

Programmation synchrone

TP1 : Prise en main d'Heptagon, opérateurs de flots, réinitialisation

Les commandes ci-dessous vous permettront d'installer Heptagon sur votre machine :

```
$ opam init
$ eval $(opam env)
$ opam install lablgtk heptagon
```

Vérifiez ensuite que la commande `heptc`, sans arguments, s'exécute sans erreur. Les options du compilateur ainsi que les constructions de langage acceptées sont décrites dans le manuel d'Heptagon, disponible dans le dossier notes du dépôt du cours.

Exercice 1 – Lecture et interprétation de code

Expliquez ce que font les noeuds `n1`, ..., `n5` suivants :

```
node n1(x: float) returns (y: float)
let
y = x *. x +. 1.0;
tel

node n2(x,y: bool) returns (z,t: bool)
let
z= if x then y else false;
t= if x
  then (if y
        then false
        else true)
  else (if y
        then true
        else false);
tel

node n3(x: int) returns (y: int)
var aux: int;
let
aux= 0 -> pre x;
y = x + aux;
tel

node n4() returns (y: int)
let
y = 0 -> (pre y) + 1 ;
tel

node n5() returns (y: int)
let
y = 0 fby (1 -> (y + pre y)) ;
tel
```

Exercice 2 – Prise en main d'Heptagon

Le code Heptagon ci-dessous implémente un compteur recevant un flot d'entiers x en entrée et produisant un flot d'entiers y en sortie. Sa spécification est la suivante : la sortie y_t vaut $\sum_{0 \leq t' \leq t} x_{t'}$.

```
node somme(x: int) returns (o: int)
let
o = x + 0 fby o;
tel
```

1. Entrez ce code source dans un fichier `tp1.ept`. Compilez à l'aide d'Heptagon ce fichier vers du code C via la séquence de commandes suivantes.

```
$ heptc -target c tp1.ept
$ cd tp1_c
$ gcc -I $(heptc -where)/c -c *.c
```

2. Pour obtenir un exécutable, il faut préciser à Heptagon le noeud principal du programme. Cherchez dans le manuel d'Heptagon quelle option utiliser à cet effet. Recompilez `tp1.ept` ainsi que le code C généré pour obtenir un binaire que vous nommerez `somme_sim`.
3. Exécutez le binaire `somme_sim` et vérifiez que vous obtenez les résultats spécifiés par le chronogramme suivant :

x	1	1	3	-5	0	10	-2	4
y	1	2	5	0	0	10	8	12
4. Un script `hept` est disponible sur le gitlab du cours. Il permet de simplifier la compilation d'un programme `prog.ept` et la simulation d'un noeud `noeud` comme suit :
`$ hept -s noeud prog.ept`
 Effectuez une simulation du noeud `somme` au moyen du script `hept`.

Exercice 3 – Alarme

Programmer un opérateur `alarme_bornes` respectant la spécification suivante.

- Entrées :
 - x , flottant, indiquant la valeur observée ;
 - lo , flottant, indiquant la valeur nominale minimale ;
 - hi , flottant, indiquant la valeur nominale maximale.
- Sortie : `alarme`, booléen.
- Fonction : `alarmet` est vrai si, et seulement si, x_t est hors de l'intervalle $[lo_t, hi_t]$.

Exercice 4 – Vitesse température

Programmer un opérateur `vitesse_temp` respectant la spécification suivante.

- Entrée : x , flottant, indiquant la température observée.
- Sortie : y , flottant.
- Fonction : y_{t+1} vaut $x_{t+1} - x_t$. Choisissez une valeur appropriée pour y_0 .

Exercice 5 – Alarme changement de vitesse de température

Programmer un opérateur `alarme_vit` respectant la spécification suivante.

- Entrée : x , flottant, indiquant la température observée.
- Sortie : `alarme`, booléen.
- Fonction : `alarmet` est vrai si, et seulement si, la vitesse de x à l'instant t est hors de l'intervalle $[-2, 2]$.

Exercice 6 – Mémoire bien initialisée

Programmer un opérateur `jafter` respectant la spécification suivante.

- Entrée : x , booléen.
- Sortie : y , booléen.
- Fonction : y_t est vrai si, et seulement si, x_{t-1} existe et est vrai.

Exercice 7 – Minimum d'un flot d'entiers

Programmer un opérateur `min_flot` respectant la spécification suivante.

- Entrée : x , entier.
- Sortie : y , entier.
- Fonction : y_t est le minimum des $x_{t'}$ pour $0 \leq t' \leq t$.

Exercice 8 – Compteur d'événements

Programmer un opérateur `compteur_evt` respectant la spécification suivante.

- Entrée : e , booléen.
- Sortie : c , entier.
- Fonction : c compte le nombre d'instants consécutifs auxquels e a été vrai depuis la dernière fois où e a été faux (ou depuis $t = 0$). Par exemple :

e	false	true	true	true	false	false	true	...
c	0	1	2	3	0	0	1	...

Exercice 9 – Automate à deux états

Programmer un opérateur `switch2` respectant la spécification suivante.

- Entrées : `orig`, `con`, `coff`, tous trois booléens.
- Sortie : `statev`, booléen.
- Fonction : `statev` passe de faux à vrai sur la commande `con`, de vrai à faux sur la commande `coff`. Tout doit se passer comme si `statev` était égal à `orig` avant le premier instant.

Exercice 10 – Demi-additionneur

Programmer un opérateur `hadd` respectant la spécification suivante.

- Entrées : x , y , tous deux booléens.
- Sorties : `cout`, s , tous deux booléens.
- Fonction : réalise l'addition binaire de x et y , i.e. à tout instant t , $x_t + y_t = 2\text{cout}_t + s_t$

Exercice 11 – Additionneur complet (3 bits)

Programmer un opérateur `fadd` respectant la spécification suivante.

- Entrées : `cin`, x , y , tous trois booléens.
- Sorties : `cout`, s , tous deux booléens.
- Fonction : réalise l'addition binaire de `cin`, x et y , i.e. à tout instant t , $\text{cin}_t + x_t + y_t = 2\text{cout}_t + s_t$.

Exercice 12 – Additionneur série

Dans cet exercice, les flots booléens sont interprétés comme des nombres entiers positifs codés en binaire petit-boutiste (*little-endian*), c'est à dire avec le bit de poids faible à t_0 , et arbitrairement longs. Ainsi, le flot $x_0x_1x_2\dots$ code $x_0 + 2^1x_1 + 2^2x_2 + \dots + 2^tx_t + \dots$

1. Quel est l'entier représenté par le flot `false` ?
2. Quel est l'entier représenté par le flot `true` \rightarrow `false` ?
3. Programmer un opérateur `sadd` respectant la spécification suivante.
 - Entrées : `x`, `y`, tous deux booléens.
 - Sortie : `s`, booléen.
 - Fonction : `s` code la somme des nombres codés par `x` et `y`.
4. Que valent `sadd(true \rightarrow false, false)` et `sadd(true \rightarrow false, true)` ?

Exercice 13 – Alarme changement de température, valeur et vitesse

Programmer un opérateur `alarme_vit_val` respectant la spécification suivante.

- Entrée : `x`, flottant.
- Sortie : `alarme`, booléen.
- Fonction : `alarmet` est vrai ssi la vitesse de `x` a été hors de l'intervalle $[-2, 2]$ lors des trois derniers instants (au moins), et que `xt` est hors de $[-10, 35]$ à l'instant `t`.

Exercice 14 – Compteur d'événements *redux*

Reprenez l'exercice "Compteur d'événements" (Exercice 8). Vous devez cette fois implémenter l'opérateur sans utiliser directement les constructions `fbv` et `pre`. Vous pouvez en revanche utiliser le nœud `somme` défini dans l'exercice 2, ainsi que la construction `reset/every`. Simulez le fonctionnement de votre implémentation.

Exercice 15 – 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, qui 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 continu 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 nœud `detecteur` et testez son comportement en simulation.