6SENG006W Concurrent Programming

FSP Process Analysis & Design Form

Name	Harini Rodrigo
Student ID	2019754
Date	19/12/2023

1. FSP Process Attributes

Attribute	Value
Name	TICKET_PRINTER
Description	Composition process containing 5 subprocesses. Three Passenger (pas1, pas2, pas3) and two technicians (paperTech, tonerTech).
Alphabet	Alphabet: {paperTech.{acquirePaperRefill, paperRefill, releasePaperRefill}, {pas1, pas2, pas3}.{acquireTicketMachine, printTicket, releaseTicketMachine}, tonerTech.{acquireTonerRefill, releaseTonerRefill, tonerRefill}}
Number of States	20
Deadlocks (yes/no)	no
Deadlock Trace(s) (if applicable)	no

2. FSP Process Code

FSP Process:

```
/*
Author: Harini Rodrigo
UoW Id: w1809819
IIT Id: 2019754
*/

// Define the set of technicians (TEC) and passengers (PAS)
set TEC = {paperTech, tonerTech}
set PAS = {pas1, pas2, pas3}

// Define constant values for machine parameters
const MAX_PAPER_LEVEL = 3
const MIN_PAPER_LEVEL = 0
const MIN_TONER_LEVEL = 0
const MAX_TONER_LEVEL = 3
const TICKET_COUNT = 3

// Define ranges for paper and toner levels
range PAPER_RANGE = MIN_PAPER_LEVEL..MAX_PAPER_LEVEL
range TONER_RANGE = MIN_TONER_LEVEL..MAX_TONER_LEVEL

// Define the TICKET_PRINTER process with paper and toner levels
TICKET_PRINTER =
TICKET_PRINTER[MAX_PAPER_LEVEL][MAX_TONER_LEVEL],
TICKET_PRINTER[paper: PAPER_RANGE][toner: TONER_RANGE] = (
    // Transition for printing a ticket when both paper and toner levels are
sufficient
    when (paper > MIN_PAPER_LEVEL & toner > MIN_PAPER_LEVEL)
        acquireTicketMachine -> printTicket -> releaseTicketMachine ->
TICKET_PRINTER[paper - 1][toner - 1] |

    // Transition for acquiring paper refill when paper level is minimum
    when (paper == MIN_PAPER_LEVEL)
        acquirePaperRefill -> paperRefill -> releasePaperRefill ->
TICKET_PRINTER[MAX_PAPER_LEVEL][toner] |

    // Transition for acquiring toner refill when toner level is minimum
    when (toner == MIN_TONER_LEVEL)
```

```
        acquireTonerRefill -> tonerRefill -> releaseTonerRefill ->
TICKET_PRINTER[paper][MAX_TONER_LEVEL]
).

// Define the PAPER_TECH process with paper fill operation
PAPER_TECH = (
    acquirePaperRefill -> paperRefill -> releasePaperRefill -> PAPER_TECH |
    exit -> END
).

// Define the TONER_TECH process with toner refill operation
TONER_TECH = (
    acquireTonerRefill -> tonerRefill -> releaseTonerRefill -> TONER_TECH |
    exit -> END
).

// Define the PASSENGER process with ticket printing operation
PASSENGER(TICKETS = TICKET_COUNT) = PASSENGER[TICKETS],
PASSENGER[ticket: 0..TICKETS] = (
    // Transition for acquiring a ticket and printing it
    when (ticket > 0)
        acquireTicketMachine -> printTicket -> releaseTicketMachine ->
PASSENGER[ticket - 1] |

    // Transition for exiting when no more tickets are needed
    when (ticket == 0)
        exit -> END
).

// Parallel composition of processes to form the TICKET_MACHINE system
|| TICKET_MACHINE = (
    pas1: PASSENGER(1) / {exit / pas1.exit}
    || pas2: PASSENGER(1) / {exit / pas2.exit}
    || pas3: PASSENGER(1) / {exit / pas3.exit}
    || paperTech: PAPER_TECH / {exit / paperTech.exit}
    || tonerTech: TONER_TECH / {exit / tonerTech.exit}
    || {PAS,TEC}::TICKET_PRINTER / //Define transition
    {
        // Action relabeling
        paperTech.acquirePaperRefill / {tonerTech.acquirePaperRefill,
PAS.acquirePaperRefill},
        paperTech.paperRefill / {tonerTech.paperRefill, PAS.paperRefill},
        paperTech.releasePaperRefill / {tonerTech.releasePaperRefill,
```

```
PAS.releasePaperRefill},
        tonerTech.acquireTonerRefill / {paperTech.acquireTonerRefill,
PAS.acquireTonerRefill},
        tonerTech.tonerRefill / {paperTech.tonerRefill, PAS.tonerRefill},
        tonerTech.releaseTonerRefill / {paperTech.releaseTonerRefill,
PAS.releaseTonerRefill},
        PAS.acquireTicketMachine / TEC.acquireTicketMachine,
        PAS.printTicket / TEC.printTicket,
        PAS.releaseTicketMachine / TEC.releaseTicketMachine
    }
).
```

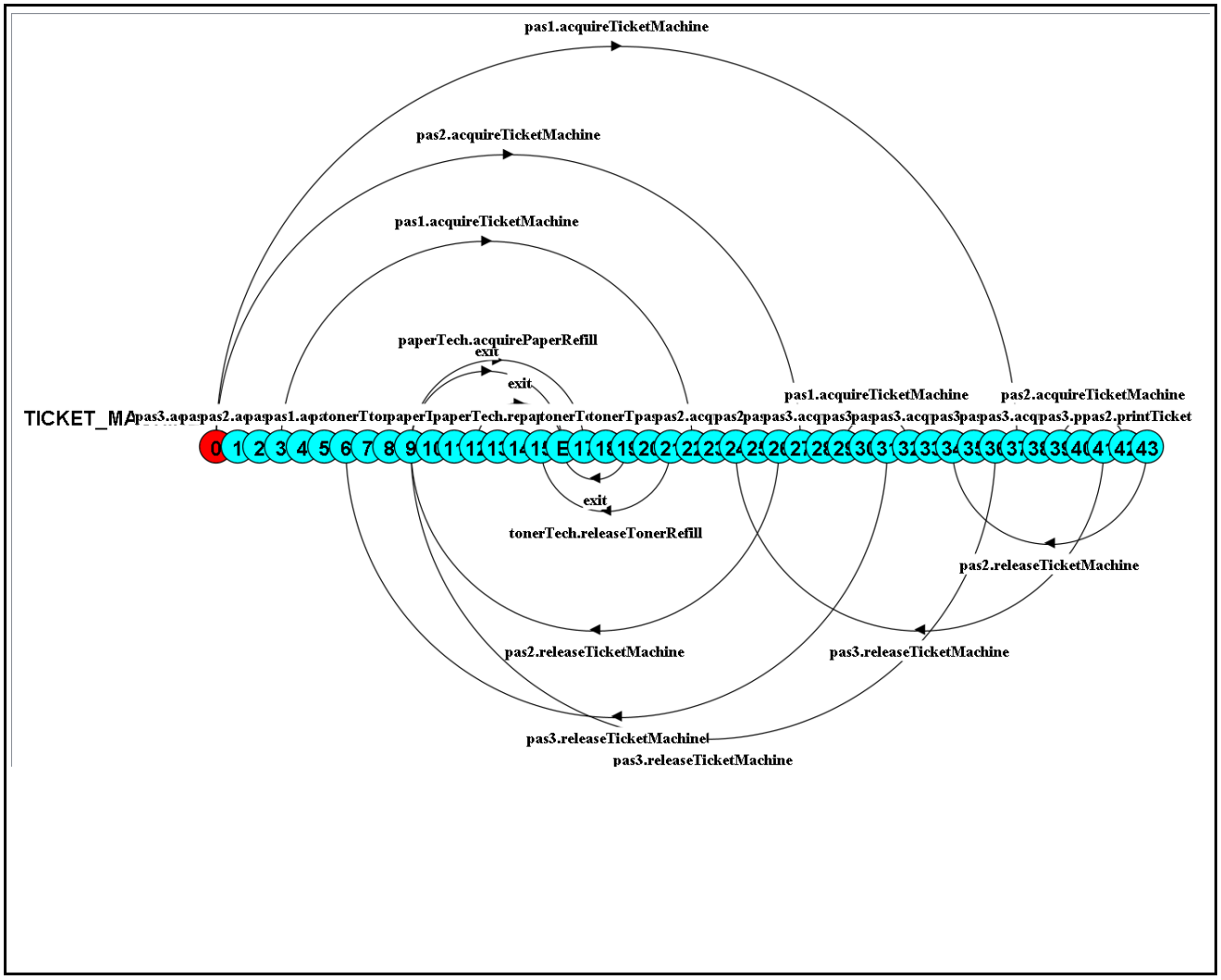
3. Actions Description

A description of what each of the FSP process' actions represents, i.e. is modelling. In addition, indicate if the action is intended to be synchronised (shared) with another process or asynchronous (not shared). (Add rows as necessary.)

Actions	Represents	Synchronous or Asynchronous
acquireTicketMachine	Requesting access to the ticket machine for printing a ticket.	Synchronous
printTicket	Initiating the ticket printing process.	Synchronous
releaseTicketMachine	Releasing the ticket machine after printing a ticket.	Synchronous
acquireTonerRefill	Requesting access to the toner refill operation.	Synchronous
tonerRefill	Performing the toner refill operation.	Synchronous
releaseTonerRefill	Releasing the toner refill operation.	Synchronous
acquirePaperRefill	Requesting access to the paper refill operation.	Synchronous
paperRefill	Performing the paper refill operation.	Synchronous
releasePaperRefill	Releasing the paper refill operation.	Synchronous
exit	Exiting the respective process	Synchronous

4. FSM/LTS Diagrams of FSP Process

Note that if there are too many states, more than 64, then the LTSA tool will not be able to draw the diagram. In this case draw small diagrams of the most important parts of the complete diagram.



5. LTS States

A description of what each of the FSP process' states represents, i.e. is modelling. If there are a large number of states then you can group similar states together &/or only include the most important ones. For example, identify any states related to mutual exclusion (ME) & the associated critical section (CS), e.g. waiting to enter the CS state, in the CS state(s), left the CS state. (Add rows as necessary.)

State	Represents
0	This is the initial state to achieve mutual exclusion (ME) access for the ticket printing machine.
1,4,7,22,25,27,30,32,35,37,40,42	In this stage, passengers are waiting to enter the critical section (CS) state. The passenger who acquires the lock will enter the CS state, where a passenger can use the printing machine to print the tickets.
2,5,8,23,26,28,31,33,36,38,41,43	This is the CS state where the passenger, who acquires the ticket machine, will print tickets.
3,6,9,24,29,34,39	This is a left CS state where the passenger, who acquires the ticket machine, will release the lock, thereby losing mutual exclusion (ME).
10,13,17,20	In this stage, technicians are waiting to enter the CS. The technician who acquires the lock will go to the CS state.
11,14,18,21	This is the CS state where the technician, who acquires the ticket machine, will refill either paper or toner.
12,15,19	This is a left CS state where the technician, who acquires the ticket machine, will release the lock. In this state, they will lose mutual exclusion (ME).
E	exit

6. Trace Tree for FSP Process

The trace tree for the process. Use the conventions given in the lecture notes and add explanatory notes if necessary.