6SENG006C Concurrent Programming

FSP Process Composition Analysis & Design Form

Name	W. Thaveesha Dheerasekera
Student ID	W1761369/ 2019483
Date	09/01/2023

1. FSP Composition Process Attributes

Attribute	**Value**
Name	PINTJOB
Description	Simulates the process of two students printing documents: student ‘a’ with 3 pages and student ‘b’ with 2 pages and technician refills the papers as required
Alphabet (Use LTSA's compressed notation, if alphabet is large.)	{a.{print.doc[1..3], student.{acquire, release}, technician.{acquire, refill, release}}, b.{print.doc[1..2], student.{acquire, release}, technician.{acquire, refill, release}}, t.{student.{acquire, print.paper, release}, technician.{acquire, refill, release}}, waiting}
Sub-processes (List them.)	PRINTER, STUDENT, TECHNICIAN
Number of States	67
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:

```
// CONSTANTS
const MAX_PAPER = 3
range PAPER_RANGE = 0 .. MAX_PAPER

set PRINTER_ACTIONS = { student.acquire , student.print.paper , student.release,
technician.acquire , technician.refill , technician.release }
set Students = { a, b }

// PRINT COMPOSITE FSM
|| PRINTJOB = ( a:STUDENT( 3 ) || b:STUDENT( 2 ) || t:TECHNICIAN || { Students, t
}::PRINTER( 3 ) )  /
        {
                waiting / { a.waiting, b.waiting, t.waiting},
                a.print.doc[ 1..3 ] / { a.student.print.paper } , b.print.doc[ 1..2 ] / {
b.student.print.paper }
        } .
```

3. Combined Sub-processes

(Add rows as necessary.)

Process	Description
PRINTER	Models the process of a printer of a printing system
STUDENT	Models a student who wants to print documents
TECHNICIAN	Models a technician whose job is refilling papers when required

4. Analysis of Combined Process Actions

- **Synchronous** actions are performed by at least two sub-processes 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.

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.)

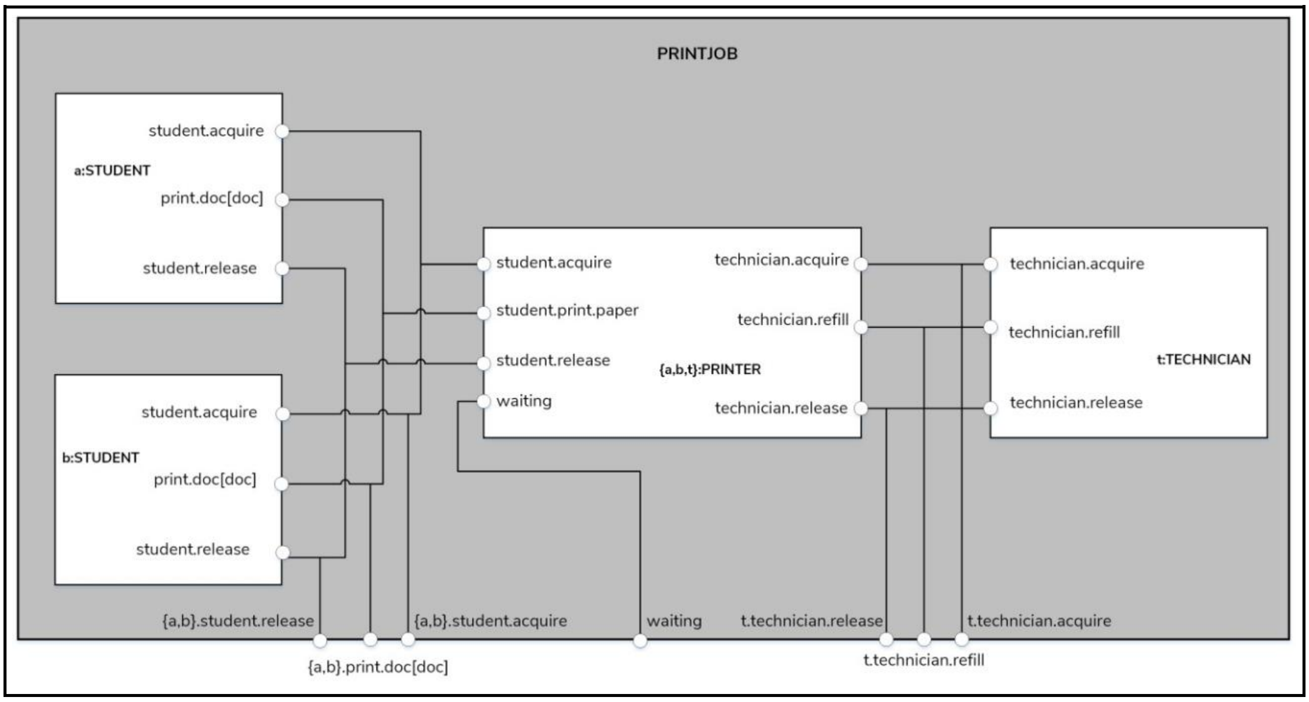
Synchronous Actions	Synchronised by Sub-Processes (List)
a.student.acquire, b.student.acquire, a.print.doc[1..3], b.print.doc[1..2], a.student.release, b.student.release	STUDENT, PRINTER
t.technician.acquire, t.technician.release, t.technician.refill	TECHNICIAN, PRINTER

Blocked Synchronous Actions	Synchronising Sub-Processes (List)	Blocking Sub-Processes
a.student.acquire, b.student.acquire, a.print.doc[1..3], b.print.doc[1..2], a.student.release, b.student.release	STUDENT, PRINTER	TECHNICIAN
t.technician.acquire, t.technician.release, t.technician.refill	TECHNICIAN, PRINTER	STUDENT

Sub-Process	Asynchronous Actions (List)
waiting	PRINTER

5. Parallel Composition Structure Diagram

The structure diagram for the parallel composition.


PRINTJOB
a:STUDENT
student.acquire
print.doc[doc]
student.release
b:STUDENT
student.acquire
print.doc[doc]
student.release
{a,b,t}:PRINTER
student.acquire
student.print.paper
student.release
waiting
technician.acquire
technician.refill
technician.release
t:TECHNICIAN
technician.acquire
technician.refill
technician.release
{a,b}.student.release
{a,b}.student.acquire
{a,b}.print.doc[doc]
waiting
t.technician.release
t.technician.acquire
t.technician.refill