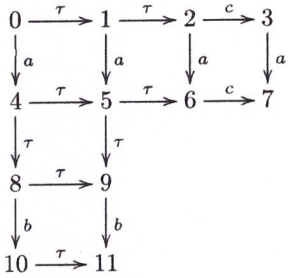
Excercises MACS 1 - part II

Jaco van de Pol

October 14, 2014

A. Confluence (process algebra): Consider the following LTS, with initial state 0:



1. Determine which of the following sets of τ-transitions form a confluent subset for this LTS.
   - the empty set $\emptyset$

     The empty set is always confluent, because the definition trivially holds (empty quantification)

   - the set $\{(8,9),(10,11)\}$

     This set is confluent: $(10,11)$ is OK (no really diverging pair). $(8,9)$ is OK as well, because the diverging steps $10 \leftarrow 8 \rightarrow 9$ can be closed with $10 \rightarrow 11 \leftarrow 9$ with the correct labels.

   - the set $\{(4,5),(4,8),(5,9),(8,9)\}$

     This is not confluent for several reasons. First $(10,11)$ is missing, so $(8,9)$ cannot be closed by a *confluent* diagram. Moreover $(5,9)$ is inherently non-confluent due to the conflict with $(5,6)$.

2. Determine the maximal confluent subset of τ-transitions.

The maximal set is $\{(0,1),(4,5),(8,9),(10,11)\}$. Checking confluence is straightforward. Checking that it is maximal: $(5,9)$ and $(5,6)$ can not be added due to the inherent divergence. Hence $(1,2)$ and $(4,8)$ cannot be added, because their diagrams cannot be completed by confluent tau-steps.

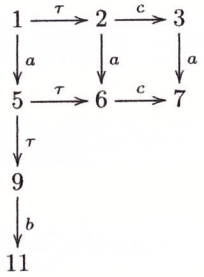
3. Determine a representant mapping φ for this example.

   This is easy and unique, because there are no tau-loops. The complete definition of ϕ with the maximal set above is:

$$\begin{cases} \phi(0) &= 1 \\ \phi(4) &= 5 \\ \phi(8) &= 9 \\ \phi(10) &= 11 \\ \phi(x) &= x, \text{ otherwise} \end{cases}$$

4. Draw the LTS after giving priority to confluent τ's and τ-compression.

The new initial state will be $\phi(0) = 1$. The reduced state space will be:



B. Data type specification Consider the following data type specification of the floors in a Dutch building[1].

```
sort Floor = struct bg | inc(dec:Floor);

map top:Floor;
    below: Floor#Floor -> Bool;
```

[1] bg = begane grond, level 0

```
var x,y:Floor;
eqn top=inc(inc(inc(bg)));
    dec(inc(x)) = x;
```

1. Extend it by defining the predicate (strictly) *below*.

   Add the three equations (like 'lt' on the slides):

   ```
   below(x,bg) = false;
   below(bg,inc(x)) = true;
   below(inc(x),inc(y)) = below(x,y);
   ```

2. Check if the resulting system is terminating, using monotone functions. In particular:
   (a) Provide the interpretation in a monotone algebra.

   One can use one of the tools, or come up with a solution by hand. For instance, take as interpretation in the natural numbers: $[\![bg]\!] = [\![false]\!] = [\![true]\!] = 0$, $[\![top]\!] = 4$, $[\![inc]\!](a) = [\![dec]\!](a) = a + 1$, and $[\![below]\!](a, b) = a + b + 1$.

   (b) Show the full computation for the recursive rule of *below*.

   Then one can compute for all rules (only the last is asked for):

   $$\begin{aligned} [\![top]\!] = 4 &> 3 = [\![inc(inc(inc(bg)))]\!] \\ [\![dec(inc(x))]\!] = x + 2 &> x \\ [\![below(x, bg)]\!] = x + 1 &> 0 = [\![false]\!] \\ [\![below(bg, inc(x))]\!] = x + 2 &> 0 = [\![true]\!] \\ [\![below(inc(x), inc(y))]\!] = x + y + 3 &> x + y + 1 = below(x, y) \end{aligned}$$

3. Check if your system is terminating using lexicographic path order.
   (a) What is the precedence?

   The precedence should have $top > inc, bg$ and $below > true, false$. For the rest it doesn't matter.

   (b) Show a derivation of $>_{\text{lpo}}$ for the recursive rule.

   For the recursive rule, we get:
   i. $inc(x) >_{\text{lpo}} x$ (subterm rule)
   ii. $inc(y) >_{\text{lpo}} y$ (subterm rule)

iii. $below(inc(x), inc(y)) >_{\text{lpo}} x$ (subterm rule and i)
iv. $below(inc(x), inc(y)) >_{\text{lpo}} y$ (subterm rule and ii)
v. $\langle inc(x), inc(y)\rangle >^{\text{lex}}_{\text{lpo}} \langle x, y\rangle$ (lexicographic order, and i)
vi. $below(inc(x), inc(y)) >_{\text{lpo}} below(x, y)$ (2nd rule, and iii,iv,v)

4. Check if the total system has overlaps, and if the critical pairs are joinable.

   There are no overlaps, hence no critical pairs

5. We now replace the second given equation by the following one:

   ```
   inc(dec(x)) = x;
   ```

   Apply Knuth-Bendix completion to obtain a complete TRS.

   Now there are several critical pairs, as can be checked with the Tyrolean tool (ttt2). One of them can be added as new rule:

   ```
   below(bg,x) -> true
   ```

   This gives a new critical pair that can be oriented anyway: `true = false`. After adding this, all critical pairs are joinable, and the system is terminating, so the system is complete. All problems are solved(?)

6. You should notice something strange. What happens? What is the cause of the problem?

   Of course, the equality `true=false` is problematic, because it makes the specification inconsistent. The problem is introduced by the incorrect equation: $inc(dec(bg)) = bg$ is not true, as $dec(bg)$ is not a valid floor.[2]

C. Linear Processes and Confluence: We analyse a lift system with 6 actions: the lift can move *up* and *down*, its doors can *open* and *close*, and there is a light, which switches *on* and *off*. The lift behaves as follows: If the lift is open, it must first be closed, and then the light must be switched on. A closed lift can move either down or up (if it is not on floor bg or top, respectively), or it can be opened after which the light must be switched off. This lift can be specified as follows:

```
act open, close, up, down, on, off;
```

```
proc lift_open(f:Floor) = close . on . lift_close(f);

    lift_close(f:Floor) =
      open . off . lift_open(f)
    + below(f,top) ->   up . lift_close(inc(f))
    + below(bg,f)  -> down . lift_close(dec(f));
```

1. We are not interested in the light. Provide the initial state of a closed lift on ground level, with the light-switching actions hidden.

   This can be specified as follows:

   ```
   init hide({on,off},  lift_close(bg));
   ```

2. Transform this process in linear process format.

   An intermediate form would be:

   ```
   proc
   lc1(f) = open . lc2(f)
          + below(f,top)->up . lc1(inc(f))
          + below(bg,f)->down . lc1(dec(f))
   lc2(f) = tau . lo3(f)
   lo3(f) = close . lo4(f)
   lo4(f) = tau . lc1(f)
   ```

   The final linear process could be:

   ```
   proc L(p:Pos,f:Floor) =
     p=1 -> open . L(2,f)
   + p=1 && below(f,top) -> up . L(1,inc(f))
   + p=1 && below(bg,f) -> down. L(1,dec(f))
   + p=2 -> tau. L(3,f)
   + p=3 -> close . L(4,f)
   + p=4 -> tau . L(1,f);

   init L(1,bg);
   ```

3. Check which summands are confluent by generating and checking the commutation formulae. Show at least one formula in detail.

   There are 2 tau-summands, so $2 \times 6 = 12$ commutation formulae. They are all trivial in this case (and they can be checked by lpsconfcheck). One example (summand 2 and 4):

   ```
   p=1 && below(f,top) && p=2
   ```

```
-> 1=2 && 3=1 && below(f,top) && (3,inc(f)) == (1,inc(f))
```

4. Generate the state space with and without confluence reduction.
   (a) Compare the size of the state spaces and the amount of reduction.

   Original: 16 states, 22 transitions. Reduced: 8 states, 14 transitions

   (b) Provide a formula of the state space sizes for a building with any number of floors.

   A building with n storeys results in $4n$ states and $6(n-1)+4 = 6n-2$ transitions. After reduction, one gets $2n$ states and $4(n-1)+2 = 4n-2$ transitions. So this is a linear state space reduction only. For a given n, one might check the result by putting the specs under B and C in lift.mcrl2 and use the following commands:

   ```
   mcrl22lps lift.mcrl2 lift.lps
   lpsconfcheck lift.lps > liftc.lps
   lps2lts -v lift.lps lift.dot
   lps2lts -vc liftc.lps liftc.dot
   ```

   Also, the dot files can be visualized nicely with dot.