Concurrency, 3C03, 2002

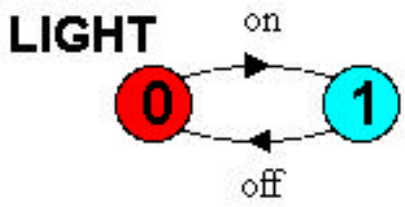
Answer Question 1 and two further questions.

Marks for each part of each question are indicated in square brackets

Calculators are NOT permitted

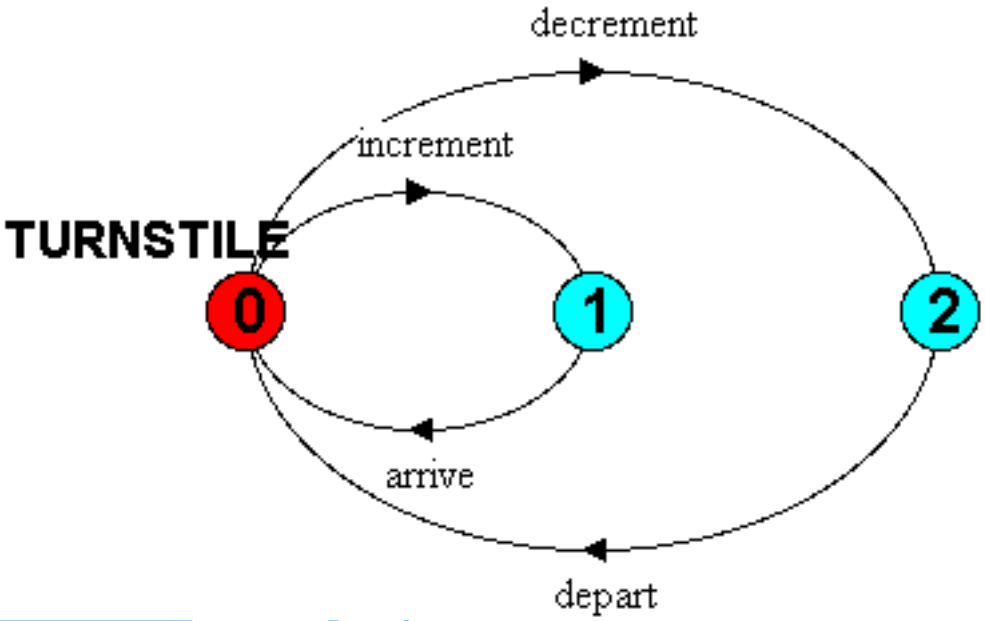
1. a. Show an equivalent labelled transition system for each of the following FSP processes.

i. `LIGHT = ( on -> off -> LIGHT).`



[2 marks]

ii. `TURNSTILE = ( increment -> arrive -> TURNSTILE`
`| decrement -> depart -> TURNSTILE ).`

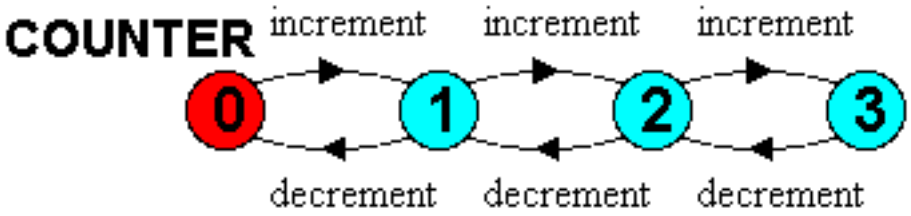


[3 marks]

iii. `COUNTER = COUNTER[0],`

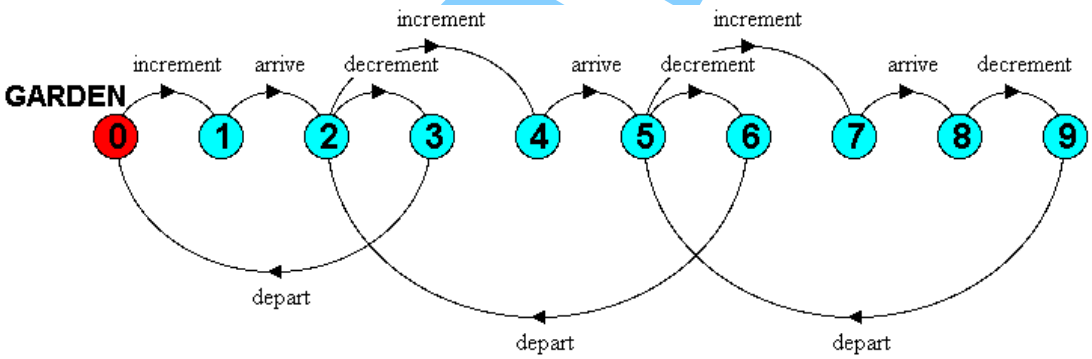
`COUNTER[x:0..3]=(when (x>0) decrement->COUNTER[x-1]`

`|when (x<3) increment->COUNTER[x+1]).`



[3 marks]

iv. `||GARDEN =(TURNSTILE || COUNTER).`



[6 marks]

[Subtotal 14 marks]

b. Web servers implement the HTTP protocol, which mainly consists of the operations `http_put` and `http_get`. Browsers need to establish a connection to a Web server in order to obtain a Web page with an `http_get` request or to send a form with an `http_put` request. Unlike FTP servers, Web servers do not maintain session information, but browsers need to `open` a TCP connection to a Web server prior to issuing a request and `close` them afterwards. Web servers might delay browsers if they are too busy.

Use FSP to model a concurrent system with one Web server and six browsers. Restrict servers to open up to four concurrent connections to browsers.

```
BROWSER = (open -> OPEN_CONN),
OPEN_CONN = (http_put -> REQUESTED
            |http_get -> REQUESTED),
REQUESTED = (close -> BROWSER).

WEBSERVER = WEBSERVER[0],
WEBSERVER[i:0..4]=(when (i<4)open->WEBSERVER[i+1]
                  |when (i>0) close->WEBSERVER[i-1]
                  |http_put->WEBSERVER[i]
                  |http_get->WEBSERVER[i]).

set Clients = {sally, alice, bob, john, peter, jill}

||WEB=(Clients:BROWSER||{Clients}::WEBSERVER).
```

[10 marks]

c. Give a safety property that you can use to prove that your Web server never opens more than four connections at a time and discuss how you would conduct the proof using the LTSA model checker.

```
property NOTOVERLOADED = COUNT[0],
COUNT[j:0..4]=(when (j<4)open->COUNT[j+1]
              |when (j>0)close->COUNT[j-1]).
```

Then use parallel composition to combine the property with the above propose.

```
||TEST=(WEB||{Clients}::NOTOVERLOADED).
```

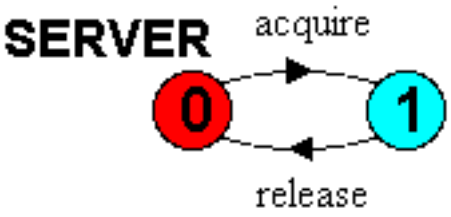
Then use reachability analysis provided by the model checker to check for reachability of the error state.

[10 marks]

[Total 34 marks]

2. a. For the following labelled transition systems, show an FSP process from which they could have been derived. Use the parallel composition operator where possible.

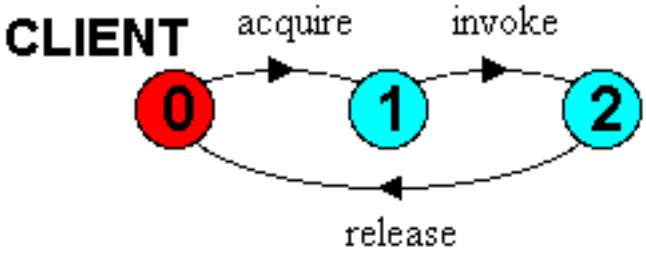
i.




```
SERVER=(acquire->release->SERVER).
```

[2 marks]

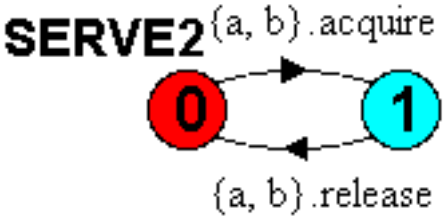
ii.




```
CLIENT = (acquire->invoke->release->CLIENT).
```

[2 marks]

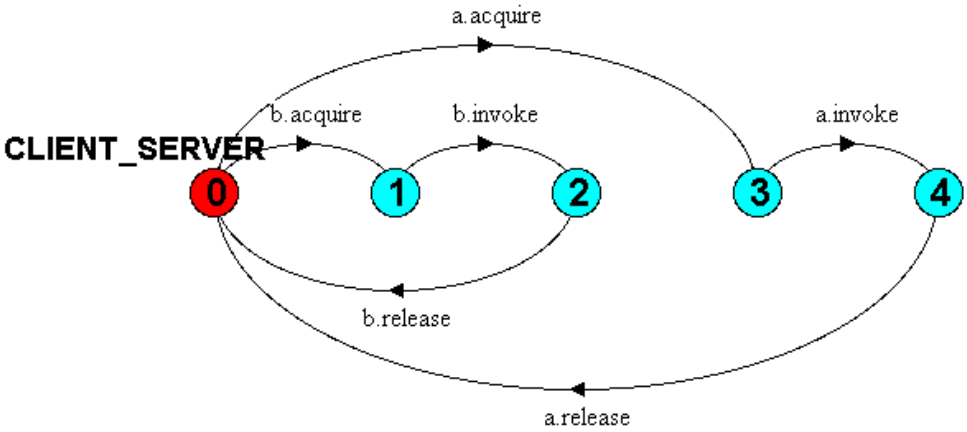
iii.




```
||SERVE2=({a,b}::SERVER).
```

[3 marks]

iv.




```
||CLIENT_SERVER = (a:CLIENT || b:CLIENT || {a,b}::SERVER).
```

[6 marks]

[Subtotal 13 marks]

b. Show the Java thread life cycle as an FSP process.

```
THREAD = CREATED,
CREATED = ( start -> RUNNING
          | stop -> TERMINATED),
RUNNING = ( sleep -> NON_RUNNABLE
          | yield -> RUNNABLE
          |{stop, end} ->TERMINATED
          | run -> RUNNING),
RUNNABLE= ( sleep -> NON_RUNNABLE
          | dispatch -> RUNNING
          | stop -> TERMINATED),
NON_RUNNABLE = ( wake ->RUNNABLE
               | stop ->  TERMINATED),
TERMINATED = STOP.
```

[4 marks]

c. Give an FSP model for a simple scheduler that could be used in a JVM for threads to achieve the interleaved model of concurrency. Assuming that you build a JVM for a single processor machine, your scheduler needs to ensure that only one thread can run at a time.

Tip: Threads are dynamically created and stopped, but in reality the JVM reuses threads from a threadpool that has a constant number of threads.

```
set ThreadPool = {t1,t2,t3,t4,t5}
JVM = ([t:ThreadPool].{start,dispatch} ->
       [t].{stop,sleep,yield} ->JVM).
||JAVA=(ThreadPool:THREAD || JVM).
```

[10 marks]

d. Give the liveness property that you would use to prove that every thread will eventually be scheduled to run.

```
progress Fair = {[ThreadPool].run}
```

[6 marks]

[Total 33 marks]

3. a. Give the alphabets for each of the following FSP processes.

i.
```
SERVER=(lock->compute->unlock->SERVER)
       \{compute}.
{ lock,
  unlock
}
```

[2 marks]

ii.
```
CLIENT = (acquire->invoke->release->CLIENT)
         @{acquire,release}.
{ acquire,
  release
}
```

[2 marks]

iii.
```
||ISOLATED=(a:CLIENT || b:CLIENT || {a,b}::SERVER).
{ a.acquire,
  a.lock,
  a.release,
  a.unlock,
  b.acquire,
  b.lock,
  b.release,
  b.unlock
}
```

[4 marks]

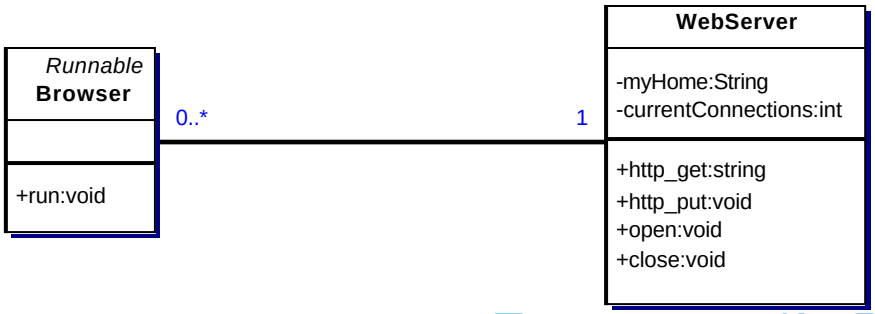
iv.
```
set Clients={a,b}
||CLIENT_SERVER = (Clients:CLIENT || Clients::SERVER)
                  /{Clients.lock/Clients.acquire,
                    Clients.unlock/Clients.release}.
{ a.lock,
  a.unlock,
  b.lock,
  b.unlock
}
```

[5 marks]

[Subtotal 13 marks]

b. Consider the Web server and browser model that you developed in response to Question 1.b.

Assuming the use of Java as a programming language, design the static structure of a concurrent system of Web servers and browsers in a UML class diagram. Assume your system has one Web server and six browsers. In the class diagram, show operations and attributes of your own classes and if necessary associations to classes that you import from the Java API.



[10 marks]

c. Show the Java code of the `open` and `close` operations of the Web server. Use condition synchronization to ensure that no more than four connections are opened at the same time.

[10 marks]

```
public class WebServer {
  ...

  public void open(){
    try {
      while (currentConnections>4) wait();
    } catch (InterruptedException e){}
    currentConnections++;
  }

  public void close() {
    currentConnections--;
    NotifyAll();
  }

  private String myHome;
  private int currentConnections;
}
```

[Total 33 marks]

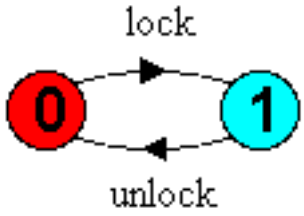
4. a. Describe the concept of transactions by discussing what atomicity, consistency, isolation and durability mean.

[10 marks]

Transactions are sequences of operations that are atomic, consistency preserving, isolated and durable. Atomicity means that the sequence is performed completely or not at all. Consistency preservation means that the sequence leads from one consistent state to another and inconsistencies are confined to within the transaction. Isolation means that the transaction is executed in isolation to any concurrent transactions and durability means that once the transaction is completed its changes will persist.

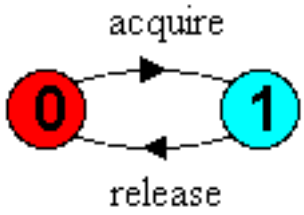
b. Compute the labelled transition systems for the following FSP processes and give them in their minimised form.

i. `SERVER=(lock->compute->unlock->SERVER)`
`\{compute}.`



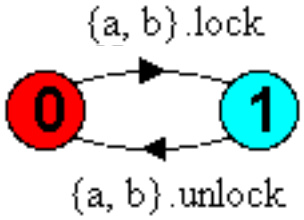
[3 marks]

ii. `CLIENT = (acquire->invoke->release->CLIENT)`
`@{acquire,release}.`



[3 marks]

iii.
```
set Clients={a,b}
||CLIENT_SERVER = (Clients:CLIENT || Clients::SERVER)
                  /{Clients.lock/Clients.acquire,
                    Clients.unlock/Clients.release}.
```



[7 marks]

[Subtotal 13 marks]

c. How can the liveness property that servers will eventually be able to `compute` a function be proven for the above `CLIENT_SERVER` process?

Using the progress property

```
progress COMPUTATION = {[Clients].compute}
```

and by removing the hiding operator from process SERVER

[10 marks]

[Total 33 marks]

5. a. In no more than 50 words each define the following concepts:

i. Semaphore

[2 marks]

ADT with encapsulated counter and waiting list. Exports two operations signal and wait. wait checks whether counter is greater than 0. If yes decrements counter, otherwise appends calling process to waiting list. If there are waiting processes, signal wakes up first of these, otherwise it increments the counter.

ii. Monitor

[2 marks]

A programming language concept that guarantees mutually exclusive access to a data structure encapsulated by the monitor.

iii. Deadlock

[2 marks]

A set of processes mutually waiting for each other.

iv. Livelock

[2 marks]

A process spinning while waiting for a condition that will never become true.

v. Condition Synchronization

[2 marks]

Condition synchronization is implemetned inside a monitor, forces calling process to wait (outside the monitor) if conditions are not yet met. Only let them proceed when conditions are met and notifies other processes if changes to data structures have occurred.

vi. Safety Property

[2 marks]

Ascertain that nothing bad will ever happen

vii. Liveness Property

[2 marks]

Ascertain that something desirable will eventually happen

[Subtotal 14 marks]

b. Give an FSP process that is behaviourally equivalent to the `||CLIENT_SERVER` below without using parallel composition.

```
SERVER=(lock->compute->unlock->SERVER)
       \{compute}.
CLIENT = (acquire->invoke->release->CLIENT)
         @{acquire,release}.
set Clients={a,b}
||CLIENT_SERVER = (Clients:CLIENT || Clients::SERVER)
                  /{Clients.lock/Clients.acquire,
                    Clients.unlock/Clients.release}.
```

Many possible answers here, e.g.

```
CS=(a.lock -> CS_LOCKED
   |b.lock -> CS_LOCKED),
CS_LOCKED = (a.unlock -> CS
            |b.unlock -> CS).
```

[11 marks]

c. It is possible to use both semaphores and monitors for achieving mutual exclusion.

i. Give a scenario where semaphores are preferrable over monitors and argue why you would not use monitors in this setting.

For synchronization of concurrent C or Assembler programs we would need to use semaphores as monitors are programming language concepts and these languages do not provide monitors.

[4 marks]

ii. Give a scenario where monitors are preferrable to semaphores and argue why you would not use semaphores in this setting.

(Condition) synchronization in Java, Occam or Ada is best done using monitors as it avoids the disadvantages of of sempahores (e.g enforcing mutual exclusion at a language level).

[4 marks]

[Subtotal 8 marks]

[Total 33 marks]

END OF PAPER