CSCM85

Modelling and Verification – CSCM85

MSc Data Science

4th November 2022

Team Members:

Shanika Hansani Kithulgodage	-	2149737
Suganja Ratheeswaran	-	2152539
Pallav Shukla	-	2154638
Salil Veeravu Ahmed	-	2150847

Question - 1

Question 1

Consider two vending machines, VM_1 and VM_2 which accept a coin as payment and deliver tea. However, VM_1 requires payment upfront whereas VM_2 has the option of getting tea first and paying later. Here are the LTSs for these two machines

$$\{(\mathrm{VM}_1, \mathrm{coin}, C_1), (C_1, \mathrm{tea}, C_0), (C_1, \mathrm{coin}, C_2), (C_0, \mathrm{coin}, C_1), (C_2, \mathrm{tea}, C_1)\}$$

$$\{(\mathrm{VM}_2, \mathrm{coin}, P), (\mathrm{VM}_2, \mathrm{tea}, M), (P, \mathrm{tea}, \mathrm{VM}_2), (M, \mathrm{coin}, \mathrm{VM}_2)\}$$

(a) Draw both LTSs (you may use FDR but drawings by hand are fine as well).

(b) Give CSP definitions of both LTSs.

(c) Show that VM_1 and VM_2 are not trace equivalent.

(d) Find a state s in the LTS for VM_2 that is bisimilar to VM_1. Show bisimilarity by giving a bisimulation that contains the pair (VM_1, s).

[30 marks]

Part (a) :

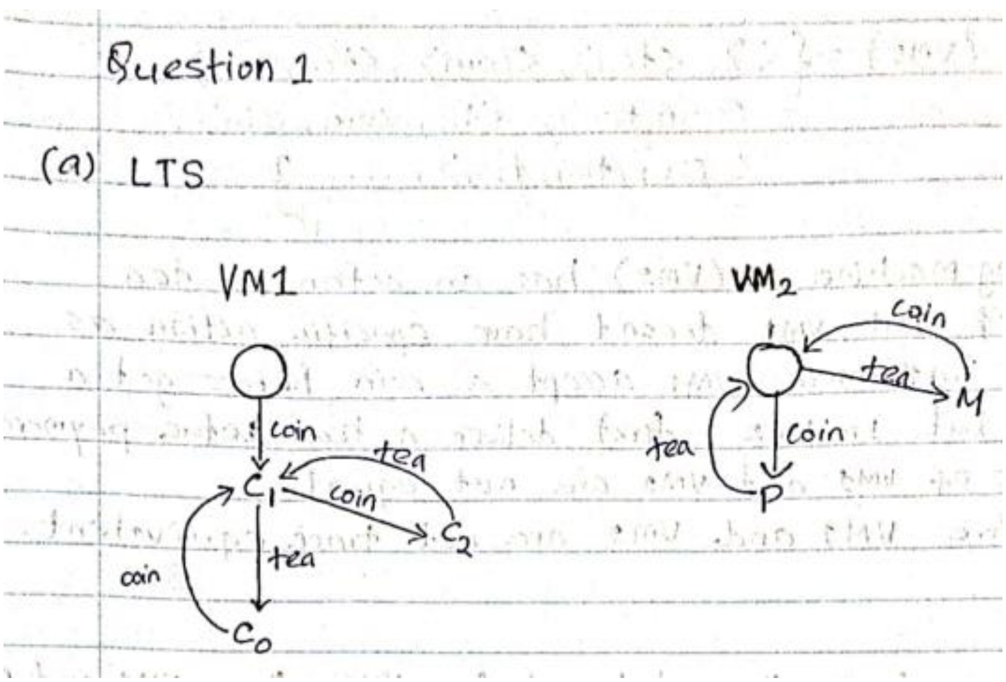


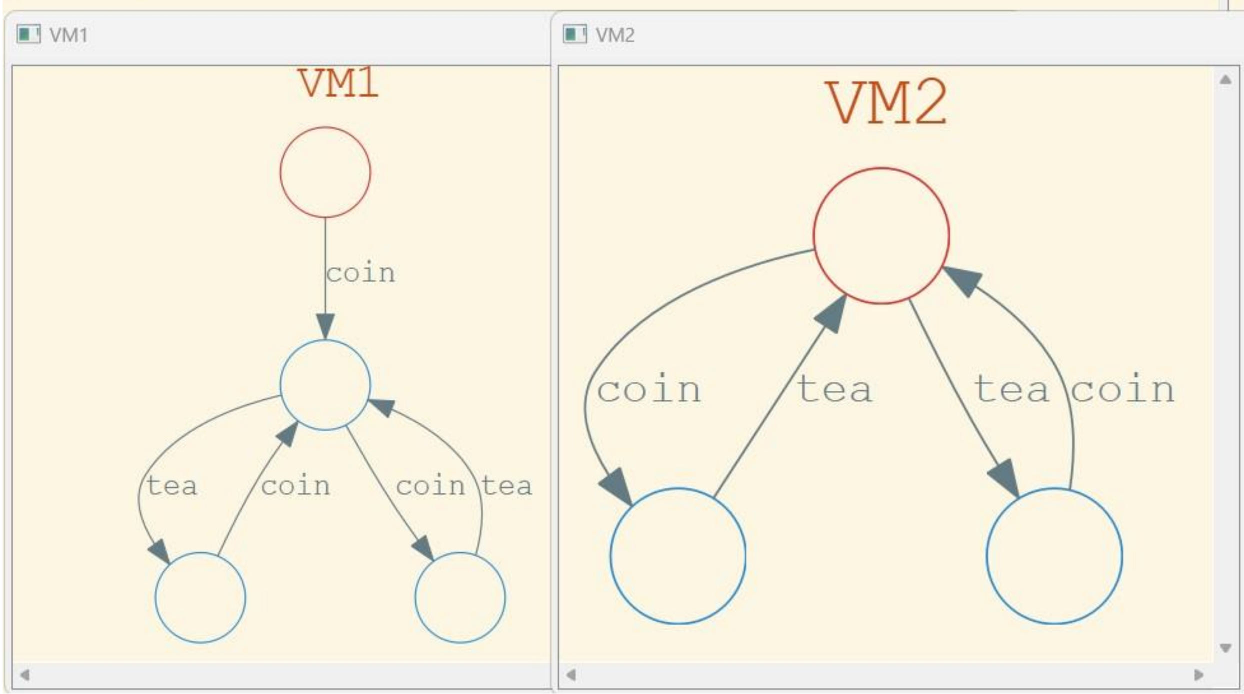
Image for (a) part is given below, Both code as well as graph

Image 1

F:/Study Swansea/AA A Term-2/AA Lecture/Modelling/Assignment/first.csp

File Help

```
Welcome to FDR 4.2.7 (6ecbe5a21b71ab020e8fcaeccfe5ebaad0599f4f)
FDR Version 4.2.7 copyright 2016 Oxford University Innovation Ltd. All Rights Reserved.
License: Academic license for non-commercial use only
Type :help for help
first.csp> :graph VM1
first.csp> :graph VM2
first.csp>
```



Part (b) :

Image : Showing the code done thorough hand then done in FDR

(b) CSP

channel coin, tea

VM1 = coin → C_1

C_1 = tea → C_0 [] coin → C_2

C_0 = coin → C_1

C_2 = tea → C_1

VM2 = coin → P [] tea → M

P = tea → VM2

M = coin → VM2

first.csp - GNU Emacs at LAPTOP-6QGMOR1P

File Edit Options Buffers Tools Help

```
-- First is declaration
channel coin, tea
-- Defining VM1 then C1 decribed C0 and C2 as sub states
VM1 = coin -> C1
C1 = tea -> C0 [] coin -> C2
C0 = coin -> C1
C2 = tea -> C1
-- Defining VM2 then P and M  as sub states
VM2 = coin -> P [] tea -> M
P = tea -> VM2
M = coin -> VM2
```

Part (c) :

Traces for VM-1

(c) Traces (VM1) = { <>, <coin>, <coin, tea>, <coin, coin>,
<coin, tea, coin>, <coin, coin, tea>,
<coin, coin, tea, tea>, <coin, tea, coin, coin>,
<coin, tea, coin, coin, tea> ..... }

Traces for VM-2

Traces (VM2) = { <>, <tea>, <coin>, <tea, coin>,
<coin, tea>, <tea, coin, coin>,
<coin, tea, tea>, ... }

Description:

Vending Machine 2 (VM2) has an action as tea ~~[illegible]~~ but VM1 doesn't have specific action as tea. Furthermore VM1 accept a coin before get a tea but in VM2, first deliver a tea before payment.
Traces of VM1 and VM2 are not equal.
Therefore VM1 and VM2 are not trace equivalent.

Part (d):

Image

(d) If we choose the state M from VM2 then VM1 and M construct bisimulation
construct R such that,
(VM1) R (M) ≡ (VM1, M) ∈ R

R = { (VM1, M), (C_1, VM2), (C_0, M), (C_2, P) }

Use game characterisation of bisimularity.

	Attacker	Defender	complete Move?
Start with,	$VM1 \xrightarrow{coin} C_1$	$M \xrightarrow{coin} VM2$	Yes
	$C_1 \xrightarrow{tea} C_0$	$VM_2 \xrightarrow{tea} M$	Yes
	$C_0 \xrightarrow{coin} C_1$	$M \xrightarrow{coin} VM2$	Yes
	$C_1 \xrightarrow{coin} C_2$	$VM_2 \xrightarrow{coin} P$	Yes
	$C_2 \xrightarrow{tea} C_1$	$P \xrightarrow{tea} VM_2$	Yes

The defender wins because the attacker is unable to carry out.
R is bisimulation.

Description:
All the moves of Attacker are being answered by the defender. So, each and every time the defender is wining Clearly Defender wins.

Question – 2

Question 2 Decide for each of the following statements whether it is true for all processes A, B:

(a) If A and B are trace equivalent and A is deadlock free, then B is deadlock free.

(b) If A and B are bisimilar and A is deadlock free, then B is deadlock free.

In each case, either prove the statement, or give a counterexample.

You may use the following definition of a deadlock:

> A process has S has a deadlock if there are a word w and a process S' such that $S \xrightarrow{w}{}^* S'$ and there is no transition from S'.

Hence, for example, part (b) can be equivalently reformulated as

(b) If A and B are bisimilar and B has a deadlock, then A has a deadlock.

[40 marks]

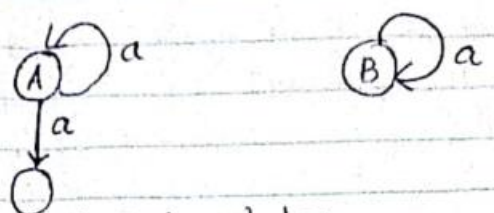
Part (a):

Image

(1) If A and B are trace equivalent and A is deadlock free, then B is deadlock free.

Statement is false

Let assume 2 states



Let $a \in A, B$ be states,

Defining Traces:

$\text{Traces}(A) = \{\langle\rangle, \langle a\rangle, \langle a,a\rangle, \langle a,a,a\rangle, \langle a,a,a,a\rangle \ldots\}$

$\text{Traces}(B) = \{\langle\rangle, \langle a\rangle, \langle a,a\rangle, \langle a,a,a\rangle, \langle a,a,a,a\rangle \ldots\}$

Traces of A and B are equal $(A =_T B)$

Process A is dead lock free because there is an action from A. But B is deadlock because no variable state from B.

$\therefore$ The statement is false.

Part (b):

(b) If A and B are bisimilar and A is deadlock free, then B is deadlock free

The statement is true.

Description in def: as stated in class by prof.

If a specific action from A and B is possible, then A and B are bisimilar. Both have same actions from states. If we consider A is deadlock ~~free~~ that means A has not any action from A. then B should be deadlock otherwise it will not act as bisimulation.

Question - 3

Question 3:

Answer 3:

Code in writing and image:

```
-- Question 3 Robot and battery that drains after 2 steps.
-- Deining the range min and max
min = 0
max = 5
-- Range description Given

Range = {min..max}
datatype Direction = L | R
-- Direction for movement defined left to right

-- Declarations
channel move : Direction
channel position : Range
channel work

-- Code provided for movement comparing it steps of robot through x
Robot(x) = position.x -> (
        (if x > min then move.L -> Robot(x-1) else STOP) []
        (if x < max then move.R -> Robot(x+1) else STOP) []
```

```
            (work -> Robot(x))
)

-- Movement checked with x greater then and less then and performing actions respectively

-- Defining Full and Empty as per movement
Full = move?x -> Full [] work -> Empty
Empty = move?x -> move?y -> Full

Battery = Robot(min) [| {|work, move|} |] Empty
```

CODE IMAGE:

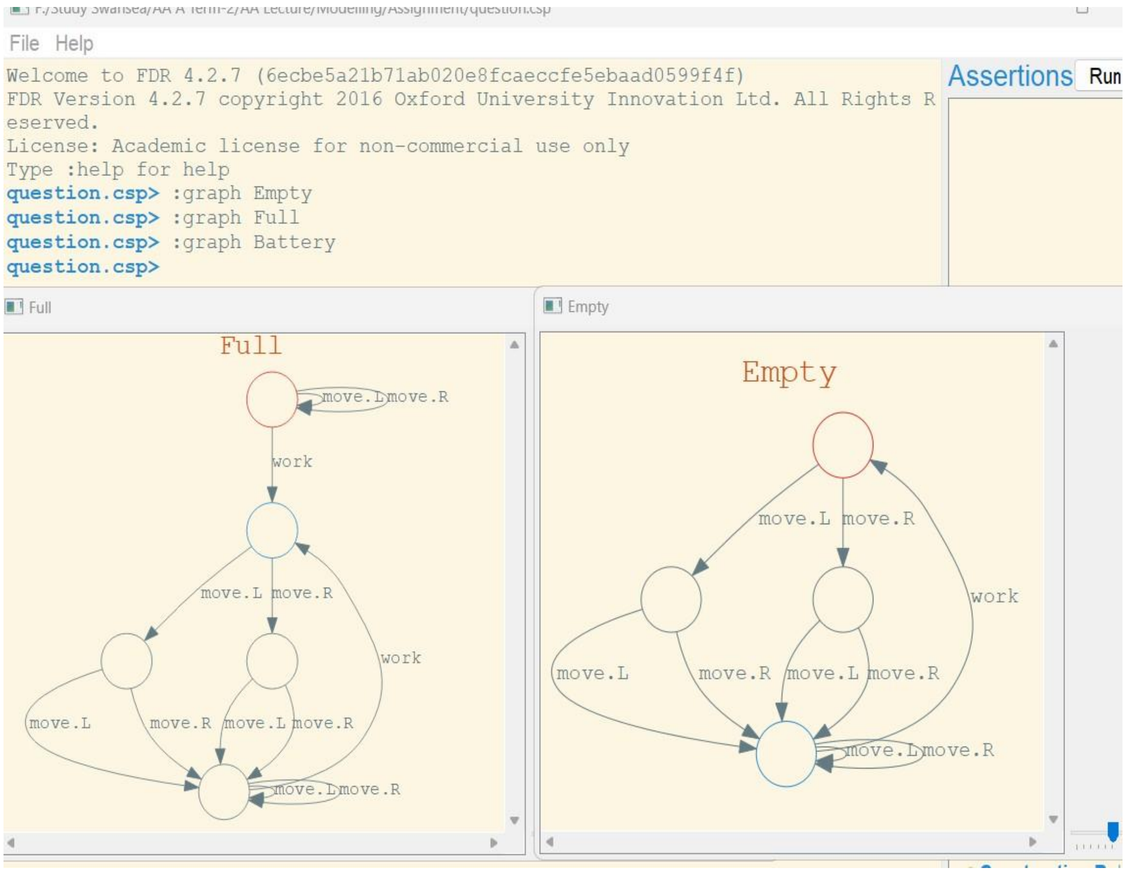
---

question.csp - GNU Emacs at LAPTOP-6QGMOR1P

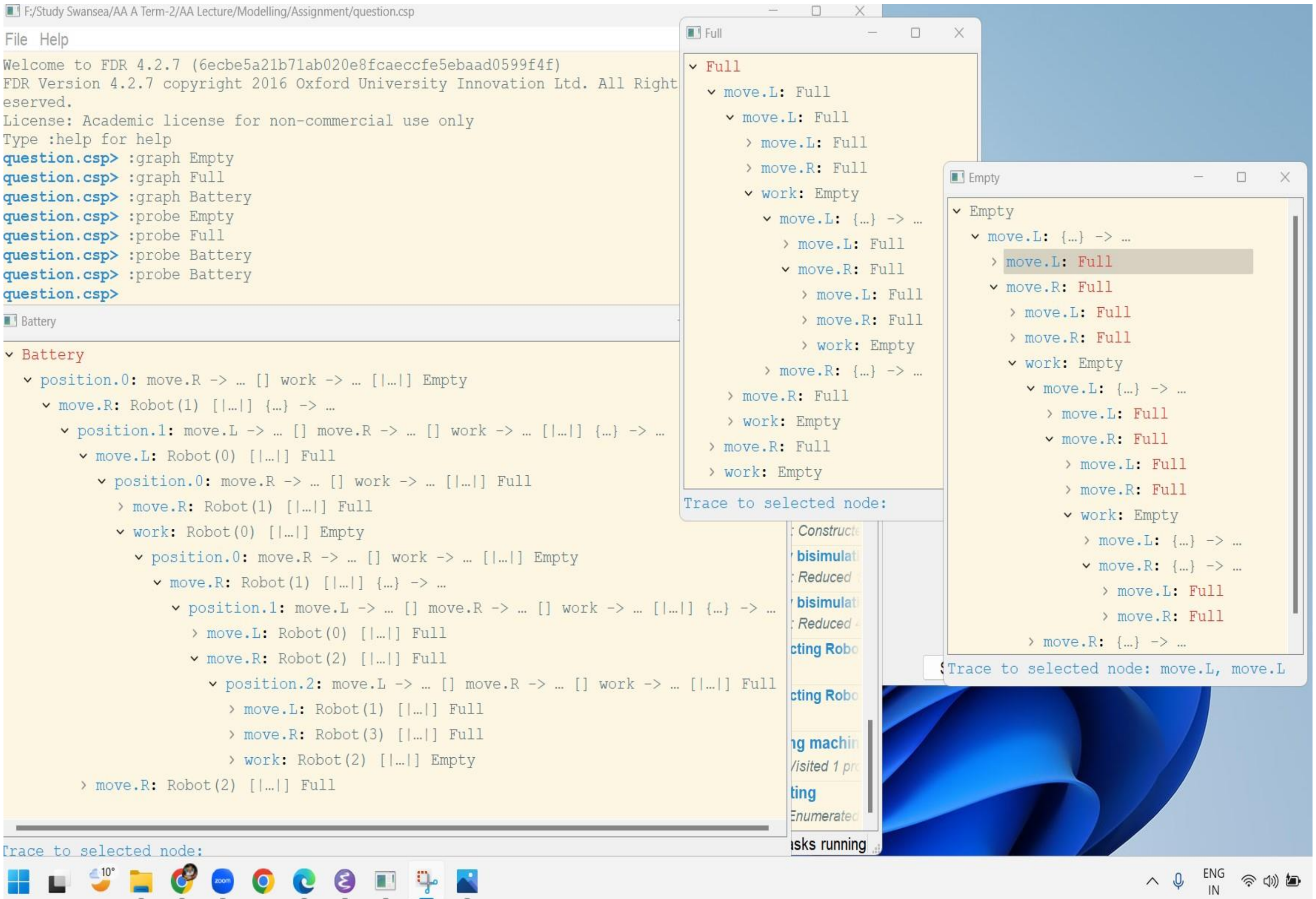
File Edit Options Buffers Tools Help

```
-- Question 3 Robot and battery that drains after 2 steps.
-- Deining the range min and max
min = 0
max = 5
-- Range description Given

Range = {min..max}
datatype Direction = L | R
-- Direction for movement defined left to right

-- Declaraions
channel move : Direction
channel position : Range
channel work

-- Code provoided for movement comapring it steps of probot through x
Robot(x) = position.x -> (
        (if x > min then move.L -> Robot(x-1) else STOP) []
        (if x < max then move.R -> Robot(x+1) else STOP) []
        (work -> Robot(x))
)

-- Movement checked with x greater then and less then and performing actions respectively

-- Defining Full and Empty as per movement
Full = move?x -> Full [] work -> Empty
Empty = move?x -> move?y -> Full

Battery = Robot(min) [| {|work, move|} |] Empty
```

GRAPH IMAGE



PROBE IMAGE



Description:

- Initially the battery is defined empty. So, there can be only two movements that are available for the machine and those are left **or** right.
- Only after making two moves the battery will be charged to full. It can choose left or right or work. When it moves left or right it can go as it is.
- But if we choose work again after making two moves then the battery is drained out to empty again. Then it can only move left or right.
- But, after making two moves it can start work again, from zero. Basically, it can keep going again and again following the rules stated.
-

In Conclusion The Machine (Robot) need to move twice in order to charge the battery to full as well as again after working.
