Concurrency Exercises 6: General exercises

This sheet consists of questions on a wide variety of topics in concurrency: Modelling, process algebra, and properties of concurrent systems.

1. Simple example:
   Write a very simple model of a VCR machine can only do play and stop actions. Model the VCR as a process, VCR, using FSP, and draw the LTS.

   **Answer:**

   
   
   ```
   VCR = PLAY,
   PLAY = (play -> SSTOP),
   SSTOP = (stop -> PLAY).
   ```

   
   **Notes:** VCR is an example of an FSP process, whereas PLAY and SSTOP are local FSP processes. They are local to the VCR process.

   Note the process name SSTOP is used instead of STOP. This is because STOP is a reserved local process name which means that there cannot be any more transitions. The action stop may be used as shown.

   By substitution you can arrive at a more concise definition of VCR:

   ```
   VCR = (play -> stop -> VCR).
   ```

   **References:** See the SWITCH example in the lectures.
2. Choice:

Extend the model of the VCR to include a pause action and draw the LTS. One can only pause after a play action, and it can be followed by either a play action or a stop action. Check that the VCR can perform the actions given by the trace:

play -> stop -> play -> pause -> play -> pause -> stop

Answer:

VCR = PLAY,
PLAY = ( play -> PLAYING ),
PLAYING = ( stop -> VCR | pause -> PAUSE ),
PAUSE = ( stop -> VCR | play -> PLAYING ).

Notes: The SSTOP process has been changed to a PLAYING process because you now have an option of either a stop action or a pause action so SSTOP is an inappropriate process name. If stop then you go back to the PLAY state allowing you to play again, If pause you go into a new state PAUSED where you can choose to stop or play.

References: See the DRINKS example in the lectures and textbook.

3. Choice:

Draw the LTS for the MICROWAVE FSP process below. What is the alphabet of the process, how many states and transitions are there?

MICROWAVE = ( put_food_in -> SETTINGS ),
COOK = ( cook -> take_food_out -> MICROWAVE ).

Answer:

Alphabet: \{ cook, put_food_in, set_heat_level, set_time, take_food_out \}

States: 6

Transitions: 7

4. Non-determinism:

Draw the LTS for the LIBRARY FSP process below. What is the alphabet of the process, how many states and transition are there?

LIBRARY = ( take_book_to_desk -> give_details -> take_book->OPTIONS ),

OPTIONS = ( renew -> OPTIONS |
            | return_on_time -> LIBRARY |
            | return_late -> FINE ),

FINE = ( pay_fine -> LIBRARY |
          | pay_fine -> never_go_back -> STOP ).

Answer:
Alphabet: {give_details, never_go_back, pay_fine, renew, return_late, return_on_time, take_book, take_book_to_desk}

States: 7
Transitions: 9

Notes: The process FINE is non-deterministic since after the action pay_fine, it may behave as either LIBRARY or (never_go_back -> STOP).

References: See the COIN example in the lectures.

5. Indexed Processes and Actions:
A function takes an integer as an input and returns the square of the input. Model the function as a process SQUARE in FSP. Use input range 0..3 and draw the LTS. There are several different approaches on how to answer this question:

1) Indexed actions using a range
2) Indexed actions explicitly stated
3) Indexed local process and a declared range type.
4) Parameterised process. (Parameter must be upper case)

Answer using any two of the above approaches.

Answer:

APPROACH 1
----------
SQUARE = ( input [i:0..3] -> square[i*i] -> SQUARE ).

APPROACH 2
----------
SQUARE = ( input [0] -> square[0*0] -> SQUARE 
).

APPROACH 3
----------
range R = 0..3
SQUARE = ( input [i:R] -> CALCULATE[i] ),
CALCULATE[i:R] = ( square[i*i] -> SQUARE ).

APPROACH 4
----------
SQUARE (R=3) = ( input [i:0..R] -> square[i*i] -> SQUARE ).
6. **Indexed Processes:**

A Celsius to Fahrenheit converter works by taking a temperature in Celsius as an input and outputs the equivalent temperature in Fahrenheit (i.e. multiply the Celsius by 9, divide by 5 and add 32). Model the converter as a process `C_F_CONVERTER` using FSP. Use an input range of 0..5.

**Answer:**

```
C_F_CONVERTER(N=5) = (celsius[i:0..N] -> fahrenheit[32 +(9*i)/5] -> C_F_CONVERTER).
```
7. Guarded Actions

Draw the LTS for the SYS FSP process below. What is the alphabet of the process, how many states and transitions are there, and what is the process doing?

SYS = (in[i:0..5] -> ( when(i*i>10) keep[i]  ->  SYS
| when(i*i<10) getrid [i]  ->  SYS
| when(i*i=10) error  ->  ERROR
)
).

**Answer:**

Alphabet: \{getrid[0..3], in[0..5], keep[4..5]\}
States: 7
Transitions: 12

This SYS FSP process takes in a range of numbers, and keeps all numbers whose square is greater than 10, and gets rid of all numbers whose square is less than 10. The last guarded action has no effect on the LTS as the guard is never true.

8. LTS to FSP

Model the LTS below in FSP. There are many different ways of doing this, you may want to declare a `const` and a `range` at the beginning.
Answer:

This is the most concise way of writing the FSP, and takes advantage of constants, ranges and indexed processes.

\[
\begin{align*}
\text{const } N &= 2 \\
\text{range } \text{Int} &= 0..N \\
\text{VARIABLE} &= \text{VAR}[0], \\
\text{VAR}[i:\text{Int}] &= ( \text{read}[i] \rightarrow \text{VAR}[i] \\
&\quad | \text{write}[v:\text{Int}] \rightarrow \text{VAR}[v] \\
),\end{align*}
\]

This is an alternative method which only uses indexed processes.

\[
\begin{align*}
\text{VARIABLE} &= \text{VAR}[0], \\
\text{VAR}[i:0..2] &= ( \text{read}[i] \rightarrow \text{VAR}[i] \\
&\quad | \text{write}[v:0..2] \rightarrow \text{VAR}[v] \\
),\end{align*}
\]

This method uses only the most basic FSP notation and all the choices are explicitly stated.

\[
\begin{align*}
\text{VARIABLE} &= \text{VAR}[0], \\
\text{VAR}[0] &= ( \text{read}[0] \rightarrow \text{VAR}[0] \\
&\quad | \text{write}[0] \rightarrow \text{VAR}[0] \\
&\quad | \text{write}[1] \rightarrow \text{VAR}[1] \\
&\quad | \text{write}[2] \rightarrow \text{VAR}[2] \\
),, \\
\text{VAR}[1] &= ( \text{read}[1] \rightarrow \text{VAR}[1] \\
&\quad | \text{write}[0] \rightarrow \text{VAR}[0] \\
&\quad | \text{write}[1] \rightarrow \text{VAR}[1] \\
&\quad | \text{write}[2] \rightarrow \text{VAR}[2] \\
),, \\
\text{VAR}[2] &= ( \text{read}[2] \rightarrow \text{VAR}[2] \\
&\quad | \text{write}[0] \rightarrow \text{VAR}[0] \\
\end{align*}
\]
9. Parallel Composition

(a) Draw the composite LTS for the SMILE and LUNCH FSP processes. What are the 3 possible traces of actions that can happen? How many states and transitions are there in the LTS?

\[
\text{SMILE} = (\text{smile} \rightarrow \text{STOP}) \]
\[
\text{LUNCH} = (\text{eat} \rightarrow \text{drink} \rightarrow \text{STOP})
\]
\[
||\text{LUNCH}||\text{SMILE}=(\text{LUNCH}||\text{SMILE})
\]

Answer:

\[
\begin{align*}
\text{States: } 6 \\
\text{Transitions: } 7 \\
\text{Traces: } & \text{eat} \rightarrow \text{drink} \rightarrow \text{smile} \rightarrow \text{STOP} \\
& \text{smile} \rightarrow \text{eat} \rightarrow \text{drink} \rightarrow \text{STOP} \\
& \text{eat} \rightarrow \text{smile} \rightarrow \text{drink} \rightarrow \text{STOP}
\end{align*}
\]

Notes: We know that the actions from the same process are executed in order. However, since processes proceed at arbitrary relative speeds, actions from different processes are arbitrarily interleaved. So you will notice that an \text{eat} action is always before a \text{drink} action, but they are arbitrarily interleaved with \text{smile} actions. This leads to the three different traces.

(b) Process Labelling

Using the LUNCH process from part (a), model a JOINT_LUNCH process using FSP, when two people have lunch together. What is the alphabet of the JOINT_LUNCH process, and how many states and transitions are there in the LTS?

Answer:

\[
||\text{JOINT_LUNCH} = (a:\text{LUNCH}||b:\text{LUNCH})
\]

Alphabet: \{a, b\}, \{\text{drink, eat}\}
States: 9
Transitions: 12
Notes: In order to have more than one copy of a LUNCH process, we need to describe a model which distinguishes the two LUNCH processes. We do not want their actions to be shared, i.e. they must have disjoint labels. To do this we use the process labelling construct as shown. From the output from the LTSA it can be clearly seen that the alphabets of the two processes are disjoint.

10. Process Labelling using a set of prefix labels:

Draw the LTS for the SYS FSP process below. What is the alphabet of the process, how many states and transition are there?

\[
A = (a \rightarrow b \rightarrow c \rightarrow A).
\]
\[
Z = (a \rightarrow c \rightarrow Z).
\]
\[
\text{SYS}(N=3) = (g[i:1..N]:A || \{g[i:1..N]}::Z).
\]

Answer:

Alphabet:\{g[1..3].\{a, b, c\}\}

States: 7

Transitions: 9

Notes:

11. Parallel composition, Action Hiding and Minimisation
\[ P = (a \rightarrow b \rightarrow P) \]
\[ Q = (c \rightarrow b \rightarrow Q) \]
\[ ||S_1 = (P||Q) \]
\[ S_2 = (a \rightarrow c \rightarrow b \rightarrow S_2 \mid c \rightarrow a \rightarrow b \rightarrow S_2) \]

(a) Show that S1 and S2 describe the same behaviour. Hint: animate the FSP descriptions using LTSA and observe the LTS. The || operator signifies the parallel composition of the processes P and Q. Type the FSP above into LTSA, check the graphs/LTSs of S1 and S2, are they the same, do S1 and S2 describe equivalent systems, i.e. can they produce equivalent action traces? Parallel composition will create a composite LTS that forces shared actions (actions having the same name in different processes) to execute simultaneously. We say that shared actions are synchronised actions. Note: that non-shared actions are not synchronised.

(b) How many states and transitions will there be if action b is hidden in the P process, which is then composed with Q? Model this in FSP and draw the LTS.

**Answer:**

\[ P = (a \rightarrow b \rightarrow P) \{b\} \]
\[ Q = (c \rightarrow b \rightarrow Q) \]
\[ ||S_1 = (P||Q) \]
States: 4
Transitions: 8
Notes: Now that the $b$ action is hidden (labelled $\tau$ in LTS), there are no more shared actions between the two processes. Therefore they can arbitrarily interleave like the LUNCH_SMILE example above.

(c) Apply the hiding operator to the above process $S1$, to remove action $b$ and draw the minimised LTS.

Answer:

\[
P = (a \rightarrow b \rightarrow P).
\]
\[
Q = (c \rightarrow b \rightarrow Q).
\]
\[
||S1 = (P||Q)\{b\}.
\]

States: 3
Transitions: 4.

(d) Use transition rules to calculate the computation trees for process $S1$ performing an $a$ action, followed by a $c$ action and finally a $b$ action. What is the transition sequence for this question (hint: there are 10 rules required for this.)

Answer:

\[
S1 \rightarrow^a ((b\rightarrow P)||Q) \rightarrow^c (((b\rightarrow P)||b\rightarrow Q)) \rightarrow^b (P||Q)
\]

12. Relabelling

Draw the LTS for the PANDQ FSP process below. What is the alphabet of the process, how many states and transitions are there?

\[
P = (x \rightarrow y \rightarrow P).
\]
\[
Q = (x \rightarrow y \rightarrow Q).
\]
\[
||PANDQ = (a[i:1..2]:P || Q)/\{a[i:1..2].x/x, a[i:1..2].y/y\}.
\]
13. Alphabet Extension

Draw the LTS for the S1 FSP process below. What effect has the alphabet extension on Q had on the LTS of S1?

What is the alphabet of S1? (Hint: First draw the LTS for P and Q ignoring the alphabet extension. Then attempt to draw the LTS for S1)

\[
P = ( \begin{array}{c} b \rightarrow P \\
| a \rightarrow e \rightarrow P \\
| c \rightarrow h \rightarrow P \end{array} ).
\]

\[
Q = ( \begin{array}{c} c \rightarrow g \rightarrow Q \\
| d \rightarrow f \rightarrow Q \end{array} +\{a, h\}.
\]

\[
||S1 = ( P || Q).
\]

Answer:
The alphabet extension on $Q$, restricts process $P$ from carrying out those actions. Therefore process $P$ cannot do actions $a$ or $h$. That is why actions $a$ or $h$ are not in the LTS for $||S_1$. Furthermore this has also prevented action $e$ from occurring as it can only take place after action $a$. The LTS is therefore small.

14. Model the MICROWAVE example from above, this time using parallel composition.
(hint: You will need to use handshaking with shared actions, so that it is not possible to produce silly action traces. eg to cook after take_food_out. There are many ways of doing this, you may want to use the following three processes: COOK, SET_HEAT and SET_TIME.)

Answer:

$$\text{COOK} = (\text{put_food_in} \rightarrow \text{cook} \rightarrow \text{take_food_out} \rightarrow \text{COOK})$$
$$\text{SET_HEAT} = (\text{put_food_in} \rightarrow \text{set_heat_level} \rightarrow \text{cook} \rightarrow \text{SET_HEAT})$$
$$\text{SET_TIME} = (\text{put_food_in} \rightarrow \text{set_time} \rightarrow \text{cook} \rightarrow \text{SET_TIME})$$
$$||\text{MICROWAVE} = (\text{COOK}||\text{SET_HEAT}||\text{SET_TIME}).$$

The processes SET_HEAT and SET_TIME require the action put_food_in so that they handshake with the MICROWAVE process, and are not allowed to just randomly interleave. The MICROWAVE process cannot cook until both the actions set_heat_level and set_time have occurred, this is achieved by the shared cook action.

15. Safety

Draw the LTS for property FRIEND=(come$\rightarrow$tea$\rightarrow$leave$\rightarrow$FRIEND).
(Hint: there are 9 transitions)

Answer:

When using the property keyword the compiler automatically generates the transitions to the error state.

References: See the COLOUR example in lecture 13
16. Safety

Draw the LTS for the SAFETY FSP process below. What action trace violates the safety property?

property SAFETY = (a -> (b -> SAFETY|a -> SAFETY)|b -> a -> SAFETY).

Answer:

Trace to property violation in SAFETY:
b->b

References: See the COLOUR example in lecture 13

17. Liveness

How many of the following progress properties are violated and why?

START = ( select -> BUFF_ONE
| select -> BUFF_TWO),
BUFF_ONE = (in[i:0..3] -> out[i] -> BUFF_ONE),

progress ONE = {out[0], in[1], out[2]}
progress TWO = {in[3]}
progress THREE = {in[1], in[3], out[2], in[5]}
progress FOUR = {in[3], out[4], out[2], in[5]}
progress FIVE = {out[4], in[5]}

Answer:
Progress properties ONE and FIVE have been violated. There are two terminal sets BUFF_ONE and BUFF_TWO.
Actions in terminal set BUFF_ONE are: in, out[0..3]
Actions in terminal set BUFF_TWO are: in, out[3..5].
For the ONE property, none of the actions in the progress set occur in the terminal set actions in, out[3..5] of BUFF_TWO, and for the FIVE property, none of the actions in the progress set occur in the terminal set actions in, out[0..3] of BUFF_ONE.

**Notes:** Checking that a progress property holds is simply checking that in each terminal set, at least one of the actions in the progress set occurs as a transition. Conversely, a progress property is violated if there exists a terminal set of states in which none of the progress set actions appear.

**References:** See the TWOCOIN example in lecture 15

18. The following 3 processes describe the actions that students undertake. Use the parallel composition operator to create a process SYS modelling the concurrent execution of these different processes. Modify the individual processes (using handshaking with shared actions) so that it is not possible to produce silly action sequences, for example that allow a student to go to lectures undressed, or, that allow a student to perform PLAY actions before WORK actions have completed.

```
PLAY = ( pub --> PLAY |
         gig --> PLAY |
         clubbing --> PLAY ) .
WORK = ( lectures --> WORK |
         laborartory --> WORK |
         library --> WORK |
         assessments --> WORK ) .
DAY = ( wake --> eat --> dress --> undress --> sleep --> DAY ) .
||STUDENT = (DAY || WORK || PLAY) .
```

**Answer:**

```
PLAY = ( donework --> TOPLAY ),
TOPLAY = ( pub --> TOPLAY |
          gig --> TOPLAY |
          clubbing --> TOPLAY |
          doneplay --> PLAY ) .
WORK = ( dress --> READY ),
READY = ( lectures --> READY |
         laborartory --> READY |
         library --> READY |
         assessments --> READY |
         donework --> WORK
```
DAY = ( wake -> eat -> dress -> doneplay
-> undress -> sleep -> DAY
).

||STUDENT = (DAY || WORK || PLAY).

19. The Rambling Bramble Ornate Garden society require a new electronic system
to monitor and control the flow of visitors through its main attraction - Wiggly
Park. This garden has a number of controlled areas, including the deer park, the
ornamental garden and tea-room, each gated by a one-way turnstile. For simplicity
of modelling, we will only consider those three areas for this prototype control
system. Visitors may enter Wiggly Park via a West entrance into a deer park, or
via an East entrance directly into the Tea-rooms. Visitors may leave via the North
exit from the deer park, the South exit from the ornamental garden area, or the East
exit from the Tea-rooms. There are also one-way turnstiles to control movement
between (i) the deer park and the ornamental garden, and (ii) the ornamental
garden and the Tea-rooms.

Arrivals into and departures from the areas are signalled by the turnstiles to a
controller, which maintains counts of the numbers of visitors in the three areas. At
opening time, the Park Director signals the controller that Wiggly Park is open.
The controller must then permit flow through the three Wiggly Park areas subject
to specific limits for each of the areas. At closing time, the Park Director signals the
controller that Wiggly Park is to close. The controller in turn stops further visitors
entering Wiggly Park and in addition closes the turnstile from the ornamental
garden to the Tea-rooms.

A generic turnstile can be modelled as follows.

Turnstile = (start -> Working),
Working = (pass -> Working | stop -> Turnstile).

Instances of these can be created for each specific turnstile. For example, for the
west entrance to Wiggly Park (into the deer park) and the south exit from the
ornamental garden, one could have:
WestParkIn = ((deerpark.west:(Turnstile/{arrive/pass}))/
  {open/deerpark.west.start,
   deerpark.stopEntrance/deerpark.west.stop})).

SouthGardenOut = ((garden.south:(Turnstile/{leave/pass}))/
  {open/garden.south.start,
   garden.stopExit/garden.south.stop})).

Note how the WestParkIn process has renamed the pass action of the Turnstile process as arrive and then labels the whole by deerpark.west. Explain the other relabellings.

Create similar instantiated processes for the North exit from the deer park, the East Tearoom entrance and exit, and the four turnstiles controlling flow between deer park and ornamental garden, and the ornamental garden and the tea-rooms.

Then, the instantiated processes can all be composed in parallel to model the set of gates.

Gates = ( WestParkIn
  || SouthGardenOut
  || EastTearoomIn
  || EastTearoomOut
  || ParkToGarden
  || GardenToPark
  || TearoomToGarden
  || GardenToTearoom ).

Now model a generic area controller that be appropriately instantiated for the deer park, the ornamental garden and the tea-rooms. You might assume that these controllers use the follow signals (actions): arrive, leave, stopEntrance, stopExit. Then, to obtain controllers for the specific areas, label and rename as appropriate.

Finally, create a model for the whole system via a suitable composition of turnstiles and controllers.