INF5140 Oblig 2, Spring 2006

1 Requirements and Hand-in

This assignment[1] should be done in groups of two students. Each group should submit one report documenting their model and results. The report should be sent by mail (to `gerardo@ifi.uio.no`) by May 9th 2006 as a *tar* file with your name as directory (e.g., if the files are in `gerardo/` type `tar -cvf gerardo.tar gerardo`). The students should also present their solution (in a PC) on Wednesday May 10th during the normal course hours (12:15 till 15:00).

The report should contain:

- Name and e-mail address of each of the participants in the group.
- A description of the Promela model, i.e. what abstractions have been made, which parts of the specification are included in the model and which are not, together with motivations. Explain also how these decisions have influenced the validation results.
- Answers and results of the exercises. Each should also include the parameter settings of the spin validations. *Note that the explanations are as important as the results of running the tool.*
- The Promela model.

Remark: The necessary theory for solving Exercise 2 will be given during the lecture on Correctness Claims (26.04.06). We recommend you start solving Exercise 1 since it will take some time to understand the model. The 19.04.2006 there will not be a "normal" lecture; the course hours will be dedicated to answer questions regarding the assignment.

2 Exercise 1

2.1 Introduction

It is often important to verify properties of a system based on a specification before the system is actually implemented. This can be done with an

[1] The first exercise is a small variation of an assignment given at a Formal Methods course at KTH (Sweden). Do not worry, you cannot find the solution in Internet.

abstract model that captures the relevant parts of the specification, and excludes those parts that are not relevant for verifying the desired properties. The verification can be done in various ways. You should use Promela and Spin (eventually XSpin) to verify some properties of the Mobile-IP protocol. The corresponding IETF Proposed Standard Protocol can be downloaded from `ftp://ftp.rfc-editor.org/in-notes/rfc3220.txt`.

2.2 Getting Started

1. Browse through the description of the protocol, so that you get a rough understanding of it. Start by reading the Protocol Overview at section 1.7, refer to the previous pages for the terminology used. When reading the rest of the protocol description, pay special attention to sections 3.3 and 3.4.
2. Walk through the draft Promela model that is enclosed in the assignment and try to understand how it works. Identify what parts of the specification is expressed in the Promela model.

2.3 Exercises

1. (a) Specify the following properties in LTL. Motivate your formalizations.
      i. Always, the mobile node eventually talks
      ii. Always, the mobile node eventually talks or moves
      iii. Always, the mobile node eventually talks or moves or timesout
   (b) Verify the above properties. Explain your results.
2. (a) Extend the Promela model so that a second mobile node (with the second agent as home agent) is running.
   (b) Verify properties 1-a-i to 1-a-iii and explain your result.
   (c) Verify the same properties with *weak fairness* and explain your result.
3. Comment out the second mobile node, so that only one mobile node is running in the Promela model.

(a) Extend home agents, so that after granting a request to their mobile node, they store (remember) at what care of agent the mobile node is registered.

(b) Specify the following property in LTL and motivate your formalization: “Always, when the mobile node talks, it is registered correctly”.

- Explanation: "Registered correctly" means that the mobile node is registered (according to the home agent) with the care of agent she "thinks" she is.
- Hint: "Registered correctly" can be defined as an agreement between a variable in the mobile node and a variable in its home agent.

(c) Verify property 3-b and explain your results.

2.4 Hints

- If verification is slow, try verification options that yield a faster verification; *weak fairness* slows verification down, while *partial order reduction* and *compression* speeds it up.
- When you are making extensions to the model, you might want to enclose your new constructions in `# if ... # endif` in order to do verification on different versions of the model. These compiler directives serve to include or exclude parts of the model in a particular validation or simulation.
- When writing properties in LTL, a useful predefined function of Spin is `procname[pid]@label` which returns true at a state only if the next statement that can be executed in the process with instantiation number `pid` is the statement that comes after the label `label` in `procname`. (It is an error if the process referred to with `pid` is not an instantiation of `procname`). To illustrate how this function can be used in expressing properties we give the following example. Property: “The mobile node will eventually move”. Expression of Property in LTL:

```
#define moves MobileNode[3]@Movement
```

$\Diamond$ *moves*

Note that `pid` is the unique process instantiation number of a process at each running. The `pid`s start with 0 and are assigned in order of creation. The single process of type `MobileNode` in the draft model is the forth created process, hence its `pid` is 3.

– When writing properties in LTL one can refer to the current value of a local variable in a similar way. The function `procname[pid]:var` refers to the current value of the local variable var in the process with instantiation number `pid`. (Again, it is an error if the process referred to with `pid` is not an instantiation of `procname`). We illustrate with an example. Property: "Eventually the value of `currentLink` in the mobile node is equal to the value of link in the first agent". Expression of Property in LTL:

```
#define areEqual MobileNode[3]:currentLink == Agent[1]:link
```

$\diamond$ *areEqual*

3 Exercise 2

1. Given the following model:

```
byte x = 2;

active proctype A()
{
          do
          :: x = 3 - x
          od
}

active proctype B()
{
progress: do
          :: x = 3 - x
          od
}
```

   (a) Are there non-progress cycles? Justify your answer and give a counter-example if it is not the case.

(b) Let the following be a slightly different version of the above model:

```
byte x = 2;

active proctype A()
{
          do
          :: x = 3 - x
          od
}

active proctype B()
{
          do
          :: x = 3 - x; progress: skip
          od
}
```

Are there non-progress cycles? Justify your answer and give a counter-example if it is not the case. Explain the difference with the previous model.