INM420 – Advanced Programming: Concurrency

2023-24 Coursework – ***To be done individually – do not discuss it, just do it!***

# Questions

1. Consider the labelled transition system (LTS) that is drawn below, which corresponds to the parallel composition of two processes, NS and WE. NS controls all actions related to North and South (i.e., n\*/s\*), while WE controls all actions related to West and East (\*w/\*e).

Provide the FSP description of each of these two processes so that their composition, S, produces the LTS drawn below.

NS = /\* ? \*/ .

WE = /\* ? \*/ .

||S = (NS || WE) .

|  |
| --- |
|  |

Figure 1: Labelled-transition system of S, the parallel composition of P & Q for question 1.

For those who have trouble reading the LTS drawn, the behaviour of S is defined as:

Sbehaviour = Q0,

Q0 = (n -> Q1), Q1 = (ne -> Q2),

Q2 = (se -> Q3), Q3 = (e -> Q4 |s -> Q11),

Q4 = (e -> Q5 |s -> Q10), Q5 = (s -> Q6),

Q6 = (ne -> Q7), Q7 = (nw -> Q8),

Q8 = (nw -> Q9), Q9 = (sw -> Q3),

Q10 = (e -> Q6), Q11 = (e -> Q10) .

[10]

1. The following FSP model is a high-level description of a drinks dispensing machine that has a brown, a red, and a green button.

minimal DRINKS = ( brown -> intern1 -> intern2 -> chocolate ->DRINKS

| red -> intern2 -> coffee ->DRINKS

| green -> intern1 -> tea ->DRINKS)

\ { intern1, intern2 } .

* 1. What is the overall probability of performing action coffee in this machine given the provided FSP model?

[5]

* 1. How much time does it take to produce tea once the green button has been pressed?

[5]

## Clan feuds

1. Two feuding clans, the Hatfields and the McCoys, need to visit the local river to water their cows. But whenever they meet at the river, carnage ensues. So, the elders of the two families need to come up with a river-sharing protocol. Before considering the protocol, model the current (bad) situation:
   1. Identify the actions of interest, and the processes involved in them *(don’t invent actions/processes, as then it’s hard to argue about your model being faithful to the actual system – stick to the brief)*.

[2]

* 1. Provide a safety property that checks that the clans are never both at the river.

[5]

* 1. Construct a structural diagram showing the processes & properties identified and how these are linked by the actions.

[3]

* 1. Provide an FSP model that describes the system. Check whether it satisfies the safety property.

[10]

1. Here is the protocol proposed by the clan elders:

*Each clan will have its own flag near the river. A clan raises its flag to show that it wants to go to the river. Then, it checks the state of the other clan's flag. If the other clan's flag is raised, then this clan lowers its flag and re-tries. If the other clan's flag is not raised, then this clan visits the river, waters its cows, and lowers its flag after they've left the river.*

* 1. Identify the actions of interest, and the processes involved in them *(again, stick to the brief)*.

[3]

* 1. Construct a structural diagram showing the processes identified and how these are linked by the actions.

[3]

* 1. Provide an FSP model that describes the system with this river-sharing protocol – change the model of the families from the one you had in Q3.d.

[20]

* 1. Check whether your FSP model satisfies the safety property from Q3.d.

[1]

1. The community hears of this solution and fears the worse as they know that both clans can be greedy. Are they right to be worried?

Define a progress property (or set of properties if appropriate) and check for progress (a) under the assumption that the clans are not greedy; and (b) under the assumption that the clans are greedy. *(your FSP code from Q4.c should be extended, not modified.)*

[5]

1. The community decides to get the help from another large clan that's currently neutral – the Petersons. So, they ask the most respected member of the Petersons, Judge Gary, to make sure that the river-sharing protocol will work no matter what. Judge Gary suggests this amended protocol:

*Each clan will have its own flag near the river as before. But now they’ll also have access to a shared turn indicator that can show either value 1 (for Hatfields) or 2 (for McCoys). When a clan wants to go to the river, then it raises its own flag and sets the indicator to the value of the* ***other*** *clan. Then, it checks the state of the other clan's flag and the value of the indicator. If the other clan's flag is raised and the indicator has the other clan's value, then they re-check the other clan's flag and value of the indicator (until either of them changes). If the other clan's flag is lowered or the indicator's value is this clan's then this clan goes to the river, waters its cows, and lowers its flag after they've left.*

*So, the Hatfields would do something like the following pseudo-code; the McCoys would behave symmetrically.****Note: instead of numeric constants 1 & 2, you can also use labels in FSP, like*** ’hatfields ***and*** ’mccoys ***, or enums in Java.***

|  |
| --- |
| while (true) {  hatfieldsFlag.raiseFlag();  turn.set(2); // set turn indicator to McCoys  while (mccoysFlag.flagIsRaised() && turn.get() == 2) {  ; // skip - wait till condition changes.  }  river.visit(hatfields);  river.leave(hatfields);  hatfieldsFlag.lowerFlag();  } |

* 1. Identify the actions of interest, and the processes involved in them (again, don’t invent actions/processes).

[2]

* 1. Construct a structural diagram showing the processes identified and how these are linked by the actions.

[3]

* 1. Provide an FSP model that describes the system with Judge Gary’s river-sharing protocol.

[30]

* 1. Check whether your FSP model in Q6.c satisfies the safety property from Q3.d.

[5]

* 1. Check whether your FSP model in Q6.c satisfies the progress properties you had provided for Q5 both when the clans are greedy and when they are not greedy.

[5]

1. Implement in Java Judge Gary's solution as modelled in Q6, using monitors and threads. Include a simple UI (could be text-based or a GUI) that shows the state of the flags and the turn indicator when they change and the actions each clan makes. Add brief comments to explain why/how your code maps to the FSP model of Q6 *(very few marks gained without a good mapping – the whole point is to produce code that follows a verified model).* Feel free to re-use GUI code from the book applets if that helps (add a comment to state which code you’ve re-used from where). The UI is not of much interest; what’s important is that your code follows the model properly.

[30]

1. When provided with Sbehaviour, how can you check with FSP that your solution to Q1 is correct, especially so if Sbehaviour was a quite large and complicated process *(assuming that Sbehaviour contains only deterministic choices)*?

[10]

## Grading

Grading is done based on whether you have demonstrably achieved the following learning objectives:

* **Construct** models from the specification; [so follow the spec’s structure!]
* **Apply** standard solutions to common concurrency problems;
* **Relate** your models to the specification; [no random processes/actions]
* **Define** safety and progress properties for your models;
* **Verify** the behaviour of your models with respect to your properties;
* **Implement** your model in Java; and
* **Relate** your implementation to your model. [no random classes/code, code implements the FSP model correctly]

Note that there are more than 100 marks (**157** actually), which correspond to 95% of the total mark (see the following section for the remaining 5%).

Of these marks, the first **40** count as a percentage mark, while the remaining ones are weighted.

## Submission – Attention to detail **5%**

* Create a folder named after your Moodle login, using lowercase letters: abcd345 (NOT ABCD345, NOT your name)
* **Copy** all files inside that folder (q1.lts q2.lts q3.lts q3c.pdf q4.lts q4b.pdf q5.lts q6.lts q6b.pdf, q8.lts, and q7/xxx.java – q7 being a Java package). Normal text answers will simply be comments /\* \*/ in the respective LTS file (with potentially no FSP model if that's not required).
* Produce a **zip** archive (a ZIP, not a RAR/TAR/7z/etc.) of your folder and submit that (abcd345.zip). On Linux: zip -r abcd345.zip abcd345/

*Better check that when you unzip the archive (in another folder) you obtain a folder named after your login and inside it all the files you wanted to submit.*

*Develop by keeping versions on gitlab/github* ***IN A PRIVATE REPOSITORY*** *– if there are problems then I’ll accept files from there,* ***if they were last modified before the deadline****.  
Files from your computer will not be accepted.*