TD
Année 2013

Systèmes synchrones
M. Pouzet

Exercices

1 Additionneur série

Ecrire un additionneur série 1-bit `somme(x,y)` qui calcule la somme `z` des deux nombres `x` et `y` représentés en binaire (poids faible en premier et tel que pour tout instant n, les n-premiers bits de z représentent la somme des n premiers bits de x et y).

Montrer, en utilisant Lesar que $\texttt{somme}(x, x) = 2x$.

2 Alarme de maison

Il s'agit de protéger une habitation divisée en deux parties: le garage et la maison. La maison est toujours sous surveillance dès que le dispositif de protection est mis sous tension. Le garage n'est sous protection qu'au bout d'un temps "delai-vigilence" à partir du moment où le dispositif est sous tension; ce délai permet au propriétaire d'avoir le temps de sortir du garage après avoir armé son dispositif de protection.

fonctionnement de l'alarme à partir du moment où elle sonne, elle le fait pendant "délai-alarme" puis s'arrête pendant "délai-reprise" et à la fin de ce dernier délai, selon la situation (présence ou non d'un intrus dans l'habitation) sonne à nouveau ou non.

arrêt du dispositif lorsque celui-ci est sous tension, on ne peut l'arrêter qu'après avoir fourni le code d'entrée. On dispose alors de "delai-vigilence" entre le moment où un code est fourni et la mise hors-circuit pendant lequel une présence dans le garage est tolérée.

entrées et sorties les entrées et sorties du système sont les suivantes:

MA bouton de mise en marche ou d'arrêt. Chaque pression sous le bouton fait changer d'état le dispositif, qui est supposé arrêté initialement.

code signal booléen devenant vrai lorsqu'un code correct a été fourni.

pb-gar signal vrai lorsqu'une présence est détectée dans le garage.

pb-hab signal vrai lorsqu'une présence est détectée dans la maison.

en-marche témoin de l'état du dispositif.

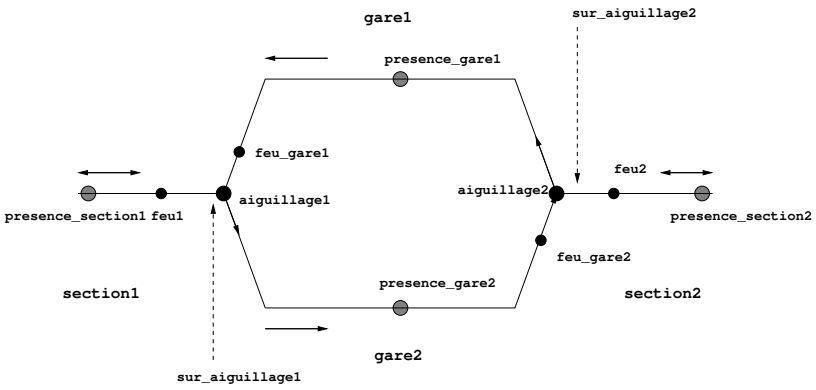
alarme sans commentaire.

constantes les identificateurs **delai-alarme**, **delai-reprise** et **delai-vigilence** désignent des constantes.

Programmer ce système dans un langage au choix (Lustre, Lucid Synchrone ou Esterel).

3 Controle d'aiguillages

On souhaite modéliser un système commandant des aiguillages permettant à des trains circulant sur une voie unique à double sens de se croiser. Le système est décrit par le schéma suivant:



Chacun des composants de ce système est décrit ci-dessous.

- La `section1` et la `section2` sont des voies uniques où les trains ne peuvent se croiser.
- Les sections `gare1` et `gare2` sont également des voient uniques. On suppera que les trains présents sur la `gare1` vont toujours de droite à gauche alors que les trains sur la `gare2` vont toujours de gauche à droite. Les trains voulant passer de la `section1` à la `section2` passent donc nécessairement par la `gare2`.
- Un système d'aiguillages `Aiguillage1` et `Aiguillage2` permet de connecter les sections entre elles. Ces aiguillages ont chacun deux positions, et peuvent être commandés par des action `actionner_haut` et `actionner_bas`.
- Le système contient également quatre feux de signalisation, `feu1`, `feu2`, `feu_gare2` et `feu_gare1`.
- Enfin, on suppose que le système dispose de plusieurs indicateurs permettant de savoir si un train est présent dans une des sections.
  - `presence_section1` indique qu'un train est présent dans la section `section1` (de même pour `presence_section2`).
  - `presence_sur_section_aiguillage1` indique qu'un train est présent sur la zone du premier aiguillage (de même pour le deuxième aiguillage).
  - `presence_gare1` indique qu'un train est présent dans la section `gare1` (de même pour la section `gare2`).

On s'intéresse au contrôle des aiguillages et des feux de signalisation. L'objectif est de définir dans un premier temps, un module ayant l'interface suivante:

```
node system
  (demande_actionner_aig1_haut,
   demande_actionner_aig1_bas,
   demande_actionner_aig2_haut,
   demande_actionner_aig2_bas:bool)
returns (aiguillage1_haut,aiguillage1_bas,
         aiguillage2_haut,aiguillage2_bas,
         feu1,feu2,feu_gare1,feu_gare2:bool);
```

et envoyant des commandes aux deux aiguillages ainsi qu'aux quatres feux de position.

Question 1 Dans un premier temps, on souhaite seulement commander les deux aiguillages `aiguillage1` et `aiguillage2` sans se préoccuper des feux. Définir le corps de ce système à partir d'un module de commande d'un aiguillage.

Question 2 On ajoute les feux `feu1` et `feu2`. Le feu `feu1` doit être allumé tant que l'aiguillage `aiguillage1` est vers le haut. De même pour le feu `feu2`.

Question 3 On ajoute les deux feux `feu_gare1` et `feu_gare2`. Le feu `feu_gare1` doit être allumé lorsque l'aiguillage `aiguillage1` est en bas.

Question 4 Vous devez maintenant tenir compte de la contrainte suivante: un seul train (au plus) peut être présent dans la section `gare1` ou dans la section `gare2`. Pour cela, vous suppposerez que le système `system` est connecté aux divers capteurs présents sur les voies.

Question 5 Même question en tenant compte cette fois de la contrainte suivante: un seul train (au plus) ne peut être présent dans la section d'un aiguillage. On ne peut donc changer l'état de l'aiguillage que s'il n'y a pas déjà de train dans la zone de l'aiguillage (sinon, celui-ci peut dérailler).

Question 6 Afin d'obtenir un système robuste insensible aux délais de transmission, on introduit un mécanisme de programmation par confirmation pour la commande des aiguillages. Son principe est d'émettre un signal de `debut_de_demande_aig_haut` et de maintenir la demande jusqu'à ce qu'un signal de confirmation `fin_de_demande_aig_haut` soit émis. Modifiez le système en conséquence.

Question 7 On souhaite maintenant implanter un autre système de controle de la présence dans la section `gare1`. Pour cela, on suppose qu'il y a deux capteurs `presence_gare1_debut` et `presence_gare1_fin`, placés en début et en fin. Définir un noeud:

```
node PresenceGare1(presence_gare1_debut,presence_gare1_fin:bool)
return (presence_gare1:bool)
```

qui indique si un train est présent dans la gare. Ajoutez ce module au système déjà défini.

Question 8 Les deux propriétés générales que l'on démontre pour ce système sont qu'un train ne peut dérailler et qu'il ne peut y avoir de collision. Pour cela, il est nécessaire de faire des hypothèses sur l'environnement (le train). Décrire, sous forme d'assertions ajoutées au contrôleur Lustre, les hypothèses suivantes:

- au premier instant, il y a au plus un train dans chacune des gares.
- le train respecte les feux.

4 Machine à café

L'objectif de cet exercice est de programmer le contrôleur d'une machine à café. Il dispose des entrées suivantes:

cafe, grand_cafe, the permettent de sélectionner une boisson.

annuler ce bouton permet d'annuler la commande et de vider le monnayeur si des pièces de monnaie ont été données.

Cette machine a café permet de commander plusieurs boissons et ne rend la monnaie que lorsque le bouton **annuler** est entré.

pièce permet d'introduire des pièces de monnaie. On supposera ici que les seules pièces possibles sont des pièces de 10cts et 20cts.

pret indique au contrôleur que la boisson demandée est prête.

miliseconde est un signal vrai toutes les milisecondes

Les sorties de cette machine sont définis ci-dessous:

preparer indique que la boisson demandée doit être préparée (cette information contrôle le mécanisme de fabrication)

sonnerie lorsque la boisson est prête, un signal sonore est émis.

boisson indique que la boisson est en train d'être préparée

monnaie elle affiche la monnaie introduite dans la machine.

vider_monnaie permet de vider le monnayeur.

Le prix des consommations est le suivant:

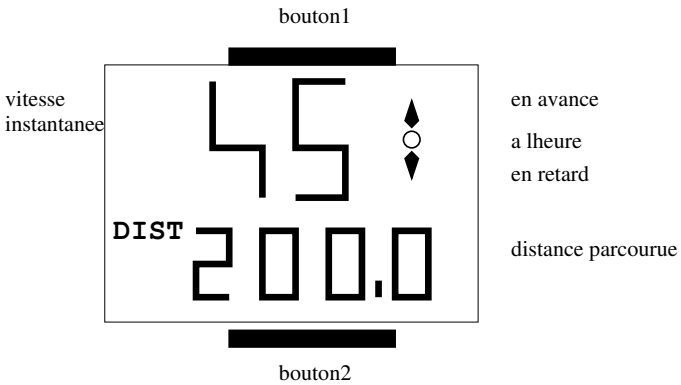
- un café coute 40cts, un grand café coute 1E et un thé coute 50cts.

Le fonctionnement de cette machine est le suivant: le consommateur introduit des pièces dans la machine puis sélectionne sa boisson. Le voyant `boisson` est alors allumé. Celui-ci s'éteint et le signal sonore `sonnerie` est émis pendant 5 secondes. Si le consommateur appuie sur le bouton `annuler` avant d'avoir sélectionné sa boisson, sa monnaie lui est rendue. Si les bouton `cafe`, `grand_cafe` ou `the` sont enfoncés avant que l'utilisateur ait introduit sa monnaie, le signal `sonnerie` est émis pendant 1 seconde. Plusieurs boissons peuvent être commandées à la suite lorsqu'il y a suffisamment de monnaie.

Programmer ce contrôleur dans un langage synchrone de votre choix.

5 Compteur de bicyclette

L'objectif de cet exercice est de programmer le contrôleur d'un compteur kilométrique de bicyclette. L'interface de ce compteur est décrite ci-dessous:



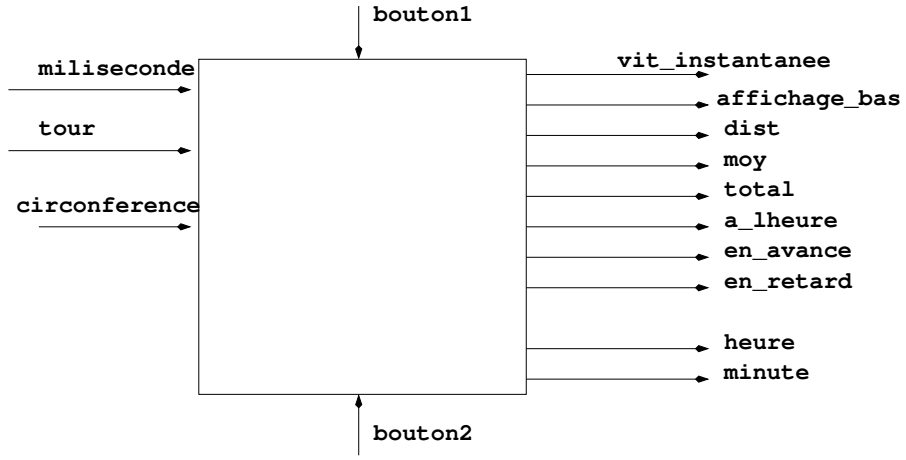
- Le compteur possède deux boutons **bouton1** et **bouton2**.
- Il affiche en permanence la vitesse instantanée
- Il affiche en permanence une information permettant de savoir si le cycliste est à l'heure, en avance ou en retard. Cette information est obtenue en comparant la vitesse instantanée avec la vitesse moyenne depuis la dernière remise à zéro.
- Sur le cadran du bas, le compteur affiche l'une seulement des informations suivantes:

- la vitesse moyenne depuis la remise à zéro. Dans ce cas, l'information `MOY` apparaît.
- la distance parcourue depuis la remise à zéro. Dans ce cas, l'information `DIST` apparaît.
- la distance totale. Dans ce cas, l'information `TOT` apparaît. Cette distance n'est pas remise à zéro (sur un compteur, il y a un bouton "caché" et un mode permettant de réinitialiser cette distance).
- l'heure sous la forme `heure:minute` (sur 4 chiffres)

On passe séquentiellement d'un affichage à l'autre (suivant l'ordre donné ci-dessus) en enfonçant le bouton du bas (**bouton2**)

- la remise à zéro s'effectue en enfonçant les boutons **bouton1** et **bouton2**.

L'objectif de cet exercice est de programmer le contrôleur de ce compteur kilométrique en Lustre. Le contrôleur est représenté par le schéma suivant:



et dont l'interface Lustre est donnée par:

```
node controleur(miliseconde, tour, bouton1, bouton2: bool;circonference: int)
     returns (vitesse_instantanee, affichage_bas:int;
              dist, moy, total: bool;
              en_avance, en_retard, a_lheure: bool);
```

Question 9 Ecrire un noeud Lustre chargé de calculer la vitesse instantanée. La vitesse instantanée est donnée par le temps écoulé entre deux tours de roues. On suppose pour cela que le temps écoulé entre deux occurrences vraies du signal `miliseconde` est égal à la miliseconde et que la circonférence est donnée en centimètre. La vitesse instantanée doit être donnée en kilomètres par heure. Le signal `tour` est vrai à chaque tour de roue.

Question 10 Que peut-il arriver avec le calcul précédent? Proposer une formulation permettant de lisser les éventuels problèmes de capteurs.

Question 11 Ecrire un noeud Lustre de calcul de la vitesse moyenne. Vous devrez être attentif aux problèmes de débordement. On supposera que la vitesse moyenne est inférieure à `vitesse_max`.

Question 12 On souhaite indiquer si le cycliste est en avance, en retard ou à l'heure. Pour cela, on compare la vitesse instantanée avec la vitesse moyenne. Le cycliste est en avance lorsque la vitesse instantanée est supérieure à la vitesse moyenne plus un; il est en retard lorsque la vitesse instantanée est inférieure à la vitesse moyenne moins un; sinon, il est dit à l'heure.

Ecrire le noeud Lustre correspondant produisant les trois sorties nécessaires.

Question 13 Ecrire un noeud de calcul de l'heure (`heure/minute`). Ce noeud a la signature suivante:

```
node heure(heur0, minute0: int; set: bool;
           mili: bool) returns (heure, minute: int);
```

`heur0` et `minute0` sont les valeurs d'initialisation et doivent être prises en compte lorsque `set` est vrai.

Question 14 Le compteur affiche en permanence la vitesse instantanée. Par contre, l'écran du dessous peut afficher plusieurs informations différences: la vitesse moyenne, la distance parcourue et la distance totale.

L'ecran du bas affiche séquentiellement la vitesse moyenne, la distance parcourue, la distance totale puis la vitesse moyenne, etc. à chaque fois que le bouton du bas est enfoncé. Ecrire un noeud dont l'interface est la suivante:

```
node affichage_sequentiel(bouton: bool;
                          vit_moy, dist_parcourue, dist_totale: int;
                          heure, minut: int)
     returns (affichage: int;
              vit, dist, tot: bool);
```

qui permet de passer d'un affichage à l'autre à chaque fois que `bouton` est vrai. Pour simplifier, on supposera que `affichage` est un entier pouvant contenir l'information `heure/minute` ou bien la vitesse moyenne ou bien la distance parcourue ou bien la distance totale.

Question 15 On peut remettre à 0 le compteur (i.e, réinitialiser la distance parcourue et la vitesse moyenne) lorsque l'on enfonce les deux boutons pendant au moins une seconde.

Ecrire un noeud dont l'interface est la suivante:

```
node raz(mili, b1, b2: bool) returns (ok: bool);
```

Question 16 Ecrire le corps du contrôleur en instanciant les divers noeuds réalisés.

Question 17 (Bonus) On veut pouvoir changer l'heure interne. Pour cela, il est possible d'entrer dans le mode de modification de l'heure en enfonçant pendant au moins cinq secondes les deux boutons. On quitte alors ce mode au bout d'une minute si aucun des deux boutons n'a été enfoncé pendant cette durée.

Dans ce mode, l'heure (`heure/minute`) est affichée. Lorsque l'on est dans ce mode, le bouton du haut permet de passer du mode de modification de l'heure au mode de modification des minutes. Chaque enfoncement du bouton du bas permet d'ajouter respectivement une minute aux minutes et une heure aux heures.

On quitte le mode de modification de l'heure en rappuyant pendant au moins cinq secondes sur les deux boutons.

Modifier le contrôleur en conséquence.

6 Sens de rotation

Soit un disque partagé en trois sections de couleur respective rouge (R), vert (V) et bleu (B), et tournant autour de son axe. Un capteur fixe est situé à la périphérie du disque et voit donc se succéder les différentes couleurs du disque. A partir de la succession de couleurs observées, celui-ci doit décider si le disque est immobile, ou déterminer son sens de rotation.

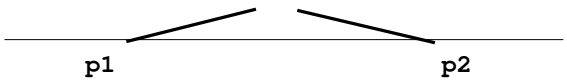
Par convention, on dira que le sens *direct* correspond à une succession "R, V, B, R ...", le sens inverse étant *indirect*. Pour être sûr que le disque est *immobile*, il faut que la couleur observée ne change pas sur au mois trois instants. Il existe des instants (dont l'instant initial) où l'observateur ne peut rien décider et est donc *indéterminé*. La spécification peut être résumée par le tableau suivant (_ désigne les valeurs avant le premier instant):

t-2	t-1	t	sens de rotation
R	R	R	immobile
B	B	B	”
V	V	V	”
_	_	x ∈ {B,V,R}	indéterminé
x ≠ R	R	R	indéterminé
x ≠ B	B	B	”
x ≠ V	V	V	”
x	B	R	sens direct
x	R	V	”
x	V	B	”
x	R	B	sens indirect
x	B	V	”
x	V	R	”

Décrire ce système en Lustre.

7 Sens d'un train

On veut décrire un dispositif qui permette de décider si un train traverse une zone Z de gauche à droite ou de droite à gauche. Pour cela, deux “pédales” p_1 et p_2 sont disposées en quinconce sur la voie. On suppose que l'on détecte le passage d'un essieu sur une pédale.



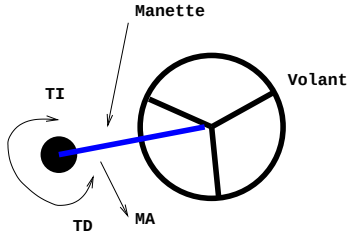
Les entrées du système sont les variables booléennes `p1` et `p2` indiquant le passage d'un train. Les sorties sont les variables booléennes `from_left_to_right` et `from_right_to_left`.

Question 18 Programmer ce système en Lustre.

8 Contrôleur de feux

On souhaite décrire en Lustre le noyau réactif d'un contrôleur de feux de voitures. Le contrôleur reçoit ses entrées via une manette et des boutons, et il contrôle en sortie les lampes des feux de la voiture.

On suppose que les lampes à contrôler sont les veilleuses, les codes et les phares.



Le conducteur dispose d'une manette de contrôle (voir figure). Il peut tourner cette manette dans un sens ou dans l'autre, et tirer sur la manette vers l'avant (elle revient alors immédiatement en arrière).

La manette peut être tournée dans le sens direct (TD) ou indirect (TI). A partir d'une situation initiale où tout est éteint, TD allume les veilleuses, un second TD éteint les veileuses et allume les

codes. Lorsqu'on est en codes ou en phares, TI les éteint et rallume les veilleuses, un second TI éteint tout.

Le fait de tirer la manette vers l'avant (MA) permet de communter entre codes et phares: lorsqu'on est en codes, MA éteint les codes et allume les phares, un second MA éteint les phares et rallume les codes.

Le conducteur ne peut pas simultanément tourner et tirer la manette.

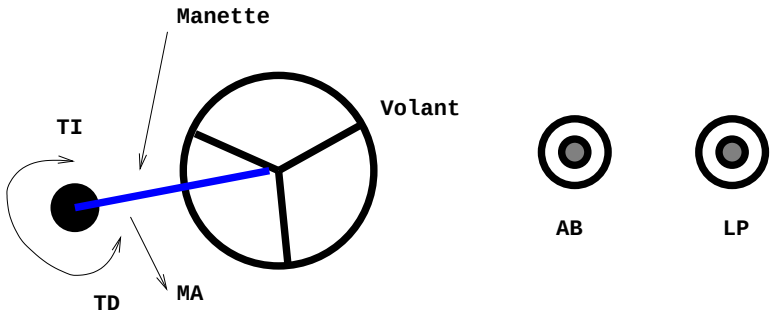
Pour les variables d'entrée, on prendra naturellement trois variables booléennes TD, TI, MA, vraies aux instants où l'action correspondante est effectuée par le conducteur.

Pour les sorties, on choisit trois flots qui représentent l'état des lampes: `veilleuses` (resp. `codes`, `phares`) est vrai tant que les veilleuses (resp. les codes, les phares) sont allumés, et eest faux tant qu'elles sont éteintes.

Question 19 Écrire le programme Lustre qui décrit le noyau du contrôleur.

Contrôleur de feux (version étendue)

On ajoute à présent des projecteurs antibrouillard et de longue portée. Deux boutons-poussoirs permettent de sélectionner ou non les antibrouillards (bouton AB) et les longue portée (bouton LP).



Le rôle de la manette est inchangé. A partir d'une situation initiale, une pression sur AB (resp. LP) sélectionne les antibrouillard (resp. les longues portées), et une seconde pression les désélectionne.

Les antibrouillards (resp. les longues portées) ne sont allumés que quand on est en codes (resp. en phares). Dans ce cas, ils ne sont allumés que s'ils sont sélectionnés.

On conserve les entrées et sorties de la version simple, auxquelles on ajoute:

- deux entrées `AB` et `LP` vraies à chaque pression du bouton correspondant;
- deux sorties `anti_b` et `longue_p` représentant l'état des projecteurs antibrouillard et de longue portée (vrai: allumé, faux: éteint).

Question 20 Ecrire un programme Lustre qui décrit le noyau du contrôleur de la version étendue.

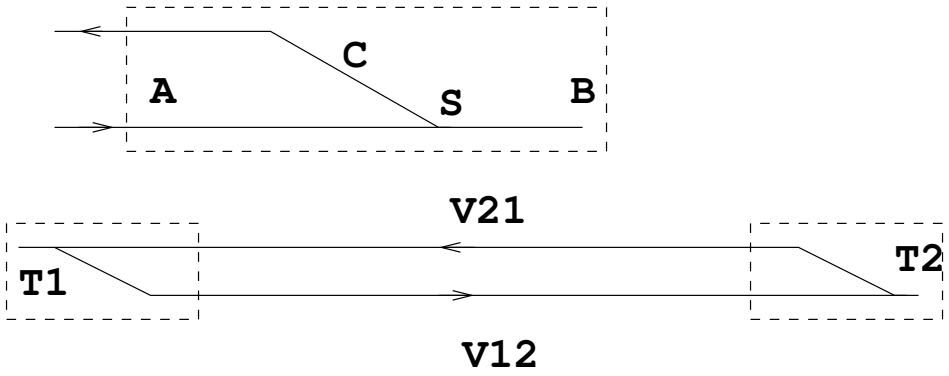
Question 21 Décrire les propriétés suivantes et faire un noeud permettant de les vérifier en utilisant Lesar:

- veilleuse, code et phare sont exclusifs.
- on ne peut être en antibrouillard que si on est en code.
- on ne peut être en longue portée que si on est en phares.

Quelles sont les hypothèses à faire sur les entrées pour que ces propriétés soient vérifiées? Ecrire les assertions correspondantes.

9 Terminus de métro

L'objectif de cet exercice est de décrire un dispositif ferrovière employé sur les lignes de métros.



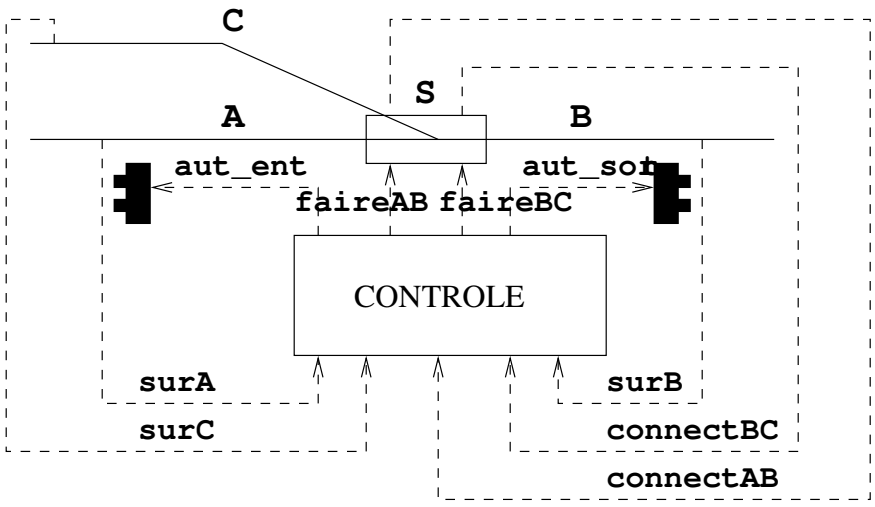
De façon shématique, un ligne de métro est constituée de deux terminus `T1` et `T2` reliés entre eux par l'intermédiaire de deux voies unidirectionnelles, empruntées par les rames de métro. La voie `V12` permet à celles-ci d'effectuer le trajet de `T1` vers `T2`, tandis que la voie `V21` leur permet d'aller de `T2` vers `T1`.

A chaque terminus, les rames pénètrent dans une "section terminale", qui leur permet de changer de voie pour repartir dans la direction opposée. Cette section est composée de trois voies `A`, `B`, `C` et d'un aiguillage `S`. La voie `A` étant la voie d'entrée dans la section et la voie `C` étant la voie de sortie, les rames allant de `A` vers `C` doivent d'abord attendre que `A` et `B` soient connectées par l'aiguillage `S`, pour transiter ensuite sur `B` et attendre à nouveau que `B` et `C` soient connectées par `S` anvant de ressortir par `C`.

Etant donné que plusieurs rames circulent en même temps sur les voies et que l'aiguillage n'est pas un dispositif sûr (il peut tomber en panne), deux genres d'accidents sont succeptibles de se produire dans une section terminale:

- si plusieurs rames sont autorisées à la fréquenter en même temps, celles-ci risquent d'entrer en collision.
- si l'aiguillage n'est pas verrouillé en position sûre au moment où une rame circule dessus, cette dernière sera victime d'un déraillement.

Il est donc clair que le contrôle d'une section terminale est une tâche critique. Le système de contrôle doit à la fois commander l'aiguillage et gérer la circulation des rames de telle sorte qu'aucun accident ne puisse se produire dans la section. L'objectif ici est de décrire un controleur recevant des informations sur le positionnement de l'aiguillage et la position des rames dans la section. Il doit alors délivrer des requêtes de positionnement à l'aiguillage et des autorisations de circulation aux rames. Graphiquement, le système se présente ainsi:



Il recoit les signaux suivants:

- `connectAB` et `connectBA` sont émis par l'aiguillage pour indiquer si celui-ci connecte `A` à `B` ou `B` à `C`. Si aucun des signaux est émis, l'aiguillage est en position indéterminée et les rames ne doivent pas être autorisées à circuler dessus.
- `surA`, `surB` et `surC` sont des signaux émis par trois capteurs placés sur les voies. Ils sont actifs pendant tout le temps où une rame circule sur la voie qu'ils surveillent. Ils indiquent donc la présence sur la voie.
- `faireAB` et `faireBC` sont des signaux de requête émis par le système pour ordonner à l'aiguillage de connecter `A` à `B` ou `B` à `C`.
- `aut_ent` et `aut_sort` sont des signaux d'autorisation de circulation des rames à l'intérieur de la section. Ils correspondent en pratique à des signaux lumineux bicolores. Le premier signal doit autoriser les rames à pénétrer dans la section terminale seulement si elle est vide et si l'aiguillage connecte `A` à `B`. Le second doit autoriser les rames à quitter la voie `B` seulement si l'aiguillage connecte `B` à `C`.

On suppose qu'initialement, il n'y a pas de rames dans la section terminale. Il sera demandé à l'aiguillage de connecter `A` à `B` à chaque fois que la section est vide, et de connecter `B` à `C` à chaque fois qu'une rame se trouve en transit sur `B`. En outre, ces requêtes devront être maintenues tant qu'elles ne sont pas satisfaites.

Question 22 Décrire le système de contrôle. Son interface sera la suivante:

```
node controle(surA,surB,surC,connectAB,connectBC:bool)
returns (aut_ent,aut_sort,faireAB,faireBC:bool);
```

On souhaite maintenant décrire les propriétés que l'environnement vérifie. Ces propriétés peuvent être décrites par des expressions Lustre vraies à chaque instant. Ces expressions sont alors rassemblées pour former des *assertions*. Si e est une expression Lustre, alors `assert(`e`)` affirme que e est troujours vrai.

Question 23 Exprimer les propriétés suivantes:

- L'aiguillage ne peut à la fois connecter `A` avec `B` et `B` avec `C`.
- A partir du moment où il est dans une position donnée, l'aiguillage ne change pas de position spontanément, à moins que le système émette une commande lui ordonnant d'atteindre la position opposée à celle couramment occupée

- Initialement, il n'y a pas de rame dans la section
- Les trains respectent les feux de signalisation. Cela peut se traduire par le fait qu'à l'instant où une rame entre dans la section, alors le feu de signalisation correspondant était vert à l'instant précédent. On peut faire le même raisonnement pour une rame sortant de la section.
- Quand un train quitte `A`, il se retrouve sur `B`. Quand un train quitte `B`, il se retrouve soit sur `A`, soit sur `C`. Ceci correspond au fait qu'une rame ne peut disparaître subitement de la section où elle se trouve !
- Rassembler ces propriétés dans une assertions.

On souhaite décrire maintenant les propriétés que doit respecter le système de contrôle.

Question 24 Exprimer les propriétés suivantes:

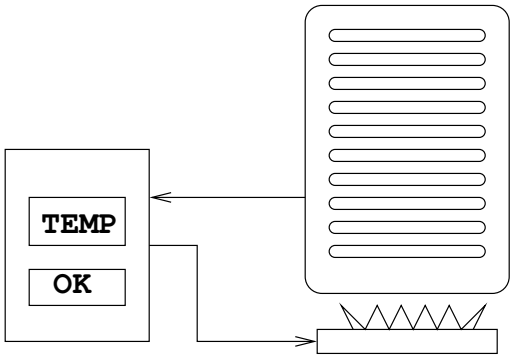
- Le premier risque d'accident concerne la collision des rames. Il faut contrôler qu'une rame n'est autorisée à pénétrer dans la section qu'à la seule condition que cette dernière soit vide. Définir le flot `non_collision` correspondant.
- Exprimer le fait que l'aiguillage est correctement commandé, c'est à dire que les requètes de commande ne sont pas activées en même temps. Définir le flot `exclusive_req` correspondant.
- L'aiguillage doit continuellement connecter `A` avec `B` à partir du moment où une rame est autorisée à pénétrer dans la section, et jusqu'à ce qu'elle se trouve en transit sur la voie `B`. Définir le flot `non_derail_AB` correspondant.
- De même, l'aiguillage doit continuellement connecter `B` avec `C` à partir du moment où une rame est autorisée à quitter la section, et jusqu'à ce que cette dernière soit à nouveau vide. Définir le flot `non_derail_BC` correspondant.
- Finalement, le système doit vérifier la conjonction de ces propriétés. Définir le flot `propriete` correspondant.

Question 25 Décrire au moyen d'un observateur, la propriété globale du système. La structure de cet observateur sera:

```
node controle_verif(surA,surB,surC,connectAB,connectBC:bool)
returns (propriete:bool);

-- assertions
.......

(aut_ent,aut_sort,faireAB,faireBC) =
           controle(surA,surB,surC,connectAB,connectBC);

-- proprietes
.......
```

10 Contrôleur de chaudière

L'objectif de cet exércice est de programmer le contrôleur d'une version simplifiée de chaudière domestique à gaz. Cette chaudière dispose d'un ballon d'eau chaude. Le schéma général de l'installation est le suivant: le contrôleur de la chaudière est relié à des capteurs venant de la chaudière qu'il commande à l'aide de diverses vannes. Le fonctionnement d'une chaudière est celui d'un bain-marie: le réseau d'eau froide passe à travers un serpentin dans un bain fermé d'eau chaude. Un capteur mesure la température de ce bain.



Ce contrôleur se charge de l'allumage de la chaudière lorsque le ballon se vide et indique que la chaudière ne fonctionne pas au cas où la température de la chaudière est supérieure à une valeur fixée ou s'il est impossible d'allumer la flamme. Les entrées du contrôleur sont les suivantes:

- `milliseconde: bool` est vrai toutes les millisecondes. `temp_eau:int`: indique la température de l'eau dans le ballon.
- `en_chauffe:bool`: indique que la flamme est allumée.
- `redemarrer:bool`: permet de redémarrer la chaudière lorsque celle-ci est en arrêt de sécurité.

Les sorties du contrôleur sont les suivantes:

- `ouverture_gaz:bool`: commande d'ouverture de l'arrivée du gaz. La commande doit être maintenue pour que la vanne reste ouverte.
- `allumage:bool`: commande d'ouverture de l'allumage (piezo-électique). La commande doit être maintenue jusqu'à ce qu'il y ait un allumage.
- `ok:bool` indique un fonctionnement normal de la chaudière.
- `nok:bool` indique un problème d'allumage du gaz. La chaudière est arrêtée et mise en arrêt de sécurité.

Les constantes du système sont les suivantes:

- `temp_max:int`: température maximale autorisée pour la chaudière. La température de celle-ci ne doit pas dépasser cette limite.
- `haut: int` et `bas`: valeurs utilisées pour éviter les problèmes d'oscillation lors du chauffage de l'eau.

Fonctionnement:

- la température dans le ballon est normalement égale à `temp_normal`. Si elle passe sous cette valeur, l'eau doit être chauffée. Pour éviter un phénomène d'oscillation, on met en place un mécanisme d'hysteresis (avec un seuil bas `temp_normal - bas` et un seuil haut `temp_normal` bas+).
- La production d'eau chaude s'effectue en allumant la flamme et en ouvrant les vannes d'arrivée du gaz. Le gaz est allumé en maintenant la commande d'allumage au plus `1 seconde`. Si le gaz n'a pu être allumé, la procédure est recommancée `3` fois (il s'écoule une demie seconde entre deux essais). Si au bout de ces essais, le gaz n'a pu être allumé, la chaudière s'arrête et l'indicateur `nok` est maintenu allumé.
- La température de la chaudière doit toujours être inférieure à une limite `temp_max`. Sinon, la chaudière s'arrête.

Question 26 Modéliser ce contrôleur en Lustre. Vous pourrez définir un noeud Lustre pour chacuns des composants décrits ci-dessus.