Answer the first question and two further questions.

1. a. Define the meaning of the following constructs of the Finite State Process (FSP) notation.

   i. action prefix ("->")

   [3 marks]

   If x is an action and P a process then `(x->P)` describes a process that initially engages in the action x and then behaves exactly as described by P.

   ii. choice ("|")

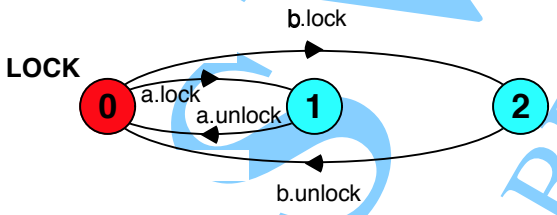
   [3 marks]

   If x and y are actions then `(x->P|y->Q)` describes a process which initially engages in either of the actions x or y. After the first action has occurred, the subsequent behavior is described by P if the first action was x and Q if the first action was y.

   [Subtotal 6 marks]

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

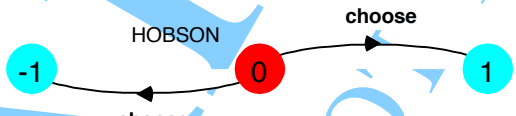
   i.



   [4 marks]

```
LOCK = (a.lock -> a.unlock -> LOCK
       |b.lock -> b.unlock -> LOCK
       ).
```

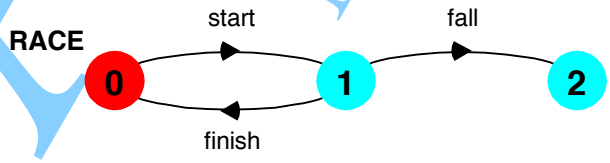
   ii.



   [3 marks]

```
HOBSON  = (choose->STOP | choose->ERROR).
```

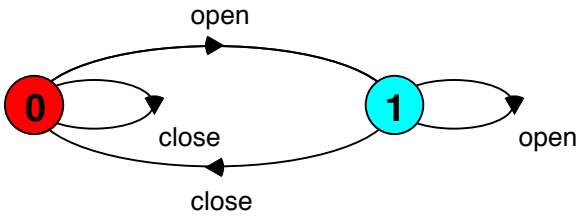
   iii.



   [3 marks]

```
RACE = (start -> (fall->STOP | finish->RACE)).
```

iv.



[4 marks]

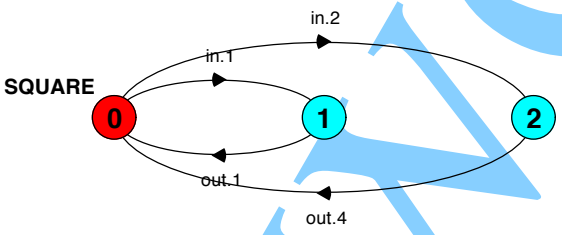
```
DOOR = CLOSED,
CLOSED = (close->CLOSED|open->OPENED),
OPENED = (open ->OPENED|close->CLOSED).
```

[Subtotal 14 marks]

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

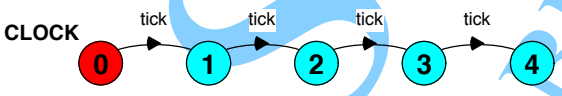
i. `SQUARE = (in[i:1..2]->out[i*i]->SQUARE).`

[3 marks]



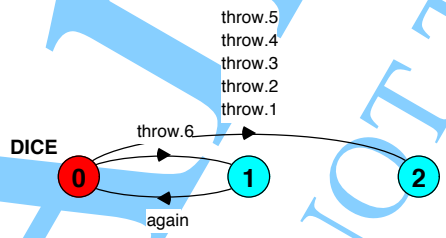
ii.
```
CLOCK          = CLOCK[0],
CLOCK[i:0..4] = (when(i<4) tick->CLOCK[i+1]).
```

[4 marks]



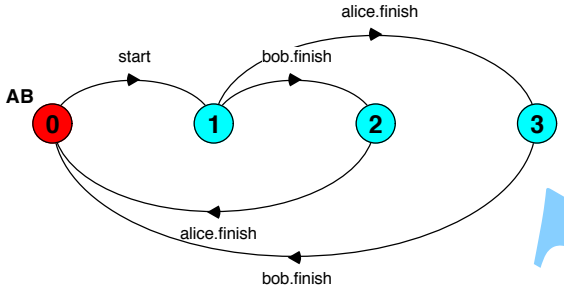
iii. `DICE = (throw[i:1..6]->(when (i==6)again->DICE)).`

[3 marks]



iv.
```
ALICE = (start->alice.finish->ALICE).
BOB   = (start->bob.finish->BOB).
||AB  = (ALICE||BOB).  //draw LTS for AB only
```

[4 marks]



[Subtotal 14 marks]

[Total 34 marks]

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

[10 marks]

The basic format for modeling a guarded action for some condition cond and action act using FSP is shown below:

```
FSP: when cond act -> NEWSTAT
```

The corresponding format for implementing the guarded action for condition cond and action act using Java is as follows:

```
Java:  public synchronized void act()
                throws InterruptedException {
    while (!cond) wait();
    // modify monitor data
    notifyAll()
}
```

b. In an automatic cable car system, each cable car has its own controller. The function of this controller is to ensure that a cable car only leaves the terminus when it is full of passengers. A cable car can hold a maximum of N passengers. After departure, the cable car arrives at the other end, all the passengers leave the cable car and new passengers may then board for another trip. The alphabet of the cable car is depicted below, together with a definition of the meaning of each action.

`board` a passenger boards the cable car.

`cardepart` the cable car departs. This action is delayed until the cable car is full.

`cararrive` the cable car arrives at the other end. This action is delayed until after departure.

Specify the behaviour of CABLECAR in FSP.

[12 marks]

```
CABLECAR(N=6) = PASSENGERS[0],
PASSENGERS[i:0..N]
    = (when(i<N) board -> PASSENGERS[i+1]
      |when(i==N)cardepart->TRIP
),
TRIP = (cararrive -> disembark -> PASSENGERS[0]).
```

c. Implement the CABLECAR specification from part b with the above three actions as monitor methods programmed in Java.

[11 marks]

```
class CableCar {
  final int N = 6;
  int passengers = 0;
  bool departed = false;

  public synchronized void board()
         throws InterruptedException{
    while(passengers==N) wait();
    ++passengers;
    notifyAll();
  }

  public synchronized void cardepart()
         throws InterruptedException{
    while (passengers!=N || departed) wait();
    departed = true;
  }

  public synchronized void cararrive()
         throws InterruptedException{
    while (passengers!=N || !departed) wait();
    passengers=0;
    departed = false;
   notifyAll();
  }
}
```

[Total 33 marks]

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

[8 marks]

`{a1,..,ax}::P` replaces every label n in the alphabet of P with the labels a1.n,...,ax.n. Further, every transition `(n->X)` in the definition of P is replaced with the transitions `({a1.n,...,ax.n}->X)`. Use this in:

```
||RESOURCE_SHARE=(a:USER || b:USER || {a,b}::RESOURCE).
```

b. The cheese counter in a supermarket is continuously mobbed by hungry customers. To restore order, the management installs a ticket machine which issues tickets to customers. Tickets are numbered in the range 1..MT. When ticket MT has been issued, the next ticket to be issued will be ticket numbered 1, i.e. the management install a new ticket roll. The cheese counter has a display which indicates the ticket number of the customer currently being served. The customer with the ticket with the same number as the counter display then goes to the counter and is served. When the service is finished, the number is incremented (modulo MT). Given the structure diagram depicted below for the cheese counter system, specify the behaviour of each of the processes (CUSTOMER, TICKET, COUNTER) and the composite process CHEESE_COUNTER in FSP.

[12 marks]

```
const MT = 4  //maximum ticket number
const MC = 2  //number of customers
range T = 1..MT
range C = 1..MC

CUSTOMER = (ticket[t:T]->getcheese[t]->CUSTOMER).

TICKET = TICKET[1],
TICKET[t:T] = (ticket[t]->TICKET[t%MT+1]).

COUNTER = COUNTER[1],
COUNTER[t:T] = (getcheese[t]->COUNTER[t%MT+1]).

||CHEESE_COUNTER =
(c[C]:CUSTOMER || {c[C]}::TICKET || {c[C]}::COUNTER).
```

c. Implement the specifications for COUNTER and TICKET in Java.

[13 marks]

```
class Ticket {
const MT = 1000;
private int next = 0;

public synchronized int ticket() {
next = next%MT + 1;
return next;
}
}

class Counter {
const MT = 1000;
private int serve = 1;

public synchronized getcheese(int ticket)
throws Interruptedexception {
while (ticket!=serve) wait();
serve = serve%MT + 1;
notifyAll();
}
}
```

[Total 33 marks]

4. a. i. Explain the terms safety property and liveness property with respect to concurrent programs.

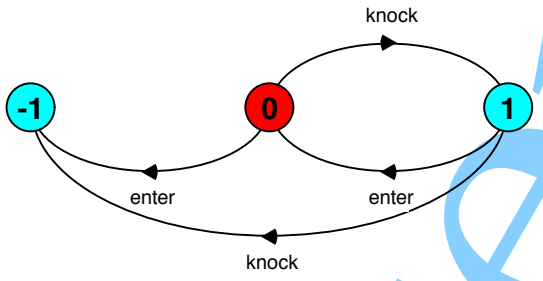
[4 marks]

Safety Properties specify that nothing bad will happen. Liveness Properties determine that something good will eventually happen.

ii. Draw the Labelled Transition System for the following safety property:

```
property POLITE = (knock->enter->POLITE).
```

[4 marks]



[Subtotal 8 marks]

b. A lift has a maximum capacity of ten people. In the model of the lift control system, passengers entering a lift are signalled by an enter action and passengers leaving the lift are signalled by an exit action. Specify a safety property in FSP which when composed with the lift will check that the system never allows the lift that it controls to have more than ten occupants.

[8 marks]

```
property LIFT_TEN = LIFT[0],
LIFT[i:0..10] = (when(i<10) enter -> LIFT[i+1]
                |when(i>0) exit -> LIFT[i-1]
                |when(i==0)exit -> LIFT[0]
                ).
```

c. Explain what is meant by the term deadlock in the context of concurrent programs and explain how LTS models can be used to check for deadlock.

[8 marks]

Deadlock occurs in a system when all its constituent processes are blocked. Another way of saying this is that the system is deadlocked because there are no eligible actions that it can perform.

In the finite state model of a process, a deadlocked state is simply a state with no outgoing transitions. A process in such a state can engage in no further actions. Analysis involves performing a complete search of the state space looking for such states.

d. It is possible for the following system to deadlock. Explain why this deadlock occurs.

```
Alice = (call.bob -> wait.chris -> Alice).
Bob   = (call.chris -> wait.alice -> Bob).
Chris = (call.alice -> wait.bob -> Chris).

||S = (Alice || Bob || Chris) /{call/wait}.
```

The following model attempts to fix the problem by allowing Alice, Bob and Chris to timeout from a call attempt. Is a deadlock still possible? If so describe how the deadlock can occur and give an execution trace leading to the deadlock.

```
Alice = (call.bob -> wait.chris -> Alice
        | timeout.alice -> wait.chris ->Alice).
Bob   = (call.chris -> wait.alice -> Bob
        | timeout.bob -> wait.alice ->Bob).
Chris = (call.alice -> wait.bob -> Chris
        | timeout.chris -> wait.bob ->Chris).

||S = (Alice || Bob || Chris) /{call/wait}.
```

[9 marks]

The first model deadlocks because each participant is trying to call someone who is trying to call someone else. Alice is trying to call Bob who is trying to call Chris who is trying to call Alice. A wait-for cycle exist.

In the model with timeouts, exactly the same situation occurs after three consecutive timeouts since waits are renamed to be calls.

```
Trace to DEADLOCK:
        timeout.alice
        timeout.bob
        timeout.chris
```

[Total 33 marks]

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

[10 marks]

```
class MyThread extends Thread {...}
Thread x = new MyThread();

class MyRun implements Runnable {...}
Thread x = new Thread(new MyRun());
```

b. A Special Savings Building Society Account is permitted to have a maximum balance of M hundred pounds. Savers may deposit one hundred pounds at a time into the account up to the maximum. They may withdraw money in multiple units of a hundred pounds so long as the account is not overdrawn. The alphabet of the process that models the savings account is depicted below, together with a definition of the meaning of each action.

```
range T = 1..M
```

`deposit`	deposit one hundred pounds. This action is blocked if the balance would exceed M.
`withdraw[T]`	withdraw an amount in the range T hundred pounds. This action is blocked if sufficient funds are not available.

Specify the behaviour of ACCOUNT in FSP.

[12 marks]

```
const M = 6
range B = 0..M
range T = 1..M

ACCOUNT       = BALANCE[0],
BALANCE[b:B] = (when (b<M) deposit          -> BALANCE[b+1]
               |when (b>0) withdraw[a:1..b] -> BALANCE[b-a]
               ).
```

c. Implement the ACCOUNT specification from part b with the two actions as monitor methods programmed in Java.

[11 marks]

```
class Account {
  int balance =0;
  int M;

  Account(int maximum) {M = maximum}

  synchronized void deposit() throws InterruptedException {
    while (balance==M) wait();
    ++balance;
    notifyAll();
  }

  synchronized void withdraw(int amount) {
  throws InterruptedException {
    while (balance-amount<0) wait();
    balance -= amount;
    notifyAll();
  }
}
```

[Total 33 marks]

END OF PAPER