Home Assignment MACS 1 - part II - 25 Oct 2014

Remarks:

1. The ultimate date for returning the assignment is **Monday, November 10, 2014 (strict).**
2. All answers can be found by hand in principle. It is allowed to use any tools, but in that case, mention which tool you used and explain the results.
3. Return your answers in two files, by E-mail to `J.C.vandePol@utwente.nl`:
   - A document **in pdf** with the answers and short explanation to the questions;
   - A **zip-file** containing files with all inputs and outputs for the tools that you used, plus a README.txt that describes exactly how I can reproduce the results.
4. You should work individually; it is not allowed to collaborate.

A. Confluence in LTS. Consider the following LTS:

$$\begin{tikzcd}
 & 0 \arrow[r, "a"] \arrow[d, "\tau"] & 1 \arrow[d, "\tau"] & \\
2 \arrow[d, "\tau"] & 3 \arrow[l, "b"'] \arrow[r, "a"] \arrow[d, "\tau"] & 4 \arrow[r, "a"] \arrow[d, bend left, "\tau"] & 5 \arrow[d, bend left, "\tau"] \\
6 & 7 \arrow[l, "b"'] \arrow[r, "a"] \arrow[d, "\tau"] & 8 \arrow[u, bend left, "\tau"] \arrow[r, "a"] \arrow[d, bend left, "\tau"] & 9 \arrow[u, bend left, "\tau"] \arrow[d, bend left, "\tau"] \\
 & 10 \arrow[r, "a"] & 11 \arrow[u, bend left, "\tau"] \arrow[r, "a"] & 12 \arrow[u, bend left, "\tau"]
\end{tikzcd}$$

1. Determine which of the following sets of τ-transitions form a confluent subset within this LTS:
   - the set $\{(4,8),(5,9)\}$
   - the set $\{(2,6),(3,7)\}$
2. Determine the maximal confluent subset of the τ-transitions.
3. Choose one representation map based on this maximal set.
4. How many different representation maps are there?
5. Draw the reduced LTS according to this representation map.

B. Designing a Queue Datatype

1. Next, we design a data type to represent and merge the contents of a queue of clients. For this we define a sort Queue over a further unspecified sort Client:

   ```
   sort Client, Queue;
   cons empty : Queue;
        insert: Client # Queue -> Queue;
   ```

   We define functions as follows:

   ```
   map merge : Queue # Queue -> Queue;
       first : Queue -> Client;
       rest  : Queue -> Queue;
       append: Queue # Client -> Queue;
   var c:Client;
       x,y:Queue;
   eqn first(insert(c,x))   = c;
       rest(insert(c,x))    = x;
       merge(empty,x)       = x;
       merge(x,empty)       = x;
       merge(insert(c,x),y) = insert(c,merge(x,y));
       append(x,c)          = merge(x,insert(c,empty));
   ```

   Question: Prove that the TRS consisting of these six equations (viewed as rewrite rules from left to right) is terminating. Do this by providing an interpretation of the function symbols `first, rest, empty, insert, merge, append` as monotone functions over natural numbers. Show for the last two rules that they will be decreasing according to your interpretation.

2. We extend our original specification with a test for emptyness on lists as follows:

   ```
   map isempty: Queue->Bool;
   var c:Client;
       x,y:Queue;
   eqn isempty(merge(empty,empty))   = true;
       isempty(merge(insert(c,x),y)) = false;
       isempty(merge(x,insert(c,y))) = false;
   ```

   Question: Show that the resulting TRS (combining all rules so far) is not confluent, by providing a term (possibly containing variables) and two reductions from that term to distinct normal forms.

3. Write down all critical pairs, and check if they are joinable.

4. Apparently, this TRS is not complete. We now restrict ourselves to the six equations that contain `merge` in their left hand side. Apply Knuth-Bendix completion, to obtain a terminating and confluent TRS extending these six equations.

5. **Bonus question:** What happens to termination and confluence if we add the equation `merge(x,merge(y,z)) = merge(merge(x,y),z)` ?

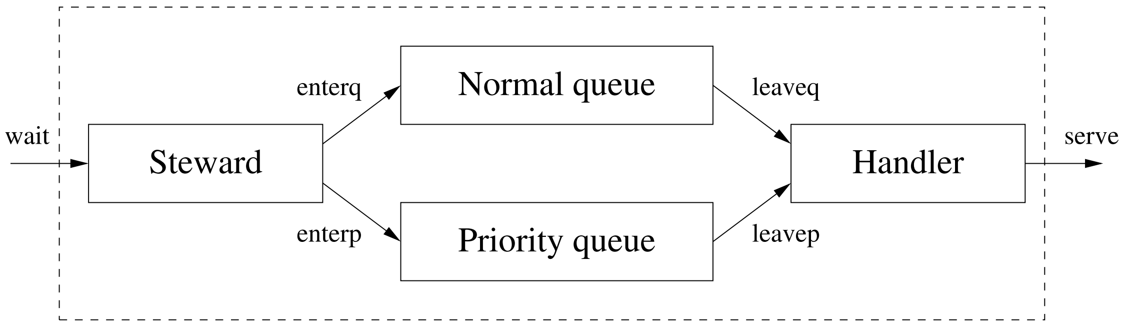

Figure 1: Airport Check-in Desk

C. Confluence at the Airport Check-in. We consider the Check-in desk at an airport. It has two queues, a normal queue and a priority queue. A mad Steward and Handler work as follows:

- The steward directs every odd waiting client to the priority queue, and every even waiting client to the normal queue.
- The handler takes a client from any of the queues in arbitrary order and serves them.

We specify the normal Queue and the mad Steward as follows, where we reuse the data types designed in Part B:

```
act wait,serve:Client;
act enterp,enterq,leavep,leaveq, Enterp, Enterq, Leavep, Leaveq: Client;

proc Q(q:Queue) =
   sum c:Client . enterq(c) . Q(append(q,c))
 + (!isempty(q))->leaveq(first(q)) . Q(rest(q));

proc Steward = sum c1:Client. wait(c1). Enterp(c1).
               sum c2:Client. wait(c2). Enterq(c2). Steward;
```

1. Provide similar definitions of the Priority Queue (P) and the mad Handler.
2. Define auxiliary actions, and provide communication, allow and hide-definitions to initalize the system as in Figure 1. The dashed box shows which actions are observable at the external system boundary.
3. Transform the system into Linear Process Specification format.
4. Next we study some confluence properties of this system. Write down the non-trivial commutation formulas for the tau-summands corresponding to entering the normal and the priority queue. Which of these formulas hold? Provide an intuitive explanation in terms of the order in which the clients are handled.