Concepts et Model Checking – TP 2

Pour ce TP, nous allons travailler avec SPIN, un outil qui sait vérifier la logique de LTL sur des modèles spécifiés dans un langage qui s'appelle PROMELA. La page web de Spin de trouve à `https://spinroot.com`.

Installation

1. Télécharger l'archive qui contient Spin dans la page web du cours et l'extraire dans un nouveau dossier (disons `spincmc`). Cet archive contient le code source de spin et quelques fichiers qu'on utilisera pour les exercices. Ouvrir un terminal et aller dans ce dossier `spincmc`.
2. Obtenir l'exécutable `spin` :
   — Si GCC est installé sur votre machine, utiliser `make` pour compiler. Vous devriez obtenir un exécutable avec le nom de `spin`.
   — Il est également possible de télécharger un exécutable précompilé pour certains plateformes depuis le github de Spin : `https://github.com/nimble-code/Spin/tree/master/Bin` Si vous utilisez un tel exécutable, le renommer `spin`.

Premiers pas

Spin sait traduire les formules de LTL en des automates de Büchi. La syntaxe utilise `[]` pour **G**, `<>` pour **F** et `&&`, `||`, `!` pour conjonction, disjonction, négation. P.ex. pour obtenir un automate pour $\mathbf{G}(p \to \mathbf{F}\, q)$, utiliser le suivant dans la ligne de commande (il est recommandé d'inclure la formule entre guillemets) :

```
spin -f '[](p -> <>q)'
```

Les automates sont représentés commes des programmes simples qui utilisent des `goto` pour aller d'un état à l'autre. Les états acceptants commencent avec `accept`. Un état spécial `accept_all` accepte n'importe quelle séquence.

1. Dessiner l'automate produit par Spin. Correspond-il à vos attentes ?
2. Idem pour l'une des formules du TD précédent : $p\, \mathbf{U}\, (q\, \mathbf{U}\, p)$

Verification avec Spin

On va utiliser l'algorithme de Dekker, une variante de celui de Peterson, qui assure l'exclusion mutuelle entre deux processus (voir Wikipédia pour une explication détaillé de l'algorithme). Une spécification de l'algorithme se trouve dans `dekker.pml`. Pourtant, une erreur s'est glissée dans cette réalisation.

Dans le dossier `spincmc` il y a deux scripts pour travailler avec Spin. Ils s'appellent `spinLTL` et `spinFairLTL`, et ils prennent le nom d'un modèle et une formule, p.ex :

```
./spinLTL dekker.pml '[]!(crit0 && crit1)'
```

Si la formule tient dans toutes les exécutions, vous aurez un message du genre `no errors found`. Sinon, vous aurez une exécution dans laquelle la formule ne tient pas. La version `spinFairLTL` ne considère que des exécutions où tout processus progresse infiniment souvent.

1. Utiliser Spin pour tester s'il est possible pour les deux processus d'atteindre leur section critique en même temps. Si c'est le cas, trouver l'erreur en suivant l'exécution fautive et corriger le modèle.
2. Utiliser Spin pour tester si le processus `p0` parvient à entrer dans sa zone critique s'il l'essaye (une fois que `flag0` est vrai, `crit0` doit finalement l'être aussi). Remarquez-vous une différence entre `spinLTL` et `spinFairLTL` ?

Algorithme de Peterson

L'algorithme de Peterson a été présenté dans le cours.

1. Créer un nouveau modèle Promela qui réalise l'algorithme de Peterson.
2. Tester si votre modèle satisfait bien les deux propriétés exigées de l'algo de Dekker.