Modélisation et vérification de protocoles en utilisant SPIN

TD 4

Ce document est disponible à l'URL `http://www.liafa.jussieu.fr/~sighirea/logver/td_spin.html`

1 Description générale

SPIN est un outil pour la vérification et la simulation des systèmes concurrents, en particulier les protocoles de communication.

Pour être étudié, un système est d'abord décrit en PROMELA (*Protocol Meta Language*), le langage de modélisation de SPIN. Promela est un langage impératif qui se ressemble au langage C agrémenté de quelques primitives de communication. Pour communiquer, les processus peuvent utiliser des canaux de communication *fifo*, prendre rendez-vous ou utiliser des variables partagées.

SPIN comporte essentiellement deux modes :

- *Simulation* : le système est exécuté pas à pas, ce qui permet de se familiariser avec son comportement.
- *Vérification* : les états du système sont explorés exhaustivement pour vérifier que le système satisfait bien certaines propriétés exprimées, par exemple, en LTL.

2 Promela

Un programme PROMELA est une liste de déclarations de *processus*, de *canaux* et des *variables*.

Déclaration de variables : On indique le type (un des `bit`, `byte`, `short`, or `int`), le nom de la variable et optionnellement sa valeur initiale.

```
bool b1 = false, b2 = false;
bit k = 0;
```

Les variables de type tableau sont déclarées comme en C, par exemple :

```
bit porteouverte[3];
```

Déclaration de canaux : On l'introduit par le mot clé **chan**, suivi du nom du canal et optionnellement de la longueur du *fifo* et du type des messages qui circulent. Par exemple :

```
chan Ouvreporte=[0] of {byte, bit},
     Transfert=[2] of {bit, short, chan};
```

`ouvreporte` est un canal *synchrone*, car sa longueur est 0, ce qui correspond à un rendez-vous ; sur ce canal circulent des messages ayant une partie `byte` et une partie `bit`. `Transfert` est un canal *asynchrone*, car il peut stocker (au plus) deux messages.

Déclaration de processus : La forme la plus simple de déclaration de processus est :

```
proctype nom ( paramètres_formels )
{ instructions }
```

Un processus est instancié en utilisant l'instruction `run` :

```
run nom ( paramètres_actuels )
```

Par exemple :

```
proctype P (bit i) {
 ...
}
proctype porte (byte i) {
 ...
}
proctype ascenseur () {
 ...
}
```

Opérations sur les canaux : Sur un canal on peut envoyer (opération "`!`") ou recevoir (opération "`?`") des messages. Par exemple :

```
ouvreporte!i,0;
ouvreporte?i,1;
ouvreporte?eval(etage),1
```

La fonction "`eval`" force l'égalité des valeurs reçues avec `etage`, la variable `etage` n'est pas changée.

Expressions : Un expression peut être utilisée comme une instruction si elle ne fait pas des effet de bord (opérations "`--`" et "`++`" de C). Dans ce cas, elle est exécutable quand sa valeur devient vraie (par le changement des valeurs des variables partagées). Par exemple :

```
(a == b);
```

est équivalent à :

```
while (a != b) skip;
```

Instruction "`init`" : L'exécution du système commence par le processus `init` (c'est le `main` de PROMELA). Par exemple :

```
init {
    run porte(1); run porte(2); run porte(3);
    run ascenseur()
}
```

Instruction "`atomic`" : L'exécution d'une séquence d'instructions préfixée par "`atomic`" peut être rendue indivisible, c'est-à-dire sans l'entrelacement des actions des autres processus. Par exemple :

```
    atomic {
        run porte(1); run porte(2); run porte(3);
        run ascenseur()
    }
```

Instruction "`if`" : Un branche de l'instruction "`if`" est exécutable si la première instruction de la branche, appelée aussi sa garde, est vraie. L'instruction "`if`" bloque jusque à ce qu'une branche devient exécutable, dans quel cas la branche est exécutée. Si plusieurs branches sont exécutables, l'une est choisie aléatoirement (*non-déterminisme*).

L'exemple suivant incrémente ou décrémente la valeur de `count` une fois.

```
if
:: count = count + 1
:: count = count - 1
fi
```

Instruction "`do`" : Similaire à l'instruction "`if`", sauf que l'instruction est exécutée et puis la sélection est répétée jusque à l'exécution d'une instruction "`break`".

```
proctype ascenseur () {
    show byte etage = 1;
    do
    :: (etage != 3) -> etage++
    :: (etage != 1) -> etage--
    :: ouvreporte!etage,1; ouvreporte?eval(etage),0
    od
}
```

3 Exercices

Exercice 1 : Les exemples donnés dans la section 2 modéliser le système d'un ascenseur pour trois étages.

- Complétez la modélisation du processus "`porte`".
- (*en TP*) Simulez la spécification complète du système.
- Ecrivez en LTL la propriété "p_1 : *si la porte du première étage est ouverte, elle sera fermée l'état suivant*".
- Ecrivez en LTL la propriété "p_2 : *inévitablement, une des trois portes sera ouverte*".
- (*en TP*) Vérifiez les deux propriétés en utilisant SPIN.

Exercice 2 : Modélisez en PROMELA et vérifiez avec SPIN la propriété d'exclusion mutuelle pour l'algorithme suivant :

vars b_1, b_2 : **bool** := **false**, k : 1..2 := 2

process P_1 **is**	||	**process** P_2 **is**
while true do begin		**while true do begin**
l'_1: **noncritical section**		l''_1: **noncritical section**
l'_2: b_1 := **true**;		l''_2: b_2 := **true**;
l'_3: **while** k != 1 **do begin**		l''_3: **while** k != 2 **do begin**
l'_4: **while** b_2 **do skip**;		l''_4: **while** b_1 **do skip**;
l'_5: k := 1;		l''_5: k := 2;
end;		**end**;
l'_6: **critical section**		l''_6: **critical section**
l'_7: b_1 := **false**;		l''_7: b_2 := **false**;
end;		**end**;

Exercice 3 : L'algorithme d'exclusion mutuelle suivant est connu sur le nom de l'algorithme de Peterson :

```
vars y1, y2 : bool := false, s : 1..2 := 1

process P1 is                             ||   process P2 is
  repeat forever begin                           repeat forever begin
    l'1: noncritical section                       l''1: noncritical section
    l'2: (y1, s) := (true, 1);                     l''2: (y2, s) := (true, 2);
    l'3: await ¬y2 + s!=1;                         l''3: await ¬y1 + s!=2;
    l'4: critical section                          l''4: critical section
    l'5: y1 := false;                              l''5: y2 := false;
  end                                            end
```

Modélisez cet algorithme en Promela et vérifiez avec Spin la propriété d'exclusion mutuelle.