Concurrent Exercises 2

Topic 2.2: FSP Modelling Concurrency

1. Draw (at least part of) the LTS for combining two people’s days, i.e. combining two ‘DAY3’ LTS diagrams, with no interaction between the two people (use actions upA, upB etc). How can you represent actions performed in ‘abstract parallel’?

2. Now consider how to model both people having to drink either tea or coffee together, i.e. they must synchronize on their tea or coffee actions.

3. If you are using concurrency in your 3rd Year Project, attempt to give abstract specifications for the concurrent components in FSP.

4. Show that $S_1$ and $S_2$ describe the same behaviour:

   $P = (a \rightarrow b \rightarrow P)$.
   $Q = (c \rightarrow b \rightarrow Q)$.
   $||S_1 = (P || Q)$.

   $S_2 = (a \rightarrow c \rightarrow b \rightarrow S_2 \mid c \rightarrow a \rightarrow b \rightarrow S_2)$.

   **Answer:**

   Hint: Construct the two LTSs by hand, and also using the LTSA tool.

5. $ELE\text{MENT} = (up \rightarrow down \rightarrow ELE\text{MENT})$ accepts an up action and then a down action. Using parallel composition and the $ELE\text{MENT}$ process describe a model that can accept up to four up actions before a down action.

   **Answer:**

   Hint: Connect the $ELE\text{MENT}$ processes together into a chain via renaming, leaving an up at one end and a down at the other, hiding other actions. Think of the processes as a bounded buffer, where the up/down action add/remove elements.

   One possible solution:

   $ELE\text{MENT} = (up \rightarrow down \rightarrow ELE\text{MENT})$.

   $\|\text{COUNT(N = 4)} = (\forall i:0..N-1) e[i]:ELE\text{MENT})$
   /
   { up/e[0].up, down/e[N-1].down,
     \forall i:0..N-2 \{ e[i].down/e[i+1].up
   } @\{up, down\}.$
Look at the LTS using the LTSA tool and RUN command. There is an alternative rather neater recursive definition:

\[
||\text{COUNTR}(N=4) = \text{if } (N == 1) \text{ then } \text{ELEMENT} \\
\text{else } (\text{ELEMENT}/\{\text{mid/down}\} || \text{COUNTR}(N-1)/\{\text{mid/up}\}) \@\{\text{up,down}\}.
\]

6. Extend the client server system \text{CLIENT\_SERVER} (see the course textbook) so that more than one client can use the server.

\textbf{Answer:}

Hint: Using process labelling to give several clients, e.g. \{a,b\}:	ext{CLIENT} the tricky part comes in connecting the client call actions to the server request actions. This can be achieved via the renaming \text{P} / \{\{a,b\}.call/request\}

\text{const N = 2}

\text{CLIENT = (call -> WAIT),}
\text{WAIT = (timeout -> continue -> CLIENT}
\text{ | wait -> continue -> CLIENT \}).

\text{SERVER = (request -> service -> reply -> SERVER).}

\[
||\text{CLIENT\_SERVER} = (\text{client}[1..N]:\text{CLIENT} || \text{SERVER})
\text{ / \{\text{client}[1..N].call/request,}
\text{ client[1..N].wait/reply \}.
\]

7. Modify the model in the previous exercise so that the call may terminate with a timeout action rather than a response from the server. What happens to the server in this situation?

\textbf{Answer:}

The simplest solution:

\text{set Clients = \{a,b\}}

\text{CLIENT = (call -> WAIT),}
\text{WAIT = (timeout -> continue -> CLIENT}
\text{ | wait -> continue -> CLIENT \}).

\text{SERVER = (request -> service -> reply -> SERVER).}

\[
||\text{CLIENT\_SERVER} = (\text{Clients: CLIENT} || \text{SERVER})
\text{ / \{Clients.call/request,}
\text{ Clients.wait/reply \}.
\]
clearly leads to a deadlock when CLIENT times out. One solution would be for SERVER also to have a timeout.

8. A roller-coaster system only permits its car to depart when it is full. Passengers arriving at the departure platform are registered with the roller-coaster controller by a turnstile. The controller signals the car to depart when there are enough passengers on the platform to fill the car to its maximum capacity of $M$ passengers. The car goes around the roller-coaster track and then waits for another $M$ passengers. A maximum of $M$ passengers may occupy the platform. Ignore the synchronisation details concerning passengers embarking from the platform and the car departure.

The roller-coaster consists of three processes: TURNSTILE, CONTROL and CAR. TURNSTILE and CONTROL interact by the shared action passenger indicating an arrival and CONTROL and CAR interact by the shared action depart signalling car departure. Provide FSP descriptions for each process and the overall composition.

**Answer:**
Here, the question is almost longer than the solution:

```c
const M = 3

TURNSTILE = (passenger -> TURNSTILE).

CONTROL = CONTROL[0],
CONTROL[i:0..M] = (when (i<M) passenger -> CONTROL[i+1]
|when (i==M) depart -> CONTROL[0])
).

CAR = (depart -> CAR).

ROLLEROASTER = (TURNSTILE || CONTROL || CAR).
```

9. A museum allows visitors to enter through the east entrance and leave through its west exit. Arrivals and departures are signaled to the museum controller by the turnstiles at the entrance and exit. At opening time, the museum director signals the controller that the museum is open and then the controller permits both arrivals and departures. At closing time, the director signals that the museum is closed, at which point only departures are permitted by the controller. Given that it consists of the four processes EAST, WEST, CONTROL and DIRECTOR, provide an FSP description for each of the processes and the overall composition.

**Answer:**

Hint: Once DIRECTOR has signalled that the museum is closed, the CONTROL process should only let visitors leave. Ideally it should continue doing this until everyone has left and so should keep a track of the number of visitors in the museum.

One solution is:

```c
const N = 5

EAST = (arrive -> EAST).
```
WEST = (leave -> WEST).

DIRECTOR = (open -> close -> DIRECTOR).

CONTROL = CLOSED[0],
CLOSED[i:0..N] = (when (i==0) open -> OPENED[0]
    |when (i>0) leave -> CLOSED[i-1])
OPENED[i:0..N] = (close -> CLOSED[i]
    |when (i<N) arrive -> OPENED[i+1]
    |when (i>0) leave -> OPENED[i-1])

||MUSEUM = (EAST || WEST || DIRECTOR || CONTROL).