Modèle et génération automatique de code

Alexandre Chapoutot

ENSTA Paris

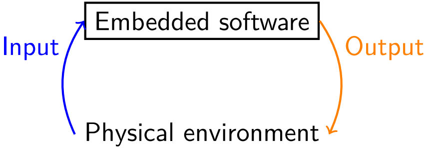
2022-2023

Part I

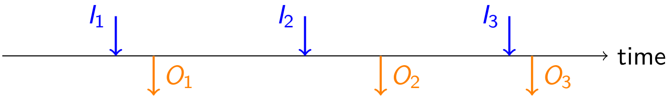
Lecture 3

Reactive software

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



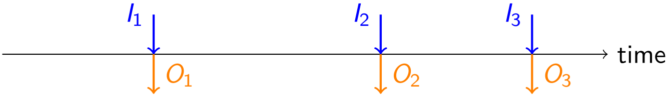
- The execution of embedded software is described by **discrete-time dynamics** *i.e.* it is a sequence of reactions.



- 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

Remark: we deal with discrete-time abstraction

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

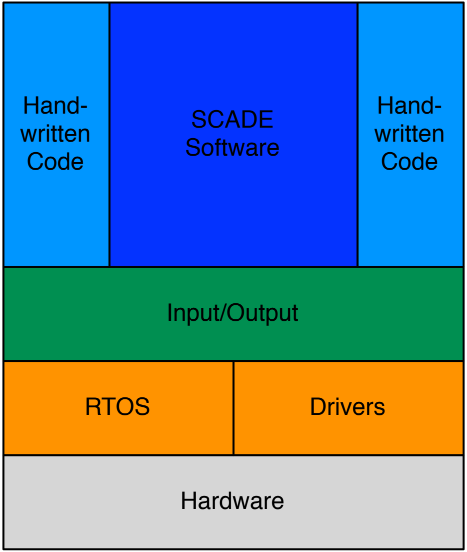
$S := S_0$
for each event **do**
  Read I
  $(S, O) = \text{step}(S, I)$
  Write O
end for

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

Examples of reactive programs

Linear filters or state machines

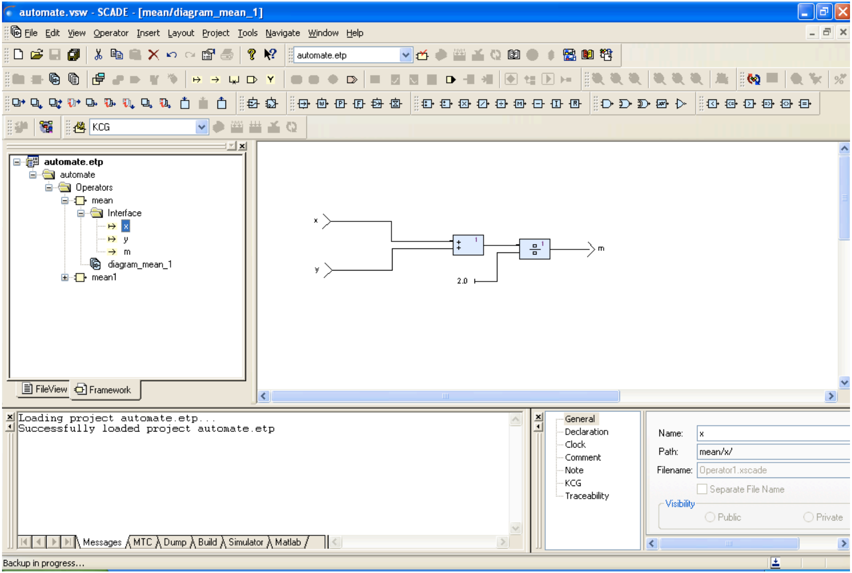
Model-based: kind of software targeted



SCADE function is based on

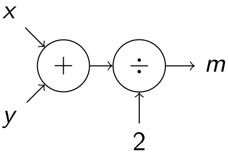
- data-flow equations
- state machines

SCADE: Safety Critical Application Development Environment



Data-flow approach

A classical approach in circuits and control theory.



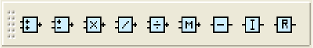
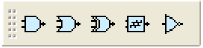
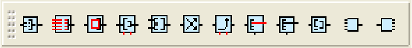
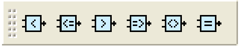
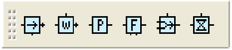
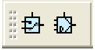

```
node mean (x, y : real)
returns (m : real);
let
   m = (x + y) / 2;
tel;
```

Synchronous interpretation:

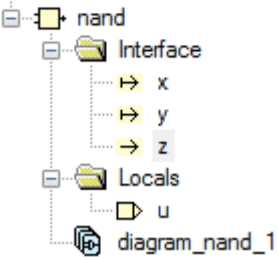
$$\forall t \in \mathbb{N}, \quad m_t = (x_t + y_t)/2$$

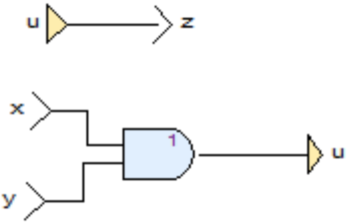
A Lustre/SCADE program is described by a set of data-flow equations.

Main language construction

Mathematical	
Logical	
Structure/Array	
Higher Order	
Comparison	
Time	
Choice	

Example in SCADE

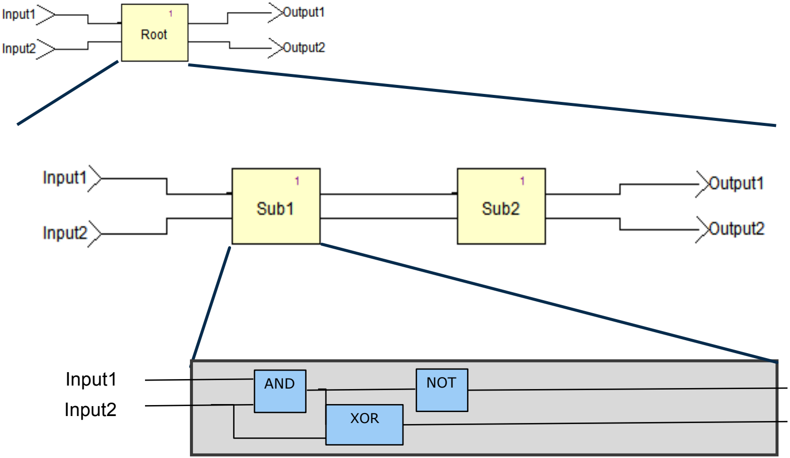




Operator hierarchy

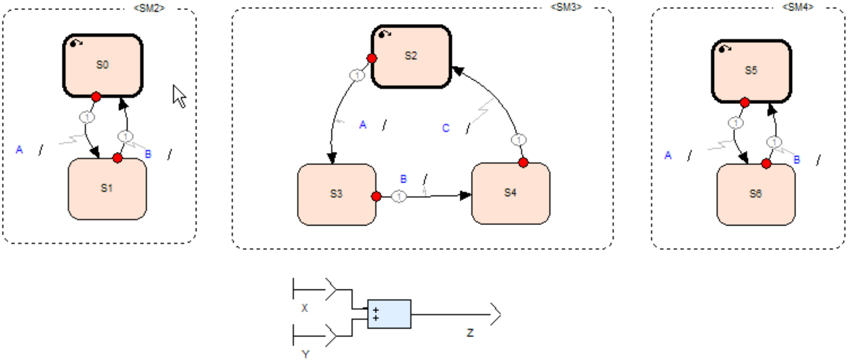
Remark

Only one root to be defined at compile time



State Machine

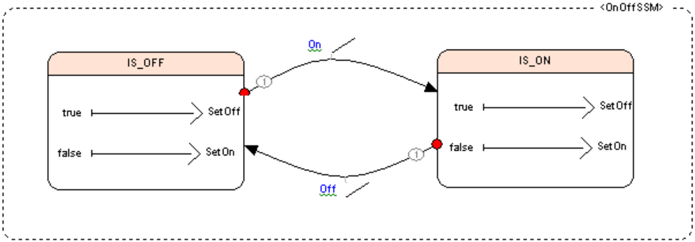
- **An operator** can be defined over a **state machine**
- It can have several state machine in "parallel" and mixed with flows
- each state machine must have **a unique initial state**



State Machine - cont'

A **state**

- is graphically represented by a rectangle with a name
- represents the memory element of a state machine
- at each cycle, a state in one state machine is either **active or not**



Note: the content, *i.e.*, the computations, of a state is defined graphically by dataflow diagrams or even other state-machines or both.

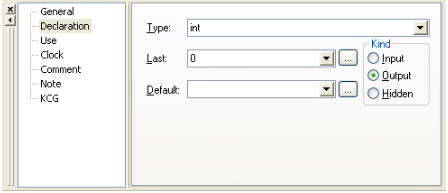
Dataflow in states

Main rules

- Equations are computed only when the state is active
- Ech declare variables (local or output) must have exactly one definition at each cycle where its scope is active

What happens when a definition is missing in a given state?

- Producing a **default value** if there is one defined for the flow
- Or maintaining the **last value** of the flow.
  Remark: If the flow is not defined at the initial cycle, the flow must have an init value for the last



State machine transitions

A transition has the general form:

$$\underbrace{\textit{SCADE textual expression}}_{\textit{Trigger}} \; / \; \underbrace{\textit{SCADE textual expression}}_{\textit{Effect}}$$

Preemption during transition

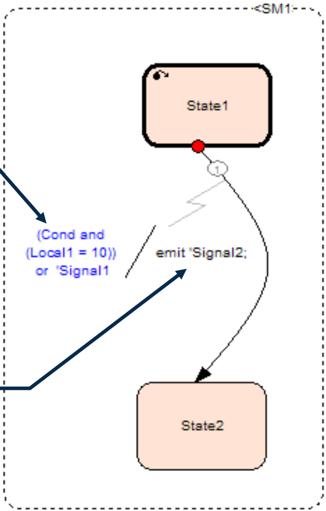
- **weak:** (**until**), when the transition is taken, the next state is activated in the next instant.
- **strong:** (**unless**), when the transition is taken, the next state is activated in the current instant.

Remark: the effects of a transition are computed in the current instant.

State machine transitions

Trigger: *Event condition*

Effect: signal emissions



State machine transitions

Triggers

are made of

- Boolean expressions
- **times operator** (presented in a few slides)

Examples: Local1 $>=$ 8

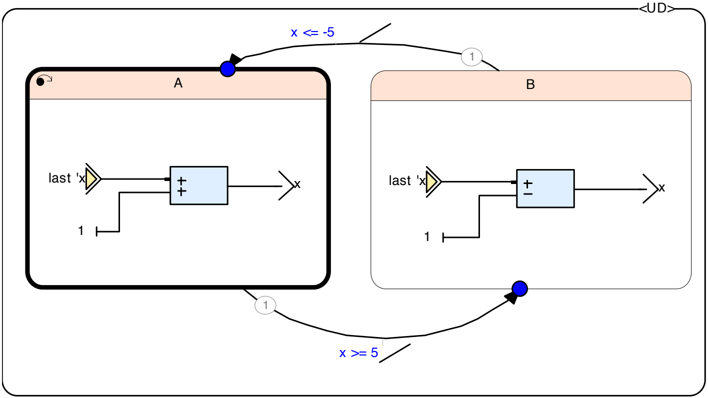
Events

are made of

- variable definitions based on any Scade expressions

Examples: Local1 $=$ 3+x;
Remark: an expression shall be terminated by a ';'

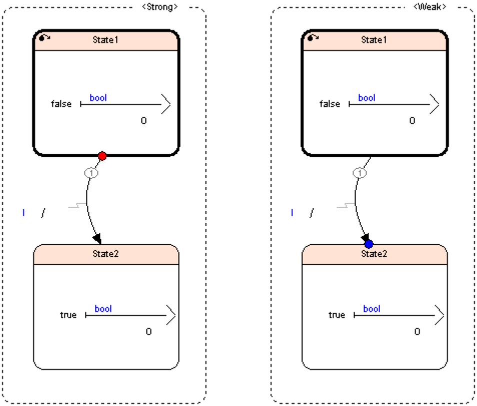
State machine transitions



Remark

The keyword **last** stands for memory that gives the value of x at the previous tick (the memory is shared between all states).

State machine transitions

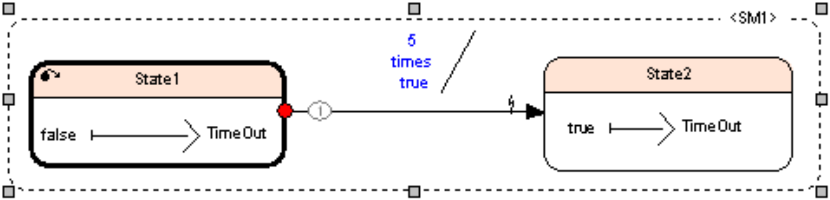


I	false	true	-
STRONG	O = false	O = true	O = true
WEAK DELAYED	O = false	O = false	O = true

State machine – factors

Factors

A factor specifies on many time a condition must be true in a guard of an automaton.
Note: can also be used in data-flow equations.



`true` in the guard can be replaced by an other Boolean flow.

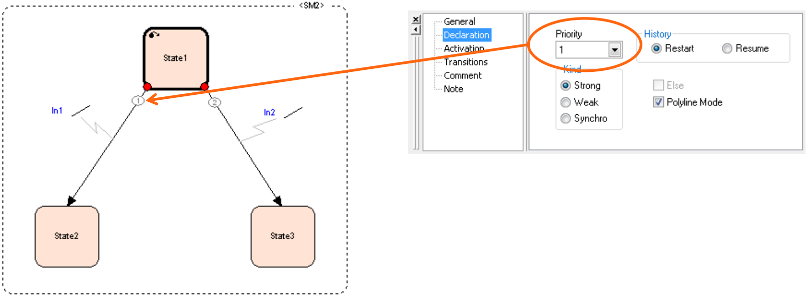
State machine in textual representation

Example

```
node UpDown () returns (x: int last=0)
let
   automaton UD
   intial state A
     x = last 'x + 1; until if x >= 5 restart B;
   state B
     x = last 'x - 1; until if x <= -5 restart A;
   returns x;
let
```

State machine transition priority

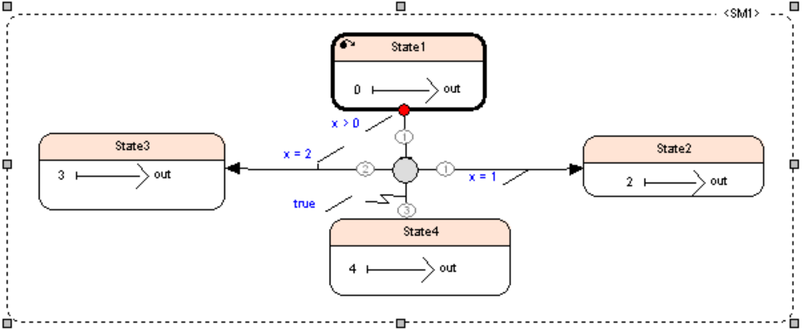
- When conditions of several transitions starting from the same active state are true, only the one with the highest priority is fired.



State machine – complex transitions

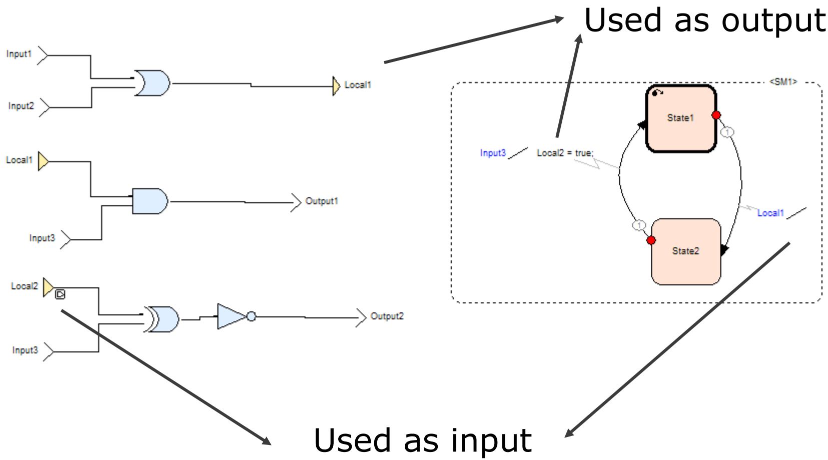
Fork

Decision point in an automaton



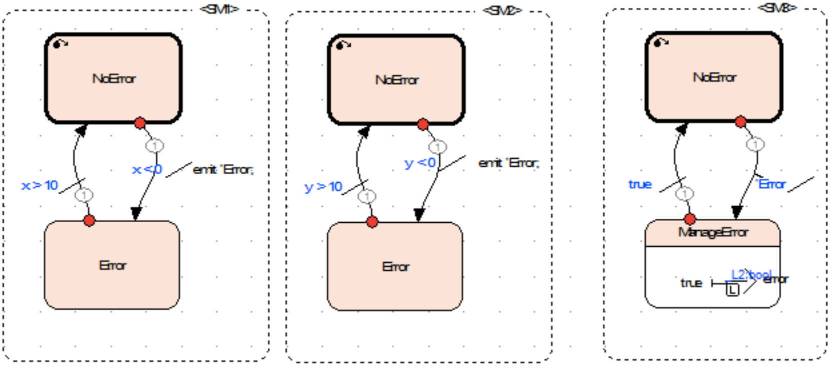
Local variable

- A local variable is only seen in the operator in which its is declared
- Can be used in `in/out` mode as many time as necessary.



Signals are a special values which are usefull to catch specific situation in several state-machines

- A signal is emitted in several parallel SSM when a condition is met
- A parallel SSM waits for the presence of the signal to respond to the event



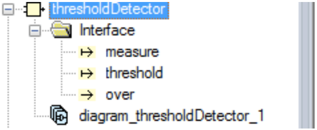
Example: Pressure controller

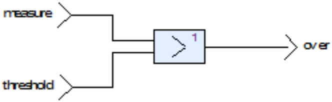
Goal of the controller

detect pression over 20 bars and set an alarm for 60 cycles.

Implementation in 3 operators

Operator 1: thresholdDetector



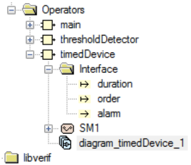


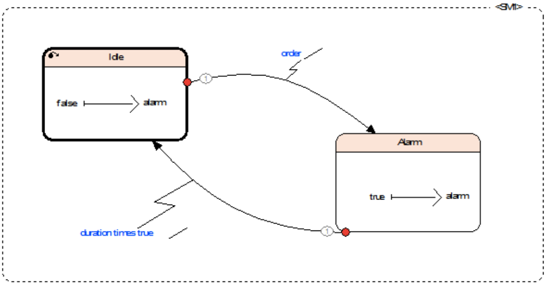
Goal of the controller

detect pression over 20 bars and set an alarm for 60 cycles.

Implementation in 3 operators

Operator 2: timedDevice





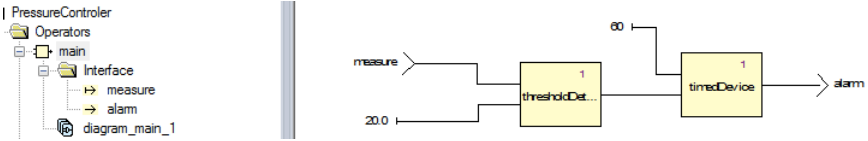
Example: Pressure controller

Goal of the controller

detect pression over 20 bars and set an alarm for 60 cycles.

Implementation in 3 operators

Operator 3: pressureController

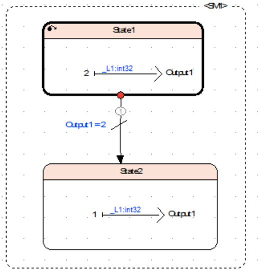


Causality loop

Definition

It is a cyclic dependencies of flow calculation, or a mix of State/Transition execution and flow calculation

KCG compiler can automatically detects them!

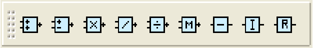
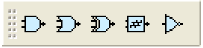
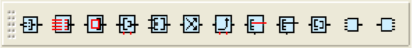
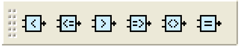
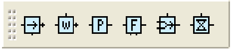
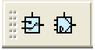


Note: this problem can be solved using **weak transition** or using **fby** operator.

1 error(s) detected - 0 warning(s) detected

Category	Code	Message
Causality Error	ERR_400	**Causality error at causality/SM1:State1:** the strong guards of state State1 depend on flow Output1 ; (causality/Output1/) the definition of flow Output1 depends on shared flow Output1 via a control block ; (causality/SM1:State1:Output1=) the definition of shared flow Output1 depends on the state of automaton SM1 via the control context ; (causality/SM1:State1:) the state of automaton SM1 depends on the strong guards of state State1 ;

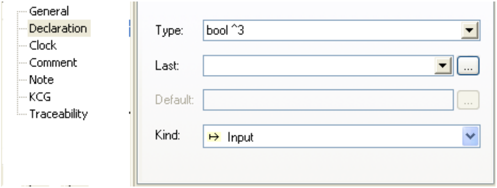
Main language construction

Mathematical	
Logical	
Structure/Array	
Higher Order	
Comparison	
Time	
Choice	

Data structure: Arrays – defintion

Restriction

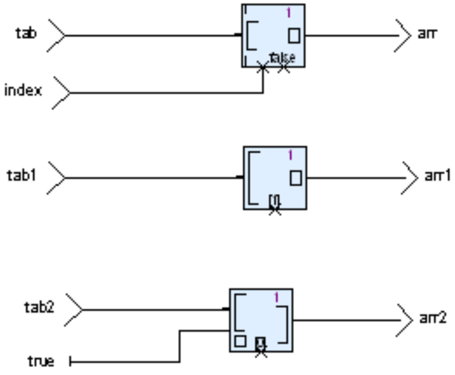
- Only static size is allowed
- First index is 0



Definition:

- Vector: **Real**^3
- Matrix: **Bool**^3^2 stands for 2 rows, 3 columns

  typedef real line_3 [3];
  typedef line_3 matrix_2_3 [2];

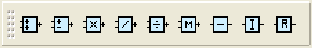
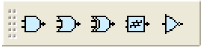
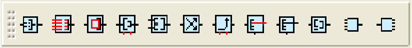
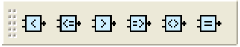
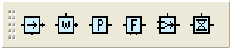
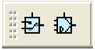


- Dynamic access in reading (with default value for out-of-bound)
- Static access in reading
- Writing

Textual notation

with square brackets x [0]

Main language construction

Mathematical	
Logical	
Structure/Array	
Higher Order	
Comparison	
Time	
Choice	

Iterators in brief – map function

Example: map

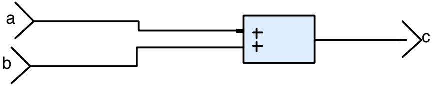
node SumScalar (a, b: int) returns (s: int) let s = a + b; tel
v = (map SumScalar «3»)(t, u);

Definition

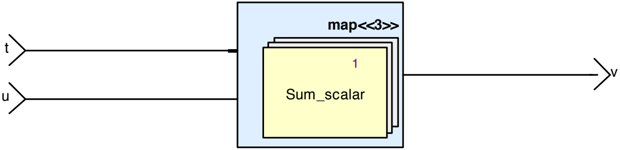
x = (map N <<dimension>>)(arguments);

- A node `N` with k arguments.
- From k arrays of dimension d we want to create a new array v of dimension d.
- The elements of v are the result of the application of `N` on the elements of the arrays in parameter.
  v = [N(x1[0],..., xk[0]); N(x1[1], ..., xk[1]); ...; N(x1[d-1],...,xk[d-1])]

Iterators in brief – map function



$$c = a + b$$



v = (map Sum_scalar<<3>>)(t, u);

Conclusion

Lustre/SCADE

Is a specialized language for critical embedded software

- having a limited but well chosen language constructions;
- mixing data-flow equations and state machines;
- with a precise and formalized semantics.

The main paradigm is the synchronicity

- assumption: computation in zero time
- time is abstracted by logical ticks