IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2013

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 Royal College of Science

PAPER C528

CONCURRENT PROGRAMMING

Thursday 2 May 2013, 10:00
Duration: 120 minutes

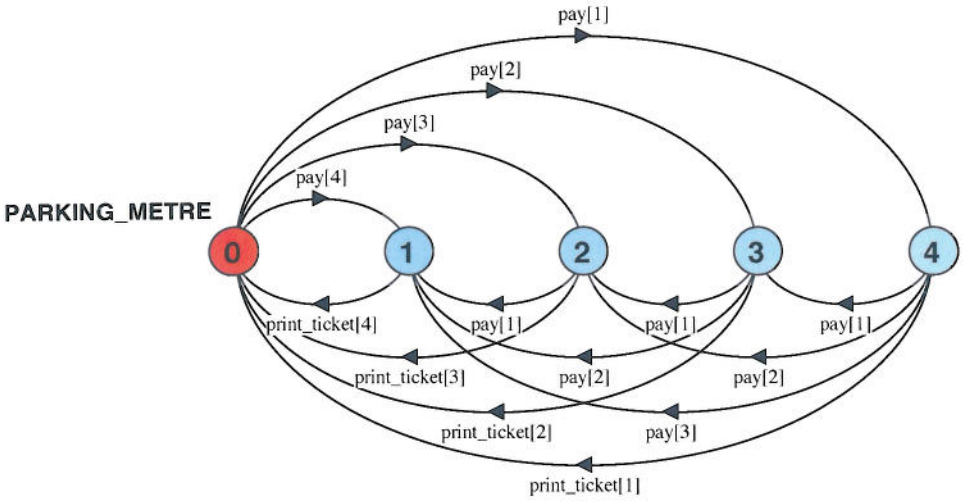
Answer THREE questions

Paper contains 4 questions
Calculators not required

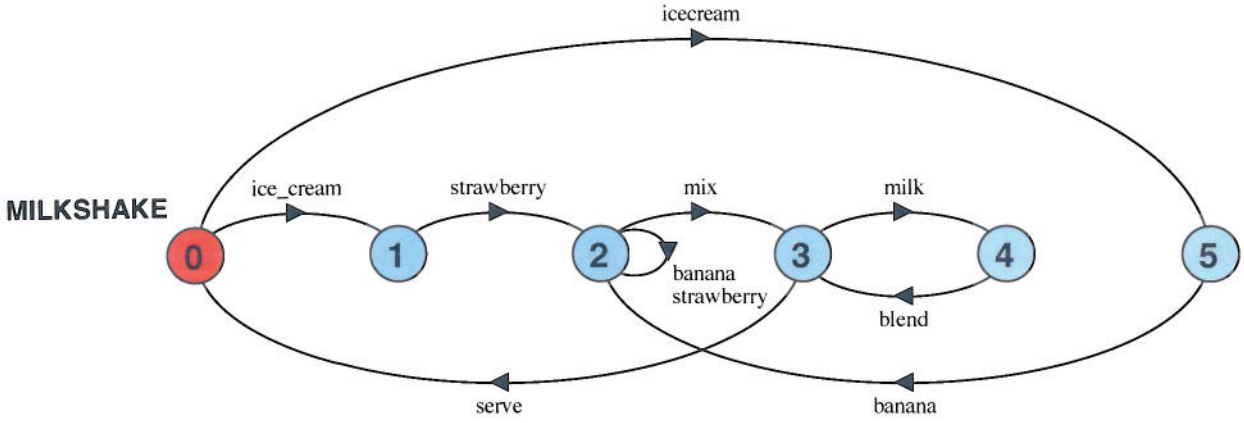
1a Briefly outline the two ways of creating a new thread in Java.

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

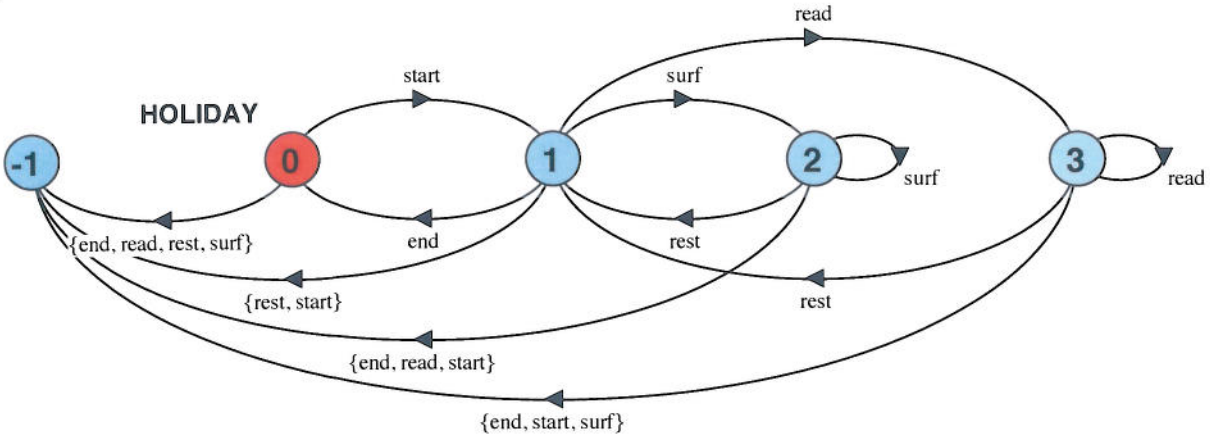
i)



ii)



iii)



1c What are the four necessary and sufficient conditions for a deadlock to occur? Briefly explain how these conditions manifest in the problem of Dining Philosophers.

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

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

i)
```
property SAFE = (lock -> LOCKED[0]),
LOCKED[i:0..1]=( when (i<1) correct_code -> unlock -> SAFE
               | when (i<1) incorrect_code -> LOCKED[i+1]
               | when (i==1) alert -> STOP).
```

ii)
```
ELECTRICITY_BILL = (enter_account_details -> OPTIONS),
OPTIONS = (enter_meter_reading -> METER
          | make_payment -> PAYMENT),
METER   = (enter_new_reading -> PAYMENT),
PAYMENT = (pay -> finish -> ELECTRICITY_BILL).
```

iii)
```
const Max = 3

PATIENT = (request-> ({receive, contact} -> PATIENT)).
PHARMACIST(N=0) = P[N],
P[n:0..3] = (request ->
             ( when (n<Max) refill[n+1] -> send -> P[n+1]
             | when (n>=Max) assess -> contact -> STOP
             | when (n>=Max) assess -> send -> PHARMACIST)).

||PRESCRIBE = (PATIENT || PHARMACIST(0))/{send/receive}.

//draw LTS for PRESCRIBE
```

iv)
```
const MAX = 3

PENALTIES = TRIES[0],
TRIES[i:0..MAX]=(when(i<MAX)team1.shoot -> team2.shoot ->
TRIES [i+1]).

TEAM(S=0)=SCORE[S],
SCORE[n:0..MAX]=( when(n< MAX)shoot -> score -> SCORE[n+1]
                | when(n< MAX)shoot -> miss -> SCORE[n]
                | end -> TEAM).

||SHOOTOUT = (team1:TEAM || team2:TEAM  || PENALTIES)
                                  @{team1.shoot, team2.shoot}.

//draw LTS for SHOOTOUT
```

2b Explain why alphabet extension is used in models of concurrent programs. Draw the LTS `HOLIDAY` resulting from the composition of the following processes.

```
READING = (read -> READING) + {program}.
WORKING = (program -> WORKING
          |think -> WORKING).

||HOLIDAY = (READING || WORKING).
```

2c When a thread calls `wait()` in a monitor, it releases the lock of that monitor and the locks of any other monitors which it holds. True or False? Explain your answer.

The three parts carry, respectively, 60%, 20%, and 20% of the marks.

3 An outline of the class **MeetingScheduler** is specified in Java and English as follows:

```
class MeetingScheduler {
  private static int capacity = 5;   //capacity of meeting room
  private int[] staff_ids = {1,2,3,4,5,6,7,8}; //list of all staff members
  private int[] attendance = new int[Capacity]; //list of staff attending the
meeting
  /* attend is invoked when a staff member wishes to attend the meeting;
     returns "true" if the staff's attendance is successfully registered;
     and blocks the calling thread if the meeting room has already reached its capacity.
   */
  public boolean attend(int id) throws InterruptedException;

  /* cancel removes a staff member's name from the attendance list;
     returns "true" if the staff's attendance is successfully cancelled;
     and blocks the calling thread if there are no attending staff members.
   */
  public boolean cancel(int id) throws InterruptedException;

 /* process_waiting_list removes n staff members from a waiting list and adds them
    to the attendance list. Before adding their ids, the scheduler must check that there are enough
    spaces in the meeting room, otherwise the calling thread will block.
  */
  public void process_waiting_list(int[] waiting_list, int n)
  throws InterruptedException;
 }
```

3a Specify the abstract behaviour of the meeting scheduler system as an FSP process, **MeetingScheduler**, which ensures that staff members are allowed to attend if there is spare capacity and that the capacity of the meeting room is never exceeded. Your model need not record whether a particular member of staff is attending or on the waiting list. The alphabet of the **MeetingScheduler** is as follows:

```
{attend[s:1..Max_id], cancel[s:1..Max_id],
process_waiting_list[i:1..Capacity]}
```

where **Capacity=5** and **Max_id=8.**

3b Specify a progress property **CanRegister** that checks that every staff member can eventually attend a meeting. This property should be preserved by your model.

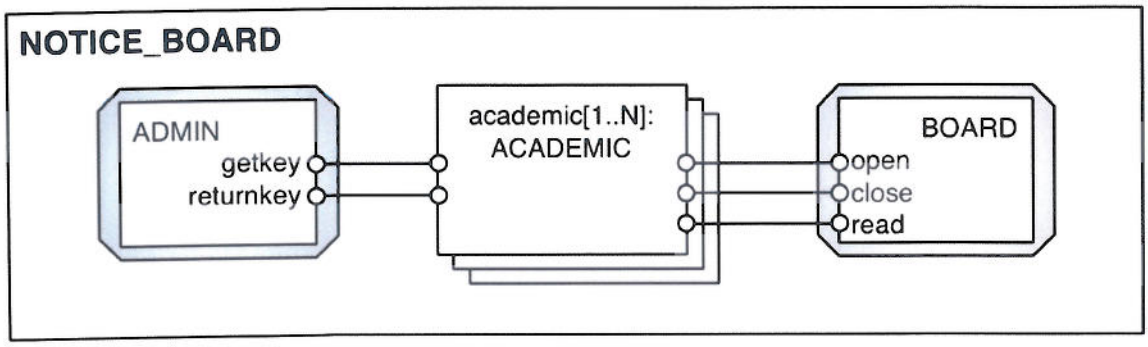
3c Specify a safety property **RegisterOnce** in FSP to check that the same staff member has not been registered twice to attend the meeting. This property should be violated in your abstract **MeetingScheduler** model. Give an example of a trace where the property **RegisterOnce** is violated.

(Hint: it is easier to provide a property for each staff member).

3d Provide the Java program that implements the class **MeetingScheduler**, satisfying all properties. Note that your Java syntax need not be perfect.

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

4a A notice board is shared by a department of academics. The notice board has a glass cover that may only be opened with the aid of a key obtained from the administrator. To change a notice, an academic must obtain the key, open the glass cover, change the notice, close the cover and return the key. Other academics can read the notice board when it is closed (i.e. not opened for a change). The structure diagram for an FSP model of the system with N academics is shown below:



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

```
ACADEMIC =  (getkey -> open -> close -> returnkey -> ACADEMIC
            |read -> ACADEMIC).
```

Specify the behaviour of each of the processes **ADMIN** and **BOARD** and the composite process **NOTICE_BOARD** in FSP.

4b Implement the specification for each of the entities **ADMIN, ACADEMIC, BOARD** in Java. Also provide the definition of a method **void build(int N)** which creates the objects required for **NOTICE_BOARD** and could be included in some main program class, e.g. applet **NoticeBoard**.

(Hint: Use the method **Math.random()** which randomly returns a floating point value between 0 and 1 to model academic's random choice of whether to change a notice or simply read the board contents, e.g. **Math.random() < 0.5** is 50% probability. Ignore details of data representations for keys, notices etc. and use **void** methods with no parameters to implement model actions.)

4c Briefly describe the operation of **wait(), notify(),** and **notifyAll()** methods when used in Java for condition synchronization in monitors. Give one example of when it would be preferable to use **notify()** instead of **notifyAll()** and another when **notifyAll()** would be safer.

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