TP1 de Programmation Lustre

Frédéric Gava
gava@u-pec.fr

Attention : ici, nous utiliserons Luste v4 qui gratuitement téléchargeable. Ce n'est pas du tout la dernière version de Lustre qui est maintenant payant et dans un environnement "scade" beaucoup plus riche. Une version d'essaie de Scade est néanmoins possible après enregistrement. D'autres langage de la société "Esterel" (comme Esterel ou Signal) sont aussi des langages pour les systèmes temps réél et synchrones.

1 Installation et mise en place

Sous Linux, téléchargez le fichier que vous trouverez ici : `http://www-verimag.imag.fr/The-Lustre-Toolbox.html?lang=en` (non testé sur Mac). Décompressez et en mode **root** recopiez le dossier dans `/usr/local/`. Ensuite, recopiez les lignes suivantes dans le fichier `.bashrc` :

```
export LUSTRE_INSTALL=/usr/local/lustre-v4-III-linux64
export PATH=$PATH:$LUSTRE_INSTALL/bin
```

Fermez votre session et reconnectez vous.

Notez que vous pourrez tout supprimer quand le TP sera terminé.

2 Tutoriel outils Lustre

Dans cette section, nous nous intéressons à la simulation de programmes Lustre à l'aide du simulateur graphique `luciole`. Dans ces premiers exercices, on va analyser l'exemple suivant :

```
node EDGE (b : bool) returns (edge : bool);
 let
  edge = false →b and not pre b;
 tel
```

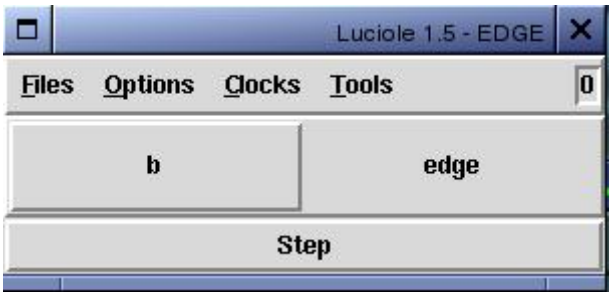
Ce programme détecte un front montant, c'est à dire le passage d'une variable booléenne de la valeur false à la valeur true.

2.1 Simulation

Exercice 1 (*Première exécution*)

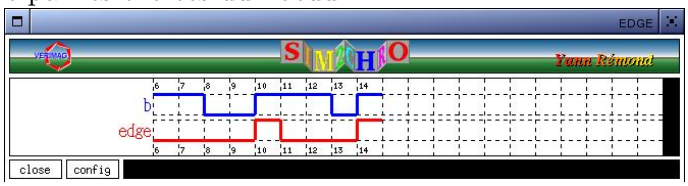
Écrire le noeud EDGE dans un fichier `edge.lus`. *Puis exécutez :*

```
$ luciole edge.lus EDGE
```



Cette commande ouvre une fenêtre de simulation. Elle présente une série de boutons correspondant aux entrées/sorties du noeud. Dans le cas du noeud EDGE, on dispose d'un bouton pour l'entrée "b" et une "lampe" pour la sortie edge (Note : une lampe est un label qui s'affiche en rouge quand edge est vrai, et en gris quand edge est faux). Par défaut, avec cette interface, cliquer sur le bouton "b" provoque un cycle de calcul avec "b" vrai, cliquer sur le bouton "Step" provoque un cycle de calcul avec "b" faux. Le résultat est renvoyé sur la lampe "edge".

Dans le menu "Tools !sim2chro" on peut ouvrir l'outil de visualisation de chronogramme. L'évolution des variables "b" et "edge" est alors automatiquement visualisée sous forme de chronogramme, indexé par les entrées du noeud.



On peut trouver 2 modes :

1. Modes auto-step et modes compose : Par défaut, l'interface d'exécution de luciole est en mode "auto-step", c'est-à-dire que le noeud est activé dès qu'on active sur une entrée booléenne ou sur Step. Dans le menu "Clocks !Compose", on peut passer en mode compose : les entrées booléennes deviennent des interrupteurs qu'on peut activer/désactiver sans provoquer un pas de calcul. Le pas de calcul est déclenché en appuyant sur le bouton Step. Ce mode est nécessaire pour des programmes qui ont plusieurs entrées, pour pouvoir composer des états où plusieurs entrées sont vraies simultanément.
2. Horloge temps-réel : Utiliser le bouton Step peut devenir gênant, on peut alors utiliser une horloge "temps-réel". Dans le menu "Clocks !Realtimeclock" on trouver de quoi permettre d'activer/désactiver le mode temps-réel. Dans ce mode, le pas de calcul est automatiquement déclenché à intervalles réguliers. La période de l'horloge peut être modifiée dans le "Clocks !Changeperiod" (exprimé en milli-secondes).

Attention ! L'aspect temps-réel est relatif et dépend de la plate-forme utilisée. On utilise un système multi-tâches multi-utilisateur qui n'offre aucune garantie temps-réel. En pratique, si la machine est un peu trop chargée, le simulateur aura du mal à soutenir une période inférieure à quelques ms.

Exercice 2 (*Amusez vous*)

Testez les différents modes !

2.2 Compilation en C

La compilation d'un programme Lustre se déroule en plusieurs phases.

(1) La pré-compilation transforme un programme Lustre en programme "Lustre noyau", aussi appelé code expansé. Cette phase est réalisée par la commande :

```
$lus2ec edge.lus EDGE
```

Elle produit un fichier `EDGE.ec`, qui dans ce cas précis est pratiquement très proche du programme initial `edge.lus`.

(2) La compilation proprement dite transforme un programme "ec" en programme C. La commande est :

```
$ ec2c EDGE.ec -v
```

L'option -v permet de passer en mode "verbeux" : des informations additionnelles sur le déroulement de la compilation sont affichées (Note : tous les outils Lustre disposent d'une option -v). Le résultat de la compilation est un fichier `EDGE.c` qui contient le noyau réactif du programme, ainsi que le fichier `EDGE.h` qui contient les informations nécessaires à l'utilisation du noyau.

(3) La compilation en C ne produit que le noyau réactif du système, c'est normalement à l'utilisateur d'écrire un programme principal qui se charge de l'acquisition des entrées et de la visualisation des sorties. Le compilateur ec2c propose une option -loop qui produit, en plus du noyau réactif, un programme principal standard. La commande :

```
$ ec2c EDGE.ec -loop -v
```

produit `EDGE.h`, `EDGE.c` plus un fichier `EDGE_loop.c` qui contient une fonction main standard. Pour des programmes simples (comme edge) ce programme principal peut être utilisé tel-quel.

(3) Pour obtenir un exécutable, il faut finalement utiliser un compilateur C, par exemple GCC sur GNU/Linux. La commande :

```
$ gcc EDGE.c EDGE_loop.c -o EDGE
```

compile les deux fichiers C, et produit l'exécutable EDGE. Le programme principal standard est en fait très rudimentaire : l'utilisateur doit taper au clavier, une par une, les entrées du programme.

(1-3) Lux permet d'enchaîner toute ces phases automatiquement. Il gère en particulier la phase de liaison avec les différentes bibliothèques nécessaires à la construction de l'exécutable. La commande :

```
lux edge.lus EDGE
```

enchaîne automatiquement toutes les phases (normalement) et produit un exécutable EDGE.

Exercice 3 (*Testez en C*)

Compilez et Testez le programme C. Regardez aussi les codes générés !

2.3 Compilation en automate

Une solution alternative pour la compilation d'exécutable consiste à générer le noyau réactif du noeud Lustre sous la forme d'un automate à états finis :

```
$ ec2oc EDGE.ec
```

Cela produit un automate, dans un format particulier appelé "oc" (dans un fichier `EDGE.oc`). On génère le code C correspondant avec la commande :

```
$ poc EDGE.oc
```

La commande "poc" dispose aussi d'une option `-loop`. On peut donc enchaîner les commandes suivante pour obtenir un programme exécutable edge :

```
$ poc EDGE.oc -loop
$ gcc EDGE.c EDGE_loop.c -o EDGE
```

Cette méthode de génération de code est assez proche de la précédente, à ceci près qu'un automate fournit un modèle pertinent pour raisonner sur les exécutions possibles du programme, et ainsi analyser son espace d'états.

Exercice 4 (*Testez les automates*)

Regardez le code des automates. Essayez de trouver une manière (avec une option ?) pour visualiser plus facilement l'automate généré.

2.4 Vérification de propriétés

Lustre dispose d'un outil de vérification formelle de propriétés. L'utilisateur exprime un ensemble de propriétés que doit vérifier le programme, et l'outil indique si ces propriétés sont satisfaites, ou fournit un contre-exemple. Cet outils est `xlesar`, une interface graphique du vérificateur `lesar`.

Mais son utilisation sort du périmètre de ce TP car il faut d'abord faire (un peu) de logique temporelle !

3 Ecriture de petits programmes

Le but de cette section est de programmer une bibliothèque d'opérateurs qui seront réutilisés dans les exercices suivants. Ces opérateurs seront regroupés dans un fichier `utiles.lus`.

Exercice 5 (*Somme*)

Ecrire et simule le noeud :

node *Sommes(X :* **int***)* **returns** *(S :* **int***)*

tel que **Sommes** *retourne, à chaque instant, la somme de toutes les données entrées dans ce noeud.*

Exercice 6 (*Avec Reset*)

De même :

node *Sommes(X :* **int***; Reset* **bool***)* **returns** *(S :* **int***)*

mais cette fois-ci, on remet à 0 la somme à chaque Reset.

Exercice 7 (*Détection de fronts montants/descendants*)

Ecrire et simuler un noeud edge *tel que :*

- *d'entrée flot X et de sortie flot Y*
- *si $Y(t)$ est vrai si et seulement si ($X(t-1)$ existe et est faux et X(t) est vrai) OU (X(t-1) existe et est vrai et X(t) est faux).*

Remarque : Les fronts descendants de x sont simplement les fronts montants de la négation de x, c'est-à-dire edge(**not** x).

Pour éviter l'ouverture/fermeture répétée des portillons de sortie du métro, la RATP a déployé à St-Lazare un mécanisme de détection des sorties des usagers. La porte reste ouverte en permanence, deux cellules photo-sensibles (notées A et B) contrôlent le passage des usagers.

Le passage s'effectue de A vers B. Tout passage de B vers A doit déclencher la fermeture du portillon, représentée par une variable booléenne alarm. Autrement dit, alarm est déclenchée si la séquence des événements captés est différente de (A. true*.B)*.

Exercice 8 (*Portillon dans le métro*)

Coder et simuler en Lustre le noeud Portillon, *il aura pour spécification :*

node *Portillon (A, B:* **bool***)* **returns** *(alarm:* **bool***);*

On fera l'hypothèse que A et B ne peuvent être simultanément vrais, du fait des contraintes physiques du système.