Théorie et pratique de la concurrence – Master 1 II
TP 1 : Spin et Promela

Spin est un outil pour la vérification et la simulation des systèmes concurrents finis (`http://spinroot com`).

Pour être étudié, un système concurrent est d'abord décrit en Promela (**Protocol Meta Language**, voir `http://spinroot com/spin/Man/promela html`), le langage de modélisation de Spin. Promela est un langage impératif qui se ressemble au langage C agrémenté de quelques instructions pour la concurrence.

Le modèle Promela peut ensuite être analysé avec Spin par :

- **Simulation** : le modèle est exécuté pas à pas, ce qui permet de se familiariser avec son comportement.
- **Vérification** : les états du modèle sont explorés exhaustivement pour vérifier que le modèle satisfait des propriétés (par exemple, exclusion mutuelle) spécifiés en LTL (**Linear Temporal Logic**).

Pour une introduction rapide à Spin et son interface graphique XSpin, voir `http://spinroot com/spin/Man/GettingStarted html`

Promela

Un programme Promela est une liste de déclarations de **types**, **constantes**, **variables** (globales) et **processus**, suivie d'un point d'entrée du programme (`init`).

Déclaration de types : Les types de base de Promela sont : `bit` ou `bool`, `byte`, `short` et `int` ; ces types n'ont pas besoin d'être déclarés. Le seul type que l'utilisateur peut être déclaré est le type énumérée `mtype` pour lequel on peut spécifier ses valeurs (constantes symboliques).

Exemple : `mtype = { ack, nak, err };`

Déclaration de constantes : Promela permet les macro-définition à la C. Il est donc possible de déclarer des constantes comme en C avec `#define`, mais aussi des macros plus compliquées.

Déclaration de variables globales : Les variables scalaires ou tableau de Promela sont déclarées comme en C : on indique le type, le nom de la variable et optionnellement sa valeur initiale.

```
bool b  = false, b2 = false;
bit k = 1;
bit porteouverte[2];
```

Une classe spéciale de variables sont les **canaux**, qui représentent des queues FIFO d'une longueur constante et dont les éléments (appelés messages) sont des tuples (typés) de valeurs scalaires ou vecteur de Promela. Les canaux sont déclarés en utilisant le mot clé `chan` suivi du nom du canal et optionnellement de sa longueur et du type des messages qui circulent. Par exemple :

```
chan Ouvreporte[1] = of {byte, bit},
     Transfert[1] = 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 }
```

où les paramètres formels sont déclarés comme en C mais séparés par des point-virgules.

Un processus peut être lancé de deux manière. La **manière implicite**, qui peut s'appliquer que

Spin et XSpin

Nous utiliserons l'interface graphique de Spin, appelée XSpin. L'appel est fait en ligne de commande :

```
> xspin &
```

XSpin comporte un (mauvais) éditeur de Promela, ainsi qu'une interface graphique aisée par le menu `Run` avec les multiples fonctions de l'outil Spin.

Pour la simulation : `Run` → `Set Simulation Parameters` .

Pour la vérification : `Run` → `LTL Property Manager` .

Propriétés LTL : Les opérateurs LTL disponibles sont ceux listés dans l'interface : les opérateurs booléens (l'implication est notée →), opérateur globalement (notation `[]`), l'opérateur dans le futur (notation `<>`) et l'opérateur **until** (notation `U`). **Attention, l'opérateur next n'est pas à utiliser pour avoir des bonnes performances !**

Les opérandes doivent être des propositions atomiques. Pour définir des propositions atomiques, utilisez des macros C dans la section **Symbols Definition**. Vous pouvez ici faire référence aux variables globales du modèle, aux variables locales des processus (en préfixant par le nom du processus, par exemple `porte[1]@ etage`), ou à une étiquette dans le corps d'un processus (par exemple `porte[1]@ l`).

Vous pouvez/devez sauvegarder les formules écrites dans des fichiers à extension `ltl` (c'est pas le même fichier que la spécification Promela, dont l'extension consacrée est `pml`).

Exercice 1: Soit l'algorithme suivant pour l'exclusion mutuelle :

```
int turn;

P :
Loop forever:
p : section NC
p : await turn==
p : section C
p : turn:=

P :
Loop forever:
q : section NC
q : await turn==
q : section C
q : turn:=
```

- Codez cet algorithme en Promela en utilisant des étiquettes pour les sections non-critiques et critiques.
- Simulez votre modèle.
- Ecrivez en LTL la propriété "**au plus un processus se trouve dans la section critique**".
- Vérifiez cette propriété sur votre modèle.

Exercice 2:

Codez l'algorithme de Dekker vu en cours et démontrez l'absence de famine (regarder ce qui se passe lorsque vous cochez ou pas la case **with weak fainess** lors de la vérification de la formule LTL).