Programmation Lustre

Compléments

Charles André – Université de Nice-Sophia Antipolis

areTheSame

- Soit X un vecteur de n bool
- areTheSame est le prédicat qui prenant X retourne true ssi toutes les composantes de X sont égales.
- Peut s'établir par récurrence:
- Soit Yk le prédicat qui retourne true ssi tous les Xj, pour j≤k sont égaux
- areTheSame = Yn-1

areTheSame (2)

- Y0 = true
- Yk = Yk-1 and (Xk = Xk-1) pour k>0
- *Ce qui se traduit en Lustre par*

```
node areTheSame(const n:int; X:bool^n)
             returns (eq:bool);
var Y:bool^n;
let
    Y[0] = true;
    Y[1..n-1] = Y[0..n-2] and (X[1..n-1]=X[0..n-2]);
    eq = Y[n-1];
tel
```

areTheSame (2)

- Y0 = true
- Yk = Yk-1 and (Xk = Xk-1) pour k>0
- Ce qui se traduit en Lustre par

```
node areTheSame(const n:int; X:bool^n)
              returns (eq:bool);
var Y:bool^n;
let
    Y[0] = true;
    Y[1..n-1] = Y[0..n-2] and (X[1..n-1]=X[0..n-2]);
    eq = Y[n-1];
tel
```

areTheSame (2)

- Y0 = true
- Yk = Yk-1 and (Xk = Xk-1) pour k>0
- Ce qui se traduit en Lustre par

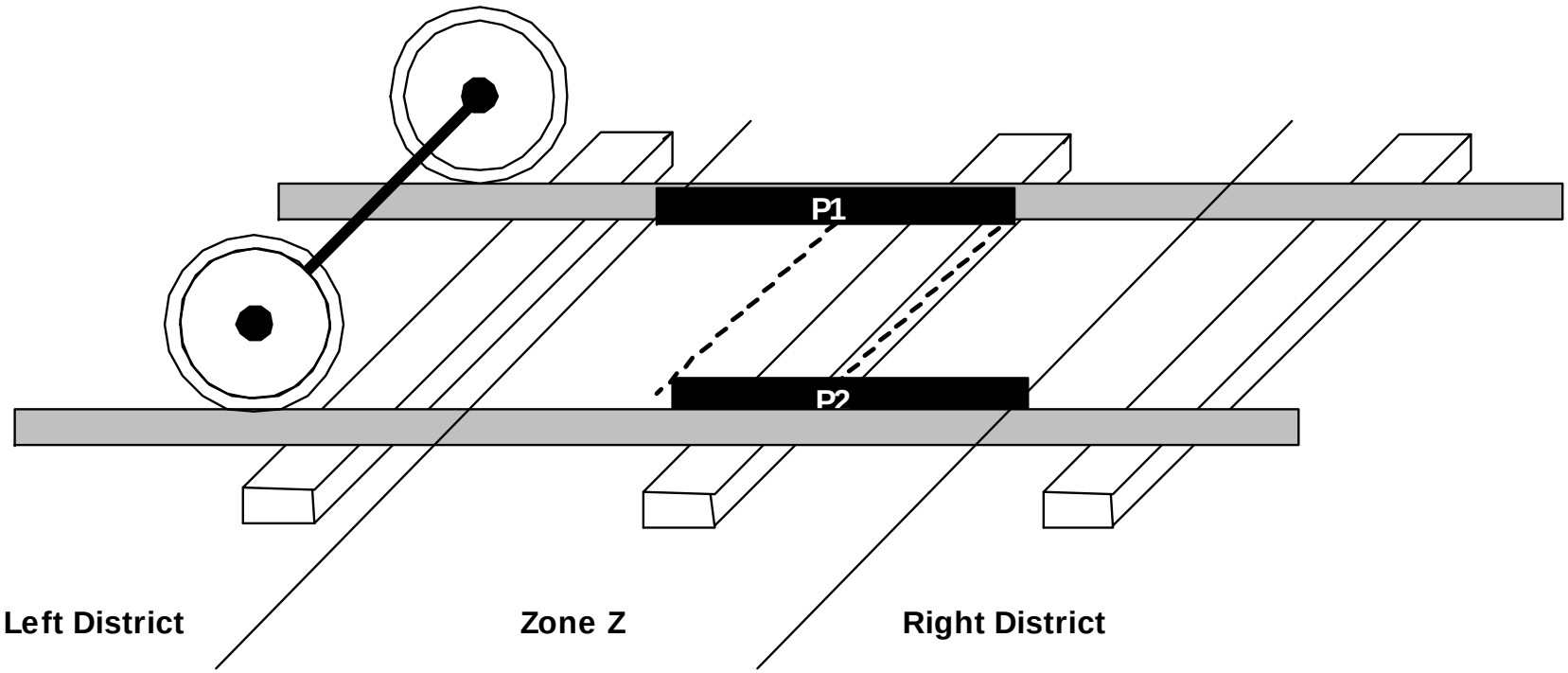
```
node areTheSame(const n:int; X:bool^n)
              returns (eq:bool);
var Y:bool^n;
let
    Y[0] = true;
    Y[1..n-1] = Y[0..n-2] and (X[1..n-1]=X[0..n-2]);
    eq = Y[n-1];
tel
```

areTheSame (2)

- Y0 = true
- Yk = Yk-1 and (Xk = Xk-1) pour k>0
- *Ce qui se traduit en Lustre par*

```
node areTheSame(const n:int; X:bool^n)
             returns (eq:bool);
var Y:bool^n;
let
    Y[0] = true;
    Y[1..n-1] = Y[0..n-2] and (X[1..n-1]=X[0..n-2]);
    eq = Y[n-1];
tel
```

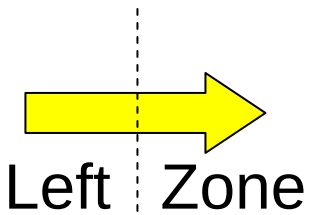
Détecteur de passage



Enter/Exit

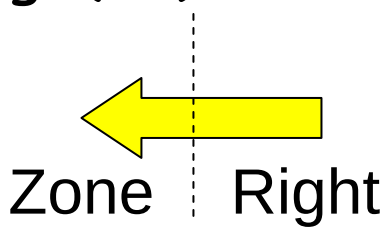
EnterL =

Edge(P1) and not P2;



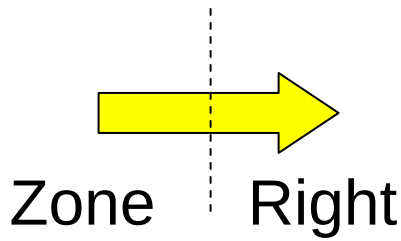
EnterR =

Edge(P2) and not P1;



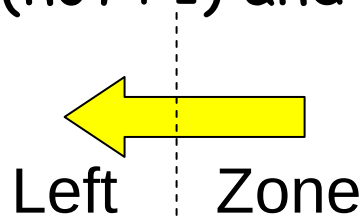
ExitR =

Edge(not P2) and not P1;



ExitL=

Edge(not P1) and not P2;



Mémorisation

```
node SR (init,set,reset:bool)  returns (s:bool);
let
   s = init ->
         if set and not pre(s) then true
         else   if reset and pre(s) then false
                else pre(s);
tel
```

```
LinL = SR(false,EnterL,EnterR) - - Last in Left
LinR = SR(false,EnterR,EnterL) - - Last in Right
```

Détecteur

```
node Detector(P1,P2:bool) returns (L2R,R2L:bool);
var LinL,LinR,EnterL,EnterL,ExitL,ExitR:bool;
let
   L2R = ExitR and LinL; - - traversée L -> R
   R2L = ExitL and LinR; - - traversée R -> L
   LinL = SR(false,EnterL,EnterR);
   LinR = SR(false,EnterR,EnterL);
   EnterL = Edge(P1) and not P2;
   EnterR = Edge(P2) and not P1;
   ExitL = Edge(not P1) and not P2;
   ExitR = Edge(not P2) and not P1;
tel
```

Assertions

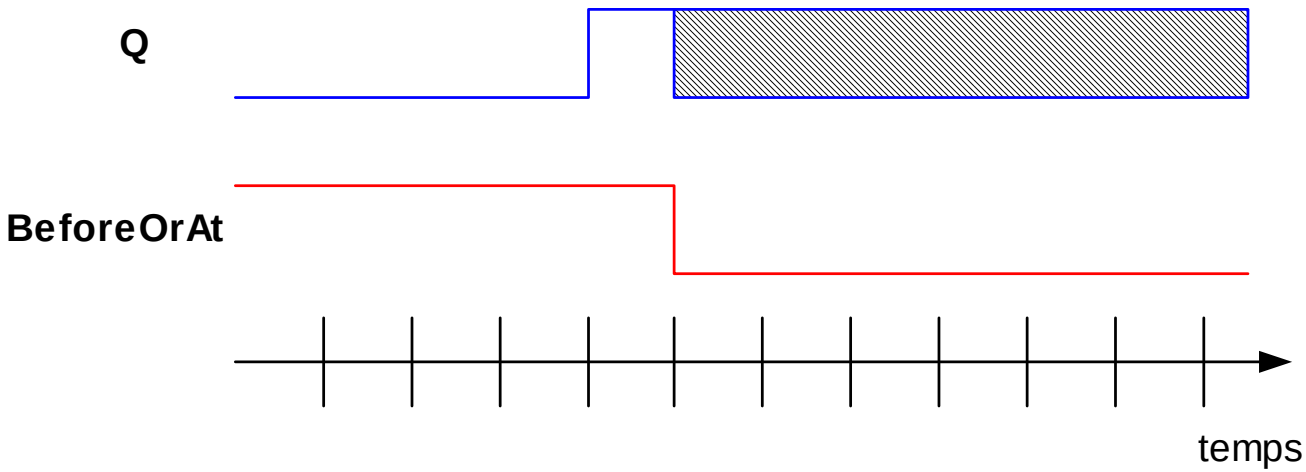
- Tout changement de pédale est perçu séparément

```
assert #(Edge(P1),Edge(P2),Edge(not P1), Edge(not P2));
```

- Initialement, la zone Z est vide

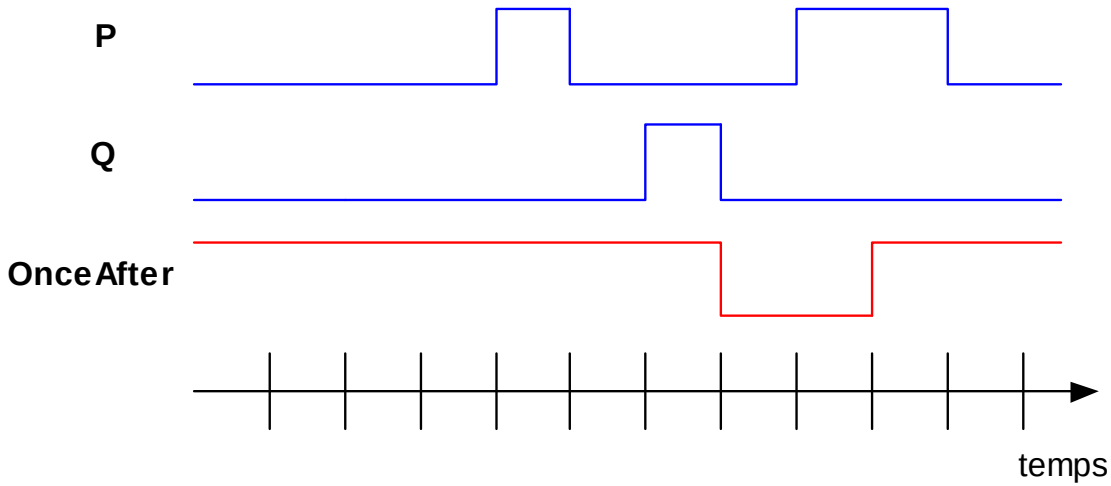
```
assert not (P1 or P2) -> true;
```

BeforeOrAt




```
node BeforeOrAt (Q:bool) returns (B:bool);
let
   B = true -> pre (not Q and B);
tel
```

OnceAfter




```
node OnceAfter (P,Q:bool) returns (B:bool);
let
   B = if BeforeOrAt (Q) then true
       else   if pre(Q) then false
              else pre (B or P);
tel
```

Propriété

- Si
  - on sort à droite (A) et
  - La dernière entrée s'est faite à gauche (B)
- Alors
  - Le train a traversé de gauche à droite (C)

$$A \wedge B \Rightarrow C \qquad \neg(A \wedge B) \vee C$$

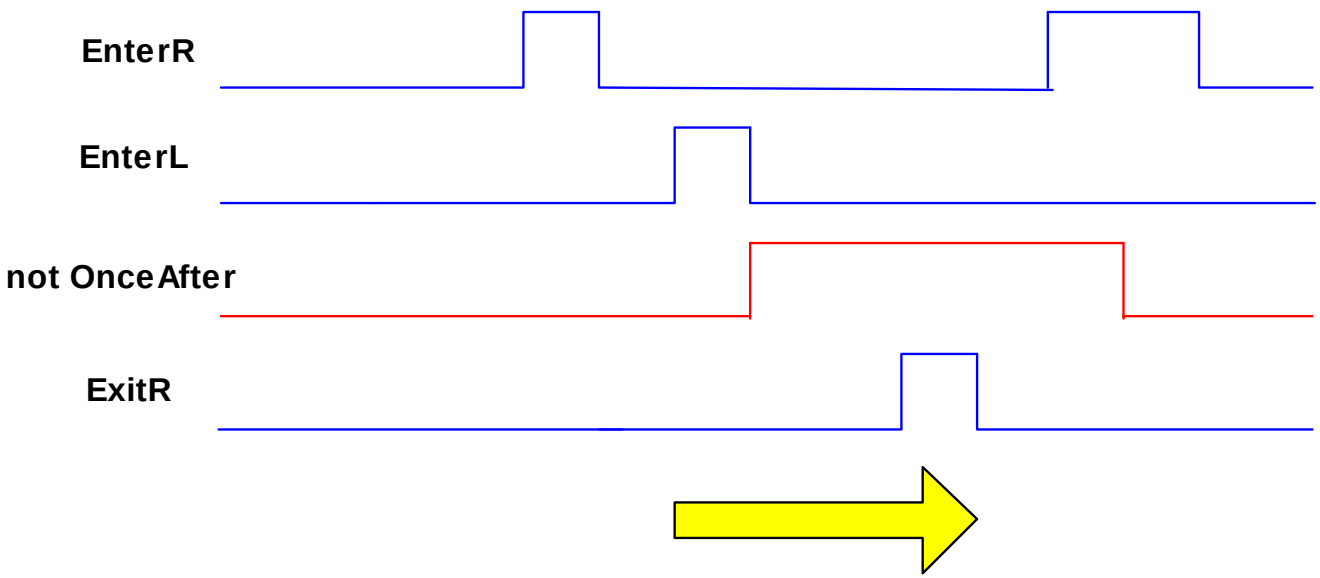
Propriété (2)

A = ExitR

B = not OnceAfter(EnterR,EnterL)

C = L2R

Ok = not(ExitR and not OnceAfter(EnterR,EnterL))
or L2R;



DELAY

- Prend un paramètre (statique) entier d $\geq$ 0 et un flot booléen X.
- Retourne X retardé de d instants
- C'est à dire

$$y_n = x_{n-d} \quad \text{pour } n > d$$

$$y_n = \texttt{false} \text{ pour } n \leq d$$

Delay

```
node DELAY (const d:int; X:bool)
  returns (Y:bool);
var A:bool^(d+1);
let
  A[0] = X;
  A[1..d] = (false^d) -> pre(A[0..d-1]);
  Y = A[d];
tel
```