6SENG006W Concurrent Programming

FSP Process Composition Analysis & Design Form

Name	Shanmugaratnam Mohanaranjan
Student ID	W1870584 / 18705841
Date	2024.01.12

1. FSP Composition Process Attributes

Attribute	Value
Name	PRINTING_SYSTEM
Description	This is a process of a printer system where two passengers print the tickets while a technician refills the printer when the printer is out of papers and toner.
Sub-processes (List them.)	PRINTER, PASSENGER, TECHNICIAN
Number of States	637440
Deadlocks (yes/no)	No
Deadlock Trace(s) (If applicable)	None

2. FSP "main" Program Code

The code for the parallel composition of all of the sub-processes and the definitions of any constants, ranges & process labelling sets used. (Do not include the code for the individual sub-processes.)

FSP Program:

```
// FSP Process Model to Ticket Machine, Passengers and Technicians.
// Shanmugaratnam Mohanaranjan || w1870584 || 18705841.

// Declaring Constants
//Maximum capacity of sheets and toner in a printer.
// Maximum number of paper on a printer.
const MAXIMUM_NUMBER_OF_SHEETS = 3
// Maximum number of toner on a printer.
const MAXIMUM_NUMBER_OF_TONER = 3

//  Number of sheets on a printer.
const NUMBER_OF_SHEETS = 0
//  Number of toner on a printer.
const NUMBER_OF_TONER = 0

// Maximum number of sheets on a printer.
const MINIMUM_OF_AMOUNT = 1
// Maximum number of sheets on a printer.
const MAXIMUM_OF_AMOUNT = 3

// Declaring Ranges
// Range representing the number of sheets in the printer.
range SHEETS_OF_RANGE = NUMBER_OF_SHEETS ..
MAXIMUM_NUMBER_OF_SHEETS
// Range representing the number of toner cartridges in the printer.
range RANGE_OF_TONER = NUMBER_OF_TONER ..
MAXIMUM_NUMBER_OF_TONER
// Range representing the number of prints.
range PRINT_AMOUNT_RANGE = MINIMUM_OF_AMOUNT .. MAXIMUM_OF_AMOUNT

// Relevant Sets
// Set of actions including acquiring prints, printing, acquiring, releasing, and passing
counts for sheets and toner.
set PRINT_ACTIONS = {acquirePrint[SHEETS_OF_RANGE][RANGE_OF_TONER], print,
acquireRefill, refill, release, passPaperCount[SHEETS_OF_RANGE],
passTonerCount[RANGE_OF_TONER]}
// Set of users including passengers and technicians.
set USERS = {passenger1, passenger2, passenger3, passenger4, toner_technician,
paper_technician}
```

```
// Ticket Machine Process
// Represents the printer process. It can acquire prints, print, acquire/refill, release, and pass counts for sheets and toner.
// It has conditions to check if there's enough paper or toner, and actions accordingly.
PRINTER(COUNT_OF_PAPER = MAXIMUM_NUMBER_OF_SHEETS, TONER_COUNT = MAXIMUM_NUMBER_OF_TONER) =
   PRINTER[COUNT_OF_PAPER][TONER_COUNT],
   PRINTER[paper: NUMBER_OF_SHEETS..COUNT_OF_PAPER][toner: NUMBER_OF_TONER..TONER_COUNT] =
                  // Condition to check the paper and toner
      if(paper == NUMBER_OF_SHEETS || toner == NUMBER_OF_TONER)
      then(acquireRefill -> refill -> release -> PRINTER[MAXIMUM_NUMBER_OF_SHEETS][toner]
      | acquireRefill -> refill -> release -> PRINTER[paper][MAXIMUM_NUMBER_OF_TONER])
      else(acquirePrint[paper][toner] -> print -> release -> passPaperCount[paper - 1] -> passTonerCount[toner - 1] -> PRINTER[paper - 1][toner - 1]).

// Paper Technician Process
// Represents the paper technician process.
// It can pass paper counts and either initiate a refill process for paper or wait.
// It includes common print actions.
PAPER_TECHNICIAN =
   (passPaperCount[paper: SHEETS_OF_RANGE] -> if(paper == NUMBER_OF_SHEETS)
         // Refill Process for Paper
   then(acquireRefill -> refill -> release -> PAPER_TECHNICIAN)
   else(wait -> PAPER_TECHNICIAN)) + PRINT_ACTIONS.

// Toner Technician Process
// Represents the toner technician process.
// It can pass toner counts and either initiate a refill process for toner or wait.
// It includes common print actions.
TONER_TECHNICIAN =
   (passTonerCount[toner: RANGE_OF_TONER] -> if(toner == NUMBER_OF_TONER)
         // Refill Process for Toner
   then(acquireRefill -> refill -> release -> TONER_TECHNICIAN)
   else(wait -> TONER_TECHNICIAN)) + PRINT_ACTIONS.

// Passenger Process
// Represents the passenger process.
// It can pass print demands and either initiate a print process or wait.
// It includes common print actions.
PASSENGER(TICKETS = 1) =
   PASSENGER[TICKETS],
   PASSENGER[demand: PRINT_AMOUNT_RANGE] =
      (acquirePrint[paper: SHEETS_OF_RANGE][toner: RANGE_OF_TONER] -> if(toner >= demand && paper >= demand)
      then(print -> release -> PASSENGER)
```

```
      else(wait -> PASSENGER)) + PRINT_ACTIONS.

// Printing System Process
// Represents the overall printing system with mutual exclusive control of the ticket
machine.
// It involves multiple passengers, paper/toner technicians, and printer processes.
// Mutual exclusive control of the Ticket Machine
|| PRINTING_SYSTEM = (
            passenger1 : PASSENGER(2)
      ||    passenger2 : PASSENGER(1)
      ||    passenger3 : PASSENGER(3)
      ||    passenger4 : PASSENGER(1)
      ||    paper_technician : PAPER_TECHNICIAN
      ||    toner_technician : TONER_TECHNICIAN
      ||    USERS :: PRINTER).
```

3. Combined Sub-processes

(Add rows as necessary.)

Process	Description
PASSNENGER1	- Put the number of tickets to print at the beginning. To print tickets, take mutually exclusive control of the ticket machine. After all tickets have been printed, end.
PASSENGER2	- Put the number of tickets to print at the beginning. To print tickets, take mutually exclusive control of the ticket machine. After all tickets have been printed, end.
PASSENGER3	- Put the number of tickets to print at the beginning. To print tickets, take mutually exclusive control of the ticket machine. After all tickets have been printed, end.
PASSENGER4	- Put the number of tickets to print at the beginning. To print tickets, take mutually exclusive control of the ticket machine. After all tickets have been printed, end.
PAPER_TECHNICIAN	- Make sure the ticket machine has enough paper by checking it again. When needed, restock the ticket machine with paper.
TONER_TECHNICIAN	- Make sure the ticket machine has enough toner by checking it again. When needed, restock the ticket machine with toner.

4. Analysis of Combined Process Actions

- **Alphabets** of the combined processes, including the final process labelling.
- **Synchronous** actions are performed by at least two sub-process in the combination.
- **Blocked Synchronous** actions cannot be performed, because at least one of the sub-processes can never preform them, because they were added to their alphabet using alphabet extension.
- **Asynchronous** actions are preformed independently by a single sub-process.

Group actions together if appropriate, e.g. if they include indexes in[0], in[1], …, in[5] as in[1..5]. Add rows as necessary.

Processes	**Alphabet** (Use LTSA's **compressed notation**, if alphabet is large.)
passenger1	passenger1.{acquirePrint[0..3][0..3], acquireRefill, {passPaperCount, passTonerCount}[0..3], {print, refill, release, wait}}
passenger2	passenger2.{acquirePrint[0..3][0..3], acquireRefill, {passPaperCount, passTonerCount}[0..3], {print, refill, release, wait}}
passenger3	passenger3.{acquirePrint[0..3][0..3], acquireRefill, {passPaperCount, passTonerCount}[0..3], {print, refill, release, wait}}
passenger4	passenger4.{acquirePrint[0..3][0..3], acquireRefill, {passPaperCount, passTonerCount}[0..3], {print, refill, release, wait}}
paper_technician	paper_technician.{acquirePrint[0..3][0..3], acquireRefill, {passPaperCount, passTonerCount}[0..3], {print, refill, release, wait}}
toner_technician	toner_technician.{acquirePrint[0..3][0..3], acquireRefill, {passPaperCount, passTonerCount}[0..3], {print, refill, release, wait}}

Synchronous Actions	Synchronised by Sub-Processes (List)
passenger1.acquire passenger1.print passenger1.release	PASSNENGER (2), PRINTER
passenger2.acquire passenger2.print passenger2.release	PASSNENGER (1), PRINTER
passenger3.acquire passenger3.print passenger3.release	PASSNENGER (3), PRINTER
passenger4.acquire passenger4.print passenger4.release	PASSNENGER (1), PRINTER
paper_technician.acquireRefill paper_technician.refill paper_technician.release	PAPER_TECHNICIAN, PRINTER
toner_technician.acquireRefill toner_technician.refill toner_technician.release	TONER_TECHNICIAN, PRINTER

Blocked Synchronous Actions	Blocking Processes	Blocked Processes
paper_technician.acquireRefill paper_technician.refill paper_technician.release	PAPER_TECHNICIAN, PRINTER	PAPER_TECHNICIAN
toner_technician.acquireRefill toner_technician.refill toner_technician.release	TONER_TECHNICIAN, PRINTER	TONER_TECHNICIAN

Sub-Processes	Asynchronous Actions (List)
Passenger1	None
Passenger2	None
Passenger3	None
Passenger4	None
Paper_technician	paper_technician.acquireRefill paper_technician.refill
Toner_technician	toner_technician.acquireRefill toner_technician.refill

5. Parallel Composition Structure Diagram

The structure diagram for the parallel composition.