Verification of Cyber-Physical System
Fall 2017

Exercice Sheet 1

Author : Sylvain Julmy

Professor : Ultes-Nitsche Ulrich
Assistant : Prisca Dotti

Exercice 1

The following *Promela* model contains two processes and a global variable (named **global**) that is initialised to 0.

```
byte global = 0;

/*
 * Indefinitely increase the "global" variable if it is lower than 255
 */
active proctype incProcess() {
    repeat :
    // If the assertion fail, it indicates that the "global" variable have reaches 255
    assert (global != 255);
    global < 255;
    global++;
    goto repeat;
}

/*
 * Indefinitely decrease the "global" variable if it is greater than 0
 */
active proctype decProcess() {
    repeat :
    // If the assertion fail, it indicates that the "global" variable have reaches 255
    assert (global != 255);
    global > 0;
    global--;
    goto repeat;
}
```

The first process increases the variable by 1 if it is lower than 255 and the second decrease the variable by 1 if it is greater than 0. Both processes are running indefinitely.

In order to use the *Spin* verifier to find out if the variable can reach the value 255, we use the assertion `assert(global != 255)`. We check the **assertion violations** property of *Spin* to

find out if the **global** variable reaches 255 or not. In our case, *Spin* find that the assertion is violated, so we know that the **global** variable can reach 255.

Exercice 2

The following *Promela* model is equivalent to the one given for the exercise 2, except it does not use `if`,`->` or `goto` statements :

```
mtype = {P,C};

mtype turn = P;

active proctype producer() {
    /* do-notation : defines a loop */

    // Guards : wait until (turn == P) because it is the only available statement
    do :: turn == P ;
        printf("Produce\n");
        turn = C;
    od
}

active proctype consumer() {
    /* do-notation : defines a loop */

    // Guards : wait until (turn == C) because it is the only available statement
    do :: turn == C;
        printf("Consume\n");
        turn = P;
    od
}
```

Exercice 3

When looking for non-progress cycles :

Does Spin detect an error when using the *week fairness* constraint ? No

Does Spin detect an error when is not using the *week fairness* constraint ? No
It looks like that Spin does not detect any non-progress cycles with the given code :

```
byte x = 2;

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

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

By the way, if we remove one progress label (either from proc A or proc B), Spin does detect a non-progress cycle when not using *week fairness* constraint:

```
byte x = 2;

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

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

In order to prove the fairness for both process, we need to use an additional process called *monitor* which is going to monitor both process A and B using the `progress` : label of Spin. Both processes will use a flag in order to indicates that they have fullfiled their job and are waiting. The monitor process wait until both A and B process have completed their work to launch them again, so we have proved the fairness for both processes.

Examples using a *monitor* :

```
byte x = 2;
byte checkProgress[4];

active proctype A(){
       do ::
           checkProgress[0]==0;
           x = 3 - x;
           checkProgress[0]++;
       od
}

active proctype B(){
       do ::
           checkProgress[1]==0;
           x = 3 - x;
           checkProgress[1]++;
       od
}

active proctype monitor(){
       repeat :
       checkProgress[0] + checkProgress[1] == 2;
       progress :
       checkProgress[0] = 0;
       checkProgress[1] = 0;
       goto repeat;
}
```