Assignment 2: Challenges of concurrency

ETH Zurich

1 Safety vs. liveness

1.1 Task

Consider the following properties.

1. What goes up must come down.
2. If two or more processes are waiting to enter their critical sections, at least one succeeds.
3. If an interrupt occurs, then a message is printed.
4. The cost of living never decreases.
5. Two things are certain: death and taxes.
6. You can always tell a Harvard man.

For each of the above properties, state whether it is a safety or liveness property. Identify the bad or good thing of interest.

1.2 Solution

With respect to the order provided in the question, the answers are as follows.

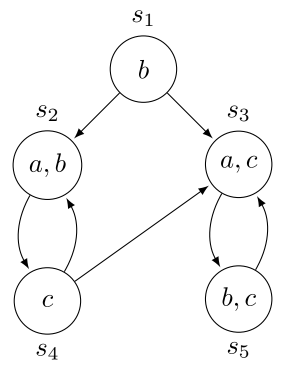
1. This is a liveness property. The good thing to happen is that the thing having gone up is coming down. Another way to think about this is, "is there a finite witness of a violation of this property?". The answer is no, because even if the thing that went up has not come down yet, it is always possible that it will come down eventually.
2. This is a liveness property. The good thing to happen is that at least one processor succeeds.
3. This statement is slightly ambiguous. Depending on how one interprets it, it could be thought of as a safety or liveness property. If the requirement is that a message needs to be printed immediately after an interrupt has occured, then this is a safety property because if the message is not printed immediately, then we already have a finite witness of a violation. However, if the requirement is that eventually a message will be printed, then this is a liveness property because even if it is not printed yet, it may be printed at some later point in time. It is quite likely to run into such ambiguities while stating properties in natural languages. Herein lies the importance of temporal logic, which allows us to state the properties more clearly.
4. This is a safety property. The "bad" thing to happen would be a decrease in the cost of living. If the cost of living decreases at some point, then we have a finite witness of a violation of the property.

5. This is a liveness property. The "good" thing to happen is death and taxes. Although one may not see death or taxes right now, but they may happen at a later point of time.
6. This is a safety property. The "bad" thing that should never happen is one not being able to tell a Harvard man.

2 LTL Models

2.1 Background

Consider the transition system $\mathcal{M}$ over the set of atomic propositions $\{a, b, c\}$:



The following notation is used in the formulae:

G	Globally	F	Future
X	Next	U	Until
$\rightarrow$	Implication	$\wedge$	Conjunction
$\vee$	Disjunction	$\neg$	Negation

2.2 Task

Decide for each of the following formulae ϕ_i in linear-time temporal logic (LTL) whether they are true in state s_1, i.e. whether $\mathcal{M}, s_1 \models \phi_i$ holds. Justify your answers in each case.

1. $\phi_1 = F\, G\, c$
2. $\phi_2 = G\, F\, c$
3. $\phi_3 = G\,(X\, a \rightarrow X\, X\, \neg a)$
4. $\phi_4 = b\, U\, (a \wedge c)\ )$
5. $\phi_5 = G\,(c \wedge X\, c \rightarrow G\, c)$

Translate each of the following properties from natural language into LTL.

6. It is never the case that a, b, and c are true at the same time.
7. Infinitely often, a is immediately followed by c.
8. After c is true, b is never true.

2.3 Solution

1. No. Consider $\pi = s_1, s_2, s_4, s_2, s_4, \ldots$.
2. Yes. On all paths starting in s_1, c is true infinitely often.
3. Yes. On all paths starting in s_1, whenever a is true in the next state, it is false in the one after that.
4. No. Consider $\pi = s_1, s_2, s_4, s_3, s_5, \ldots$.
5. Yes. On all paths starting in s_1, if c is true twice in a row, it is forever true.
6. $\phi_6 = G\,\neg(a \wedge b \wedge c)$
7. $\phi_7 = G\,F\,(a \wedge X\,c)$
8. $\phi_8 = G\,(c \rightarrow X\,G\,\neg b)$

3 Introduction to Spin

3.1 Background

Model checking is a verification technique that, given the model of a system and a correctness property, tells us whether the former satisfies the latter.

Spin [1] is a state-of-the-art model checker which can be used for proving the correctness of concurrent programs. In order to verify a system in Spin, one first needs to model it in a language called Promela (**Pr**ocess **Me**ta **La**nguage). Promela is different from a traditional programming language in the sense that it has less number of constructs which makes it less expressive. While this may not be good for implementing a system, it is very efficient in modeling a system at an abstract level and using a model checker to verify its correctness.

3.2 Installation

The easiest way to install Spin is to download the OS specific precompiled executable from http://spinroot.com/spin/Bin/index.html. The binary has the version number appended to it. Rename the binary to **spin** and put it to your bin folder. If this does not work, you can also find the instructions for compiling the source at http://spinroot.com/spin/Man/README.html. There is a GUI for Spin called iSpin. Before you install iSpin, make sure that you have Tcl/Tk installed (either version 8.4 or 8.5). You can download it from http://www.tcl.tk/. To install iSpin, you need to download the source. It comes together with the source of Spin which you can download at http://spinroot.com/spin/Src/index.html. Go to the subdirectory iSpin and run the script **install.sh**.

3.3 Tutorial

Here is a brief tutorial explaining the basic Spin functions you will need to know in order to do the exercises. I assume that you have installed iSpin as explained above.

- To open a Promela model, select Edit/View from the top left corner of the window and click on open. Browse to the location of your pml file and open it.
- To view the State Machine Diagram of the Promela model, click on Automata View on right of the window. It will generate an automata for every process in your model and you can click on the process name to view it.

- To run a simulation of your model, select Simulate/Replay from the menu bar on the top and click (Re)Run. There are a lot of options on this window but you can leave them to the defaults.
- To verify your model, select Verification from the top menu bar and click on the Run button at the top middle part of the window. In this window, there are several options which let you choose the properties you want to verify. You can leave the options in the Search Mode section to the defaults.

  If the verifier finds an error, an error trail is logged. In such a situation, go back to the Simulate/Replay option and select the Mode as "Guided, with trail". The trail file is generated to indicate at what depth the error occured. Running the simulation with this setting will generate a counter example which helps you to trace down the cause of the error.
- Here is the manual for LTL specifications in Spin. Additionally, also refer to the Spin manual here.
- Note that in Promela, the semicolon is not a statement terminator, but it is a statement separator. As a result, in all models, the last statement in a set of statements does not have any ";". Also note that → is also used to separate two statements. The difference between ";" and → is that the latter is used to indicate a causal relationship among the two separating statements.

3.4 Task

In this exercise, you will verify the Dining Philosopher's problem with Spin. Consider three philosophers sitting around a table with a bowl of rice in the middle. They can either think or eat the rice. There are three forks on the table. A philosopher needs two forks for eating the rice.

- Download the promela model for this system and open it in iSpin.
- Generate the automata for all the processes in the model and run some simulations.
- Verify the model and describe your observations. Was the verification successful? Explain your observations.
- If the verification was not successful, correct the model and verify it. Justify your approach.

3.5 Solution

The original model was not correct. The verification failed. Upon re-running the simulation with the trail, it can be seen that there is a deadlock when each of the philosophers picks up the left fork and keeps waiting for the right one. Refer to the solution code to see how the model can be corrected and verified to be deadlock free. The correct model ensures that before picking up the forks, the philosopher checks whether both of them are available or not. If yes, then she picks them up, first the left then the right, otherwise she waits for both of them to be available. In the original model, this was not the case. She would pick up the left fork even if the right one was not free. This would lead to a situation where each philosopher would end up with the left fork only and the system would not make any progress.

References

[1] http://spinroot.com/spin/whatispin.html