Examen
Langages pour le temps réel : le langage LUSTRE
ENSEEIHT 3SN 2022-2023

IMPORTANT :
- Vous répondrez dans l'énoncé, que vous rendez ensuite à l'issue de l'exam.
- Vous ne dégraferez pas les feuilles.
- Vous marquerez votre nom sur la dernière feuille (et uniquement sur celle-ci) à l'endroit indiqué.
- Et vous replierez la dernière feuille le long des pointillés de manière à masquer votre nom.

Question 1: 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 (S11, S12 : int; 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);
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 $X = (1, 1, 1, 1, 1, 1 \dots)$. Quelle sont les valeurs de :

accumulateur(X), B1, S11, S12, B2, S21, S22 ?

On répondra par un chronogramme. Vous répondrez directement sur la page suivante. Si vous souhaitez ajouter un commentaire, faites les également sur la page.

Réponse à la question 1 (petite question de compréhension)

Chronogramme à compléter

X	1	1	1	1	1	1	1	1	1	1	1	1
accumulateur(X)												
B1												
accumulateur(X when B1)												
accumulateur(X) when B1												
S11 = current(accumulateur(X when B1))												
S12 = current(accumulateur(X) when B1)												
B2												
accumulateur(X when B2)												
accumulateur(X) when B2												
S21 = current(accumulateur(X when B2))												
S22 = current(accumulateur(X) when B2)												

Commentaires (optionnel) :

Question 2: 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 ...;
        Z4: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... Z4 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").

Vous répondrez sur la page suivante. Vous préciserez l'horloge de tous les flots (y compris les flots d'entrée)

Réponse à la question 2 (calcul d'horloge)

Programme à compléter

clk(B1) =
clk(B2) =
clk(H1) =
clk(H2) =
clk(X1) =
clk(X2) =
clk(Y) =

clk(Z1) =
clk(Z2) =
clk(Z3) =
clk(Z4) =

clk(S) =

Explications :

Question 3: Un logiciel avionique.

On souhaite calculer l'altitude barométrique d'un avion par la formule

$$Z = \left(\frac{PS}{P0}\right)^a$$

où PS est la pression statique mesurée (par un capteur) à l'altitude de l'avion et $P0$ la pression au niveau de la mer (donnée par le contrôle aérien), et a une constante (dont la valeur n'est pas donnée ici).

Le calculateur qui calcule Z reçoit en entrée PS et $P0$, et produit en sortie Z et un booléen de validité BZ qui doit être vrai en fonctionnement normal et faux lorsqu'une anomalie est détectée.

On considère qu'on est en fonctionnement normal si PS est supérieure à $P0$, si $P0$ est strictement positive, et si la différence entre deux valeurs successives de PS ne dépasse pas en valeur absolue un seuil appelé *maxDeltaPS*, c'est-à-dire:

$$\forall n \textit{ instant d'échantillonnage}, PS_n \geq P0_n > 0 \textit{ et } \mid PS_n - PS_{n+1} \mid \leq \textit{maxDeltaPS}$$

Toute situation inverse (qui ne satisfait pas cette formule) est considérée comme anormale.

Il est demandé d'écrire un programme Lustre, appelé Baro, qui

- lorsqu'on est en fonctionnement normal, calcule Z selon la formule ci-dessus, et retourne true pour le booléen de validité BZ ;
- lorsqu'on est en fonctionnement anormal, retourne pour Z la dernière valeur correcte de Z, sans provoquer une erreur d'exécution, et retourne false pour le booléen de validité BM.

Le squelette de ce programme est donné en page suivante. Complétez ce programme directement sur la page. Si vous avez besoin de plus de place pour répondre à la question, utilisez le verso de la page de réponse.

Réponse à la question 3 (un logiciel avionique)

Programme à compléter

```
const a : real;
const maxDeltaPS : real;
node Baro (
        PS : real;
        P0 : real)
returns (
        Z : real;
        BZ : bool; )
var     \\ (optionnel)


let




tel.
```

Question 4: Un contrôleur d'atterrissage.

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 (appelé gears).

L'interface de ce controleur en LUSTRE est donnée par :

```
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;  )
```

Il 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.

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.

La logique est la suivante :

- Lorsque l'ordre de descente des train 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.

Question 4.1 : Ecrire en Lustre ce contrôleur (sur la feuille en annexe).

Question 4.2 : On veut vérifier la propriété d'intégrité ci-dessous, c'est-à-dire les trains ne doivent être manoeuvrés (montée ou descente) que si les portes sont verrouillées en position ouverte. Exprimer cette propriété en Lustre dans un noeud "verif" (sur la feuille en annexe).

Réponse à la question 4.1 (un contrôleur d'atterrissage)

Programme à compléter

```
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     \\ (optionnel)



let



tel.
```

Réponse à la question 4.2 (verif)

Programme à compléter

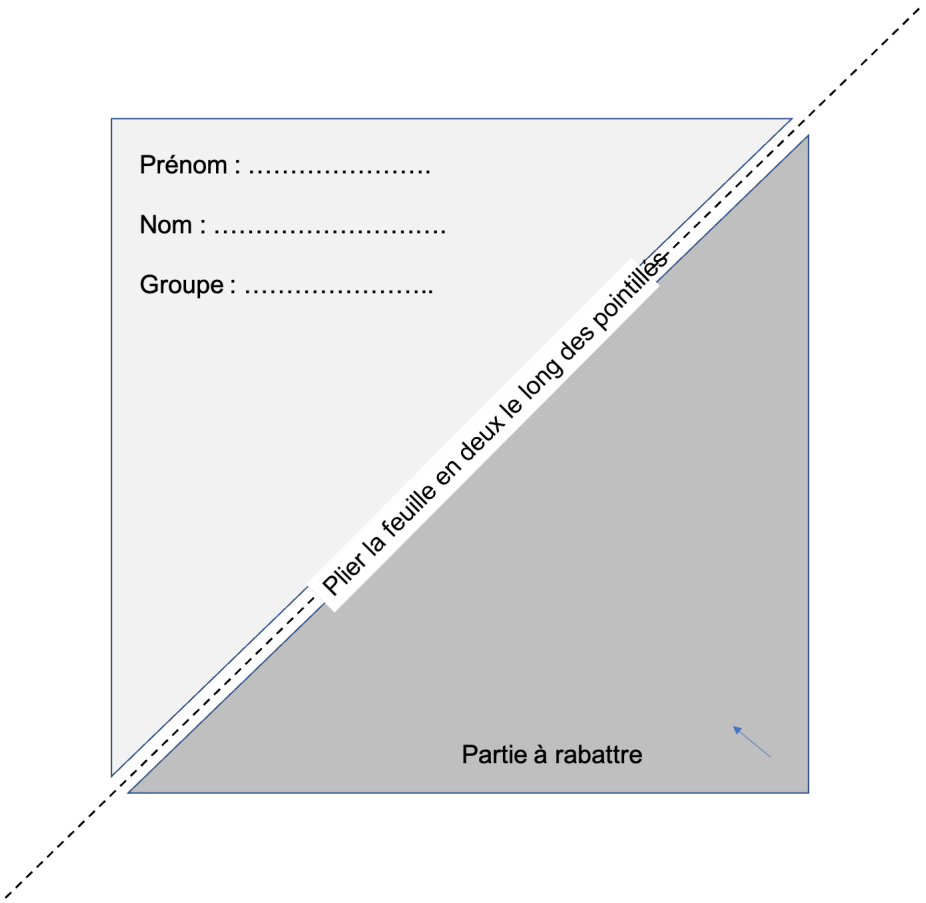
```
node verif  (
        order_down : bool ;
        doors_locked_open : bool ;
        doors_locked_closed  : bool ;
        gears_locked_down : bool ;
        gears_locked_up  : bool ;)
returns (
        B : bool; )
var
        open, closed  : bool;
        down, up : bool;


let

        (open, closed , up, down) = controller(order_down,
                                               doors_locked_open, doors_locked_closed ,
                                               gears_locked_down, gears_locked_up);

        B =

tel.
```


Prénom : ……
Nom : ……
Groupe : ……..
Plier la feuille en deux le long des pointillés
Partie à rabattre