Assignment 1: Crossing the river – in mCRL2

José Proença

Arquitectura e Cálculo – 2016/2017

To do: Write a report using LaTeX including the answers to the exercises below.

To submit: The report in PDF by email.

Deadline: 23 March 2017 @ 14h (Thursday)

Modelling the wolf-sheep-cabbage problem

Exercise 1. Recall last semester's specification of the wolf-sheep-cabbage problem using SMV.

```
%%%%  SMV specification.  %%%%
VAR
        barqueiro : boolean;
        lobo : boolean;
        ovelha : boolean;
        couve : boolean;
IVAR
        thing : {n,l,o,c};
ASSIGN
        init(barqueiro) := FALSE;
        init(lobo)      := FALSE;
        init(ovelha)    := FALSE;
        init(couve)     := FALSE;
        next(barqueiro) := !barqueiro;
        next(lobo)      := thing = l & barqueiro = lobo   ? !lobo   : lobo;
        next(ovelha)    := thing = o & barqueiro = ovelha ? !ovelha : ovelha;
        next(couve)     := thing = c & barqueiro = couve  ? !couve  : couve;
DEFINE
        bad := (lobo = ovelha & lobo != barqueiro) | (ovelha = couve & couve != barqueiro);
INVAR
        !bad
TRANS
        thing = l -> lobo = barqueiro &
        thing = o -> ovelha = barqueiro &
        thing = c -> couve = barqueiro
```

We will encode the same problem using mCRL2's process algebra. This algebra focus on *actions* rather than *state*, making it less optimal for this particular problem. However, it will help clarifying the key differences between a state-based approach and an action-based approach to model. We start with a simplified (but incomplete) version `barqueiro1.mcrl2` below.

```
%% file: barqueiro1.mcrl2
act
  ld,le,od,oe,cd,ce,                      % acções pelos passageiros
  bld,bod,bcd,barqd,ble,boe,bce,barqe, % acções pelo barqueiro
  lobod,loboe,oveld,ovele,couvd,couve, % acções pelo sistema
  winl,wino,winc,win;                     % acções para detectar vitória

proc
  Lobo = ld.(le+winl).Lobo ;
  Ovel = od.(oe+wino).Ovel ;
  Couv = cd.(ce+winc).Couv ;
  Barq = (bld+bod+bcd+barqd).(ble+boe+bce+barqe).Barq ;

init
  allow(
    { lobod,loboe,oveld,ovele,couvd,couve,barqe,barqd,win },
  comm(
    { ld|bld → lobod, le|ble → loboe,
      od|bod → oveld, oe|boe → ovele,
      cd|bcd → couvd, ce|bce → couve,
      winl|wino|winc|barqe → win
    },
    Lobo || Ovel || Couv || Barq
  ));
```

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. Produce the labelled transition system (LTS) of this mCRL2 specification using (1) `mcrl22lps` to linearise the system and (2) `lps2lts` to produce the final LTS. Finally, visualise the resulting LTS with `ltsgraph` and **show a screenshot of the LTS (make sure it is understandable).**

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

1.3. By omitting the restrictions *allow* and *comm* **would you obtain more or less states** than with the original specification? **Why?**

Exercise 2. We now present a new specification for the same problem consisting of a single process `Estado` that keeps the state information. This new specification includes more advanced features of mCRL2, including: a data structure, actions with data parameters, processes have parameters, and user defined functions **inv** and **ok**.

```
%% file: barqueiro2.mcrl2
sort
  Posicao = struct esq | dir;
map
  inv : Posicao → Posicao ;
  ok  : Posicao # Posicao # Posicao # Posicao → Bool ;
var
  b,l,o,c: Posicao;
```

```
eqn
  inv(esq) = dir ;
  inv(dir) = esq ;
  ok(b,l,o,c) = %% (1) %% ;

act
  lobo,ovel,couv,barq : Posicao; % acções do sistema, parameterisadas na posição
  win;                           % acção para detectar vitória
proc
  Estado(b:Posicao,l:Posicao,o:Posicao,c:Posicao) = % (barqueiro,lobo,ovelha,couve)
     ((b==l && ok(inv(b),inv(l),o,c)) -> lobo(inv(l)).Estado(inv(b),inv(l),o,c))
   + ((b==o && ok(inv(b),l,inv(o),c)) -> ovel(inv(o)).Estado(inv(b),l,inv(o),c))
   + ((b==c && ok(inv(b),l,o,inv(c))) -> couv(inv(c)).Estado(inv(b),l,o,inv(c)))
   + (         ok(inv(b),l,o,c)       -> barq(inv(b)).Estado(inv(b),l,o,c))
   + ((b==dir && l==dir && o==dir && c==dir)   -> win.Estado(esq,esq,esq,esq));
init
  Estado(esq,esq,esq,esq);
```

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 as `barqueiro2.mcrl2`. **Show your new definition of the function ok.**

2.2. Without modifying the process `Estado`, adapt the specification by adding a new process `Contador(n:Nat)` that runs in parallel with `Estado(esq,esq,esq,esq)` and counts the number of traversals made by the boat. Save the resulting specification as `barqueiro3.mcrl2` and **show your new specification**. (hint: it could be useful to use a bound for the `Contador`), i.e., do not allow n to be bigger than a certain number.)

LTS Equivalence

Exercise 3. Recall the action-based `barqueiro1.mcrl2` specification from Exercise 1 and the state-based `barqueiro2.mcrl2` specification from Exercise 2.

3.1. Modify the initial process (***init***) of both `barqueiro1.mcrl2` and `barqueiro2.mcrl2` to hide all allowed actions except `win` (using *hide*), and save them as `barqueiro1-tau.mcrl2` and `barqueiro2-tau.mcrl2`, respectively. In `barqueiro2-taus.mcrl2` redefine the function **ok** by setting it to true, i.e., define `ok(b,l,o,c)=true;`. **Show the resulting *init* block from each file.**

3.2. Generate the `.lts` files corresponding to `barqueiro1-tau.mcrl2` and `barqueiro2-tau.mcrl2`, and compare them using strong bisimulation using the following command. **What can you conclude?**

```
$ ltscompare --equivalence=bisim barqueiro1-taus.lts barqueiro2-taus.lts
```

3.3. Using `ltsconvert`, minimise the LTS for `barqueiro2-taus.mcrl2` with respect to branching bisimulation, using the command below. **Include a screenshot of the minimised LTS and describe what can we conclude from this LTS.**

```
$ ltsconvert --equivalence=branching-bisim barqueiro2-taus.lts
```

Verification of the wolf-sheep-cabbage problem

Exercise 4. Recall the LTSs from Exercise 1 and Exercise 2 (after completing Exercise 2.1. You will now verify properties of these systems. In mCRL2, a property can be written in a text file `prop.mcf`, and verified against a system `system.mcrl2` using the following two commands.

```
$ mcrl22lps system.mcrl2 system.lps
$ lps2pbes  system.lps -f prop.mcf system.pbes
$ pbes2bool system.pbes
```

4.1. What does the property "`[true*]<ready>true`" mean? Does it hold in any of these LTSs?

4.2. What does the property "`[true*.lobo(dir).win]false`" mean? Does it hold in `barqueiro2.lts`?

4.3. Recall that `barqueiro1.lts` is less complete than `barqueiro2.lts`, because it fails to include some important invariants. **Write a property** that exemplifies an invariant that is fails in `barqueiro1.lts` but succeeds in `barqueiro2.lts`. Verify it using the mCRL2 toolset.

4.4. Consider now the extended system `barqueiro3.mcrl2` produced in Exercise 2.2. In this example there is a an extra process called `Contador(n:Nat)`. Using this extra process, **define the following two properties**:

1. It is possible to win after exactly 7 moves.
2. It is not possible to win in less than 7 moves.