TME 8 (CPS) : Modélisation de la concurrence

Exercice 1 : Feu de signalisation

Question 1.1.

On souhaite modéliser (dans un fichier `feu.pml`) un processus Promela décrivant le comportement d'un feu de signalisation. Le comportement attendu est le suivant :

- avant initialisation, la couleur du feu est indéterminée
- initialement, le feu est en mode initialisation avec une couleur orange clignotante.
- après initialisation, le feu est actif et de couleur Rouge
- le feu peut changer de couleur, selon les règles de circulation française
- à tout moment, le feu peut tomber en panne, il est alors orange clignotant et ne peut sortir du mode panne.

Dans le modèle Promela, on souhaite étudier tous les franchissements possibles, et pour cela tous les changements d'états seront purement non-déterministes. En particulier on souhaite que le modèle soit *sans deadlock* et sans notion de temps explicite.

Pour vérifier ce fait, on utilisera les commandes suivante :

- `spin -a feu.pml` pour générer un analyseur
- `gcc -DSAFETY -O2 pan.c -o pan` pour compiler l'analyseur généré
- `./pan` pour vérifier les erreurs de sûreté éventuelles, en particulier la présence ou non d'un deadlock.

(*Remarque* : le `-DSAFETY` optimise le générateur pour ne détecter que des problèmes de sûreté comme les *deadlocks* ou les assertions fausses).

On pourra afficher l'automate non-déterministe du processus avec les commandes suivantes :

- `./pan -D > feu.dot` génère le graphe de l'automate au format 'dot"
- `dot -Tpdf feu.dot -o feu.dot.pdf` génère le PDF de l'automate.

Question 1.2.

On souhaite vérifier sous la forme d'assertions que le modèle `Promela` du feu vérifie les propriétés suivantes :

- le feu ne peut pas passer du rouge à l'orange
- le feu ne peut pas passer de l'orange au vert
- le feu ne peut pas passer du vert au rouge
- le feu ne peut pas être clignotant et rouge ou vert

Pour cela, on définit un process d'observation qui doit détecter *depuis l'extérieur* si les séquences sont effectuées. Pour signaler une erreur une simple assertion fausse suffit. Lors des changements de couleur, le feu devra émettre un message à l'observateur qui fera alors ses vérifications (attention à l'atomicité).

Donner une version avec observateur du feu de signalisation dans une fichier `feu_safe.pml`.

Si le vérificateur `pan` trouve une erreur, on peut rejouer le chemin d'exécution qui conduit à l'erreur avec la commande :

```
spin -t -p feu_safe.pml
```

Question : l'erreur provient-elle du feu ? de l'observateur ?

Exercice 2 : Exclusion mutuelle

Les fichiers `mutex1.c` et `mutex2.c` sur le site de l'UE sont des "solutions" proposées pour le problème de l'exclusion mutuelle *sans* utilisation de primitive spécifique.

L'objectif de cet exercice est de modéliser les deux propositions en `Promela` (fichiers `mutex1.pml` et `mutex2.pml`) et de vérifier si le problème d'exclusion mutuelle est bien résolu.

Voici quelques conseils dans le passage du code `C` (compilable et exécutable) au modèle `Promela`.

- les temps d'attent (`usleep`) ne sont pas modélisés
- les boucles de répétitions (placées pour les tests) deviennent des "boucles" réactives (sans terminaison). Donc la présence d'un *deadlock* correspond à une erreur.
- l'exclusion mutuelle se vérifie par une assertion portant sur un compteur (cf. assertions du code C).

Pour "simuler" le `if (<cond>) <then> else <else>` du `C` on pourra écrire en Promela :

```
if
::<cond> -> <then>
::else  -> <else>
fi
```

Une boucle d'attente active :

```
while (<expr>) {
   /* attente active */
}
```

S'écrit tout simplement en `Promela` :

```
<expr>  ;
```

Car les expressions booléenne sont bloquantes si elles sont en position instruction et qu'elles sont fausses.