Correction de l'exam « Langages pour le temps réel – LUSTRE » de l'année 2022-20223

<u>Question 1</u> Petite question de compréhension

Soit les programmes Lustre TEST1 et TEST2 suivants :

```
node accumulateur (X: int) returns (S:int);
     let
          S = X + (0 -> pre(S));
     Tel

node TEST1 (X: int) returns (S12,S12:int; B1:bool);
     var B1:bool;
     let
          B1 = true -> pre(false -> pre(B1));
          S11 = current (accumulateur (X when B1));
          S12 = current (accumulateur (X) when B1);

     Tel

node TEST2 (X: int) returns (S21,S22:int; B2:bool);
     var B2:bool;
     let
          B2 = true -> pre(true -> not pre(B2));
          S21 = current (accumulateur (X when B2));
          S22 = current (accumulateur (X) when B2);

     tel
```

Soit le flot constant `X = (1,1,1,1,1,1…)`. Quelle sont les réponses de :

`accumulateur(X)`, `TEST1(X)` et `TEST2(X)` ?

On répondra par un chronogramme. Vous pouvez répondre directement sur la feuille donnée en annexe, ou vous inspirer de ce chronogramme donné en annexe et répondre sur votre copie.

Correction

instant	0	1	2	3	4	5	6
X	1	1	1	1	1	1	1
accumulateur(X)	1	2	3	4	5	6	7
B1	T	F	T	F	T	F	T
accumulateur(X when B1)	1		2		3		4
accumulateur(X) when B1	1		3		5		7
S11=current(accumulateur(X when B1))	1	1	2	2	3	3	4
S12=current(accumulateur(X) when B1)	1	1	3	3	5	5	7
B2	T	T	F	F	T	T	F
accumulateur(X when B2)	1	2			3	4	
accumulateur(X) when B2	1	2			5	6	
S21=current(accumulateur(X when B2))	1	2	2	2	3	4	4
S22=current(accumulateur(X) when B2)	1	2	2	2	5	6	6

Explications pour B1 :

Le flot booléen B1 est défini avec une double initialisation et un double pre. La double initialisation dit qu'il vaut true à l'instant 0 et false à l'instant 1, puis se répète en appliquant un double pre. B1 est donc une alternance de true et de false.

Explications pour B2 :
Le raisonnement est le même pour B2. Il est défini avec une double initialisation et un double pre. La double initialisation dit qu'il vaut true à l'instant 0 et à l'instant 1, puis se répète en appliquant un not sur un double pre. B1 est donc une alternance de true true et de false false.

Explications pour accumulateur(X when B1) :
L'important est de voir que l'opération when est appliquée à X, c'est-à-dire à l'entrée de accumulateur. Le flot X when B1 est égal à X aux instants où B1 est vrai, et est absent aux instants où B1 est faux. Donc lorsque B1 est faux, la fonction accumulateur ne reçoit aucune entrée (puisque X when B1 est absent), et donc accumulateur n'est pas activée à ces instants là. Du coup, accumulateur n'est appelé que à l'instant 0, puis à l'instant 2, puis 4, puis 6… Et comme accumulateur incrémente sa sortie à chaque fois qu'il est appelé, alors accumulateur(X when B1) vaut 1 à l'instant 0, puis 2 à l'instant 2, puis 3 à l'instant 4, pour 4 à l'intant 6…

Explications pour current(accumulateur(X when B1)) :
Le current est une opération de sur-échantillonnage (c'est l'inverse du when). Il ramène sur l'horloge de l'horloge du flot auquel il est appliqué. Ici, il est appliqué à accumulateur(X when B1). Ce dernier a comme horloge B1 (il est présent lorsque B1 est vrai). Donc current(accumulateur(X when B1)) est sur l'horloge de l'horloge de accumulateur(X when B1), c'est-à-dire sur l'horloge de B1, c'est-à-dire sur l'horloge de base. (car B1 est sur l'horloge de base, B1 ayant une valeur à tous les instants).
Donc, pour tous les instants de l'horloge de base, current(accumulateur(X when B1)) vaut la valeur de accumulateur(X when B1) lorsque ce dernier est présent (aux instants pairs), et vaut la dernière valeur connue de accumulateur(X when B1) lorsque ce dernier est absent (aux instants impairs).

Explication pour accumulateur(X) when B2 :
L'important est de voir ici que le when est appliqué sur la sortie de accumulateur(X). Pour construire le flot accumulateur(X) when B2 il suffit de prendre accumulateur(X) et l'échantillonner sur B2.

Explication sur current(accumulateur(X) when B2) :
On suit le même raisonnement que pour le current précédent. L'horloge de accumulateur(X) when B2 est B2. Donc l'horloge de current(accumulateur(X) when B2) est l'horloge de B2, c'est-à-dire l'horloge de base. Donc à tous les instants de l'horloge de base, si accumulateur(X) when B2 est présent, current(accumulateur(X) when B2) a pour valeur la valeur de accumulateur(X) when B2, sinon il a pour valeur la dernière valeur connue de accumulateur(X) when B2.

<u>Question 3</u> Calcul d'horloge

Soit le programme Lustre défini partiellement par

```
node prog1 (   B1, B2 : bool ;
               H1 : bool when B1 ;
               H2 : bool when B2 ;
               X1 : int when H1 ;
               X2 : int when H2 ;
               Y : int when (B1 and B2))
returns (S : int when …) ;
var  Z1:int when …;
     Z2:int when …;
```

```
        Z3:int when …;
    let
        Z1 = current(current(X1)) ;
        Z2 = current(current(X2)) ;
        Z3 = (Z1 + Z2) when (B1 and B2) ;
        S = current(Z3 + Y) ;
    tel.
```

Il manque dans ce programme les informations d'horloge. Lorsqu'elles existent, précisez les horloges de Z1… Z3 et S. (Note : pour indiquer qu'un flot F est sur l'horloge de base on peut écrire indifféremment « F : int » ou « F : int when true »).

Correction

```
    node prog1 (   B1, B2 : bool ;
                   H1 : bool when B1 ;
                   H2 : bool when B2 ;
                   X1 : int when H1 ;
                   X2 : int when H2 ;
                   Y : int when (B1 and B2))
    returns (S : int when true) ;
    var  Z1:int when true;
         Z2:int when true;
         Z3:int when (B1 and B2);
    let
         Z1 = current(current(X1)) ;
         Z2 = current(current(X2)) ;
         Z3 = (Z1 + Z2) when (B1 and B2) ;
         S = current(Z3 + Y) ;
    tel.
```

Explication : l'important dans ce programme est de comprendre les horloges des flots d'entrée.

- B1 et B2 sont sur l'horloge de base, ils sont reçus à chaque top de l'horloge de base (avec une valeur qui peut être vraie ou fausse)
- H1 est reçu que lorsque B1 est vraie, donc H1 a comme horloge B1
- H2 est reçu que lorsque B2 est vraie, donc H2 a comme horloge B2
- X1 est reçu que lorsque H1 est vraie, donc X1 a comme horloge H1
- X2 est reçu que lorsque H2 est vraie, donc X2 a comme horloge H2
- Y est reçu que lorsque B1 et B2 sont vraies en même temps, donc Y a comme horloge B1 and B2.

A propos de Z1 :
Current(X1) sur-échantillonne X1 sur l'horloge de son horlge. Hors, l'horloge de X1 est H1, et l'horloge de H1 est B1. Donc current(X1) est a comme horloge B1. Du coup, current(current(X1)) sur-échantillonne current(X1) sur l'horloge de son horloge. L'horloge de current(X1) est B1, l'horloge de B1 est la base, donc current(current(X1)) est sur la base.

Idem pour Z2.

A propos de Z3 :
Dans l'équation Z3, on fait Z1+Z2. Il faut donc vérifier qu'on a le droit de faire Z1+Z2, c'est-à-dire que Z1 et Z2 sont du même type (ce qui est le cas) et que Z1 et Z2 ont la même horloge (ce qui est le cas aussi). Ensuite, on applique un when. Il faut donc vérifier que Z1+Z2 et B1 and

B2 ont la même horloge, ce qui est le cas (c'est l'horloge de base). Donc le flot Z3 peut être calculé, et son horloge est B1 and B2.

A propos de S :
Comme pour Z3, il faut vérifier que Z3+Y sont de même type et ont la même horloge, ce qui est le cas. Donc S peut être calculé. Le current ramène sur l'horloge de l'horloge de Z3+Y. L'horloge de Z3+Y est B1 and B2. L'horloge de B1 and B2 est la base. Donc l'horloge de current(Z3+Y) est la base. Donc S a comme horloge la base.

Question 3 Un logiciel avionique
Voir commentaires faits en cours

```
const a : real;
cont : maxDeltaPS : real;

node abs(x:real)returns (absx:real)
let
     absx = if x>0 then x else -x;
tel

node never(x:bool)returns (ok:bool)
let
     ok = (not x) -> (not x and pre(OK));
tel

node BaroSimple(PS : real; P0 : real) returns (Z : real)
let
     Z = (PS/P0)**a;
tel

node Baro (PS : real; P0 : real) returns (Z : real; BZ : bool)
let
     BZ = (PS >= P0)
          and (P0 > 0)
          true -> (abs(PS - pre(PS)) <= maxDeltaPS);

     Z = if never(BZ) then 0.
          else current(BaroSimple((PS, P0) when BZ));
tel.
```

Commentaires additionnels :

- le « ** » n'existe pas en Lustre. Il faudrait le coder, ce qu'on le fait pas ici (et qui n'était pas demandé dans l'exam) par souci de simplicité.
- Ici, par rapport à la solution discutée en cours, on utilise un nœud appelé BaroSimple qui ne fait aucune vérification sur la validité des entrée. En revanche, on fait cette vérification dans le nœud Baro, qui appelle BaroSimple en appliquant le when BZ sur ses entrées. Ce qui revient à n'appeler BaroSimple que lorsque BZ est vrai.

Rappel : ici, ce que l'on veut, c'est ne pas faire le calcul (PS/P0)**a lorsque les conditions exprimées dans l'expression BZ sont fausses. Faire ce calcul lorsque BZ est faux risque de provoquer une erreur dans le processeur et un arrêt de celui-ci. C'est pour ça qu'on applique le when sur les entrées de BaroSimple et non sur sa sortie.

Question 4 Un contrôleur de train d'atterrisage

On souhaite écrire en Lustre le contrôleur des trains d'atterrissage d'un avion. Les trains sont composés de portes (appelées doors), qui ferment le logement dans lequel sont placés les trains lorsqu'ils sont remontés, et des trains eux-mêmes (appelés gears).

Ce contrôleur reçoit

- Un ordre de montée ou de descente des trains (flot booléen order_down : égal à vrai lorsque le pilote de l'avion demande la sortie du train, et faux lorsqu'il demande la montée du train).
- La position des portes :
    - flot booléen doors_locked_open : égal à vrai pour dire que les portes sont ouvertes et verrouillées, et égal à faux pour dire que les portes (ou au moins une d'entre elles) ne sont pas ouvertes ou ne sont pas verrouillées en position ouverte.
    - flot booléen doors_locked_closed : égal à vrai pour dire que les portes sont fermées et verrouillées, et égal à faux pour dire que les portes (ou au moins une d'entre elles) ne sont pas fermées ou ne sont pas verrouillées en position fermée.
- La position des trains :
    - flot booléen gears_locked_down : égal à vrai pour dire que les trains sont baissés et verrouillés, et égal à faux pour dire que les trains (ou au moins un d'entre eux) ne sont pas baissés ou ne sont pas verrouillés en position basse.
    - flot booléen gears_locked_up : égal à vrai pour dire que les trains sont remontés et verrouillées, et égal à faux pour dire que les trains (ou au moins un d'entre eux) ne sont pas remontés ou ne sont pas verrouillées en position haute.

La logique est la suivante :

- Lorsque l'ordre de descente des trains est vrai, si les trains ne sont pas déjà en position basse verrouillée, le contrôleur ordonne l'ouverture des portes, puis lorsque les portes sont ouvertes et verrouillées, alors il arrête d'ordonner l'ouverture des portes, et il ordonne la descente des trains, puis lorsque les trains sont verrouillés bas, alors il arrête d'ordonner la descente des trains.
- Inversement, lorsque l'ordre de descente des train est faux (dans ce cas, il faut les rentrer), si les trains ne sont pas déjà en position haute verrouillée et si les portes ne sont pas déjà en position fermée verrouillée, le contrôleur ordonne la montée des trains, puis lorsque les trains sont verrouillés haut, alors il arrête d'ordonner la montée des trains, puis il ordonne la fermeture des portes, puis lorsque les portes sont fermées et verrouillées, alors il arrête d'ordonner la fermeture des portes.

Attention, le contrôleur doit ne pas violer certaines propriétés d'intégrité : si les portes ne sont pas verrouillées ouvertes (c'est-à-dire si elles sont soit fermées soit quelque part entre fermées et ouvertes), alors il ne faut pas bouger les trains, sinon il risque d'y avoir une collision entre les trains et les portes. Il faut d'abord ouvrir et verrouiller les portes, avant de bouger les trains.

Pour ce faire, le contrôleur produit les flots booléens :

- open : égal à vrai pour ordonner l'ouverture des portes
- closed : égal à vrai pour ordonner la fermeture les portes
- down : égal à vrai pour ordonner la descente des trains
- up : égal à vrai pour dire pour ordonner la montée des trains.

Il faut voir ces flots comme des commandes continues sur un système hydraulique. Par exemple, si on veut ouvrir les portes, il faut positionner open à vrai tant que les portes ne sont pas ouvertes. Lorsque open redevient faux, la pression hydraulique sur les vérins d'ouverture

des portes cesse, et donc l'ouverture des portes cesse aussi. Autrement dit, si on veut ouvrir une porte, il faut que open soit vrai jusqu'à ce que les portes soient ouvertes et verrouillées.

Ecrire en Lustre ce controleur.

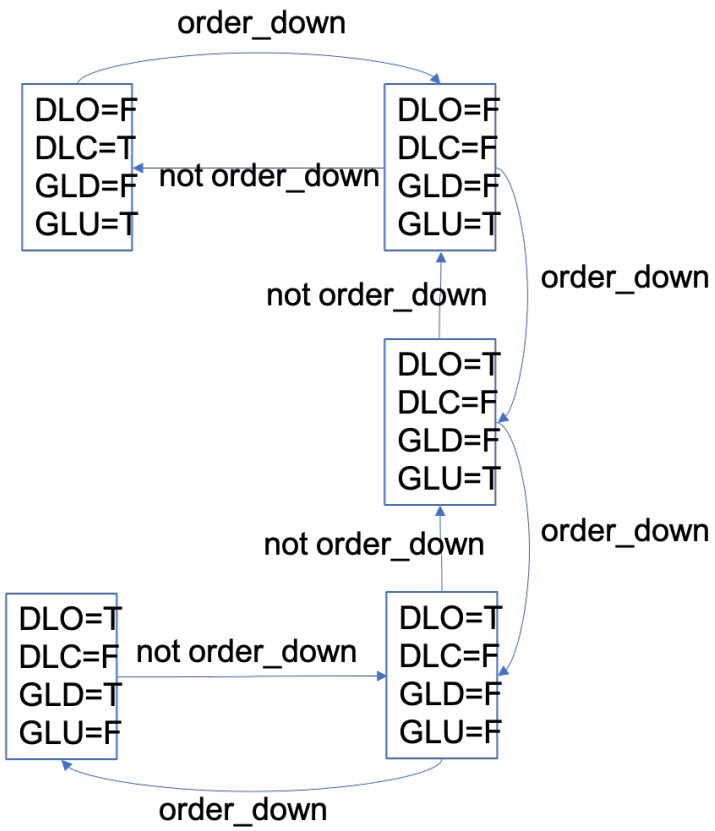
Correction

```
node controller  (order_down : bool ;
                  doors_locked_open : bool ;
                  doors_locked_closed : bool ;
                  gears_locked_down : bool ;
                  gears_locked_up : bool ;)
returns (open, closed : bool;
         down, up : bool; )
var …
let
     open = ( (not doors_locked_open)
               and order_down
               and (not gears_locked_down)) ;

     closed = ( (not doors_locked_closed)
               and (not order_down)
               and gears_locked_up) ;

     down = ( (not gears_locked_down)
               and order_down
               and doors_locked_open) ;

     up = ( (not gears_locked_up)
               and (not order_down)
               and doors_locked_open) ;
tel.
```

Puis, par un petit raisonnement, essayer de montrer que les trains sont toujours dans un état verrouillé (en bas ou en haut) lorsque les portes ne sont pas verrouillées ouvertes.
On suppose

- qu'il n'y a pas de panne matérielle,
- qu'il y a uniquement deux états de départ possibles : trains verrouillés haut et portes verrouillées fermées, ou trains verrouillés bas et portes verrouillées ouvertes,
- et que lorsqu'on ordonne l'ouverture (resp. fermeture) d'une porte qui était verrouillée fermée (resp. ouverte), alors elle devient déverrouillée et elle finit par devenir verrouillée ouverte (resp. fermée),
- et de même lorsqu'on ordonne la descente (resp. montée) d'un train qui était verrouillé haut (resp. bas), alors il devient déverrouillé et il finit par devenir verrouillé bas (resp. haut).

Correction : une façon de raisonner est d'explorer l'ensemble des états possibles à partir des deux états initiaux possibles


order_down
DLO=F
DLC=T
GLD=F
GLU=T
not order_down
DLO=F
DLC=F
GLD=F
GLU=T
order_down
not order_down
DLO=T
DLC=F
GLD=F
GLU=T
order_down
not order_down
DLO=T
DLC=F
GLD=T
GLU=F
not order_down
DLO=T
DLC=F
GLD=F
GLU=F
order_down