Software Quality Exercise 1

Model Checking

1 Information

1.1 Dates

- Release: 29.02.2016 12.15pm
- Deadline: 09.03.2016 23.59pm
- Discussion: 14.03.2016

1.2 Formalities

While this exercise can be solved and handed in in groups of three, every member of a group must be able to answer questions about the group's solution.

Please package the files containing your solution into a *zip* (or *tar.gz*) file and submit it via email to `charrada@ifi.uzh.ch`. The subject of the email must begin with *[FS 16 SWQ]*. Answer the questions in a *pdf* document and include the source code (*java*, *pml* and *ltl* files) of your solutions. Do not include binary files such as *class* files, metadata (such as *.svn* folders or *.project* files) nor derived files like *pan.c*.

1.3 Daiquiri

To support these exercises, we have created a *trac environment* on *Daiquiri*, one of our internal servers[1]: `http://daiquiri.ifi.uzh.ch/trac/swq16/`. Trac provides a wiki, on which you will find pointers to the documentation of the various tools used in these exercises. Please register on this website before March 9th. As username, choose your UniAccess username (s*xxyyyzz*, where *xxyyyzz* is your student number without the last digit).

[1] To access it, you need to be connected to the university network, either physically (in lab rooms or via wifi) or through a VPN connection.

1.4 SPIN

These exercises are based on *spin*, a model checker. You can find pointers to documentation about this tool on `http://daiquiri.ifi.uzh.ch/trac/swq16/wiki/SPIN`. The tool has been installed on the computers in the lab (room: BINZ 0.B.04 - first row). Here are some helpful commands for these exercises:

- Simulate a model (append `-p -g` for more verbose output)

  ```
  spin model.pml
  ```

- Verify a property on a model.

  ```
  spin -a -F property.ltl model.pml
  gcc -o pan pan.c
  ./pan -a
  ```

- Replay a trace found by the verifier (append `-p -g` for more verbose output).

  ```
  spin -t model.pml
  ```

Note: In some MacOSX and Linux distributions SPIN has problems to process LTL files given with the -F modifier. This issue can be avoided by specifying the LTL formula directly in the command line within double quotes after the -f modifier. For example:

```
spin -a -f " [] greenLight" model.pml
```

The LTL formula can then be saved in an LTL file before submitting the solution.

2 Colony of Chameleons

A friend of yours is the director of a zoo. He has described you the latest acquisition he was considering for his zoo: a rare colony of chameleons. This colony of chameleons includes 20 red, 18 blue, and 16 green individuals. Whenever two chameleons of different colors meet, each changes to the third color. The color mutation is an unique feature and your friend expects to attract many visitor with this colony. Still, the director has a doubt: if, by any chance, all 54 chameleons are in the same color, there will not be any color change any more and the colony would loose its value.

2.1 State Space

a) Give a rough estimation of the number of states in which the colony can be.
b) How many states present the undesired property?

2.2 Promela Model of the Colony

Download the file *Colony.pml.txt* on the website of the exercises and rename it to *Colony.pml*. Study this model with the help of Promela documentation.

a) The behavior of chameleons is not deterministic, in the sense that they meet each other randomly. How is this non-determinism expressed in the Promela model of the colony?
b) Which other aspect of Promela is not deterministic?
c) Explain briefly what are *d_step* blocks and the reason of their presence in the Promela model of the colony. In which sense is the first instruction of such a block different than the others?
d) Despite their relation with the color change phenomenon, the duration of a color change and the probability of a change have not been modeled. Explain, in maximum 5 sentences, why they are not relevant for the problem at hand.

2.3 Simulation and Verification

a) Run a few simulations of the model. Have you ended up in a state where all chameleons are of the same color?
b) Express the property *there are always at least two chameleons of different color* in an LTL formula using the variables defined in the Promela model of the colony. Save this formula in a textfile named `ColorChange.ltl`. Note that you cannot define predicates in an *ltl* file; you have to define them as macros in the Promela model. For example:

```
#define noRedChameleonLeft (!nRed)
```

c) Is it a *safety* or a *liveness* property? Why?
d) Using *spin*, investigate whether the colony can reach a state where this property does not hold. Note that *spin* looks for executions that satisfy an LTL property, so, you have to negate the formula in `ColorChange.ltl`. What will you recommand to your friend?
e) In the report of *pan*, the number of *states stored* is the number of reachable states. How many states were reachable? Compare with your answer in 2.1.1.a).
f) In the report of *pan*, the number of *transitions* is the number of transitions that have been visited during the search. How many transitions were investigated by *pan*?
g) A few days later, the director of the zoo informs you: a green chameleon escaped during the delivery. Has he any reason to worry?
h) What is the depth of the trace (that is, the number of steps) found by *pan*?

2.4 Extending the Model

After a few weeks, you visit the zoo. Looking at this chameleon colony, you feel a strong relief: there are still chameleons of every color and they change colors happily. Still, by observing them, you figure out that your friend did not describe the behavior of the chameleons very precisely. You make the following additional observations:

- When 3 chameleons of different colors meet, they argue violently and kill eachother.
- When 2 chameleons of the same color meet, they give birth to a new chameleon of their color.
- At the day of your visit, there were 20 red, 18 blue, and 16 green individuals.

a) Extend the Promela model of the colony with 2 processes so that it reflects your new observations.
b) Run a few simulations of this model. What happens? Explain.
c) Let's make an assumption here: chameleons only give birth if there are less than Nchameleons of that color in the colony (Initially, set N to 15). Modify the model accordingly.
d) Express the property *soon or later, there will be no chameleon left* in an LTL formula. Save this formula in a textfile named `Extinction.ltl`.
e) Is it a *safety* or a *liveness* property? Why?
f) Using *spin*, investigate whether the colony can die off. Are chameleons threatened with extinction?
g) Modify you model again so that chameleons only give birth if there are in total less than Nchameleons in the colony (initially, set N to 30). Are chameleons still threatened with extinction?
h) Change the the number N to 350 instead of 30. As you did previously, investigate whether the colony can die off. What happens? Why?
i) Set the max search depth to 5'000'000 steps when performing the verification. Do you obtain a different result? What is the depth of the trace? Which depth has been reached during the search?
j) Change the order of process declarations in the Promela model by swapping the *fight* process with the *birth* process and re-verify the model. What is the depth of the trace and which depth has been reached during the verification? Explain the difference with your previous results. Your explanation must account the fact that the behavior of the model remains unchanged by your modification.

2.5 Scalability

This exercise is based on the extended model of the colony built in the exercise 2.1.4. No LTL property will be checked on the model. While keeping the max search depth to 5'000'000 steps, perform an exhaustive search on the model for $N = 60$, $N = 120$, $N = 180$, $N = 220$ and $N = 400$. Note that you have to invoke *pan* without the option `-a`, since no LTL property was specified. For every execution of *pan*, report the following information in a table:

- the time spent for the verification.
- the number of transitions explored.
- the length of the longest trace explorated.
- the number of reachable states.
- the memory used for the verification.

3 Petri Nets

Petri nets can easily be translated into Promela models. For example, the Petri net in figure 1 can be translated in the following Promela code:

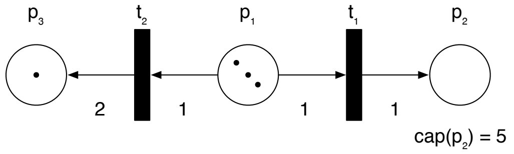

Figure 1: Example of Petri net

```
byte p1, p2, p3; /* assumption: at most 256 tokens on each place */
init
{
  p1 = 3; p2 = 0; p3 = 1; /* initial marking */
  do
  /*t1*/ :: d_step {p1 >= 1 && p2 <= 4; p1 = p1 - 1; p2 = p2 + 1;}
  /*t2*/ :: d_step {p1 >= 1; p1 = p1 - 1; p3 = p3 + 2;}
  od
}
```

Create a Promela model based on the Petri net depicted in figure 2. Using *spin*, verify whether (a) this Petri net is free of deadlock, (b) the transition t_4 can be fired at least once, (c) the transition t_3 can be fired an infinite number of times and whether (d) as soon as p_4 receives a token, it never gets empty again. If applicable, give an execution trace to justify your answer. In each case, explain briefly how you did the verification with *spin*.

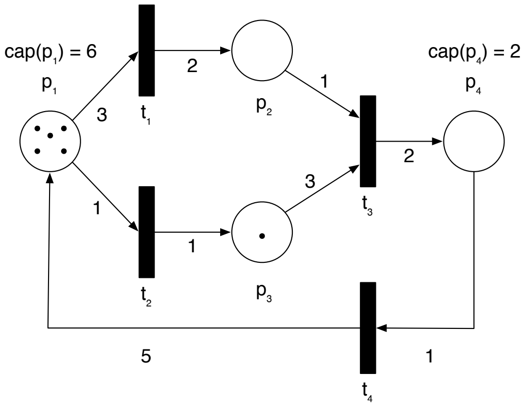

Figure 2: Petri net to be translated in Promela