A1: Modelling behaviour

David Pereira & José Proença

RAMDE – 2021/2022

To do: Produce a report as a PDF document including the answers to the exercises below.

To submit: The PDF report and the 3 files requested in the exercises: `farmer1.mcrl2`, `farmer2.mcrl2` and `farmer3.mcrl`. All files should be in your group's git repository. ALL students should push commits.

Deadline: 12 Dec 2021 @ 23:59 (Sunday)

Auxiliary files: `https://cister-labs.github.io/ramde2122/assignments/farmer.zip`

Modelling the farmer-fox-goose-beans problem

A **farmer** wants to transport a **fox**, a **goose**, and some **beans** across a river (from the **left** margin to the **right** margin). Unfortunately, he can only carry one at a time. Furthermore, if the farmer is not present, the fox will eat the goose and the goose will eat the beans. The problem is solved if the farmer can carry all animals across the river.

```
%% file: famer1.mcrl2
act
  fr,fl,gr,gl,br,bl,                              % actions by the passengers
  ffr,fgr,fbr,farmerr,ffl,fgl,fbl,farmerl,        % actions by the farmer
  foxr,foxl,gooser,goosel,beansr,beansl,          % actions by the system
  winf,wing,winb,win;                             % actions to detect winning conditions

proc
  Fox = fr.(fl+winf).Fox ;
  Goose = gr.(gl+wing).Goose ;
  Beans = br.(bl+winb).Beans ;
  Farmer = (ffr+fgr+fbr+farmerr).(ffl+fgl+fbl+farmerl).Farmer ;

  Sys = allow(
          { foxr,foxl,gooser,goosel,beansr,beansl,farmerl,farmerr,win },
        comm(
          { fr|ffr → foxr,   fl|ffl → foxl,
            gr|fgr → gooser, gl|fgl → goosel,
            br|fbr → beansr, bl|fbl → beansl,
            winf|wing|winb|farmerl → win
          },
          Fox || Goose || Beans || Farmer
        ));

init
  Sys;
```

Draft

Exercise 1. We will encode the same problem using mCRL2's process algebra. Start by downloading the auxiliary files for this assignment at `https://cister-labs.github.io/ramde2122/assignments/farmer.zip`, where you will find the `farmer1.mcrl2` file above. This is a simplified (but incomplete) specification of our farmer-fox-goose-beans problem.

The specification is split into three sections: ***act***, a declaration of 24 actions, ***proc***, the definition of 4 processes, and ***init***, the initialisation of the system.

1.1. Create a new project `farmer1` using `mcrl2ide`, and add the resulting project folder to your git repository. Produce the labelled transition system (LTS) of this mCRL2 specification and **show a screenshot of the LTS (make sure it is understandable).**

1.2. This specification is not complete yet, i.e., it does model the puzzle completely. **Explain informally why this specification is not complete**, by explaining what is being modelled and what is still missing.

1.3. If you replace the ***init*** block by only `Fox || Goose || Beans || Farmer` (i.e., without the restrictio0ns *allow* and *comm*) **would you obtain more or less states** than with the original specification? **Why?**

Exercise 2. We present below a new specification for the same problem consisting of a single process `State` that keeps the state information, found in the provided auxiliary file `farmer2.mcrl2`. This new specification includes more advanced features of mCRL2, including: a *data structure*, actions with *data parameters*, processes with *data parameters*, and user defined *functions* **inv** and **ok**.

```
%% file: farmer2.mcrl2
sort
  Position = struct left | right;
map
  inv : Position -> Position ;
  ok  : Position # Position # Position # Position -> Bool ;
var
  fm,f,g,b: Position;
eqn
  inv(left)   = right ;
  inv(right)  = left ;
  ok(fm,f,g,b) = %% (1) %%;

act
  fox,goose,beans,farmer : Position; % system actions, parameterised on the position
  win;                              % actions to detect the winning condition
proc
  State(fm:Position,f:Position,g:Position,b:Position) = % (farmer,fox,goose,beans)
     ((fm==f && ok(inv(fm),inv(f),g,b)) -> fox(inv(f))    .State(inv(fm),inv(f),g,b))
   + ((fm==g && ok(inv(fm),f,inv(g),b)) -> goose(inv(g))  .State(inv(fm),f,inv(g),b))
   + ((fm==b && ok(inv(fm),f,g,inv(b))) -> beans(inv(b))  .State(inv(fm),f,g,inv(b)))
   + (          ok(inv(fm),f,g,b)       -> farmer(inv(fm)).State(inv(fm),f,g,b))
   + ((fm==right && f==right && g==right && b==right)  -> win.State(left,left,left,left));

   Sys = State(left,left,left,left);

init
  Sys;
```

Draft

2.1. This new specification has a hole in the definition of **ok**, marked with *%% (1) %%*. Extend the given mCRL2 definition by replacing this hole with the code that describes the desired state invariant and save the resulting specification in a new project named `farmer2`. **Show your new definition of the function ok.**

2.2. Without modifying the process `State`, adapt the specification by adding a new process `Counter(n:Nat)` that runs in parallel with `State(left,left,left,left)` and counts the number of traversals made by the boat. Save the resulting specification in a new project `farmer3` and **show your new specification**. (hint: it could be useful to use a bound for the `Counter`), i.e., do not allow n to be bigger than a certain number.)

Modelling a vending machine

Exercise 3. Specify two interacting processes in mCRL2:

- **a vending machine** with 2 products, apples and bananas, costing 1eur and 2eur respectively; and
- **a user** who can insert 1eur or 2eur coins and request for products.

Provide a specification of this system and include them in a `vending.mcrl2` file, according to the requirements below. Try to keep the specifications simple. **Submit this file in your git repository.**

Requirements:

- The user must be able to get apples and bananas;
- The machine accepts up to 3eur, and not more than that;
- The machine must give change back when applicable;
- The machine can be powered off and powered on;

Self-peer-evaluation

Exercise 4. In a scale from 0-5, where 5 is better than 0, give a mark to you and each of your team groups for each of the following criteria:

- **Effort** (time spent)
- **Quality** (of the work produced)
- **Collaboration** (how easy it was to meet and interact)

Send this information individually by e-mail or via Teams to David Pereira and José Proença.

Draft