Projet NuSMV
Gestion de feu de circulation

Chloé Bensoussan

Universite Nice Sophia-Antipolis
Master 1 Informatique

1 Simple traffic light

Dans cet exercice, il faut gérer le feu de circulation des voitures en fonction du temps. En effet, ce feu change de couleur au bout d'un temps prédéfini : 5 temps au feu rouge, 2 temps au feu jaune puis 10 temps au feu vert.

J'ai implémenté cette fonction en créant un module `counter_tick(time, activate)` qui prend en paramètre un entier représentant le temps (`time`) et un boolean `activate`. Tant que `activate` est différent de `vrai`, le compteur ne se déclenche pas. Ce module retourne `vrai` lorsque `time` est atteint par le compteur. Ainsi, c'est en fonction de ce résultat que la couleur du feu de circulation changera.

Par ailleurs, on ajoute à ce feu de circulation, un feu de piéton. Ce dernier change de couleur seulement en fonction du feu de voiture. En effet, lorsque le compteur du feu rouge de voiture est terminé, le feu de piéton passe au rouge, puis dès que le compteur du feu jaune se termine, le feu de piéton passe au vert.

Cette fonction est implémentée dans le module `pedestrian(car_red_bool, car_yellow_bool)`. C'est dans ce dernier que le feu des piétons est géré. Ainsi, le module main ne gère que le feu de circulation.

2 Smart traffic light

Ce deuxième exercice consiste à ajouter un bouton au feu de piéton. Il servira pour signaler la présence d'un piéton voulant traverser la route.

En effet, le feu de circulation se base sur un compteur de temps implémenté dans l'exercice précédent. Cependant, dans cet exercice, le feu de circulation restera vert même après la fin de son compteur tant qu'aucun piéton ne veuille traverser (c'est-à-dire que le bouton n'est pas appuyé). Dès qu'un piéton appuie sur le bouton, le feu de circulation passe au rouge pour le laisser traverser. En revanche, le feu vert aura toujours un temps minimum d'écoulement

avant de changer de couleur. Cette dernière fonction se fait par l'implémentation d'un module `counter_tick_green(time, activate, button)`. Il compte jusqu'à l'entier `time`, puis restera sur la même valeur tant que bouton soit faux. Si le compteur se termine et que le bouton est vrai, il réinitialise le compteur.

La gestion du bouton de piéton se fait dans le module `pedestrian_button(push)`. Lorsque `push` devient vrai (une personne appuie sur le bouton), l'état du bouton devient vrai tant que le feu de piéton est différent de vert. Il est possible d'appuyer plusieurs fois sur le bouton, l'état du bouton restera toujours vrai. Toutefois, lorsqu'on appuie sur le bouton alors que le feu de piéton est vert, l'état du bouton sera faux (le piéton a la possibilité de traverser donc on retient pas sa demande).

3 Propriétés

Dans ce projet il est demandé de respecter deux propriétés importantes : la sûreté et la vivacité de circulation autant à pieds qu'en voiture.

3.1 Sûreté

Il est nécessaire que les piétons puissent traverser en toute sécurité. Pour cela, il ne faut qu'à aucun moment, les feux de voiture et de circulation soient vert au même moment. C'est pourquoi je vérifie cette propriété grâce à l'expression :

```
G(pedestrian_light = green -> car_light = red) &
 G(car_light = green -> pedestrian_light = red)
```

Le feu de piéton est vert si et seulement si le feu de voiture est rouge, puis vice versa, le feu de voiture passe au vert si et seulement si le feu de piéton est rouge.

3.2 Vivacité

Un feu de circulation se doit d'alterner entre le feu vert et le feu rouge. S'il reste toujours vert pour les voitures, et toujours rouge pour les piétons, il n'y a aucune vivacité et plus personne n'utilisera ce carrefour. C'est pourquoi, je vérifie que chaque feu alterne correctement avec les expressions :

```
        F(car_light = green) <-> F(car_light = red)
F(pedestrian_light = green) <-> F(pedestrian_light = red)
```