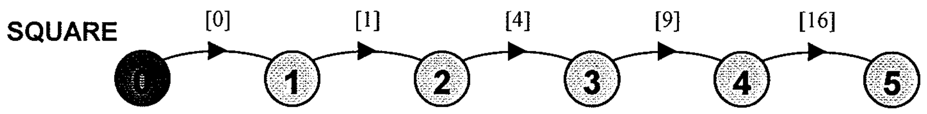
ii)



iii)



iv)



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

i)
```
OVERFLOW = CAPACITY[0],
CAPACITY[i:0..3] = (drip->CAPACITY[i+1]).
```

ii)
```
CAR = (start->(stop->STOP | faster->slower->CAR)).
```

iii)
```
BLOCK = STOP + {north}.
TRAIN = (north->move->TRAIN | south->move->TRAIN).
||SYS = (BLOCK || TRAIN).  //draw LTS for SYS
```

iv)
```
EXIT  = (in->out->EXIT).
||OR = (a:EXIT || b:EXIT)/{in/{a.in,b.in}}.
//draw LTS for OR
```

The three parts carry, respectively, 20%, 40%, 40% of the marks.

2a Briefly explain how a *guarded action* in an FSP specification is translated into part of a Java program that implements that specification.

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

```
interface Buffer {

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

  /* 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;

  /* blocks calling thread until the buffer is full
   *  then  returns entire contents of the buffer
   */
  public char[] get() throws InterruptedException;

}
```

Specify the abstract behaviour of the buffer as an *FSP* process **`BUFFER`** with the alphabet **`{put, get}`**.

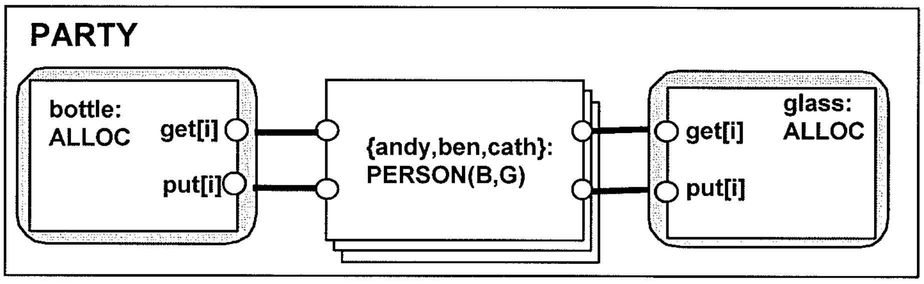
c List the program for a class that implements the **`Buffer`** interface.

d Extend the **`BUFFER`** model you specified in part b to add the action **`getone`** which models a method that gets a single character from the buffer and blocks when the buffer is empty.

The four parts carry, respectively, 15%, 35%, 40%, 10% of the marks

3a Explain briefly how a resource, shared by a set of processes can be modelled in FSP.

b When they arrive, people attending a very organised party must first request the number of bottles of refreshment that they require from the bottle allocator and then the number of glasses into which to pour the refreshing drinks from the glass allocator. When the bottles are empty, they return bottles and glasses to the allocators where they are respectively refilled and cleaned. A person may request again. If sufficient bottles or glasses are not available, a person must wait until other people return theirs. The party goes on forever. The structure of the system as modelled in FSP is depicted below:



Given that the behaviour of **PERSON** is defined by:

```
const NB  =  3 //total number of bottles that can be allocated
const NG  =  2 // total number of glasses that can be allocated
set All   =  {bottle.{get,put}[1..NB],
                glass.{get,put}[1..NG]}

PERSON(B=1,G=1)
    = (bottle.get[B] -> glass.get[G] -> drink ->
       bottle.put[B] -> glass.put[G] -> PERSON)+ All.
```

Specify the behaviour of the process **ALLOC** and the composite process **PARTY** in FSP. Assume that *Andy* requires 1 bottle and 1 glass, *Ben* 2 bottles and 1 glass and *Cath* 1 bottle and 2 glasses.

c Implement the specifications for each of the entities (**PERSON, ALLOC**) in Java. Include the definition of a method `void build(int NB, int NG)` which creates the objects required for **PARTY**.

d Briefly explain why the alphabet extension (`+ All`) is used in the definition of **PERSON**.

(The four parts carry, respectively, 15%, 25%, 45%, 15% of the marks

4a Explain what is meant by *safety* and *liveness* properties with respect to concurrent programs.

b An academic department, due to government cuts, can only afford a single bathroom that can hold a maximum of *MP* people. The bathroom can be used by both men and women, but not at the same time. Given the following definitions:

```
const BM  = 2         //maximum number of people allowed in bathroom
const Max = 7
set Men   = {man[1..Max]}     //set of all men in the department
set Women = {woman[1..Max]} //set of all women in the department

PERSON = (enter -> wash -> exit -> PERSON).
||PEOPLE = (Men:PERSON || Women:PERSON).
```

Specify a process **BATHROOM** in *FSP* that ensures that a maximum of *BM* people are allowed into the bathroom at any one time and that the bathroom cannot be occupied by both men and women at the same time.

c Specify the follow safety properties in *FSP*:

i) **UNISEX** checks that the bathroom is occupied either by men or women, but not both simultaneously.

ii) **OVERFLOW** checks that more than *BM* people do not occupy the bathroom at the same time.

Give the *FSP* composition for the system that combines people, bathroom and the safety properties.

d Specify two progress properties in *FSP* that check, respectively, that women eventually get to use the bathroom and that men eventually get to use the bathroom. Give the specification for a system that models the situation in which there is a heavy demand for the bathroom. Would your progress properties be violated in this system?

The four parts carry, respectively, 10%, 30%, 40%, 20% of the marks