Modèle et génération automatique de code

Alexandre Chapoutot

ENSTA Paris

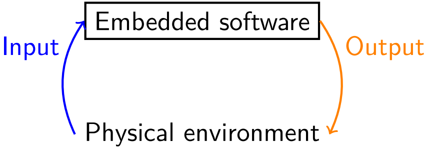
2022-2023

Part I

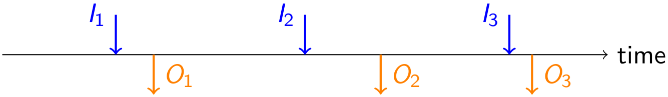
Lecture 2

Reactive software

- **Embedded software** are also known as **reactive programs**: they continuously produce outputs in response to inputs coming from the physical environment.



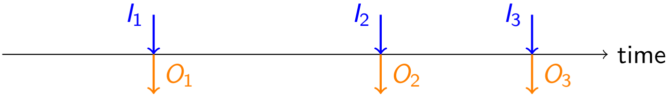
- The execution of embedded software is described by **discrete-time dynamics** *i.e.* it is a sequence of reactions.



- Ideally we should have that:
  - Output O_i should be emitted before input I_{i+1} and no important input I_i is missed.
  - The software is deterministic: same input produces same output.
  - A finite amount of memory is used.

An ideal abstraction: synchronicity

- The execution of embedded software is described by **discrete-time dynamics** *i.e.* it is a sequence of reactions.
  We assume that the computation time is zero



- **Conceptually**
  - Output are produced infinitly quickly
  - All the computation are done in parallel
- **Verification of the hypothesis**
  - Compute WCET and check that input are not faster than WCET

Classical implementation

A reactive software is mainly an infinte loop of the form

Two possible implementations: **sampled-base** or **event-based**

$S := S_0$
for each tick **do**
  Read I
  $(S, O) = \text{step}(S, I)$
  Write O
end for

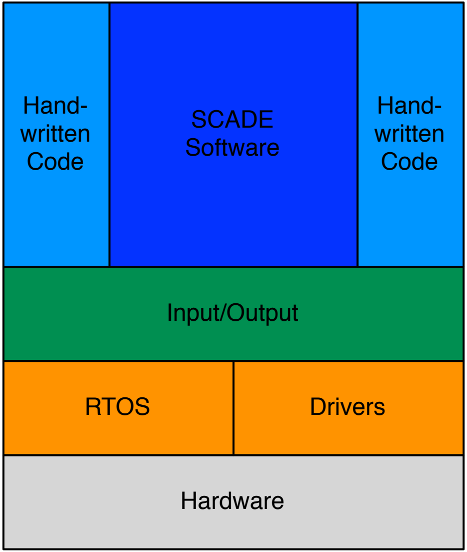
$S := S_0$
for each event **do**
  Read I
  $(S, O) = \text{step}(S, I)$
  Write O
end for

The function *step* is the targeted applications of SCADE language

Examples of reactive programs

Linear filters or state machines

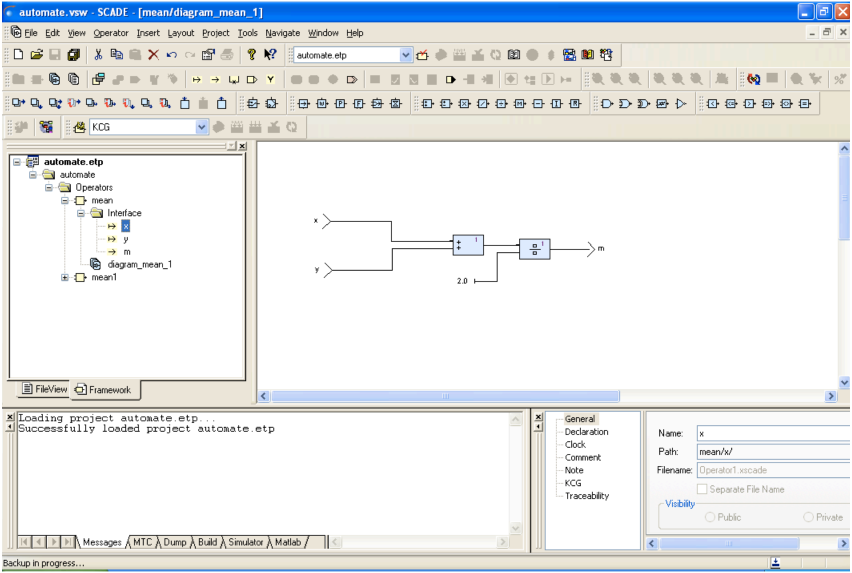
Model-based: kind of software targeted



SCADE function is based on

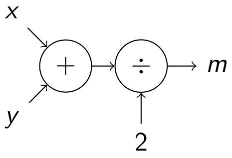
- data-flow equations
- state machines

SCADE: Safety Critical Application Development Environment



Data-flow approach

A classical approach in circuits and control theory.




```
node mean (x, y : real)
returns (m : real);
let
   m = (x + y) / 2;
tel;
```

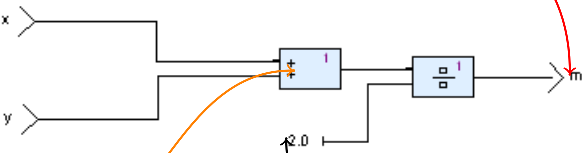
Synchronous interpretation:

$$\forall t \in \mathbb{N}, \quad m_t = (x_t + y_t)/2$$

A Lustre/SCADE program is described by a set of data-flow equations.

Mean example in Scade

- output
- input



- operator
- constant or textual expression

Flows

Definition

A flow is an infinite sequence of values of the same type.

- **All the Lustre/SCADE variables** are flows.
- Type of flows: `bool`, `int8`, `int16`, `int32`, `int64`, `float32`, or `float64`.

Flow example

- true $\equiv$ true, true, true, ...
- $1 \equiv 1, 1, 1, \ldots$
- $3.14 \equiv 3.14, 3.14, 3.14, \ldots$

Operations on flows

- An operator is **applied on flows** of particular type and produce an other flow of a particular type.

Operator example

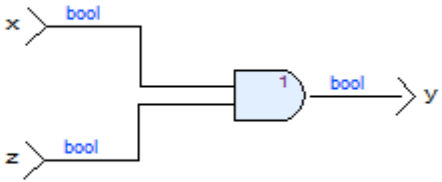
"and" is an operator applied on two Boolean flows and produces an other Boolean flow.

- Operators are applied point-wisely.

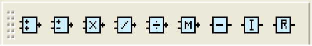
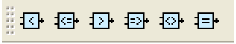
Example

	0	1	2	3	4	5	6	7
x	true	false	true	false	true	false	true	false
y	true	true	false	true	false	false	true	true
x and y	true	false	false	false	false	false	true	false

Example in SCADE



A set of atomic opertions is offered by SCADE:

- Arithmetic 
- Logical 
- Comparison 

Remark: SCADE Suite can be configured to display type variable

Lustre/SCADE program

A Lustre/SCADE program is made of a set of equations such that:

- **The order of equations is not important**
- **follows the substitution principle**

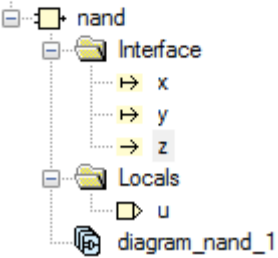
Example

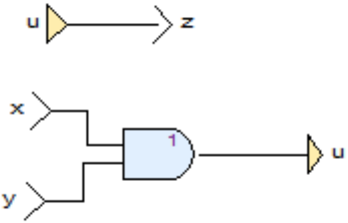
```
node nand (x, y: bool) returns (z: bool);
var u: bool;
let
   z = not u;
   u = x and y;
tel
```

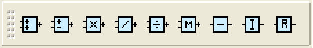
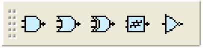
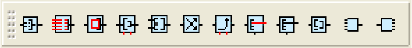
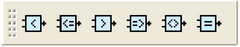
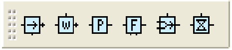
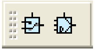
Execution:

x	true	true	false	true	true	false	...
y	false	true	false	false	true	false	...
u	false	true	false	false	true	false	...
z	true	false	true	true	false	true	...

Example in SCADE





Mathematical	
Logical	
Structure/Array	
Higher Order	
Comparison	
Time	
Choice	

if expression

Example

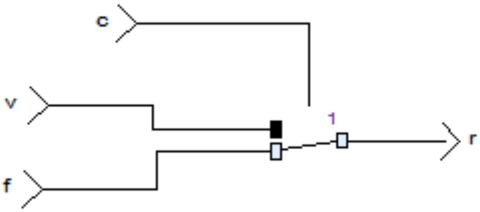
```
node max (x, y: int) returns (m: int);
let
  m = if (a >= b) then a else b;
tel
```

Remark: if expression as in functional language

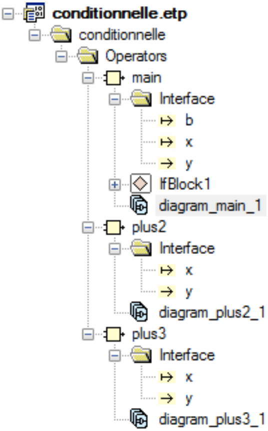
- `if`: (bool flow) $\times$ (t flow) $\times$ (t flow) $\rightarrow$ (t flow)

Remark: always `then` **and** `else` $\Rightarrow$ **determinism**.

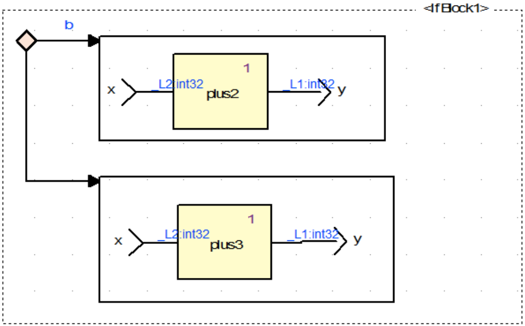
Example in SCADE



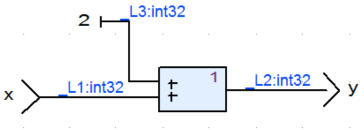
if block



conditionnelle.etp
conditionnelle
Operators
main
Interface
b
x
y
IfBlock1
diagram_main_1
plus2
Interface
x
y
diagram_plus2_1
plus3
Interface
x
y
diagram_plus3_1



<IfBlock1>
b
x
_L2:int32
plus2
1
_L1:int32
y
x
_L2:int32
plus3
1
_L1:int32
y



2
_L3:int32
x
_L1:int32
1
_L2:int32
y

-> operator

Solve the initialisation problem of `pre` operator by fixing the initial value.

$$(x \text{ -} > y)_i = \begin{cases} x_i & \text{if } i = 0 \\ y_i & \text{if } i > 0 \end{cases}$$

- **Warning** the **dates** i **are absolute** and not relative to the current instant.

Example

x	1	1	1	0	...
$0 \text{ -} > x$	0	1	1	0	...

What is the value of: $0 -> (0 -> 1)$?

pre operator

- pre: retains in memory previous values of a flow.

$$(\texttt{pre}(\texttt{e}))_i = \begin{cases} \bot & \text{if } i = 0 \\ e_{i-1} & \text{if } i > 0 \end{cases}$$

Memory size: number of embedded pre operators.

Example

e	1	0	1	0	1	...
pre e	$\bot$	1	0	1	0	...

Remark: Initialisation problem of pre operator which solves using -> operator.

Example: min and max

min/max program

```
node minmax (x: int)
returns (min, max: int);
let
    min = x -> if (x < pre(min)) then x else pre(min);

    max = x -> if (x > pre(max)) then x else pre(max);
tel
```

Execution:

x	12	5	7	-2	21	0	...
min	12	5	5	-2	-2	-2	...
max	12	12	12	12	21	21	...

fby (followed by) operator

Idea

Combination of the two operators: `pre` and `->`

Syntax

fby(exp; delay ; init)

- `exp`: flow expression;
- `delay`: number of delay instants;
- `init`: initial values.

Example

y = fby(x;1;0) + 1;
equivalent to
y = (0 −> **pre**(x)) + 1;

Definition of recursive flows

Corrects definition

- The sequence of values can be defined step by step
- *i.e.*, the recursion is not related to the past
- *i.e.*, no short circuit:
  e.g., equation $x = x + 1$ has no solution
  Remark: in some cases the recursion has a solution
  e.g., $x = 1/(2 - x)$
  but the computation is unbounded.

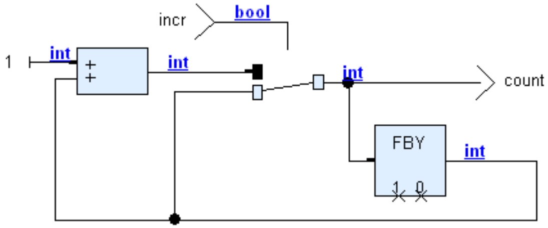
Example of recursive flows

```
alt = false -> not pre alt
```

$\Rightarrow$ built flow: false true false true ...

A graphical representation

Lustre/SCADE is mostly used with this graphical representation



its textual representation

```
count = fby(count + if incr then 1 else 0; 1; 0)
```

SCADE operator (Lustre node)

- **Equations** define the **output values** by constraining the input flows.
- Instantaneous evaluation and the order of equations is not important
- the value of output flow must be uniquely defined.

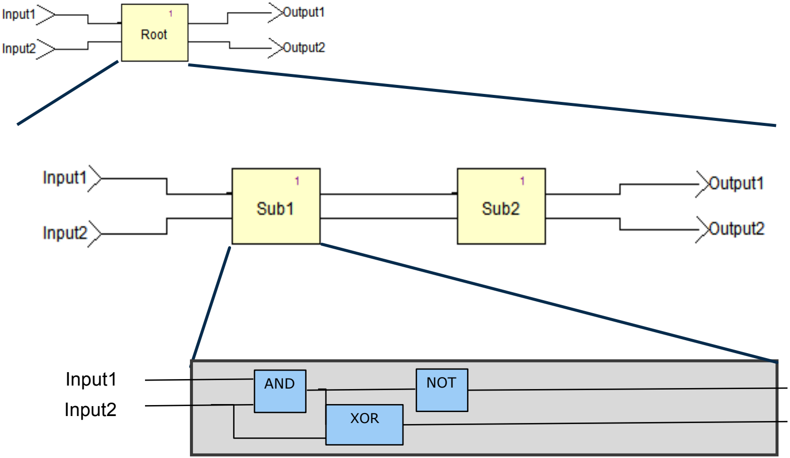
Node example

```
node voter (e1, e2, e2: bool) returns (s: bool);
var tmp1, tmp2: bool;
let
   tmp1 = e1 and e2;
   s = tmp1 or (e1 and tmp2);
   tmp2 = e2 or e3;
tel
```

Operator hierarchy

Remark

Only one root to be defined at compile time



Operator semantics

- A Lustre/SCADE node is a specification of constraints between input and output flows.
- The semantics of one node is then a set of input and output flows which are admissible for these constraints.
- Every node defined by the user can be reuse.

Remark

a node without state should be declared as a function.

Example

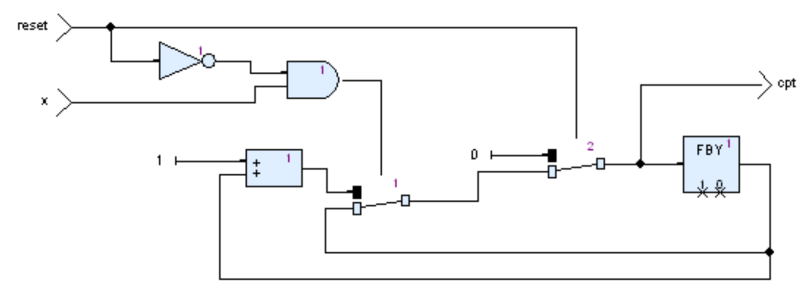
- the input flow `X` is taken into account only if it is maintained more than *n* hundredths of a second;
- the input flow `cs` is true each hundredth of a second.
- the output flow `y` is true when the input X is maintained more than *n* hundredths of a second;

Example - 1

Two nodes are needed:

- CounterReset: increases a counter when `X` is true and it is reset when `reset` (priority) is true;
- Detector:

CounterReset node

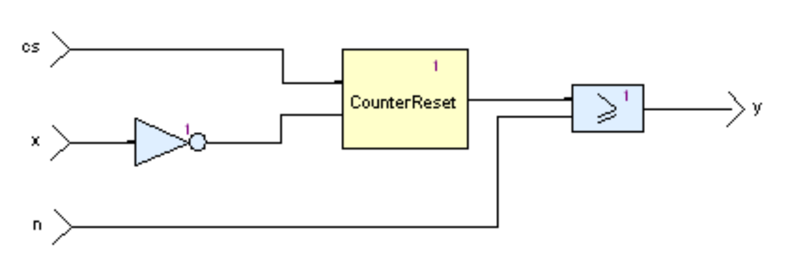


Example - 2

Two nodes are needed:

- CounterReset: increases a counter when `X` is true and it is reset when `reset` (priority) is true;
- Detector:

Detector node



Clocks and sampling operator

Sampling operator: `when`
uses to define a slower rate flow than its output.

Example of sampling

X	4	1	−3	0	2	7	8	...
C	true	false	false	true	true	false	true	...
X `when` C	4			0	2		8	...

Remark: when `C` is false, X `when` C **does not exist**.

Warning: operators are applied on flows on the same clock.
e.g., x + (x when c) is not allowed

Remark 2: we can sample a sampled flow.

Clocks and nodes

```
node cpt (x: bool) returns (y: int);
var cpt: int;
let
   y = 0 -> if x then pre cpt + 1 else pre cpt
end
```

- Sampling input is not equivalent to sampling output.

Sampling examples

C	true	true	false	false	true	false	...
cpt (true when C)	0	1			2		...
cpt (true) when C	0	1			4		...

merge operator

Bring back a low rate flow on a faster clock.

Definition

$$\text{merge}(h; x^1; ...; x^p) = \begin{cases} x_n^1 & \text{if } h \text{ match } e^1 \\ \vdots & \\ x_n^p & \text{if } h \text{ match } e^p \end{cases}$$

h is an element of enumerated type among $e^1, \ldots, e^p$.

Projection example

X	2	−2	2	−2	2	−2	2	...
Y	−1	1	−1	1	−1	1	−1	...
C	true	false	false	true	true	false	true	...
U = X `when` C	2			−2	2		2	...
V = Y `when not` C		1	−1			1		...
N = `merge`(C;U;V)	2	1	−1	−2	2	1	2	...

Conditional activation of an operator

Definitions

- **activate** N **every** clock_expr
  N is activated when clock clock_expr is true.
- **activate** N **every** clock_expr **default** exp
  Idem except that the value of expr2 is returned when clock_expr is false.
- **activate** N **every** clock_expr **initial** **default** exp
  N is activated when clock_expr is true. And when clock_expr is false the result is set with the value of expr2 at the first instant then it is the latest value of N which is used.

```
node integr (X: int) retruns (Y: int)
let
  Y = X + (0 -> pre(Y));
tel
```

Conditional activation of an operator

Definitions

- **activate** N **every** clock_expr
  `N` is activated when clock `clock_expr` is true.
- **activate** N **every** clock_expr **default** exp
  Idem except that the value of `expr2` is returned when `clock_expr` is false.
- **activate** N **every** clock_expr **initial** **default** exp
  `N` is activated when `clock_expr` is true. And when `clock_expr` is false the result is set with the value of `expr2` at the first instant then it is the latest value of `N` which is used.

Example

t = activate(integr every C) (X)

X	1	2	3	4	5	6	...
C	false	true	false	false	true	true	...
t		2			7	13	...

Conditional activation of an operator

Definitions

- **activate** N **every** clock_expr
  N is activated when clock clock_expr is true.
- **activate** N **every** clock_expr **default** exp
  Idem except that the value of expr2 is returned when clock_expr is false.
- **activate** N **every** clock_expr **initial** **default** exp
  N is activated when clock_expr is true. And when clock_expr is false the result is set with the value of expr2 at the first instant then it is the latest value of N which is used.

Example

t = activate(integr every C default 0) (X)

X	1	2	3	4	5	6	...
C	false	true	false	false	true	true	...
t	0	2	0	0	5	11	...

Conditional activation of an operator

Definitions

- **activate** N **every** clock_expr
  N is activated when clock clock_expr is true.
- **activate** N **every** clock_expr **default** exp
  Idem except that the value of expr2 is returned when clock_expr is false.
- **activate** N **every** clock_expr **initial** **default** exp
  N is activated when clock_expr is true. And when clock_expr is false the result is set with the value of expr2 at the first instant then it is the latest value of N which is used.

Example

t = activate(integr every C initial default 0) (X)

X	1	2	3	4	5	6	...
C	false	true	false	false	true	true	...
t	0	2	2	2	5	11	...

Conditional activation of a set of equations

Definition

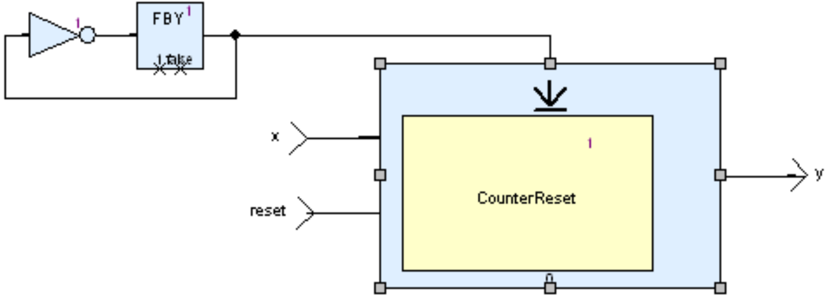
- **activate if** exp **then** equation_set1 **else** equation_set2 ;
  If `expr` is true then `equation_set1` is evaluated else `equation_set2` is evaluated.

Example

```
node N (e:int; h:bool) returns (s: int; t:int last=0)
let
   s = integr(e);
   activate if h then t = integr(e); else t = last 't;
   returns t;
tel
```

e	1	2	3	4	5	6	7	...
h	false	true	false	false	true	true	false	...
s	1	3	6	10	15	21	28	...
t	0	2	2	2	7	13	13	...

Example of activated node



restart operator

Definition

- (**restart** N **every** c)(e);
  is used to set the node N in is initial state.

Example

```
node S (e: int) returns (sum: int)
    let  sum = 0 -> e + pre sum;  tel

node Count () returns (x: int)
    let x = 0 -> (1 + pre(x));  tel

s = (restart S every (0 -> pre s >= 10)) (Count());
```

Count()	0	1	2	3	4	5	6	7	8	9	...
s	0	1	3	6	10	0	6	13	0	9	...