# Synchronous languages

## Lecture 2:
## LUSTRE (simplified version)

ENSEEIHT 3A – parcours E&L
2021/2022

Frédéric Boniol (ONERA)
frederic.boniol@onera.fr



# LUSTRE: introduction…

## Motivation

"Simple" and "safe" programming model for control systems.

## Style

Based on the notions used by engineers in the field of control theory:

⇒ Block diagrams

⇒ Data flows

=> Sampled systems

## => LUSTRE

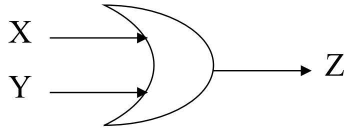
Formal programming language defined in 1985 by P. Caspi et N. Halbwachs (Grenoble, Vérimag)

- Commercial version: SCADE - Esterel Technologie
- Industrial users: Airbus, Dassault Aviation, Thales, Schneider Electric…

# LUSTRE: overview…

## Example :

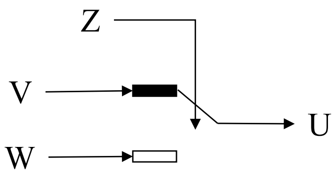
<u>"Or" gate</u>




```
Z = X or Y;
```

for all $n \geq 0$, $Z_n = X_n$ *or* $Y_n$

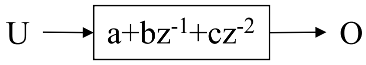
<u>"Switch"</u>




```
U = if Z then V
      else W;
```

for all $n \geq 0$, $U_n =$ *if* $Z_n$ *then* $V_n$ *else* $W_n$

<u>"Filter"</u>




for all $n \geq 2$, $O_n = aU_n + bU_{n-1} + cU_{n-2}$

```
O = a*U
    + b*pre(U)
    + c*pre(pre(U));
```

*(Remark: uncomplete equation)*

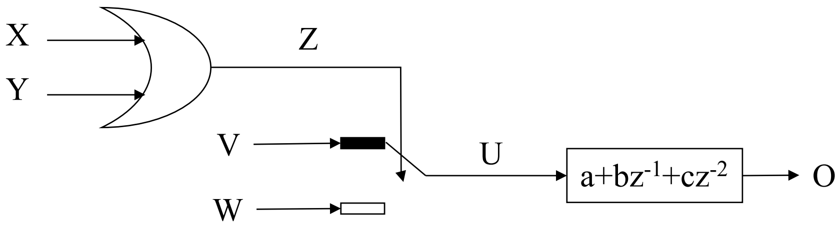


# LUSTRE: overview…

## Generalisation :

=> Lustre program = network of nodes operating on data flows

Example :




```
Z = X or Y;
U = if Z then V else W;
O = a * U + B * pre(U) + c * pre(pre(U));
```

*(Remark: uncomplete equations)*

# LUSTRE: basics…

## Generalisation :

– Data flow =

$X$ = sequence of values $X_n$ for $n \geq 0$ (flow = infinite sequence of values)

$X_n$ = value of X at instant $n$

Examples :

- `1` is the infinite flow (`1,1,1,1,1,`…)
- `true` = (`true`, `true`, `true`, `true`,…)

– Local and output flows are defined by equations

```
O = X op Y
```

defines $O_n$ with respect to $X_n$ and $Y_n$

=> $O_n$, $X_n$, $Y_n$ are synchronous (they are produced at the same time $n$)

=> LUSTRE program =

- A function receiving and producing data flows at each tick of a global clock (i.e., the sampling clock)
- Defined by a set of equations

⇒ LUSTRE = declarative language

- Similar to functional languages
- Definitions instead of assignments

=> Simple + modular language



# LUSTRE: basics…

## LUSTRE program =

```
[declaration of external functions]
-- comments
node name (declaration of input flows)
returns (declaration of output flows)
[var declaration of local flows]
[assertions]
let
   system of equations defining once each local flow and output
   depending on them and the inputs
tel.
[others nodes]
```

## Data flow declaration:

```
Name_of_the_flow : Type_of_the_flow;
```

## Constant flow declaration :

```
const Name_of_the_flow : Type_of_the_flow = value ;
```

## Data types :

– Basic types: `int`, `bool`, `real`

– Tabular: `int^3`, `real^5^2`…

# LUSTRE: basics…

## Equation

– Equation = mathematical definition

$$\begin{cases} X = Y + Z \\ Z = U \end{cases}$$

means

$$\text{For all } n \geq 0,\ X_n \stackrel{def}{=} Y_n + Z_n \quad and \quad Z_n \stackrel{def}{=} U_n$$

=> Substitution principle: An equation defines a mathematical egality.

Any flow can be replaced by its definition in all the equations of the node

$$\begin{cases} X = Y + Z \\ Z = U \end{cases} \quad \text{is equivalent to} \quad \begin{cases} X = Y + U \\ Z = U \end{cases}$$

=> Equations are not ordered

$$\begin{cases} X = Y + Z \\ Z = U \end{cases} \quad \text{is equivalent to} \quad \begin{cases} Z = U \\ X = Y + Z \end{cases}$$

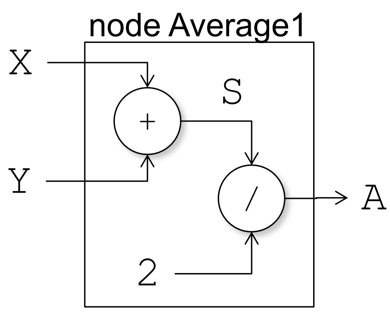


# Example: binary average computation

```
node Average1 (X, Y : int)
returns (A : int)
var S : int
let
   A = S / 2 ;
   S = X + Y;
tel.
```

- Two input flows (X, Y)
- One output flow (A)
- One internal flow (S)

=> Two equations to define

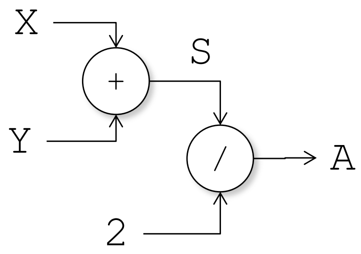



=> For all $n \geq 0$,

- $S_n = X_n + Y_n$
- $A_n = S_n / 2$

# Example: binary average computation – simplified version

```
node Average2 (X, Y : int)
returns (A : int)
let
  A = (X + Y) / 2 ;
tel.
```




⇒ Average2 = Average1 where S has been substituted by its definition

⇒ Average1 and Average2 are semantically equivalent



# Example: a NAND node…

```
node Nand (X, Y : bool) returns (Z : bool)
var U : bool ;
let
  U = X and Y ;
  Z = not U ;
tel.
```

Top of the global (sampling) clock

| | | n=0 | n=1 | n=2 | n=3 | n=4 | n=5 | n=6 | … |
|---|---|---|---|---|---|---|---|---|---|
| Input flows | $X_n$ | true | true | false | true | true | false | vrai | |
| | $Y_n$ | false | true | false | false | true | false | false | … |
| Local flow | $U_n$ | false | true | false | false | true | false | false | … |
| Output flow | $Z_n$ | true | false | true | true | false | true | true | … |

Equivalent to (by substitution) =>

# Example: a NAND node…

```
node Nand (X, Y : bool) returns (Z : bool)
let
   Z = not (X and Y) ;
tel.
```

Top of the global (sampling) clock

| | | n=0 | n=1 | n=2 | n=3 | n=4 | n=5 | n=6 | … |
|---|---|---|---|---|---|---|---|---|---|
| Input flows | $X_n$ | true | true | false | true | true | false | vrai | |
| | $Y_n$ | false | true | false | false | true | false | false | … |
| Output flow | $Z_n$ | true | false | true | true | false | true | true | … |



# LUSTRE: operators…

Classical operators:

Arithmetical:

Binary : +, –, *, `div`, `mod`, /, **

Unary : –

Logical:

Binary : `or, xor, and, =>`

Unary : `not`

Comparison:

=, <>, <, >, <=, >=

Control:

`if.then.else`

Temporal operators:

**`pre`** (previous): operator which allows to work on the past of a flow

**`->`** (followed by): operator which allows to initiate a flow

**`when`**: under sampling operator

**`current`**: over sampling operator

**`condact(B, F(X, Y, …), Init)`** (condact): conditional activation

# LUSTRE: pre

**pre** (previous): operator which allows to work on the past of a flow

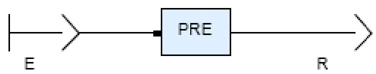
`pre(E)` returns the previous value of E

Let E be the flow

$(E_0, E_1, \ldots, E_n, \ldots)$

then

`R = pre(E)` is defined as the new flow $(nil, E_0, E_1, \ldots, E_n, \ldots)$



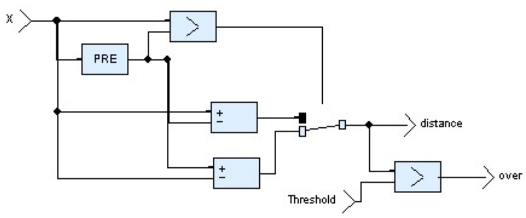

Generalisation

`(R, R') = pre(R, R')` means $R_0 = nil$, $R'_0 = nil$
for all $n \geq 1$, $R_n = E_{n-1}$ and $R'_n = E'_{n-1}$

Example :

- Let X an input flow
- Let OVER a boolean flot
- OVER must be set to true whenever the difference between two consecutive values on X is greater then Threshold

```
distance = if (X>pre(X))
           then X-pre(X)
           else pre(X)-X ;
OVER = (distance > Threshold) ;
```








# LUSTRE: ->

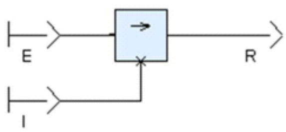
**->** (followed by): operator which allows to initiate a flow

Let E and I be then flow

$(E_0, E_1, \ldots, E_n, \ldots)$ and $(I_0, I_1, \ldots, I_n, \ldots)$

then

`R = I -> E` is defined as the new flow $(I_0, E_1, \ldots, E_n, \ldots)$




Generalisation

`(Z, Z') = (Y, Y')->(X, X')` means $Z_0 = Y_0$, $Z'_0 = Y'_0$
for all $n \geq 1$, $Z_n = X_n$ and $Z'_n = X'_n$

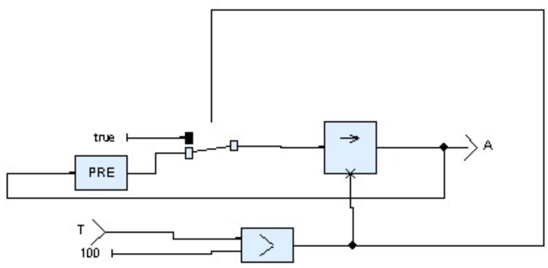
Example: temperature monitoring

```
A = (T>100) ->
        if (T>100)
        then true
        else pre(A) ;
```

means:

$$A_0 = (T_0 > 100)$$

$$A_n = \begin{cases} \text{true if } (T_n > 100) \\ A_{n-1} \text{ otherwise} \end{cases}$$

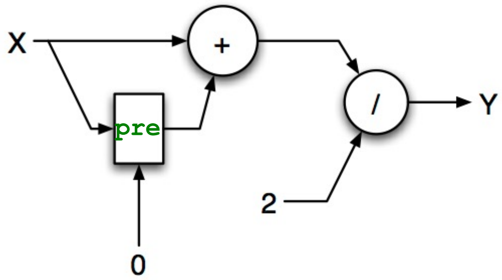


# LUSTRE: pre and ->

## Example:

- Convolution filter

$$Y_1 = X_1/2$$
$$Y_n = (X_n + X_{n-1})/2$$




```
node Convolution (X: real) returns (Y: real);
let
    Y = (X + 0 -> pre X)/2;
tel
```

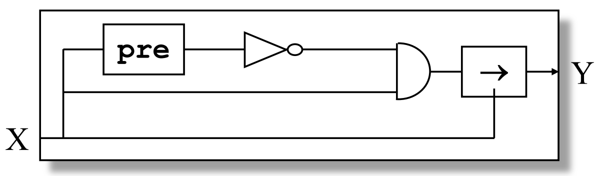


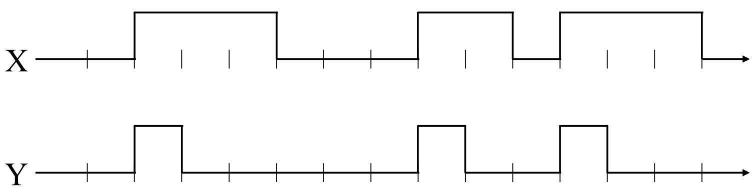
# LUSTRE: pre and ->

## Exemple: Raising edge node

Let X a boolean flow
Y must set to true whenever X performs a raising edge

```
node EDGE (X : bool) returns (Y : bool) ;
let
  Y = X -> (X and not pre(X));
tel ;
```



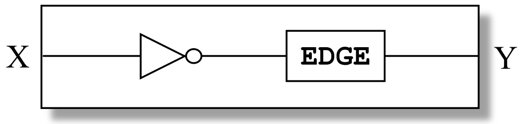
# LUSTRE: pre and ->

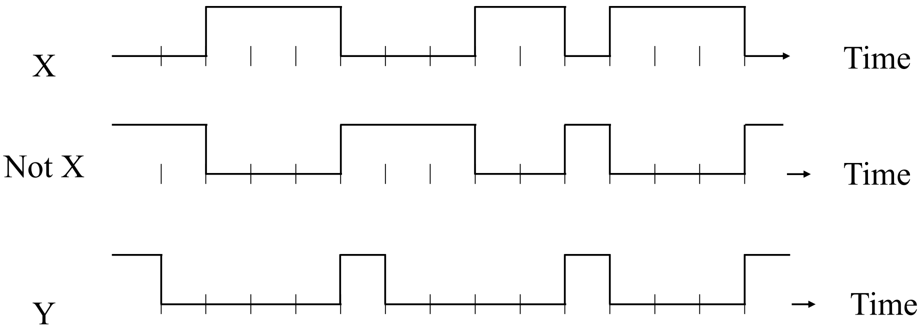
## Example: Falling edge

Let X a boolean flow

Y must set to true whenever X performs a falling edge

```
node FALLING_EDGE (X : bool) returns (Y : bool) ;
let
   Y = EDGE (not X) ;
tel ;
```



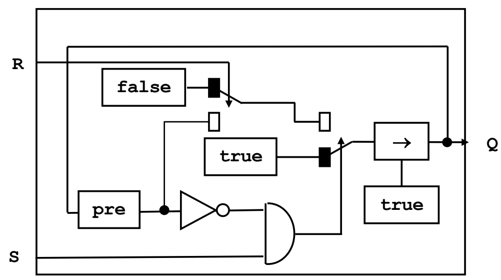






# LUSTRE: pre and ->

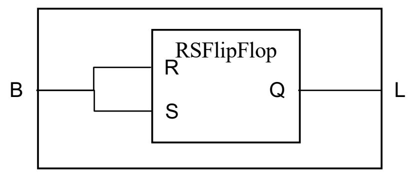
## Example: RS Flip-Flop

```
node RSFlipFLop (R, S : bool) returns (Q : bool) ;
let
   Q = true -> if S and not pre(Q)
               then true
               else if R then false
                         else pre(Q) ;
tel.
```




## Example: PushButton

```
node PushButton (B : bool) returns (L : bool) ;
let
   L = RSFlipFlop(B,B);
tel.
```



# LUSTRE: pre and ->

- Exercice1: write a resetable counter
  - top, reset: boolean flows (input)
  - n: integer flow (output)
  - Spec: n is the number of occurrences of true values on top since the last true value on reset

```
node counter (top, reset : bool) returns (n : int)
let
    n = if reset then (if top then 1 else 0)
        else (if top then (1-> (pre(n))+1)
                else 0->pre(n))
tel.
```

| top | False | True | True | False | True | True | True | False | True |
|---|---|---|---|---|---|---|---|---|---|
| **reset** | **True** | **False** | **False** | **False** | **True** | **False** | **False** | **True** | **True** |
| **n** | 0 | 1 | 2 | 2 | 1 | 2 | 3 | 0 | 1 |



# LUSTRE: pre and ->

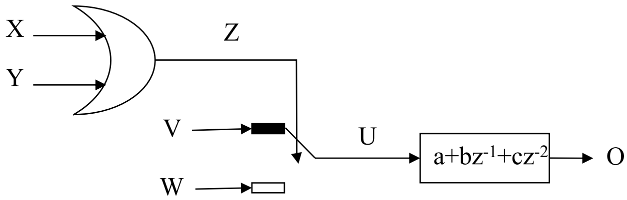
- Exercice2: write a delay operator
  - Input set:bool activation of the timer (Boolean flow)
  - Output set_delayed:bool
  - Constant delay : int duration of the timer (w.r.t. the global clok)

```
const D : int;
node timer1 (set : bool) returns (set_delayed : bool)
var …
let
    …
tel.
```

| D | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|
| set | False | True | False | False | True | True | False | False | False |
| set_delayed | False | False | False | True | False | False | False | True | False |

# LUSTRE: pre and ->

- Exercice3:




Z = X or Y;
U = if Z then V else W;
O = a * U + B * pre(U) + c * pre(pre(U));

Is this program correct? Why? How to fix it?

- Exercice4:
  - Write the Fibonacci sequence

```
node fibonacci () returns (F : int) ;
var …;
let
  …
tel.
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **F** | 1 | 1 | 2 | 3 | 5 | 8 | … |



# LUSTRE: assert

## Assert

To formalize hypotheses on input / local / output flows

=> To optimize the generated code
=> To take into account hypotheses in the verification process

**Example:**

```
assert (not (X and Y))
```

means that X and Y are never true at the same time

```
assert (true -> not (X and pre(X)))
```

means that there are never two true consecutive values on X

# LUSTRE: a full example…

## Example: Justin, the wolf, the goat and the cabbage…

A farmer (Justin) wants to get

- his cabbage,
- his goat
- and a wolf

across a river.

- He has a boat that only holds two.
- If he leaves the cabbage and the goat alone, then the goat eats the cabbage.
- If he leaves the wolf and the cabbage alone, then the wolf eats the goat.

**=> Write a LUSTRE node modeling this system**



# LUSTRE: a full example…

## Example: Justin, the wolf, the goat and the cabbage…

Input flows

- `m : bool` to mean that Justin crosses the river alone
- `mw : bool` to mean that Justin crosses the river with the wolf
- `mg : bool` to mean that Justin crosses the river with the goat
- `mc : bool` to mean that Justin crosses the river with the cabbage

Output flows

- `J : int` the side of the river where Justin is
- `W : int` the side of the river where the wolf is
- `G : int` the side of the river where the goat is
- `C : int` the side of the river where the cabbage is

$X_n = 0$ means that X is on the intial side of the river at time $n$

$X_n = 1$ means that X is on the second side of the river at time $n$

$X_n = 2$ means that X is dead at time $n$ (it has been eaten at time $n$ or before)

# LUSTRE: a full example…

## Example: Justin, the wolf, the goat and the cabbage…

```
node justin(m, mw, mg, mc : bool) returns (J, W, G, C : int);
assert (true -> (m or mw or mg or mc));
assert ( (not (m or mw or mg or mc)) -> true);
assert( not (m and mw));
assert( not (m and mg));
assert( not (m and mc));
assert( not (mw and mg));
assert( not (mw and mc));
assert( not (mg and mc));
assert( true -> not (mw and not (pre(J)=pre(W))));
assert( true -> not (mg and not (pre(J)=pre(G))));
assert( true -> not (mc and not (pre(J)=pre(C))));
let
   J = 0 -> 1 - pre(J);
   W = 0 -> if mw then 1 - pre(W) else pre(W);
   G = 0 -> if pre(G) = 2 then pre(G)
            else if mg then 1 - pre(G)
                 else if (pre(G)=pre(W) and not mw) then 2
                      else pre(G);
   C = 0 -> if pre(C) = 2 then pre(C)
            else if mc then 1 - pre(C)
                 else if (pre(C)=pre(G) and not mg) then 2
                      else pre(C);
tel.
```



# LUSTRE: a full example…

## Example: Justin, the wolf, the goat and the cabbage…

```
…
J = 0 -> 1 - pre(J);
W = 0 -> if mw then 1 - pre(W) else pre(W);
G = 0 -> if pre(G) = 2 then pre(G)
         else if mg then 1 - pre(G)
              else if (pre(G)=pre(W) and not mw) then 2
                   else pre(G);
C = 0 -> if pre(C) = 2 then pre(C)
         else if mc then 1 - pre(C)
              else if (pre(C)=pre(G) and not mg) then 2
                   else pre(C);
```

=> Solution of the Justin's problem

| | t=0 | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 |
|---|---|---|---|---|---|---|---|---|
| **m** | false | false | **true** | false | false | false | **true** | false |
| **mw** | false | false | false | **true** | false | false | false | false |
| **mg** | false | **true** | false | false | **true** | false | false | **true** |
| **mc** | false | false | false | false | false | **true** | false | false |
| **J** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **W** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| **G** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| **C** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

# LUSTRE: a full example…

## Example: Justin, the wolf, the goat and the cabbage…

```
…
J = 0 -> 1 - pre(J);
W = 0 -> if mw then 1 - pre(W) else pre(W);
G = 0 -> if pre(G) = 2 then pre(G)
              else if mg then 1 - pre(G)
                   else if (pre(G)=pre(W) and not mw) then 2
                        else pre(G);
C = 0 -> if pre(C) = 2 then pre(C)
              else if mc then 1 - pre(C)
                   else if (pre(C)=pre(G) and not mg) then 2
                        else pre(C);
```

What happens if Justin decides to cross the river alone at initial time?

| | t=0 | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 |
|---|---|---|---|---|---|---|---|---|
| **m** | false | **true** | | | | | | |
| **mw** | false | false | | | | | | |
| **mg** | false | false | | | | | | |
| **mc** | false | false | | | | | | |
| **J** | 0 | 1 | | | | | | |
| **W** | 0 | 0 | | | | | | |
| **G** | 0 | 2 | | | | | | |
| **C** | 0 | 2 | | | | | | |



# LUSTRE: arrays…

## Array in LUSTRE

=> Examples

**bool^4**, = boolean vector of dimension 4

**int^n** = integer vector of dimension n where n is a constant value

**real^4^8** = …

# LUSTRE: arrays…

## Example: another timer node

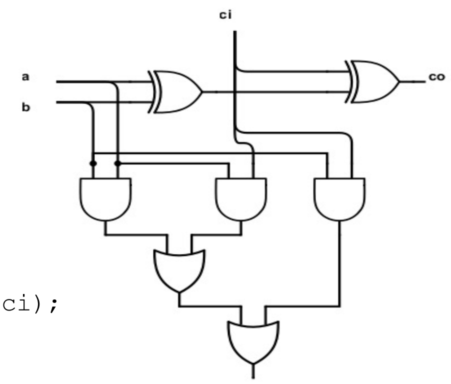
```
node Tdelay (const d : int; x : bool) returns(y : bool);
var A : bool^(d+1);
let
   A[0] = x;
   A[1..d] = false^d -> pre(A[0..d-1]);
  -- For all i = 1..d, A[i] = false -> pre(A[i-1])
   y = A[d];
tel


node Main (A : bool) returns (A_delayed : bool);
let
    A_delayed = Tdelay(10,A);
tel
```





# LUSTRE: arrays…




## Example

```
node ADD1 (a, b, ci : bool)
returns(s, co : bool);
let
   s = (a xor b) xor ci;
   co = (a and b) or (a and ci) or (b and ci);
tel

node ADD (const n : int; A, B : bool^n)
returns (S : bool^n; carry : bool)
var C : bool^n;
let
   (S[0], C[0]) = ADD1(A[0], B[0], false);
   (S[1..n-1], C[1..n-1]) = ADD1(A[1..n-1], B[1..n-1], C[0..n-2]);
   carry = C[n-1];
tel

const size : int = 4;
node Main (A,B : bool^size) returns (S : bool^size);
var carry : bool;
let
    (S, carry) = ADD(size, A, B);
tel
```

# LUSTRE: Causality and Initialization rules

Causality rule: A flow can not depend on it-self at the same time

Example:

`Y = X + Y` is a non causal equation

Example

```
Y = X + Z
Z = W + Y
```

is a set of non causal equations

Intialization rule: all flow expression must be initialized

`Y = X + pre(Y)` is not initialized

=> Example of correct equations

```
Z = X + pre(Y)
Y = X -> Z
```



End of lecture 2

⇒Next lecture: LUSTRE (full version)