A2: Analysing behaviour – Part I

David Pereira & José Proença & Eduardo Tovar

RAMDE – 2021/2022

To do

Produce a report as a PDF document including the answers to the exercises below. Continue to work in your git repository – you will need the 3 files produced in your last assignment: `farmer1.mcrl2`, `farmer2.mcrl2` and `farmer3.mcrl`.

What to submit

The PDF report and the new files required by the exercises placed in your group's git repository: `farmer12.mcrl2`, `vending1.mcrl2` and `vending2.mcrl2`. Recall that **all students should push commits**.

Deadline:

9 Jan 2022 @ 23:59 (Monday) – together with Part II (to appear)

Verification of the farmer-fox-goose-beans problem

Recall the specifications in the `farmer1`, `farmer2`, and `farmer3` projects from the modelling exercises (`https://cister-labs.github.io/ramde2122/assignments/a2-modelling.pdf`) You will now verify properties of these systems. In `mcrl2ide`, a property can be written using `Tools>Add Property`. There are 2 types of properties: **Equivalence** and **Mu-Calculus**, covered by this assignment.

LTS Equivalence

Exercise 1. Create variations of the `Sys` processes in `farmer1` and `farmer2` and compare them to the originals as follows.

1.1. Create a new process `SysHide` in both `farmer1` and `farmer2` equal to `Sys` but hiding all allowed actions except `win` (using *`hide`*). **Show the resulting `SysHide` processes for each file.**

1.2. Combine both specifications of `farmer1` and `farmer2` in a single specification `farmer12`. Rename `Sys`/`SysHide` from `farmer1` to `Sys1`/`SysHide1`, and similarly for `Sys`/`SysHide` from `farmer2`. Redefine the function **ok** by setting it to true, i.e., define **ok**`(fm,f,g,b)=true;`.

Visualise the processes `SysHide1` and `SysHide2`. Compare them using strong bisimulation by adding a new **Equivalence** property that compares them. **What can you conclude?**

Verification of properties

Exercise 2. Answer the questions below on the use of mu-calculus for specifying properties in mCRL2.

2.1. What does the property "`[true*]<ready>true`" mean? Does it hold for `farmer1` and for `farmer2`?

2.2. Does the property "`[true*.foxr.win]false`" holds for `farmer1`? Does the equivalent property "`[true*.fox(right).win]false`" holds for `farmer2`? What can you conclude?

2.3. Recall that `farmer1` is less complete than `farmer2`, because it fails to include some important invariants. Write a **single property** for `farmer2` to capture that:

- no bad state is reached, and
- the goal is reached (everyone can cross).

Add this property to your project and **verify** it using mCRL2 toolset. **Reformulate** this property for `farmer1` and add it to that project, and **verify** if it holds.

2.4. Consider now the extended system `farmer3`. In this example there is a an extra process called `Counter(n:Nat)`. **Define the following two properties** over actions of this counter:

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

Modelling a vending machine

Exercise 3. Specify two interacting processes in mCRL2:

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

Provide two variations of this system and include them in files `vending1.mcrl2` and `vending2.mcrl2`, respectively, according to the requirements below. Try to keep the specifications simple. **Submit this file in your git repository.**

3.1. Specify in `vending1.mcrl2` a system such that the properties below hold.

```
[true*.pay2eur.pay2eur] false
[true*.pay2eur.pay1eur] false
[true*.pay2eur]<(!pay1eur && !pay2eur)*.getApple
<true*.pay2eur.true*.getBanana> true
```

Show your specification and **show a screenshot of its LTS**.

3.2. Specify another system in `vending2.mcrl2` such that the properties below hold.

```
<true*.pay2eur.pay2eur> true
<true*.getApple> true
<true*>[true*.getApple] false
```

Show your specification and **show a screenshot of its LTS**.

3.3. Now it is your turn to formalise the requirements. Write a set of requirements using mCRL2's formulas that capture the informal requirements stated below. Do not implement the mCRL2 model. If necessary, include an explanation of the actions and assumptions that you used.

> *I would like the vending machine to sell 3 items: apples, bananas, and chocolates. It should be possible to buy chocolates for 2€ and fruit for 1€. Only 1€ and 2€ coins are accepted. The machine has a maximum capacity for 1€ coins and for 2€ coins. The machine does not accept coins if its capacity is full. The machine should give change back when buying fruit after inserting 2€. If the machine has already 2€ inserted, it refuses another coin. If the machine has no 1€ coins, it cannot not sell fruit with a 2€ coin. The user can request the money back after inserting coins.*

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 as before by email or Teams to David Pereira and José Proença. No justification is needed – e.g., *"Group 3: João: Effort 5, Quality 4, Collaboration 5; Maria: ..."*.