TD 6 Introduction à Esterel

Exercice 1 – ZUDNB

Signaux valués, introduction de variables, instructions temporelles et instruction `present`

Le but de cet exercice est de modéliser un compteur avec reset.
Spécification : un signal `ZERO` sert de reset, qui remet un compteur à zero ; ensuite, chaque occurrence d'un signal d'entrée `UN` reçu incrémente ce compteur ; de même le signal `DEUX` augmente le compteur jusqu'à la prochaine occurrence du signal `ZERO`.

Un signal `NB` valué avec la valeur du compteur est émis lorsque le compteur vaut 3 ou un multiple de 3.

Question 1

Ecrire un squelette de programme prenant en compte le signal `ZERO`.

Question 2

Compléter votre squelette en rajoutant les instructions d'initialisation d'une variable `compt`, de prise en compte de `UN` et `DEUX` et d'émission de `NB`.

Question 3

Ecrire un fichier de test en précisant ce qui est attendu

Exercice 2 – Feux tricolores

Dans cet exercice, on souhaite écrire un contrôleur de feux tricolores pour un croisement simple de 2 voies est-ouest (`EO`) et nord-sud (`NS`).

Pour modéliser ce mécaniseme on utilisera les signaux de sortie `RNS, ROE, VNS, VEO, ONS, OEO` pour commander l'allumage des différentes composantes. Deux signaux `ACNS` et `ACEO` émis par le programme activent alternativement les deux voies

On considère que les feux sont tous au rouge initialement par sécurité, et que la voie Nord-Sud passe la première au vert.

On doit écrire deux contrôleurs de feux, un pour chaque voie ; chaque contôleur déroule la séquence vert orange rouge pour sa voie. Un contrôleur actif ne signale que ses changements, l'autre voie est au rouge pendant ce temps.

Question 1

Les feux correspondant à chaque voie restent au vert 4 instants, puis passent 1 instant à l'orange avant de passer au rouge ; par sécurité, le passage au vert d'une voie se fait un instant plus tard que le passage au rouge de la voie perpendiculaire.

On a ici un fonctionnement du type coopératif : un thread déroule la séquence d'un pas de boucle, puis signale à l'autre qu'il peut démarrer et se met lui-même en attente de son propre signal de démarrage. Lors de l'exécution, une seule branche avance à chaque instant, (sauf aux instants de basculement) c'est simple.

Question 2

La question précédente fait l'hypothèse implicite que l'allumage d'une couleur sur un feu tricolore éteint forcément la couleur précédente. De fait des signaux doivent aussi être émis pour provoquer l'extinction ; on les nomme `RNS_E` `OEO_E`. Compléter le programme précédent pour assurer un fonctionnement correct.

On ajoute des signaux (pour éteindre explicitement une couleur avant d'afficher la suivante) pour insister sur le fait que plusieurs signaux peuvent être emis au même instant.

Question 3

On néglige de nouveau dans la suite les signaux d'extinction pour ne pas alourdir la structure des programmes

Signaux d'entrée, instruction abort :Un signal de Reinitialisation (Reset) bloque immédiatement les deux voies en feu rouge (par exemple pour passage de vehicules d'urgence dispensés du respect des feux) et attend un signal `ACNS` pour débloquer d'abord la voie Nord-sud et reprendre le fonctionnement alterné.

Question 4

On introduit maintenant deux autres branches parallèles comptant les véhicules qui arrivent (sous forme de signaux extérieurs fournis par l'utilisateur en cours de la simulation, par des capteurs dans la réalité) sur chacune des voies pendant la durée de son feu rouge . On ajoute donc les signaux d'entrée AN, AS, AE, AO qui indiquent l'arrivée d'une Auto sur l'une des quatre directions.

Variables : la durée du feu vert sur chaque voie est 2 au minimum et 2 + le nombre de voitures arrivée pendant la durée du feu rouge précédent (on suppose que la circulation est assez fluide pour que tous les véhicules attendant au feu puisse passer lors du feu vert). On introduit pour cela une variable `duree(N ou E)`. Comme une variable ne peut être partagée, un signal valué (`DNS` ou `DEO`) communique la durée lors de l'activation du contrôleur de la voie, prêt à passer au vert.

Ecrire la surveillance des arrivées et ajouter la communication nécessaire aux contrôleurs

Exercice 3 – Producteur-consommateurs : signaux valués multiples

Le problème assemble un producteur, deux consommateurs et le gérant du stock de l'unique produit : il met à jour le nombre de produits disponibles.

Les signaux d'entrée sont : C1 et C2 pour indiquer la demande d'un des deux consommateurs, arrivant aléatoirement de l'extérieur et FIN qui arrête le travail de la journée.

Dans un premier temps on accepte une seule demande par instant : écrire déclaration et relation.

On impose le comptage des produits reçus par chaque consommateur, le comptage des produits fabriqués par le producteur et évidemment le stock. Les variables `stock`, `nbprod`, `nbconso1` et `nbconso2` sont sorties en fin de programme grâce respectivement aux signaux de sortie valués `FS`, `FP`, `FC1` et `FC2`.

Question 1

Donner la structure générale du programme.

Question 2

Ecrire le producteur : il produit dès le départ, émettant le signal de sortie `P` après 3 tick (destiné au gérant), incrémente sa production et attend pour recommencer à produire le signal P_S (produit stocké) émis par le gérant.

Question 3

Un consommateur recevant l'entrée qui lui est associée (C1 ou C2) envoie un signal C valué par son numéro : il demande ainsi un produit au gérant et attend C_S(num) (consommateur servi) avant de reprendre toute commande. Lorsque le signal C_S(num) le concerne, il incrémente le nombre de produits reçus. Une demande C1 n'est pas nécessairement satisfaite, le gérant répond alors par C_S(0).Ecrire le consommateur 1.

Question 4

Le gérant est plus complexe. On impose que le traitement des signaux du producteur et du consommateur soit réalisé par des branches parallèles : `stock` ne peut être partagée -même en lectuire seule- et doit être consultée et modifiée dans un traitement unique. Pour cela, le signal multiple valué `S` peut être émis plusieurs fois pour indiquer les modifications souhaitées par les branches (+1 pour accepter un produit et -1 pour livrer un produit) ; il présente le bilan de traitement des branches et permet la mise à jour du stock en fin d'instant. De la même manière, des signaux internes, locaux au gérant, `MAX_L` et `VIDE_L` émis permettent d'avertir les branches des conditions de blocage temporaire de leur activité.

Le point délicat concerne la reprise de la production lorsque le stock est plein. Rappel : sauf pour les signaux multiples, un *signal d'entrée* deja présent dans l'instant ne peut y être émis de nouveau.

Exercice 4 – Threads concurrents et synchronisation en Esterel

On veut simuler le rayon `fruits` d'une supérette : on suppose qu'il y a trois étalages de fruits (Bananes, Pommes, Oranges), initialement fournis chacun avec 10 kilos de fruits et au plus deux clients qui se servent simultanément dans le même bac. Un gérant automatique de chaque étalage connaît à chaque instant la quantité disponible de chaque fruit, il doit prévenir un employé lorsqu'il reste moins de 2 kilos d'un fruit pour qu'un cageot soit ajouté et éviter ainsi la rupture de stock. Le signal d'entrée `B1, O1`, ou `P1` (resp. `B2, O2`, ou `P2`) indique que le client 1 (resp client 2) souhaite se servir un kilo du fruit B, O ou P. Bien sur, un client ne peut se servir que d'une sorte de fruit à la fois. Un instant plus tard, le client dispose de son kilo de fruit, chaque balance intelligente émet l'un des signaux valués `B(1), O(1), P(1)` (destiné au gérant automatique) ; bien sûr, plusieurs clients peuvent émettre ce signal au même instant. Le gérant des bananes émet le signal `MB` pour indiquer qu'il risque de manquer de bananes lorsqu'il en reste 2 kilos ou moins, celui des oranges émet `MO`, celui des pommes `MP` dans les mêmes circonstances. Plusieurs de ces signaux peuvent être émis au même instant. Il reçoit un signal interne (ou déclaré output pour être observable en phase de test) AB (resp. AO ou AP) lorsque 10 kilos de bananes (resp. oranges, pommes) ont été ajoutés à l'étalage. Ces signaux sont émis par l'employé lorsqu'il a rechargé l'étalage.

Question 1 – Déclarations et structure générale

Déclarer les signaux nécessaires. Indiquer l'organisation générale du programme : montrer tous les blocs parallèles (en indiquant leur rôle par une ligne de commentaire).

Question 2 – Les clients (en supposant qu'un étalage n'est jamais vide)

Ecrire les deux clients. Quelles sorties obtient-on avec la séquence d'ensembles d'entrées O1 ;B1 O2 ; P1 ; ?
Proposer une séquence d'ensembles d'entrées en incluant des instants où aucun client n'est présent au rayon fruit.

Question 3 – Les gérants

Déclarer les variables nécessaires. Ecrire le gérant des oranges. Proposer une séquence d'ensembles d'entrées menant à l'émission des signaux MO puis AO

Question 4 – L'employé, récepteur des signaux MB, MO, MP et émetteur de AB, AO, AP

Ecrire l'employé, en supposant qu'il peut recharger plusieurs étalages dans le même instant. Si on lève cette hypothèse, un étalage peut se trouver vide ; quelles modifications proposez-vous ?

Question 5

Les gérants sont complètement superposables sauf par le nom des signaux qu'ils échangent avec l'extérieur. Proposer une écriture du module gérant et trois instanciations dans le programme initial.

Exercice 5 – Calculs en C appelés dans esterel

Esterel n'est pas limité à l'émisson de signaux : par exemple, des valeurs de régulation à transmettre peuvent faire appel à des calculs complexes, appelant des bibliothèques de calcul scientifique C. L'appel d'une fonction C se fait par l'instruction call fonction(liste param résultat)(liste param d'entrée). La liaison est évidemment facilitée par le fait que le traducteur esterel produit du C et l'incorporation de procédures se fait facilement lors de l'édition de lien. Cependant, cette étape doit être soigneusement préparée, d'une part par une déclaration dans le code esterel séparant en deux listes distinctes les paramètres de retour et les paramètres d'entrée (dans cet ordre) et d'autre part dans un fichier nom.h (syntaxe C) portant le même nom que le programme esterel nom.strl. Les paramètres ont des valeurs simples (entiers, flottants, chaînes, booléens) ; dans la fonctions C associée, les données d'entrée sont des valeurs et les résultats sont des pointeurs sur les types correspondant. Le premier exercice émet simplement des signaux à valeurs aléatoires entre 1 et une valeur maximum. Voici le programme C permettant l'initialisation du générateur et la production d'une nouvelle valeur entre 0 et borne - 1 :

```
/********** alea.c **********/

#include <stdio.h>
#include <sys/types.h>
#include <time.h>
#include <limits.h>
#include <stdlib.h>

void init (int *sortieinutile, int entreeinutile) {
  time_t t1;

  (void) time(&t1);
  srand((long) t1);
}

void alea (int *resalea, int borne) {
  int rd;

  rd = rand();
  *resalea = (((double) rd) / RAND_MAX) * borne;
}
```

Question 1

Ecrire le programme `delaialea.strl` comportant une boucle qui, lors de chaque pas :

- demande le calcul d'un nouveau nombre entre 0 et une borne (- 1) donnée,
- émet la valeur obtenue dans un signal de sortie valué S,
- utilise ce nombre pour faire une attente de 1 à borne `tick`.

Deuxième exemple : On veut simuler avec cette technique l'arrivée aléatoire de 3 types de clients numérotés de 1 à 3 :

- Une branche demande le calcul d'un nombre aléatoire entre $[0, 3[$ à l'aide de la procédure `alea()` et émet la valeur obtenue au travers d'un signal valué signalant l'arrivée du client de type indiqué par cette valeur.
- Trois autres branches comptent simplement le nombre d'arrivée de chacun des clients avec respectivement 3 variables `n1`, `n2` et `n3`.
- Un signal `FIN` arrête le processus et provoque l'affichage des trois compteurs.

Question 2

Ecrire une version formée de 4 boucles se déroulant en parallèle.

`alea.c` contient les définitions de `alea()` et `init()`.
`clientalea.h` doit contenir les déclarations C des fonctions `init()` et `alea()`.
Compiler avec `gcc -ansi -Wall -c alea.c` pour créer `alea.o`.
`esterel -simul clientalea.strl` crée `clientalea.c` comme d'habitude.
`gcc -o clientalea clientalea.c alea.o ../libcsimul.a` crée `clientalea`, prêt à executer ...
avec la séquence `;;;;;;;;;;;;;;;;;;;;;;;;...;;FIN;`

exemple de sortie : `F1(446) F2(454) F3(467)`

Question 3

On peut remarquer que chaque boucle exécute exactement un pas lors de chaque instant ; on peut donc écrire une seule boucle contenant des branches parallèles. Modifier le programme précédent pour mettre en œuvre cette solution.

Exercice 6 – Une machine à café modulaire

On veut écrire un module attend_prix réutilisable dans de mutiples distributeurs pour recevoir une certaine somme fixée par une constante prix_produit, 4 par exemple. Dans un premier temps on accepte uniquement des pièces de 1, 2 unités monétaires (signal valué d'entrée PIECE, de sortie PIECE_REJETEE pour une autre pièce). Le module cumule les entrées jusqu'à ce que le prix soit atteint ou dépassé, tout en surveillant l'annulation (signal d'entrée ANNULER) qui renvoie les pièces déjà émises (signal de sortie valué RETOUR).
Lorsque la somme cumulée atteint -ou dépasse le prix de la boisson, le module émet PRIX_ATTEINT ; si la somme cumulée dépasse le prix le module rend la monnaie (signal de sortie valué MONNAIE).
Il attend que la boisson soit prise (signal PRODUIT_SERVI) ou pour reprendre au début, attendre de nouvelles pièces.

L'instruction trap (condition portant sur des variables) permet de sortir de la phase d'acquisition des pièces et de rendre la monnaie (signal de sortie valué MONNAIE) si nécessaire ; l'instruction abort (portant sur les signaux) permet de traiter le retour de la somme déjà acquise (signal de sortie valué RETOUR).

Question 1
Ecrire l'ensemble du module.

Question 2
Le second module servir_produit est plus simple : il reçoit PRIX_ATTEINT, NUM_PRODUIT et émet SERVIR avec ce numéro du produit (signal de sortie valué NUM_PRODUIT) ; une pause de durée déterminée par la constante durée simule la préparation/livraison paramétrable et émet PRODUIT_SERVI.

Les signaux de communication entre les deux modules sont donc PRIX_ATTEINT et PRODUIT_SERVI.

Question 3
Ecrire une machine à boissons chaudes mach_boisson_run (on peut supposer que le numéro 1 est associé à un café, 2 à un thé, 3 à un chocolat) qui utilise ces modules. Veiller à ce que les signaux circulant (émis par l'un utilisé par l'autre) entre les modules soient internes au module de la machine. Les signaux entrants et sortants doivent être redéclarés dans la machine avec une correspondance assurée dans l'instruction run. Donner une séquence de signaux d'entrée pour tester le fonctionnement.

Annexe

Syntaxe de l'instruction `run`

```
run nommodule [ signal   exterieurs1/locals1, exterieurs2/locals2 ;
                constant exterieurc1/localc1 ;
                <et type, function, procedure, task> ]
```

Compilation d'un programme comportant des modules :

1) `esterel nom.strl` permet de contrôler syntaxe et erreurs grossières
2) pour compiler un programme comportant des modules, on doit donner la liste de tous les modules et le nom choisi pour le programme c (en général le nom du module principal racine des instanciations des autres modules)

```
esterel -simul nom1.strl nom2.strl -B nom3
```

Le programme C créé se traduit en exécutable comme tout autre programme Esterel :

```
gcc -m32 -o feux_tricolores4 feux_tricolores4.c -I \$ESTEREL/include -L \$ESTEREL/lib
```

