6SENG002W Concurrent Programming

FSP Process Composition Analysis & Design Form

Name	Dulshan Ratnayake
Student ID	2017256/w1697707
Date	30/12/2020

1. FSP Composition Process Attributes

Attribute	Value
Name	PRINTING PROCESS
Description	The model is used to simulate the process in which a printer is shared among two students and a technician for their own usage thus showing how mutually exclusive principles have been applied as well.
Alphabet (Use LTSA's compressed notation, if alphabet is large.)	{std1.print.aquire, std2.print.aquire, std1.print[3], std1.print[2], std1.print[1], std2.print[3], std2.print[2], std2.print[1], tech.print.aquire, tech.print[3], tech.print[2], tech.print[1], std1.release, std2.release, tech.release , std1.refill.aquire, std2.refill.aquire, tech.refill.aquire, std1.refill, std2.refill, tech.refill, std1.terminate, std2.terminate, std1.not.empty.acquire, std1.not.empty, std2.not.empty.acquire, std2.not.empty, tech.not.empty.acquire, tech.not.empty, }
Sub-processes (List them.)	PRINTER, STUDENT, TECHNICIAN
Number of States	114
Deadlocks (yes/no)	No
Deadlock Trace(s) (If applicable)	N/A

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 other sub-processes.)

FSP Program:

```
//constants
const MAX_PAPER_LEVEL = 3
const MIN_PRINT_PAPER_LEVEL=1
const MIN_PAPER_LEVL=0
const MIN_PRINT_DOCUMENT = 1
const MIN_DOCUMENT=0
//ranges
range PRINT_Paper_Level =  MIN_PRINT_PAPER_LEVEL .. MAX_PAPER_LEVEL
range PAPER_Level = MIN_PAPER_LEVL .. MAX_PAPER_LEVEL
set PRINT_Actions = { print[PRINT_Paper_Level],
print.acquire,refill.acquire,release,refill,not.empty.acquire,not.empty }
set PRINTER_Users = { tech, std1,std2}

//printer FINITE STATE MACHINE
PRINTER = PRINTER_FREE[MAX_PAPER_LEVEL] ,
PRINTER_FREE[papers_available : PAPER_Level] =
(when(papers_available>MIN_PAPER_LEVL)print.acquire -> PRINT[papers_available]
|when(papers_available>MIN_PAPER_LEVL) not.empty.acquire->
EMPTY_PRINTER[papers_available]
|when(papers_available==MIN_PAPER_LEVL) refill.acquire->
REFILL_PRINTER[MAX_PAPER_LEVEL]) ,

EMPTY_PRINTER[papers_available : PRINT_Paper_Level]=(not.empty->
RELEASE_PRINTER[papers_available]),
REFILL_PRINTER[refill_papers : PAPER_Level]=(refill-> RELEASE_PRINTER[refill_papers]),
PRINT[papers_available : PRINT_Paper_Level] =(print[papers_available] ->
RELEASE_PRINTER[papers_available-1]),

RELEASE_PRINTER[papers : PAPER_Level]=(release->PRINTER_FREE[papers]).


//student FINITE STATE MACHINE
STUDENT( DOCUMENTS = 3 ) = STUDENT_PRINT[DOCUMENTS],
STUDENT_PRINT[document_number : MIN_DOCUMENT..DOCUMENTS] =
(when(document_number>MIN_DOCUMENT) print.acquire -
>STUDENT_PRINT_DOCUMENT[document_number]
| when(document_number==MIN_DOCUMENT) terminate ->END),

STUDENT_PRINT_DOCUMENT[document_number:MIN_PRINT_DOCUMENT ..
DOCUMENTS] = (print[PRINT_Paper_Level] -
```

```
>STUDENT_RELEASE_PRINTER[document_number-1]),
STUDENT_RELEASE_PRINTER [document :MIN_DOCUMENT..DOCUMENTS] = (release-
>STUDENT_PRINT[document])
+PRINT_Actions.



//technician FINITE STATE MACHINE
TECHNICIAN = ( refill.acquire-> TECHNICIAN_REFILL| not.empty.acquire-
>TECHNICIAN_NOT_EMPTY),

TECHNICIAN_REFILL = (refill -> TECHNICIAN_PRINTER_RELEASE),
TECHNICIAN_NOT_EMPTY = (not.empty -> TECHNICIAN_PRINTER_RELEASE),

TECHNICIAN_PRINTER_RELEASE = (release -> TECHNICIAN)
 +PRINT_Actions.

//COMPOSITE PROCESS
|| PRINTER_PROCESS = ( PRINTER_Users :: PRINTER|| std1 : STUDENT ( 3 )|| std2 :
STUDENT ( 2 ) ||tech : TECHNICIAN).
```

3. Combined Sub-processes

(Add rows as necessary.)

Process	Description
TECHNICIAN	Describes the technician sub-process in which the technician attempts to refill the papers of the printer when it has run out of paper
STUDENT(3)	Describes a student attempting to print three documents
STUDENT(2)	Describes a student attempting to print two documents
PRINTER	Describes the shared resource printer that being used by the students and the technician

4. Analysis of Combined Process Actions

- **Synchronous** actions are performed by at least two sub-process in the combination.
- **Blocked Synchronous** actions cannot be performed, since at least one of the sub-processes cannot 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, for example if they include indexes,
e.g. in[0], in[1], …, in[5] as in[1..5].

(Add rows as necessary.)

Synchronous Actions	Synchronised by Sub-Processes (List)
std1.print.acquire, std1.print[1..3], std1.release	PRINTER, STUDENT(3)
tech.refill.acquire, tech.refill, tech.release, tech.not.empty.acquire, tech.not.empty	PRINTER, TECHNICIAN
std2.print.acquire, std2.print[1..3], std2.release	PRINTER, STUDENT(2)

Blocked Synchronous Actions	Synchronised by Sub-Processes (List)	Blocked due to Sub-Process (by alphabet extension to sub-process)
tech.print.acquire	PRINTER, TECHNICIAN	TECHNICIAN
tech.print[1..3]	PRINTER, TECHNICIAN	TECHNICIAN
std1.refill.acquire, std1.refill, std1.not.empty.acquire, std1.not.empty	PRINTER, STUDENT(3)	STUDENT(3)
std2.refill.acquire, std2.refill, std2.not.empty.acquire, std2.not.empty	PRINTER, STUDENT(2)	STUDENT(2)

Sub-Process	Asynchronous Actions (List)
PRINTER	No
STUDENT(3)	std1.terminate
STUDENT(2)	std2.terminate
TECHNICIAN	No

5. Parallel Composition Structure Diagram

The structure diagram for the parallel composition.