Programmation synchrone (PSYN9)

Programmation Synchrone
Flots et automates

Exercice 1 – *Détecteur d'intrusion*

On se propose de programmer un détecteur d'intrusion simple qui, lorsqu'il est activé, lève une alarme en cas de mouvement. Son interface est la suivante :

- un flot booléen `mvt` en entrée, qui fournit la sortie du détecteur de mouvements ;
- un flot booléen `onoff` en entrée, qui signale qu'on veut activer ou désactiver le détecteur ;
- un flot booléen `hs` en entrée, fournit une indication sur le passage du temps physique (voir ci-dessous) ;
- un flot booléen `alarme` en sortie, qui signale qu'une intrusion a été détectée.

Le détecteur doit obéir au cahier des charges détaillé ci-dessous.

- Le détecteur commence son exécution désactivé.
- Le capteur de mouvement est bruité : il se déclenche sporadiquement en l'absence de mouvement. Pour cette raison, le détecteur d'intrusion ne doit lever une alarme que si le capteur indique un mouvement continuel pendant plus de *cinquante millisecondes*.
- Le flot booléen `hs` est vrai une fois par cycle de *dix millisecondes*. On suppose donc que la durée d'un cycle d'exécution est plus courte que cette période.
- Une fois déclenchée, l'alarme doit être maintenue jusqu'à la désactivation du détecteur.

Programmez le noeud `detecteur` et testez son comportement en simulation.

Exercice 2 – *Détecteur d'intrusion redux*

Réimplémentez le détecteur d'intrusion de l'Exercice 1 en utilisant les automates d'Heptagon.

Exercice 3 – *Chronomètre numérique simple*

Le but de cet exercice est de programmer un chronomètre numérique simple à deux boutons (déclenchement/interruption et réinitialisation). Son interface est la suivante :

- l'entrée booléenne `start_stop` transmet les pressions sur le bouton de déclenchement/interruption, contrôlant le basculement du chronomètre de marche à arrêt et vice-versa,
- l'entrée booléenne `rst` transmet les pressions sur le bouton de réinitialisation, qui remet à zéro le temps affiché même si le chronomètre est à l'arrêt,
- l'entrée booléenne `hs` est vraie un cycle synchrone toutes les dix millisecondes—on suppose donc que la durée d'un cycle synchrone est plus petite que dix millisecondes,
- la sortie entière `time` transmet à l'écran numérique le temps à afficher.

Écrivez un noeud `chrono2` obéissant à la spécification ci-dessus **en utilisant les automates** d'Heptagon. Définissez un automate à deux états, `Stop` et `Running`, et utilisez le mot clé `last`.

Exercice 4 – *Chronomètre numérique évolué*

Le chronomètre digital élaboré enrichit le chronomètre digital simple avec la fonctionnalité d'enregistrer un temps intermédiaire.

Ainsi, il gère deux temps :

- un temps interne, `internal_time`, calculé par la montre simple,
- un temps intermédiaire, `displayed_time`, qui reste constant quand la montre est "gelée" et qui est égal à `internal_time` sinon.

Pour contrôler la nouvelle fonctionnalité, nous considérons deux solutions.

1. L'opérateur `chrono3`[1] dispose d'un bouton `pause` qui permet de changer l'état "gelé/non gelé" de l'affichage :
   - initialement, l'état est non gelé,
   - si `pause` est enfoncé et que l'affichage n'est pas gelé alors il devient gelé,
   - si `pause` est enfoncé et que l'affichage est gelé, alors il devient non gelé.

2. L'opérateur `chrono2bis` a les mêmes entrées que `chrono2`, on surcharge le bouton `rst` pour changer l'état "gelé/non gelé" de l'affichage :
   - initialement, l'affichage est non gelé,
   - si `rst` arrive alors que l'affichage est gelé, l'affichage devient non gelé,
   - si `rst` arrive alors que la montre est en marche et l'affichage non gelé, alors l'affichage devient gelé.

   Le bouton `rst` est interprété comme une remise à zéro uniquement quand la montre est à l'arrêt et l'affichage non gelé.

Programmez chacune de ces deux solutions à l'aide des automates d'Heptagon. Nous vous encourageons vivement à réutiliser le nœud `chrono2` réalisé précédemment.

1. Appelé ainsi du fait de ses 3 boutons.