Exercises werkcollege 6

Exercise 1

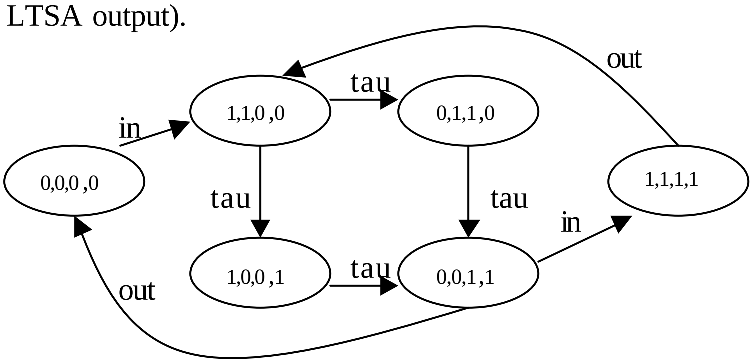
Consider the following FSP definitions:

```
BUFFER    = (in -> out -> BUFFER).

||SYNC_IN = (a:BUFFER || b:BUFFER)/{in/{a.in,b.in}}.
||SYNC_OUT= (c:BUFFER || d:BUFFER)/{out/{c.out,d.out}}.

||SYSTEM  = ( SYNC_IN/{sync.ac/a.out,sync.bd/b.out}
            ||SYNC_OUT/{sync.ac/c.in,sync.bd/d.in})
           @{in,out}.
```

a) Give a structured graph of the labelled transition system of `SYSTEM`. Label the states with tuples (*i*,*j*,*k*,*l*), where *i*,*j*,*k*,*l* are the respective local states of the processes `a:BUFFER`, `b:BUFFER`, `c:BUFFER` and `d:BUFFER`, who collectively determine the global state of `SYSTEM` (so you can't just copy the the LTSA output).



b) Give a minimal automaton that is observation equivalent to `SYSTEM`. Give a sequential FSP process (i.e. without parallel composition or hiding) that is observation equivalent to `SYSTEM`.

```
DBUFFER = LEEG,
LEEG = (in -> HALFVOL),
HALFVOL = (in -> VOL | uit -> LEEG),
VOL = (uit -> HALFVOL).
```

Exercise 2

Complete the MAZE example given in lecture 6 (slide 13). A path out of the maze is called *balanced* if and only if the number of north/south steps equals the number of east/west actions in the path. Modify your model such that for each initial square a shortest balanced path out of the maze, if it exists, is produced as a deadlock trace of the model. Determine the squares for which a balanced exit path exists.

```
MAZE(Start=8) = P[Start],
P[0] = (north->STOP|east->P[1]),
P[1] = (east ->P[2]|south->P[4]|west->P[0]),
P[2] = (south->P[5]|west ->P[1]),
P[3] = (east ->P[4]|south->P[6]),
P[4] = (north->P[1]|west ->P[3]),
P[5] = (north->P[2]|south->P[8]),
P[6] = (north->P[3]),
P[7] = (east ->P[8]),
P[8] = (north->P[5]|west->P[7]).

BALANCE = B[0],
B[i:-100..100] = ({east,west}->B[i+1]
                 |{north,south}->B[i-1]
                 |when (i!=0) loop -> B[i]).

||SOLUTION = (MAZE || BALANCE).
```

This solution will also generate safety errors because the index variable can be out of range, which will be visible as an `ERROR` state. By choosing the bound sufficiently great (e.g. 100) we can be sure that the shortest counter-examples will lead to the desired deadlock states.

Exercise 3

One solution to the dining philosophers problem permits only 4 philosophers to sit down at the table at the same time. Specify a BUTLER process that, when composed with the model presented in lecture 6 (slide 9), permits a maximum of 4 philosophers to be seated concurrently at the table. Show that this system is deadlock-free.

```
const N=5

PHIL = (sitdown->right.get->left.get
          ->eat->left.put->right.put
          ->arise->PHIL).

FORK = (get -> put -> FORK).

||DINERS =
   forall [i:0..N-1]
   (phil[i]:PHIL
   ||{phil[i].left,phil[((i-1)+N)%N].right}::FORK).

BUTLER = B[0],
B[i:0..4] = (when (i>0) arise -> B[i-1]
```

```
            |when (i<4) sitdown -> B[i+1]).

\\ Note that the index of BUTLER counts the number of
\\ philosophers that is seated.

||WAITEDPHILS = (DINERS || phil[0..N-1]::BUTLER).
```

Exercise 4

What action trace violates the following safety property?

```
     property PS = (a->(b->PS|a->PS)|b->a->PS).
```

The traces of even length ending in `b->b.`

Exercise 5

A lift has a maximum capacity of ten people. In the model of the lift control system, passengers entering a lift are signalled by an `enter` action and passengers leaving the lift are signalled by and `exit` action. Specify a safety property in FSP that when composed with the lift will check that the system never allows the lift to have more than 10 occupants.

```
property CONTROL = C[0],
C[i:0..10] = (when (i>0) exit -> C[i-1]
            |when (i<10) enter -> C[i+1]).
```