mCRL2 exercises – Vending machines

José Proença
Arquitectura e Cálculo – 2015/2016

January 29, 2016

Modelling a vending machine

Exercise 1. We specify a very primitive vending machine and a user in mCRL2 below. After inserting a coin of 50 cents, the user can push the button for an apple.

```
act
  ins50, optA, acc50, putA, coin, ready ;
proc
  User = ins50 . optA . User ;
  Mach = acc50 . putA . Mach ;
init
  allow(
    { coin, ready },
    comm(
      { ins50|acc50 → coin, optA|putA → ready },
      User || Mach
  ) ) ;
```

The specification is split into three sections: ***act***, a declaration of actions of 6 actions, ***proc***, the definition of 2 processes, and ***init***, the initialisation of the system.

1.1. Produce the labelled transition system (LTS) of this specification using (1) `mcrl22lps` to linearise the system and (2) `lps2lts` to produce the final LTS.

1.2. Visualise the previous LTS with `ltsgraph`. **Show a screenshot of LTS (after rearranging as needed.**

1.3. Specify and visualise the LTS of a similar specification, obtained by omitting the restrictions *allow* and *comm*. **How many states do you expect (and count), and why?**

Exercise 2. Next, we add a chocolate bar to the assortment of the vending machine. A chocolate bar costs 1 euro, an apple 50 cents. The machine will now accept coins of 50 cents and 1 euro. The scenarios allowed are (i) insertion of 50 cents and purchasing an apple, (ii) insertion of 50 cent twice or 1 euro once and purchasing a chocolate bar. Additionally, after insertion of money, the user can push the change button, after which the inserted money is returned.

2.1. Extend the following mCRL2 specification to describe the vending machine sketched above, and **save the resulting specification as `vm2.mcrl2`**. The actions that are involved, and a possible specification of the `Mach` process have been given. The machine is required to perform a `prod` action for administration purposes.

```
act
  ins50, ins100, acc50, acc100, coin50, coin100, ret50, ret100 ;
  optA, optC, chg50, chg100, putA, putC, prod,
  readyA, readyC, out50, out100 ;

proc
  User =
    %% 1 %%

  Mach =
    acc50.( putA.prod + acc50.( putC.prod + ret100 ) + ret50 ).Mach +
    acc100.( putA.prod.ret50 + putC.prod + ret100 ).Mach ;

init
  %% 2 %%
```

2.2. Linearise your specification, build the LTS, and visualise the result LTS using `mcrl22lps`, `lps2lts`, and `ltsgraph`. **Show a screenshot of the obtained graph.**

Exercise 3. The same vending machine is now specified below using data parameters to capture different coins and products.

```
sort Coin = struct c50ct | c1eur;
     Product = struct Apple | Chocolate;

act
  ins, acc, coin, ret, chg, out: Coin;
  opt, put, ready: Product;
  prod;

proc
  User =
    ins(c50ct).( opt(Apple) + ins(c50ct).( opt(Chocolate) + chg(c1eur) )
              + chg(c50ct) ).User +
    ins(c1eur).( opt(Apple).chg(c50ct) + opt(Chocolate) + chg(c1eur) ).User ;

  Mach =
    acc(c50ct).( put(Apple).prod + acc(c50ct).( put(Chocolate).prod + ret(c1eur) )
              + ret(c50ct) ).Mach +
    acc(c1eur).( put(Apple).prod.ret(c50ct) + put(Chocolate).prod + ret(c1eur) ).Mach ;

init
  allow(
    { coin, ready, out, prod },
    comm(
      { ins|acc → coin,
        chg|ret → out,
        opt|put → ready },
      User || Mach
  ) ) ;
```

3.1. Modify this specification such that all coins of denomination 2eur, 1eur, 50ct, and 20ct can be inserted. The machine accumulates upto a total of 2.50 eur. If sufficient credit, an apple or chocolate bar is supplied after selection. Money is returned after pressing the change button. **Show this updated specification.**

3.2. Visualise and include a screenshot of the new machine.

LTS Equivalence

Exercise 4. Recall the `vm2.mcrl2` specification from Exercise 2 of a system performing `coin50` and `coin100` actions as well as so-called τ-steps.

4.1. Build a specification `vm2-taus.mcrl2` that is the same as `vm2.mcrl2` after hiding the actions `readyA`, `readyC`, `out50`, `out100`, `prod`. **Show this specification.**

4.2. Using the `ltscompare` tool, compare your model under branching bisimilarity with the LTS of the system `vm2-taus` with the LTS of the system `vm2` after hiding the actions `readyA`, `readyC`, `out50`, `out100`, `prod`. For that use the following command.

```
$ ltscompare --equivalence=branching-bisim \
             --tau=out50,out100,readyA,readyC,prod vm2.lts vm2-taus.lts
```

4.3. Using `ltsconvert`, minimise the LTS for `vm2.mcrl2` with respect to branching bisimulation after hiding the same actions as before (using the *`hide`* operation in mcrl2). **Visualise and include a screenshot of the minimised LTS.** Compare the minimised LTSs and `vm2-taus.lts` visually using `ltsgraph`.

Verification of the vending machines

Exercise 5. Recall the simple LTS from Exercise 1.

5.1. What does the property “`[true*]<ready>true`” mean? Does it hold?

5.2. What does the property “`[true*.ready.!coin]false`” mean? Does it hold? Note: `!coin` represents the complement of `coin`, i.e., any action that is not `coin`.

5.3. Write a property that expresses that, at any moment, it is possible to reach a state where a coin can be inserted.

5.4. Use the mCRL2 toolset to verify if the 3 properties above hold (from Exercises 5.1, 5.2, and 5.3). This involves 2 steps for each property:

1. use “`lps2pbes <lps-file> -f <file-with-formula> <output.pbes>`” to produce a system of boolean equations, and

2. use “`pbes2bool <output.pbes>`” to evaluate if these equations have a solution.

5.5. Adapt the 3 properties (if needed) to the vending machine specified in Exercise 2 and verify them using the mCRL2 toolset. **Show these properties and the result of the verification.**