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`.

Solution:

```
module ZUDNB :

input  UN, ZERO, DEUX;
output NB : integer,
       RESET;

var compt := 0 : integer  in
  loop
     emit RESET;
     compt := 0;
     loop
        await [ UN or DEUX ];

        present UN then
          compt := compt + 1
        end present;

        present DEUX then
          compt := compt + 2
        end present;

        if compt >= 3 and (compt mod  3) = 0 then
           emit NB(compt)
        end if;
     end loop

  each ZERO
```

```
end var                 %compt

end module


%%%%%%%%%%

%{

$ esterel -simul ZUDNB.strl
$ gcc -m32 -o ZUDNB ZUDNB.c -I $ESTEREL/include -L $ESTEREL/lib -lcsimul
$ ZUDNB
ZUDNB> ;
--- Output: RESET
ZUDNB> UN DEUX; DEUX;
--- Output: NB(3)
--- Output:
ZUDNB> UN;
--- Output: NB(6)
ZUDNB> ZERO; UN DEUX;
--- Output: RESET
--- Output: NB(3)
ZUDNB>

}%
```

Question 3
Ecrire un fichier de test en précisant ce qui est attendu
Solution:

```
;
UN DEUX
;
UN;
ZERO; UN DEUX;
;
```

Exercice 2 – Machine à café élémentaire

thème : Restriction sur les ensembles d'états, attente sélective

Spécification : une machine automatique peut servir du café, du thé ou du chocolat (signaux d'entrée `BCAFE, BTHE, BCHOCO` - B pour Bouton - et signaux de sortie `SERV_CAFE, SERV_THE, SERV_CHOCO`) ;

Toutes les boissons sont au même prix, avec un jeton (représenté par le signal `PIECE`) ; le jeton doit être reçu avant l'appui du bouton sélectionnant la boisson (qui reste sans effet sinon).

Un bouton `ANNULER` peut être appuyé à tout instant ; si un jeton a été inséré et non utilisé, le signal `RETOUR_PIECE` est émis.

On précise que le jeton ne peut être reçu en même temps que l'appui d'un bouton de boisson, et bien sûr, on ne peut demander deux boissons simultanément.

Question 1
Ecrire le module MCA.
Solution:

```
module MCA :

input  PIECE,BTHE,BCAFE,BCHOCO;
input  ANNULER;
output SERV_THE, SERV_CAFE, SERV_CHOCO, RETOUR_PIECE;
relation PIECE # BTHE # BCAFE # BCHOCO;

loop
   var paye := false : boolean in
     abort
         await immediate PIECE;
         paye := true;
        weak abort
         await                          %un seul execute
            case PIECE  do emit RETOUR_PIECE
            case BTHE   do emit SERV_THE
            case BCAFE  do emit SERV_CAFE
            case BCHOCO do emit SERV_CHOCO
         end await
        when [BCAFE or BTHE or BCHOCO]
     when ANNULER do
       if (paye) then emit RETOUR_PIECE end if
     end abort

   end var
end loop

end module
```

Question 2
Ecrire un fichier de test.
Solution:

```
;;PIECE;;
BCAFE;;
BCAFE;;
PIECE;;; ANNULER;;
```

Exercice 3 – 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.

Solution:

```
module NS :

input  ACNS;
output ACEO, RNS, ONS, VNS;

loop
   abort
      sustain RNS
   when ACNS;

   emit RNS; pause;     % par securite, rouge encore un instant

   repeat 4 times
      emit VNS; pause
   end repeat;
   emit ONS; pause;
   emit ACEO
end loop

end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module EO :

input  ACEO;
output ACNS, REO, OEO, VEO;

loop
   abort
      sustain REO
   when ACEO;

   emit REO; pause;     % par securite, rouge encore un instant
```

```
   repeat 4 times
      emit VEO; pause
   end repeat;
   emit OEO; pause;
   emit ACNS
end loop

end module

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

module feux :

input  ACNS, ACEO;
output RNS, REO, ONS, OEO, VNS, VEO;

relation ACNS # ACEO;

[
  await 5 tick;  % Les feux tricolores ne demarre qu'apres 5 tick.
  emit ACNS      % Et on commence par le Nord-Sud
  ||
  run NS[signal ACNS/ACNS]
  ||
  run EO[signal ACEO/ACEO]
]

end module


%%%%%%%%%%

%{

$ esterel feux_tricolores1.strl -simul
$ gcc -m32 -o feux_tricolores1 feux_tricolores1.c -I $ESTEREL/include -L $ESTEREL/lib
$ feux_tricolores1
feux> ;;;;;;;;;;;;;;;;;;
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO ONS
--- Output: RNS REO
--- Output: RNS VEO
--- Output: RNS VEO
```

```
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS OEO
--- Output: RNS REO
feux> ;;
--- Output: REO VNS
--- Output: REO VNS
feux>

}%
```

Solution:

```
module NSEO :

input  ACNSEO;
output ACAUTRE, R, O, V;

loop
   abort
      sustain R
   when ACNSEO;

   emit R; pause;       % par securite, rouge encore un instant

   repeat 4 times
      emit V; pause
   end repeat;
   emit O; pause;
   emit ACAUTRE
end loop

end module

%%%%%%%%%%

module feux :

input  ACNS, ACEO;
output RNS, REO, ONS, OEO, VNS, VEO;

relation ACNS # ACEO;

[
  await 5 tick;  % Les feux tricolores ne demarre qu'apres 5 tick.
  emit ACNS      % Et on commence par le Nord-Sud
  ||
  run NSEO[signal ACNS/ACNSEO, ACEO/ACAUTRE, RNS/R, ONS/O, VNS/V ]
  ||
```

```
  run NSEO[signal ACEO/ACNSEO, ACNS/ACAUTRE, REO/R, OEO/O, VEO/V]
]

end module


%%%%%%%%%%

%{

$ esterel feux_tricolores1_1.strl -simul
$ gcc -m32 -o feux_tricolores1_1 feux_tricolores1_1.c -I $ESTEREL/include -L $ESTEREL/
$ feux_tricolores1_1
feux> ;;;;;;;;;;;;;;;;;;
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO ONS
--- Output: RNS REO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS OEO
--- Output: RNS REO
feux> ;;
--- Output: REO VNS
--- Output: REO VNS
feux>

}%
```

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.

Solution:

```
module NSEO :

input  ACNSEO;
output ACAUTRE, R, O, V, RE, OE, VE;
```

```
loop
   abort
      sustain R
   when ACNSEO;

   emit R; pause;       % par securite, rouge encore un instant
   emit RE;

   repeat 4 times
      emit V; pause
   end repeat;
   emit VE;
   emit O; pause;
   emit OE;
   emit ACAUTRE
end loop

end module

%%%%%%%%%%

module feux :

input  ACNS, ACEO;
output RNS, REO, ONS, OEO, VNS, VEO;
output RNS_E, REO_E, ONS_E, OEO_E, VNS_E, VEO_E;

relation ACNS # ACEO;

[
  await 5 tick;  % Les feux tricolores ne demarre qu'apres 5 tick.
  emit ACNS      % Et on commence par le Nord-Sud
  ||
  run NSEO[signal ACNS/ACNSEO, ACEO/ACAUTRE, RNS/R, ONS/O, VNS/V,
                  RNS_E/RE, ONS_E/OE, VNS_E/VE ]
  ||
  run NSEO[signal ACEO/ACNSEO, ACNS/ACAUTRE, REO/R, OEO/O, VEO/V,
                  REO_E/RE, OEO_E/OE, VEO_E/VE ]
]

end module


%%%%%%%%%%

%{

$ esterel feux_tricolores2.strl -simul
$ gcc -m32 -o feux_tricolores2 feux_tricolores2.c -I $ESTEREL/include -L $ESTEREL/lib
$ feux_tricolores2
feux> ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: REO VNS RNS_E
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO ONS VNS_E
--- Output: RNS REO ONS_E
--- Output: RNS VEO REO_E
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS OEO VEO_E
--- Output: RNS REO OEO_E
--- Output: REO VNS RNS_E
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO ONS VNS_E
--- Output: RNS REO ONS_E
--- Output: RNS VEO REO_E
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS OEO VEO_E
--- Output: RNS REO OEO_E
--- Output: REO VNS RNS_E
--- Output: REO VNS
feux>

}%
```

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.

Solution:

```
module NSEO :

input  ACNSEO, RESET;
output ACAUTRE, R, O, V;

loop
   loop
      abort
```

```
         sustain R
      when ACNSEO;

      emit R; pause;    % par securite, rouge encore un instant

      repeat 4 times
         emit V; pause
      end repeat;
      emit O; pause;
      emit ACAUTRE
   end loop
each RESET

end module

%%%%%%%%%%

module feux :

input  ACNS, ACEO, RESET;
output RNS, REO, ONS, OEO, VNS, VEO;

relation ACNS # ACEO;

[
  await 5 tick;  % Les feux tricolores ne demarre qu'apres 5 tick.
  emit ACNS      % Et on commence par le Nord-Sud
  ||
  run NSEO[signal ACNS/ACNSEO, ACEO/ACAUTRE, RNS/R, ONS/O, VNS/V ]
  ||
  run NSEO[signal ACEO/ACNSEO, ACNS/ACAUTRE, REO/R, OEO/O, VEO/V]
]

end module


%%%%%%%%%%

%{

$ esterel feux_tricolores3.strl -simul
$ gcc -m32 -o feux_tricolores3 feux_tricolores3.c -I $ESTEREL/include -L $ESTEREL/lib
$ feux_tricolores3
feux> ;;;;;;;; RESET;;;;;;;;;;;ACNS;;;;;;;;;
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: REO VNS
--- Output: REO VNS
```

```
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO VNS
--- Output: REO ONS
--- Output: RNS REO
--- Output: RNS VEO
--- Output: RNS VEO
feux>

}%
```

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

Solution:

```
module NSEO :

input  ACNSEO, RESET;
output ACAUTRE, R, O, V;

input D : integer;

loop
   abort
      sustain R
   when ACNSEO;

   emit R; pause;       % par securite, rouge encore un instant
```

```
   await immediate D;
   repeat ?D times      % on sait que
      emit V; pause
   end repeat;
   emit O; pause;
   emit ACAUTRE
end loop

end module

%%%%%%%%%%

module feux :

input  ACNS, ACEO, RESET, O;
output RNS, REO, ONS, OEO, VNS, VEO;

input  AN, AS, AE, AO;
output DNS : integer, DEO : integer;

relation ACNS # ACEO;

[
  await 5 tick;  % Les feux tricolores ne demarre qu'apres 5 tick.
  emit ACNS      % Et on commence par le Nord-Sud
  ||
  run NSEO[signal ACNS/ACNSEO, ACEO/ACAUTRE, RNS/R, ONS/O, VNS/V, DNS/D ]
  ||
  run NSEO[signal ACEO/ACNSEO, ACNS/ACAUTRE, REO/R, OEO/O, VEO/V, DEO/D ]
  ||
  loop          % pour compter les voitures NS
     await RNS;
     var duree := 2 : integer in
       abort
         loop
            present AN then duree := duree + 1 end present;
            present AS then duree := duree + 1 end present;
         each tick
       when [not RNS];
       emit DNS(duree)
     end var;
  end loop
  ||
  loop          % pur compter les voiture EO
     await REO;
     var duree := 2 : integer in
       abort
         loop
            present AE then duree := duree + 1 end present;
            present AO then duree := duree + 1 end present
         each tick
       when [not REO];
```

```
       emit DEO(duree)
     end var;
  end loop
]

end module


%%%%%%%%%%

%{

$ esterel feux_tricolores4.strl -simul
$ gcc -m32 -o feux_tricolores4 feux_tricolores4.c -I $ESTEREL/include -L $ESTEREL/lib
$ feux_tricolores4
feux> ;;;;;;;;;;
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: RNS REO
--- Output: REO VNS DNS(2)
--- Output: REO VNS
--- Output: REO ONS
--- Output: RNS REO
feux> ;;;
--- Output: RNS VEO DEO(2)
--- Output: RNS VEO
--- Output: RNS OEO
feux> AN AS; AN AS;;;
--- Output: RNS REO
--- Output: REO VNS DNS(4)
--- Output: REO VNS
--- Output: REO VNS
feux> AE; AE AO; AE AO;;;
--- Output: REO VNS
--- Output: REO ONS
--- Output: RNS REO
--- Output: RNS VEO DEO(7)
--- Output: RNS VEO
feux> ;;;;;;
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS VEO
--- Output: RNS OEO
feux> ;;;
--- Output: RNS REO
--- Output: REO VNS DNS(2)
--- Output: REO VNS
```

```
feux>

}%
```

Question 5

Il y a donc toujours plusieurs processus qui se déroulent en parallèle : contrôle des feux d'une voie, surveillance des voitures arrivant sur la voie bloquée ; c'est moins simple ; attention au 'immediate' ou pas !

Instruction `run`, construction mdulaire. Le traitement des deux voies Est-Ouest et Nord-Sud est tout a fait similaire. On revient au problème initial plus simple, en oubliant les véhicules

a) Introduire un signal `DebutE`, émis juste avant le premier passage au rouge de la voie Nord-Sud après un redémarrage ; ce signal est inutile mais il permet de traiter de manière symétrique les deux voies.
b) Définir le suivi des feux d'une voie sous forme de module `tricolorVoie` : il est essentiel de déterminer quels sont les signaux d'entrée (provenant le plus souvent de l'extérieur) et les signaux de sortie communiquant avec l'extérieur (d'autres parties du programme) et visualisant les commandes électriques effectuées.
c) Ecrire le programme `tricolorRun.strl`, de manière à activer deux instanciations du module `tricolorVoie` en parallèle : il suffit de mettre en correspondance les signaux formels du module et les signaux de l'environnement.

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
```

