IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2010

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

CONCURRENCY

Tuesday 27 April 2010, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

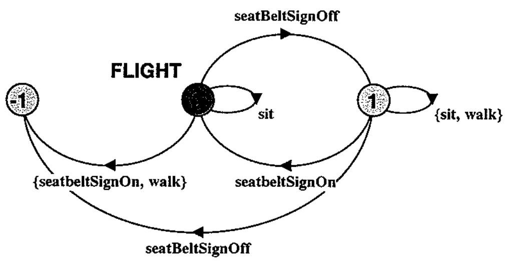
Answer Sections A and B in separate answer books.

SECTION A

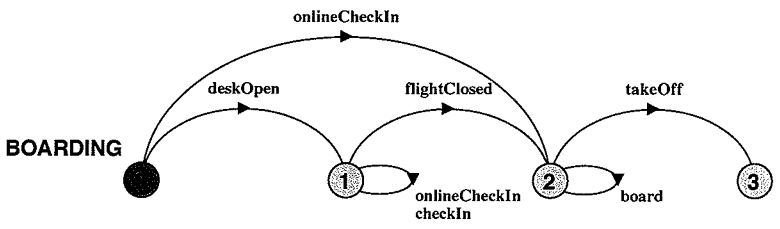
1a If a process *P* has *n* states and a process *Q* has *m* states, give a condition over these processes that guarantees that *P* || *Q* has *n* times *m* states? Exemplify.

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

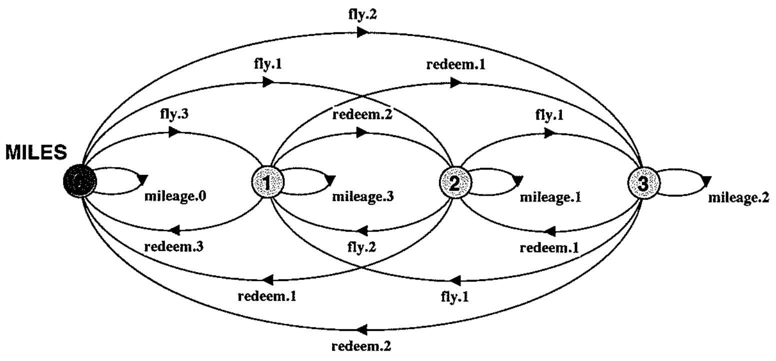
i)



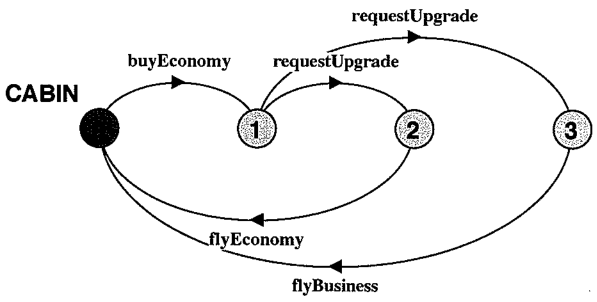
ii)



iii)



iv)



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

i)
```
PROGRAM = (write -> WRITTEN),
WRITTEN = ( compile -> compile_time_error -> PROGRAM |
            compile -> ok -> COMPILED),
COMPILED = (run -> run_time_error -> PROGRAM |
            run -> STOP).
```

ii)
```
property PROTOCOL = CLOSED,
CLOSED = (open-> OPEN),
OPEN  = ({read, write} -> OPEN | close -> CLOSED).
```

iii)
```
const MAX = 3
ENTER_PIN = ENTER_PIN[0],
ENTER_PIN[n:0..MAX-1] =
(pin ->
    (ok -> ENTER_PIN | nok -> ENTER_PIN[n+1])),
ENTER_PIN[MAX] = (confiscate_card -> STOP).
```

iv)
```
const MAX = 1
VAR = VAR[0],
VAR[i:0..MAX] = (read[i] -> VAR[i] |
                 write[j:0..MAX] -> VAR[j]).

PROCESS(N=1) = (write[N]->read[i:0..MAX]->STOP)
                               + {write[i:0..MAX]}.

||INTERFERE = ( a:PROCESS(0) ||
                b:PROCESS(1) ||
                {a, b}::VAR)
                    /{read/{a.read, b.read}}.

//draw LTS for INTERFERE
```

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

2a What is the *alphabet* of a process specified in FSP?

b A timer service is required that cyclically counts from 0 to 59 seconds and then returns to zero. The timer increments its count every time it is signalled by the external **tick** action. Processes using the timer service can request to be executed when the timer reaches a particular count using the action **at[t]** – this delays the invoking process until the counter within the timer service has the value **t**. Give the FSP model **TIMER** for the timer service using the following definitions:

```
const MAX = 60
range TIME = 0..MAX-1
```

You may assume that actions **at[t]** have high priority.

c The definition of a system model which includes processes that use the **TIMER** service is given below:

```
PROCESS(I=2) = (at[I] -> run -> STOP) + {at[TIME]}.

||SYS = ({a,b}::TIMER || a:PROCESS(2) || b:PROCESS(7))
        /{tick/{a,b}.tick}
        <<{{a,b}.at[t:TIME]}.
```

i) Give a trace of this system that includes ten tick actions.

ii) Explain clearly why the alphabet extension is required in the definition of **PROCESS.**

iii) Suppose that the alphabet extension was removed from **PROCESS**. What traces would the resulting **SYS** model allow?

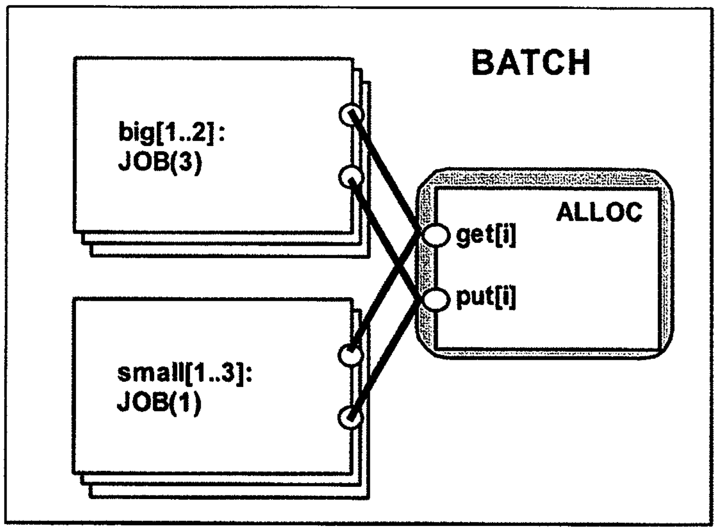
d Give an implementation as a Java class of the timer service (part b) that you modelled in FSP as the process **TIMER.**

The four parts carry, respectively, 10%, 25%, 30%, 35% of the marks.

SECTION B

3a Briefly outline the two different ways of creating a new thread in Java.

b The overall model structure of a batch processing system is shown below:



A job entering the system must acquire memory blocks from an allocator before it can run and returns the blocks when it has finished running. A job is blocked until sufficient blocks become available. There are two types of jobs: big jobs require three blocks and small jobs require only one. Given the models for job and allocator:

```
const NB = 4 //maximum number of allocatable blocks
set ALL = {{get,put}[1..NB]}

JOB(B=1) = (get[B]->run->put[B]->JOB)+ALL.

ALLOC = ALLOC[NB],
ALLOC[i:0..NB] = (when (i>0) get[j:1..i] -> ALLOC[i-j] |
                    put[j:1..NB] -> ALLOC[i+j] ).
```

i) Specify the composite process **BATCH** in FSP.

ii) Specify two properties, one for small jobs and one for big jobs that assert that jobs eventually get allocated memory blocks.

iii) Specify a **BATCH** system with adverse scheduling conditions to model a heavily loaded system.

iv) Briefly outline a scenario in which one of the properties you have specified in ii) might be violated in the system you have specified in iii)

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

The three parts carry, respectively, 10%, 40%, 50% of the marks.

4a Explain what is meant by liveness and progress properties with respect to concurrent programs. Give an example, using natural language, of each sort of property.

b A remote login system should 1) allow up to 4 remote users logged in at the same time and 2) ensure that only enabled users are allowed to log in.

The action **login[u]** models the successful login by user **u**, while **logout[u]** indicates that user **u** has logged out where **u** is in the range **U=1..N**. Similarly actions **enable[u]** and **disable[u]** are used by administrator to configure users that are allowed to login to the system.

Specify the two requirements set out above using safety properties in FSP. (*Hint: The second of these properties for all users is best specified using parallel composition of the property for a single specific user*).

c i) Explain what is meant by recursive locking in Java.

ii) Given the following declarations:

```
const N = 3
range P = 1..2   //thread identities
range C = 0..N   //counter range for lock
```

Model a Java recursive lock as the *FSP* process RECURSIVE_LOCK with the alphabet **{acquire[p:P],release[p:P]}**.

acquire[p] acquires the lock for thread **p** and **release[p]** releases it.

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