Concurrence – Master 1
TP 4
Programmation et modélisation d'un ramasse-miettes parallèle

Exercice 1 : Cet exercice a pour but de vous faire modéliser en Promela le ramasse-miettes (GC) parallèle vu en cours.

Indications :

1. Pour vous entraîner avec Spin, consulter les exemples donnés sur le site Web du cours.
2. Écrire la première version du GC vue en cours (avec deux couleurs) :
   (a) Le Collector et le Mutator sont deux processus actifs de Promela.
   (b) L'initialisation des variables globales doit être faite avant que le Mutator et la boucle du Collector tournent ; pour cela, vous pouvez mettre cette initialisation dans le Collector et synchroniser sa fin avec le début du Mutator.
   (c) Le Mutator doit faire une boucle où il applique les opérations *add*, *del* et *new* sur les parties gauche ou droites des cellules ; prenez différents scénarios pour son activité.

   ATTENTION : la sémantique des boucles **do** en Promela est autre que celle des boucles **do** vue en cours.

   Vous pouvez ensuite simuler cette version en utilisant la simulation pas à pas de Spin.

   Pour vérifier une propriété invariant I avec Spin, il faut écrire la formule :

   ```
   [] I
   ```

   avec I défini comme macros C dans la partie "*Symbols defintions*" du "*LTL Property Manager*" de Spin. Par exemple, pour la propriété FREE, on doit écrire :

   ```
   [] FREE
   #define FREE (left[free] != endfree)
   ```

   Pour vérifier GC, il faut écrire un processus qui calcule les accessibles à partir du `root` et qui affecte une variable globale `gc` à 0 si `endfree` devient accessible. La propriété à vérifier devient :

   ```
   [] GC
   #define GC (gc == 1)
   ```

   Vérifier que cette propriété n'est pas vraie pour cette implantation.
3. Modifier la version précédente pour arriver à une version correcte du GC. Essayer d'éliminer le plus de sections critiques pour augmenter le parallélisme du système.

Exercice 2 : Cet exercice a pour but de vous faire programmer en Java le ramasse-miettes (GC) parallèle vu en cours.

Indications :

1. Écrire une classe pour représenter les variables globales. Un objet de cette classe sera crée et modifié (prévoir des méthodes pour cela) par le Mutator et le Collector. Prévoir des méthodes d'affichage et de calcul des accessibles afin de vous faciliter le debougage.
2. Écrire une classe Mutator qui implémente l'interface `Runnable` (la méthode `run`) dont le comportement est constitué de différents scénarios de manipulation de la mémoire.
3. Écrire une classe Collector qui implémente l'interface `Runnable` (la méthode `run`) dont le comportement est celui du Collector.
4. Les sections critiques en Java sont les méthodes déclarées `synchronized`.
5. Écrire une classe contenant le méthode `main` qui :
   - crée l'objet pour la mémoire globale, m
   - crée les deux objets représentant le code du Mutator et du Collector sur la mémoire globale m (référence passée en paramètre des constructeurs)
   - crée deux threads représentant les deux processus et les active.