6SENG002W Concurrent Programming

FSP Process Composition Analysis & Design Form

Name	Ravishan Lakshitha Premerathne
Student ID	W1790297
Date	1/12/2023

1. FSP Composition Process Attributes

Attribute	Value
Name	PRINTING_SYSTEM
Description	This model represents a Printing System and is made up of 4 smaller processes - 2 for Students, 1 for a Technician, and 1 shared Printer process that is accessible by both Students and Technician. It allows for exclusive use of the shared Printer by those who wish to use it.
Alphabet (Use LTSA's compressed notation, if alphabet is large.)	alphabet (PRINTING_SYSTEM) = { student1.{{acquireToPrint, acquireToRefill, cannotFill, fill}, printDocument[1..3], release}, student2.{{acquireToPrint, acquireToRefill, cannotFill, fill}, printDocument[1..2], release}, technician.{acquireToPrint, acquireToRefill, cannotFill, fill, release}, terminate }
Sub-processes (List them.)	PRINTER, student_a : STUDENT(3), student_b : STUDENT(2) TECHNICIAN
Number of States	80
Deadlocks (yes/no)	No
Deadlock Trace(s) (If applicable)	Not applicable

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:

```
range PAPER_TRAY = 0..3
const EMPTY_PAPER_TRAY = 0
const FULL_PAPER_TRAY = 3

set Students = { student1, student2 }
set Users = { Students, technician }


||PRINTING_SYSTEM = ( student1: STUDENT(3) || student2: STUDENT(2) || technician: TECHNICIAN || Users :: PRINTER )
                        / { terminate / Users.terminate }.
```

3. Combined Sub-processes

(Add rows as necessary.)

Process	Description
student1.{acquireToPrint, acquireToRefill, cannotFill, fill, release}	student_a: STUDENT (3), PRINTER
student2.{acquireToPrint, acquireToRefill, cannotFill, fill, release}	student_b: STUDENT (2), PRINTER
technician.{acquireToPrint, acquireToRefill, cannotFill, fill, release}	TECHNICIAN, PRINTER
terminate	Student_a: STUDENT(3), student_b: STUDENT(2), 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)
student1.{acquireToPrint, acquireToRefill, cannotFill, fill, release}	Student_a: STUDENT(3), PRINTER
student2.{acquireToPrint, acquireToRefill, cannotFill, fill, release}	Student_b: STUDENT(2), PRINTER
technician.{acquireToPrint, acquireToRefill, cannotFill, fill, release}	TECHNICIAN, PRINTER
terminate	Student_a: STUDENT(3), student_b: STUDENT(2), TECHNICIAN

Sub-Process	Asynchronous Actions (List)
Student_a : STUDENT(3)	Student_a.printDocument[1..3]
Student_b : STUDENT(2) s	Student_b.printDocument[1..3]
PRINTER	None
TECHNICIAN	None