6SENG002W Concurrent Programming

FSP Process Composition Analysis & Design Form

Name	Bombuwalage Gayashan Kalhara Bombuwala
Student ID	W1582950
Date	06/12/2018

1. FSP Composition Process Attributes

Attribute	Value
Name	SHARED_PRINTER
Description	Models a scenario between a printer, technician and two students. One of the students (s1) is trying to print two documents, while the other student (s2) is trying to print three documents. Technician (tcn) refills the printer when the printer runs out of paper.
Alphabet (Use LTSA's compressed notation, if alphabet is large.)	{ s1.acquire, s1.empty, s1.print[1], s1.print[2], s1.print[3], s1.refill_printer, s1.release, s2.acquire, s2.empty, s2.print[1], s2.print[2], s2.print[3], s2.refill_printer, s2.release, tcn.empty, tcn.print[1], tcn.print[2], tcn.print[3], tcn.refill_printer, tcn.release, terminate }
Sub-processes (List them.)	PRINTER, STUDENT, TECHNICIAN
Number of States	56
Deadlocks (yes/no)	No deadlocks/errors
Deadlock Trace(s)	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 sub-processes.)

FSP Program:

```
const MIN_SHEET_COUNT        =      1
const MAX_SHEET_COUNT        =      3
range DOC_COUNT              =      MIN_SHEET_COUNT .. MAX_SHEET_COUNT
range SHEET_STACK            =      0 .. MAX_SHEET_COUNT

set All_Users = {s1, s2, tcn}
set PRINT_Actions = {acquire, print[DOC_COUNT], release, empty}

PRINTER(SHEETS_AVAILABLE  = MAX_SHEET_COUNT) =
PRINTER_AVAILABLE[MAX_SHEET_COUNT],
PRINTER_AVAILABLE[sheets_available: SHEET_STACK] =
(
when(sheets_available > 0)acquire -> print[DOC_COUNT] -> release ->
PRINTER_AVAILABLE[sheets_available - 1] |
when(sheets_available == 0)empty -> release ->
PRINTER_AVAILABLE[MAX_SHEET_COUNT]
).

STUDENT(DOCS_TO_PRINT = 1) =  PRINT[DOCS_TO_PRINT],
PRINT[doc_count: 0 .. DOCS_TO_PRINT] = (
when (doc_count > 0)  acquire -> print[DOCS_TO_PRINT + 1 - doc_count] -> release ->
PRINT[doc_count - 1] |
when (doc_count == 0) terminate -> END
) + PRINT_Actions.

TECHNICIAN = (empty -> refill_printer -> release -> TECHNICIAN | terminate -> END) +
PRINT_Actions.

|| SHARED_PRINTER = (s1: STUDENT(2) || s2: STUDENT(3) || tcn : TECHNICIAN ||
All_Users :: PRINTER)
/ {terminate/s1.terminate,terminate/s2.terminate,terminate/tcn.terminate}.
```

3. Combined Sub-processes

(Add rows as necessary.)

Process	Description
PRINTER	Represents a simple printer which can hold three sheets of a time
STUDENT(2)	Represents a student who is trying get two documents printed
STUDENT(3)	Represents a student who is trying get three documents printed
TECHNICIAN	Represents a technician who refills the printer when the printer runs out of paper (i.e. refills three papers at a time)

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 perform them, because they were added to their alphabet using alphabet extension.
- **Asynchronous** actions are preformed independently by a single sub-process.

(Add rows as necessary.)

Synchronous Actions	Synchronised by Sub-Processes (List)
s1.acquire, s1.print[1], s1.print[2], s1.release	STUDENT(2), PRINTER
s2.acquire, s2.print[1], s2.print[2], s2.print[3], s2.release	STUDENT(3), PRINTER
tcn.empty, tcn.refill_printer, tcn.release	TECHNICIAN, PRINTER
terminate	STUDENT(2), STUDENT(3), TECHNICIAN

Blocked Synchronising Actions	Synchronising Sub-Processes	Blocking Sub-Processes
tcn.print[1], tcn.print[2], tcn.print[3]	TECHNICIAN, PRINTER	TECHNICIAN
s1.empty	STUDENT(2), PRINTER	STUDENT(2)
s2.empty	STUDENT(3), PRINTER	STUDENT(3)

Sub-Process	Asynchronous Actions (List)
TECHNICIAN	tcn.refill_printer
PRINTER	None
STUDENT(2)	None
STUDENT(3)	None

5. Parallel Composition Structure Diagram

The structure diagram for the parallel composition.