UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science

Advanced Operating Systems

Class Exercise - Lecture 2

August 8, 2013

Your Questions:

1. You are given the following FSP definition of two processes, with parallel composition:

```
TINA=(requestFrom.donna->wait.donna->TINA).
DONNA=(requestFrom.tina->wait.tina->DONNA).
||GIRLS=(TINA || DONNA).
```

Draw the composition state space, showing the transitions and actions between the different states.

ANSWER: The required composition state space diagram as follows:

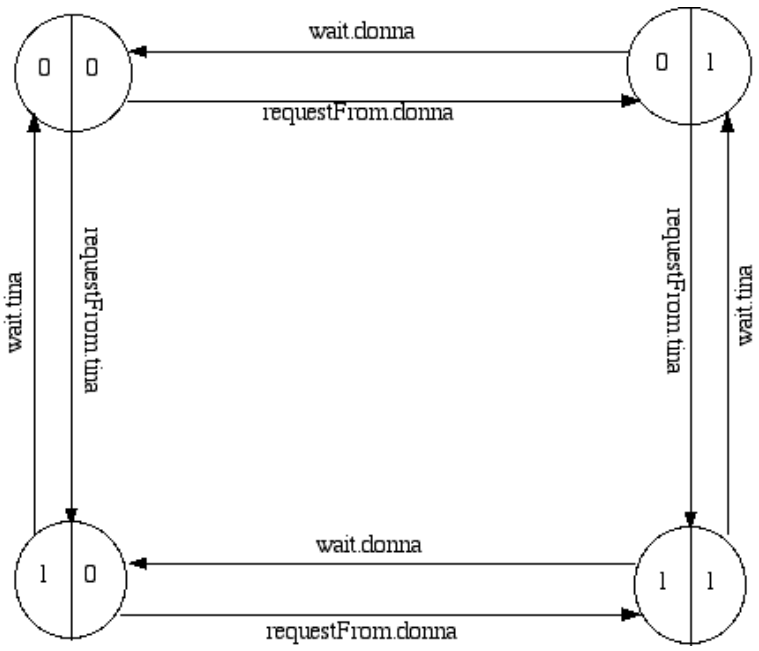

Figure 1: GIRLS Composition State Space Diagram

2. Using the composition state space diagram you drew in the previous question, draw the finite state machine for the composite process, `GIRLS`.

   ANSWER: The required FSM as follows:

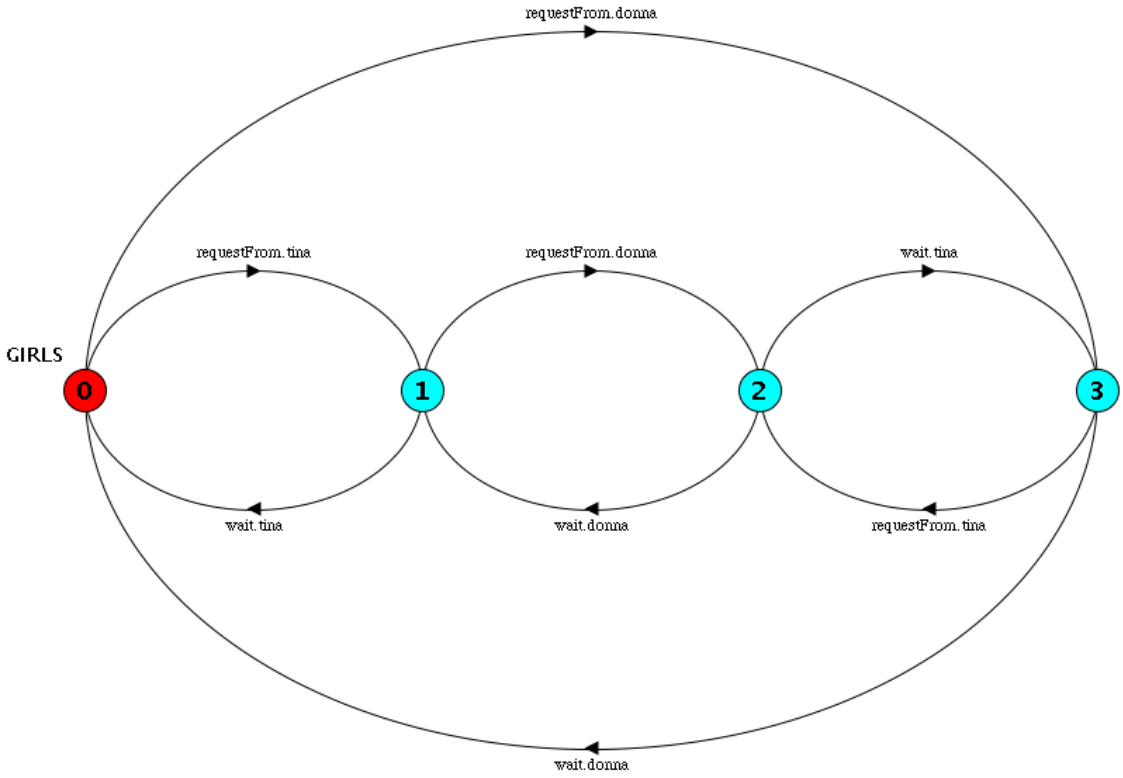

Figure 2: GIRLS Finite State Machine

3. Draw a composition state space diagram for the following composite process, `COUPLE` (adapted from [1]).

```
FLO = (start->flo.finish->FLO).
ANDY = (start->andy.finish->ANDY ).
||COUPLE = (FLO || ANDY ).
```

   ANSWER: The required state space diagram as follows:

4. Provide an equivalent finite state machine for the composition state space you drew in the previous question.

   ANSWER: The equivalent FSM as follows:

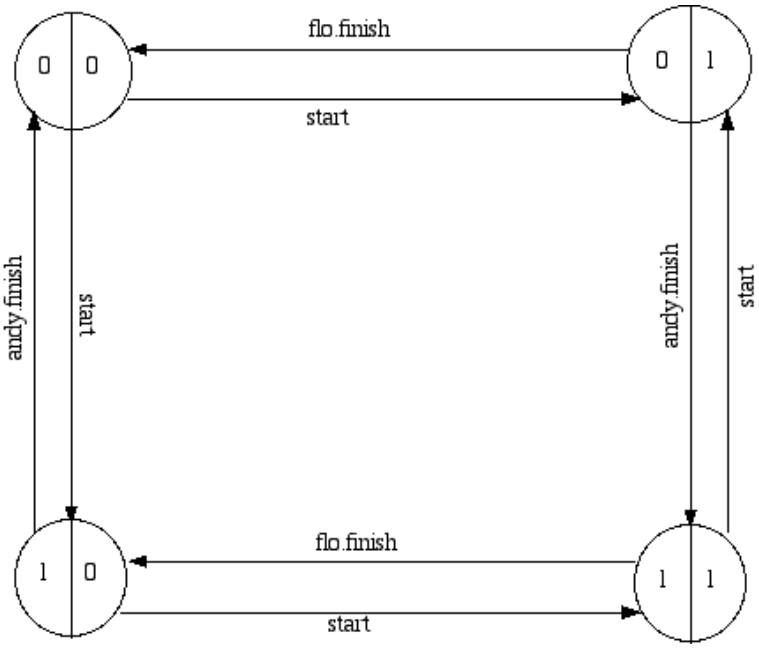

Figure 3: COUPLE Composition State Space Diagram

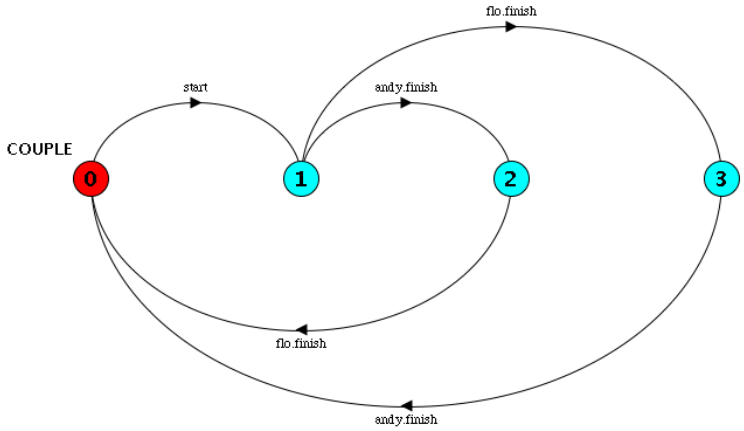

Figure 4: COUPLE finite state machine

5. 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 consists of the following actions:

   `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. The action is delayed until after departure.

   Write the FSP definition for the `CABLECAR` process ([1]).

ANSWER: The FSP definition as follows:

```
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]).
```

6. Draw the finite state machine that corresponds to your definition of `CABLECAR` in the previous question.

   ANSWER: The LTS as follows:

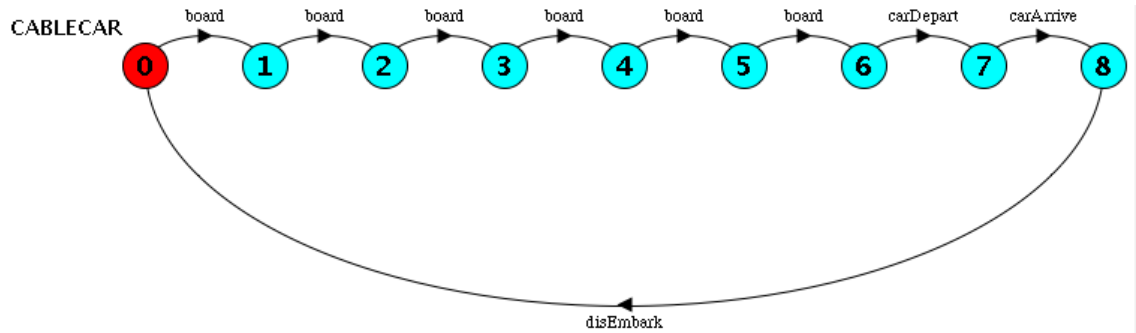

Figure 5: TRIP Finite State Machine

7. The cheese counter in a supermarket is continuously mobbed by hungry customers. To restore order, the management has installed a ticket machine which issues tickets to customers. Tickets are numbered in the range `1..MT`, where `MT` is some maximum ticket number. When ticket `MT` has been issued, the next ticket to be issued will be numbered 1, at which point a new ticket roll is installed by management. The cheese counter has a display that indicates the ticket number of the customer currently being served. When the number first comes up, the customer whose ticket number is the same as the one being displayed on the counter goes to the counter to be served. After s(he) has been served, the number is incremented (`modulo MT`).

   Using FSP, define the behaviour of each of the following processes. For each process, draw its corresponding finite state machine.

   To begin with with, use the following declarations:

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

- CUSTOMER process.
- TICKET process.
- COUNTER process.
- CHEESE_COUNTER, the composite process.

ANSWER: The FSPs and FSMs as follows:

(a) The CUSTOMER process FSP and FSM as follows:

i. The CUSTOMER FSP

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

ii. The CUSTOMER FSM

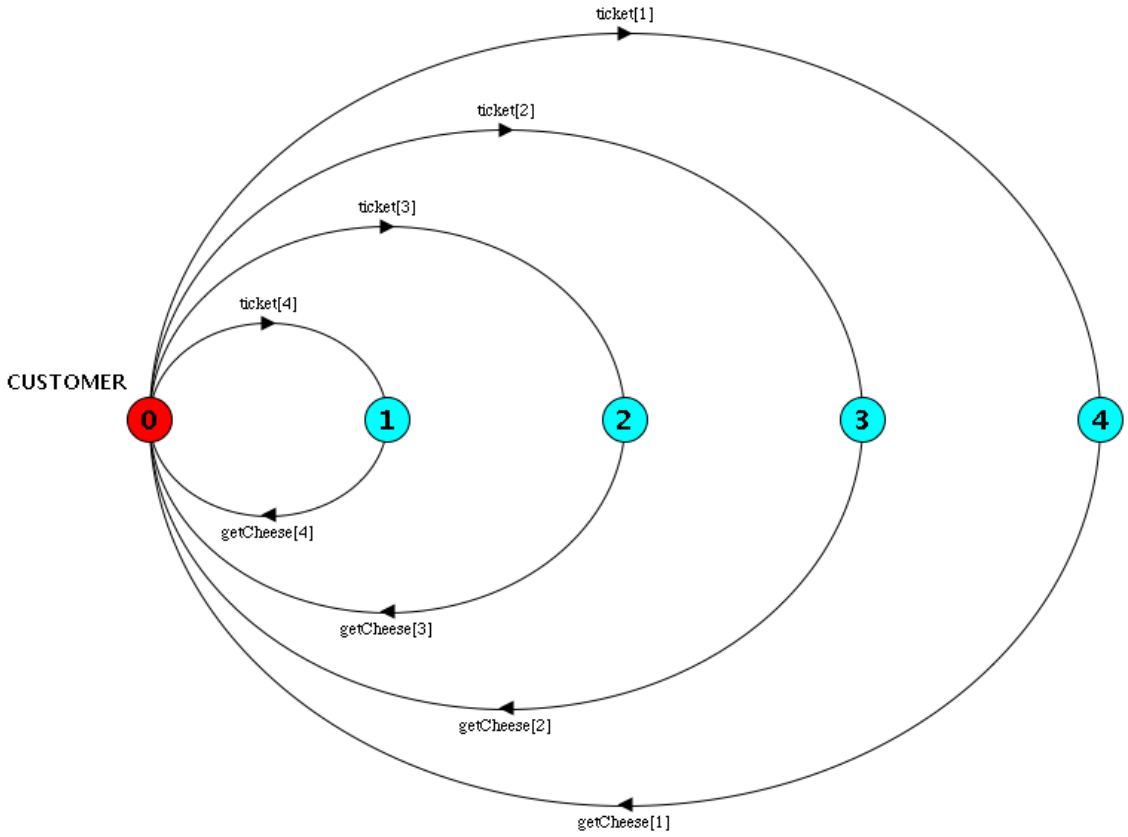

Figure 6: CUSTOMER Finite State Machine

(b) The TICKET process FSP and FSM as follows.

i. The TICKET FSP

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

ii. The `TICKET` FSM

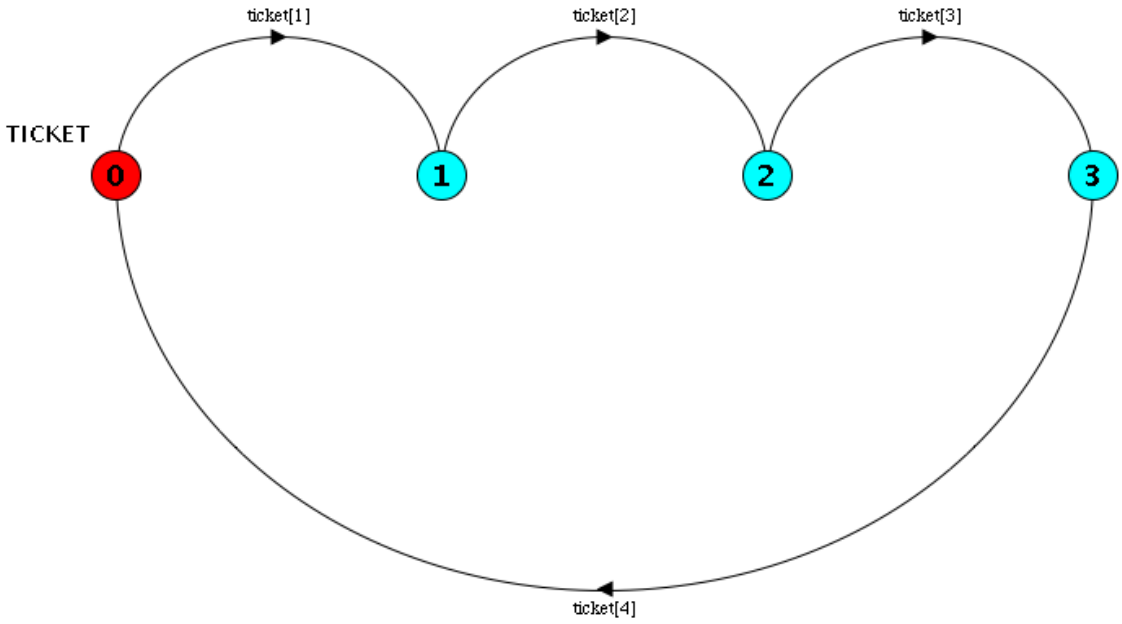

Figure 7: TICKET Finite State Machine

(c) The `COUNTER` process FSP and FSM.

i. The `COUNTER` FSP.

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

ii. The `COUNTER` FSM

(d) The `CHEESE_COUNTER` FSP and FSM.

i. The `CHEESE_COUNTER` FSP

```
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],
```

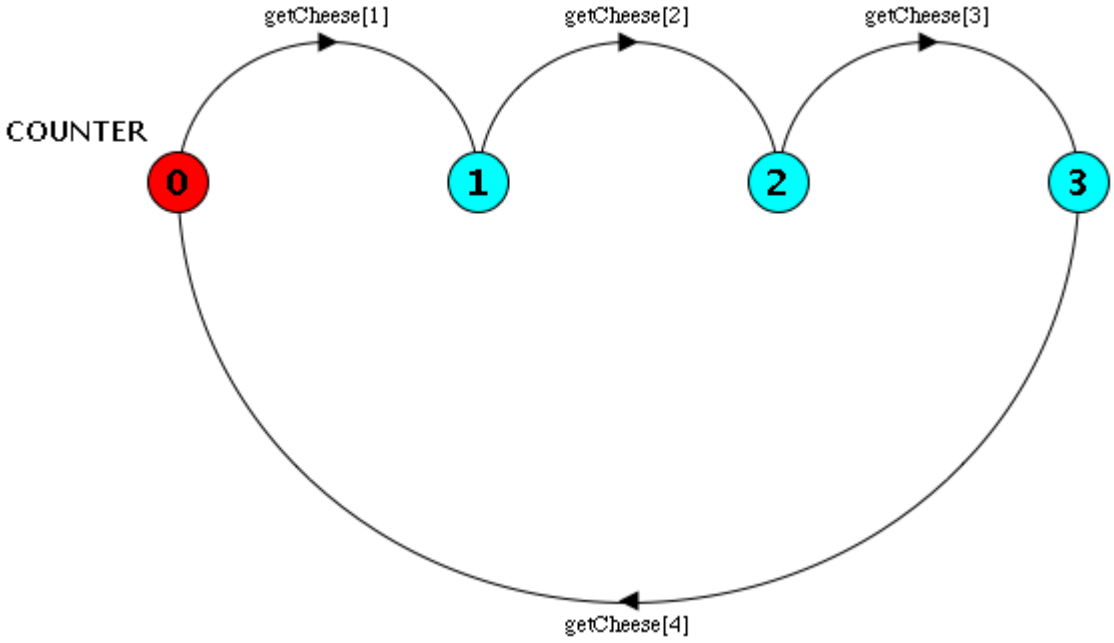

Figure 8: COUNTER Finite State Machine

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

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

ii. The `CHEESE_COUNTER` FSM

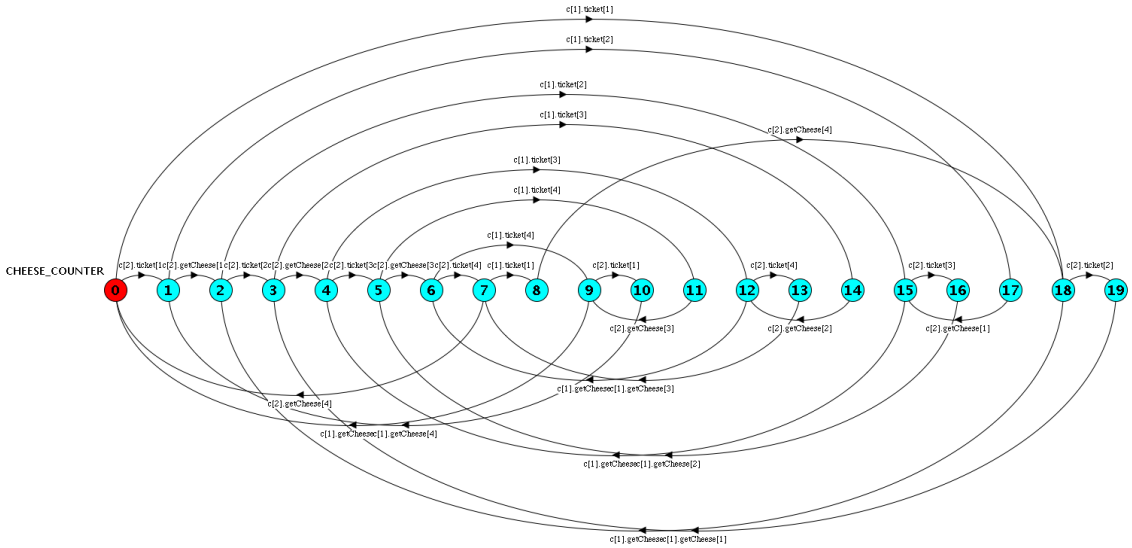

Figure 9: CHEESE_COUNTER Finite State Machine

8. The finite state machine for process `A` is shown below:

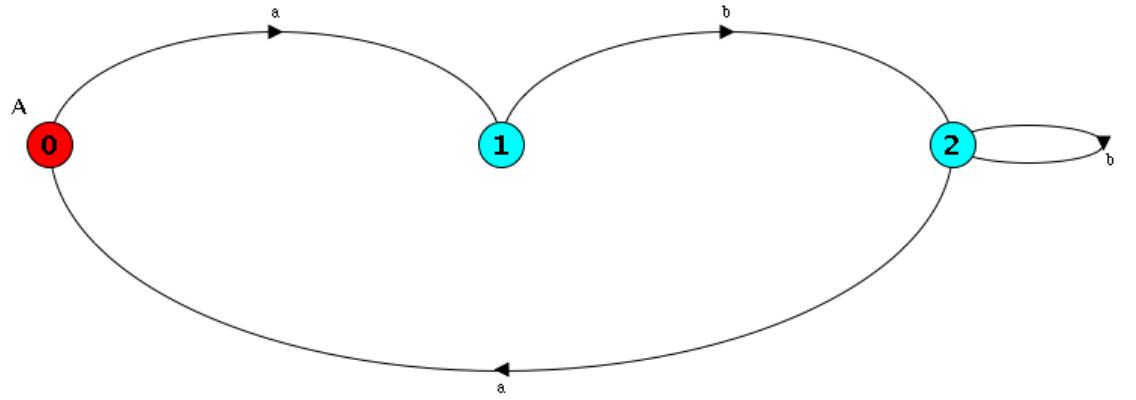

Figure 10: Process A's Finite State Machine

Write the corresponding FSP definition.

ANSWER: The required FSP as follows:

```
A=(a->b->B),
B=(b->B | a->A).
```

9. Study the following FSP definition of a composite process, `DO_IT`.

```
DO=(first->second->DO).
IT=(third->IT).
||DO_IT=(DO || IT)\{second}.
```

Draw the composition state space for this FSP.

ANSWER: The required state space as follows:

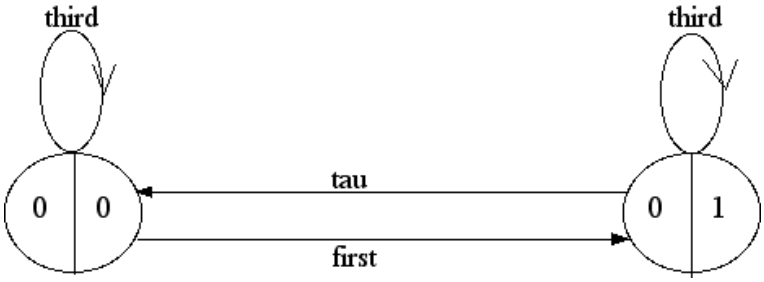

Figure 11: DO_IT Composition State Space

10. Draw the finite state machine corresponding to the FSP defined in the previous question.

    ANSWER: The required FSM as follows:

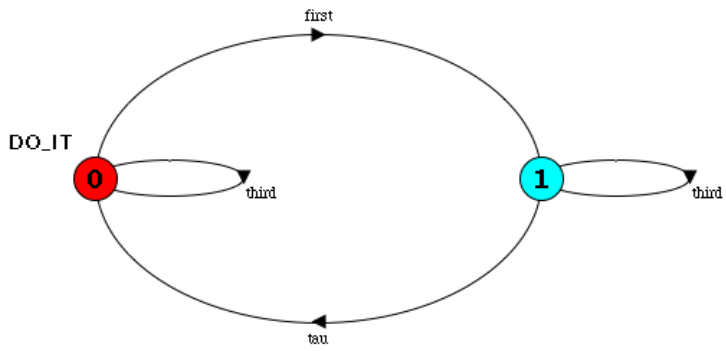

Figure 12: DO_IT Finite State Machine

11. Draw the equivalent finite state machines for the following FSP processes:

    (a) The `COUNTER` process

```
COUNTER=COUNTER[0],
COUNTER[x:0..3]=(when (x>0) decrement->COUNTER[x-1]
                | when (x<3) increment->COUNTER[x+1]).
```

    ANSWER: The required LTS as follows:

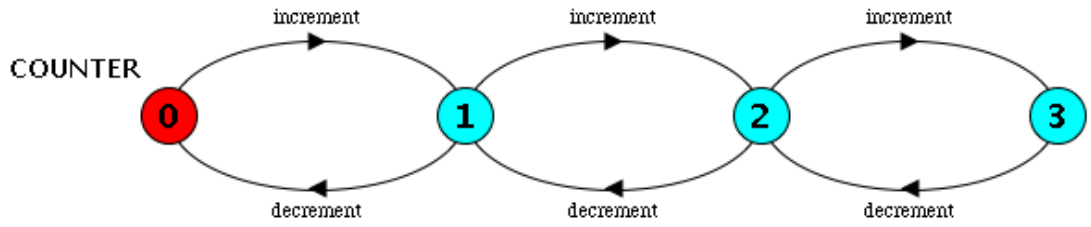

Figure 13: COUNTER Finite State Machine

    (b) The `TURNSTILE` process

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

    ANSWER: The required LTS as follows:

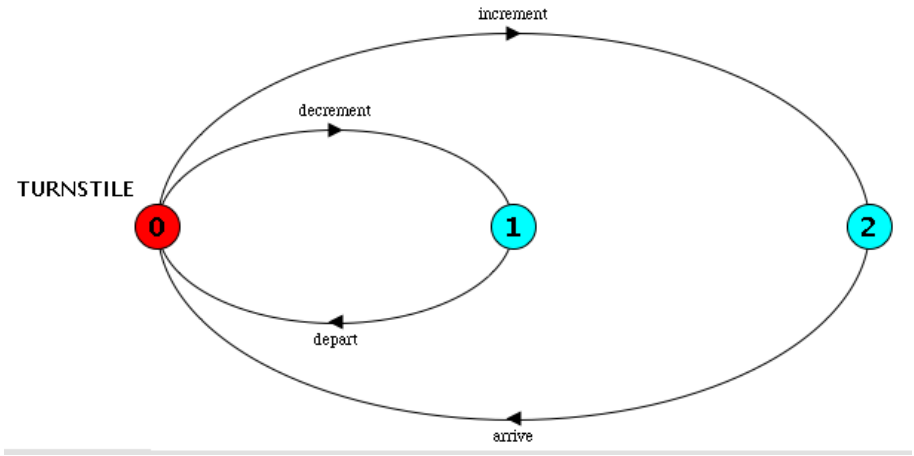

Figure 14: TURNSTILE Finite State Machine

(c) The composite process, `GARDEN`

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

ANSWER: The LTS as follows:

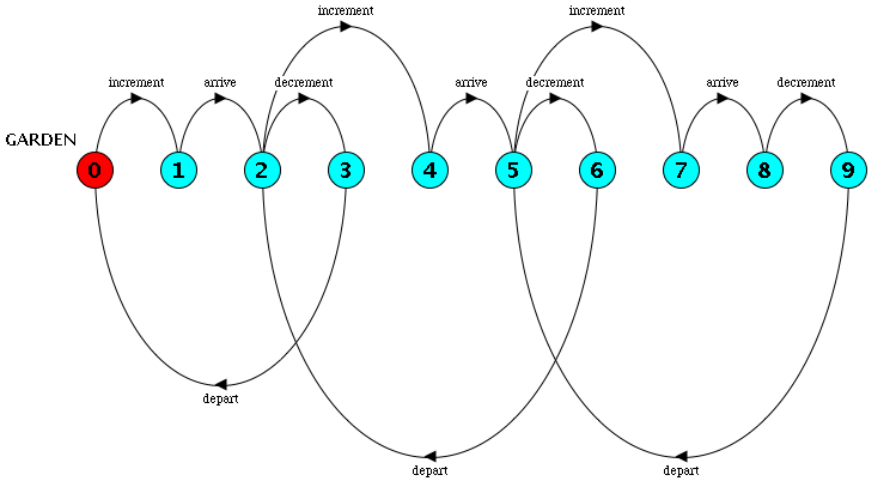

Figure 15: GARDEN Finite State Machine

12. Consider a stack of fixed capacity. The stack has two operations, `push` and `pop`. Use FSP to model the behaviour of a stack in such a way that your stack process guarantees that never more elements are popped from the stack than have previously been pushed onto it. **ANSWER:** The

required FSP as follows:

```
const int SIZE = 5

STACK = STACK[0],
STACK[x:0..SIZE] = (when (x<SIZE) push->STACK[x+1]
                   | when (x>0) pop->STACK[x-1]).
```

ANSWER: The corresponding LTS as follows:

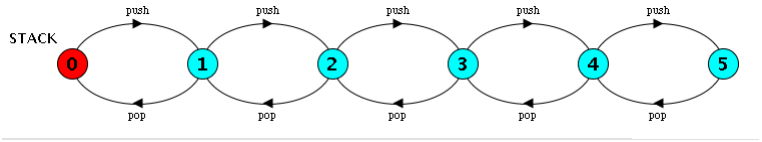

Figure 16: STACK Finite State Machine