CVFP (Software Design and Formal Verification)
TP 1 : Let's play with SPIN

SPIN (Simple PROMELA Interpreter; `http://spinroot.com/`) is one of the most used model checker. This tutorial aims giving it an introduction.

Installation There exists some graphical interface for SPIN, but for the simplicity of the tutorial we'll stick to a command line version. Download and install SPIN using the link `http://spinroot.com/spin/Man/README.html`.

Exercise 1 `Playing with SPIN and PROMELA`

SPIN takes as input a program writen in PROMELA (Process Meta Language). It's a C-like language including parallelism and LTL formulae. You can get a documentation of this language there[1]: `http://spinroot.com/spin/Man/Quick.html`.

Example Here is a small (patched) example taken from the documentation[2].

```
bool turn, flag[2];
byte nbcriticalSection = 0;

proctype user() {
    assert(_pid == 0 || _pid == 1); // _pid is the identifier of the current thread.

    do :: (1) ->
        atomic {
            flag[_pid] = 1; turn = _pid; };

        // This waits until the given formula is satisfied.
        (flag[1 - _pid] == 0 || turn == 1 - _pid);

        nbcriticalSection++;
        // Dangerous part!  There shall only be one thread there at any time.
        nbcriticalSection--;

        flag[_pid] = 0;
    od;
}
```

[1] Or you can call me, but you might not be alone needing help. ⌣̈

[2] In two pages!

```
init {
    run user(); run user()
}

ltl alwaysone { [] (ncrit <= 1) }
```

1.1. What does the example do?

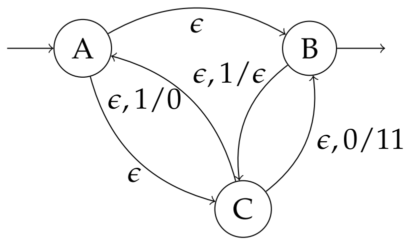
This example defines a formula `alwaysone` we can ask SPIN to check. To do so run `spin -a your\ file.pml`. This generates a C file `pan.c` you can compile with `cc -o pan pan.c`. You than can execute it by `./pan -N alwaysone`.

1.2. Run the example.

Then remove the waiting formula `(flag[1 - _pid] == 0 || turn == 1 - _pid)` of the program, rerun SPIN and compare the results.

In case of errors, SPIN generates a `your\ file.pml.trail` file corresponding to a failing evaluation. To understand this particular execution, you can run `spin -psrvlg -k your\ file.pml.trail your\ file.pml`.

1.3. Write a (non-deterministic) program emulating the following stack automaton:



In practise, every PROMELA model is finite. We'll ignore the problem by using a `byte` to store the stack.

1.4. Actually, SPIN translates formulae into automata. You can for instance play with `spin -f '[]<>!p'` (or with any bigger formula) to see what's inside SPIN.

You can also visualise the control flow computed by SPIN using the following command: `./pan -D | dot -Tps -o pan.ps` and openning the resulting `pan.ps` file.

Exercise 2 `The Dinning Philosophers.`
The goal of this exercice is to make some experiments with the dinning philosophers problem. You should find a file at `http://people.irisa.fr/Martin.Bodin/instruado/2013/CVFP/philosophers.pml`.

2.1. Complete the file and run SPIN on it. Fix the problem so that every infinitely waiting phisolopher eats infinitely often.