Automata in the Family of Synchronous Languages

Florence Maraninchi
VERIMAG / INPG, Grenoble, France
Contents

- Introduction: reactive systems and the synchronous approach
- Argos
- Argos + Lustre
- Mode-Automata
- Other kinds of automata and related work...
reactive systems and the synchronous approach
A Regulation System

\[
\text{actuator}_n = F(\text{sensor}_n, \text{sensor}_{n-1}, \ldots, \text{sensor}_0)
\]
An Event-Driven System
Example: a Digital Watch

OnSegment1\(_n\) = F (buttons\(_n\), buttons\(_{n-1}\), ... buttons\(_0\))
The Synchronous Approach: Typical Program

Initializations

while true

Get inputs $i$

Compute outputs $o$ ← Reactive kernel

Emit outputs $o$

$\Rightarrow$ Execution cycles = instants of a discrete logical time $N$

At instant $n \in N$, input $i_n$ and output $o_n$
Input/output Relations (I)

- General case: \( \forall n, o_n = f(i_0, i_1, \ldots, i_n) \)

  Remarks:
  \[ \begin{cases} 
  o_n \text{ may not depend on the future } i_{n+}\ldots \text{ (causality)} \\
  o_n \text{ may depend on } i_n \text{ (synchrony hypothesis)} \\
  \text{The history of inputs is not bounded} 
  \end{cases} \]

- Combinational circuits: \( \forall n, o_n = f(i_n) \)
  No memory
Input/output Relations (2)

- **Bounded-Memory** Systems (or Sequential Circuits):

\[
\forall n, \; o_n = f(Abs(i_0, i_1, ..., i_n))
\]

Where \( Abs \) is an abstraction function, such that \( \exists B \mid \text{Image}(Abs) \mid < B \)
Bounded-Memory Systems: Program Scheme

Initialization of the memory $M$

while true

**Invariant:** $M = \text{Abs}(i_0, i_1, \ldots, i_{n-1})$

Get input $i_n$

Compute output $o_n = f(M, i_n)$

Update memory $M = g(M, i_n)$

Property here: $M = \text{Abs}(i_0, i_1, \ldots, i_n)$

Emit output $o_n$

The abstraction function must be such that:

$\exists g, \text{Abs}(i_0, i_1, \ldots, i_n) = g(\text{Abs}(i_0, i_1, \ldots, i_{n-1}), i_n)$
Synchronous Languages for the Reactive kernel

- **dataflow** (lustre, Signal): Systems of Equations between the flows of values of i, o, M

- **Control Structures** (Esterel): parallel composition, watchdogs, ...

- **Automata** (Statecharts, Argos): explicit memory M and f (output function) and g (transition function) given in extension

- **Automata+dataflow** (Mode-Automata): dataflow equations attached to the states of an automaton.

F. Maraninchi, M. Jourdan (Phd 91–94)
References

http://www-verimag.imag.fr/~maraninx

- [CONCUR’92] : Definition, semantics, congruence
- [PROCOMET’94] : General framework
- [FTRTFT’96] : compilation into Boolean equations
- [ACM-SIGPLAN’94 and ’95] : examples
- [ICCL’94] : Argos + Lustre
- [CAV’93] : Timed Argos
- Habilitation (in french), pages 11-40
- The Journal of Computer Languages, 2001
Motivations (from Statecharts [Harel 87])

- Identify basic Statecharts constructs
- Define a synchronous semantics
- Compile programs into something that can be analysed by model-checking tools or equivalence-checking tools
- Produce executable code (at least for deterministic specs.)
- Study synchronous compositions of automata in a simple and general framework
Example: mod. 8 ‘a’ counter with start and stop commands
Statecharts version of the counter
Argos version of the counter

- Zero \(\rightarrow c \rightarrow c/\text{mod8} \rightarrow \text{One}\)
- Zero \(\rightarrow b \rightarrow b/c \rightarrow \text{One}\)
- Zero \(\rightarrow a \rightarrow a/b \rightarrow \text{One}\)

\(b, c, \text{mod8}\)

Start \(\rightarrow \text{OFF}\)

\(\text{Stop} \rightarrow \text{mod8/hi}\)
Programming with explicit states.
Mouse clicks: the problem

Inputs: click, top
Outputs: simple, double

Spec: Emits double whenever two clicks are received in less than 4 units of time, as defined by the occurrences of top. Emits single when a click is not followed by another click within 4 units of time.
Mouse clicks: a “flat” solution
Mouse clicks: a structured solution (1)

Idea: in state Counting, the program counts tops to 4 and then expires.
Mouse clicks: a structured solution (2)
Mouse clicks: a structured solution (3)

\[
\begin{align*}
\text{Counting} & : \text{int} : 0 \\
(cpt < N) & \rightarrow \text{top} / (cpt++) \\
(cpt = N) & \rightarrow \text{top} / \text{exp}
\end{align*}
\]

\[
\begin{align*}
\text{WaitFirstClick} & \rightarrow \text{click/double} + \\
\text{exp.--click/simple} & \rightarrow \text{CounterHas Expired}
\end{align*}
\]
Programming with explicit states.

Digital watch: the problem

Boolean Inputs: four EXCLUSIVE buttons $b_1, b_2, b_3, b_4$

$\forall i, j \in [1, 4]. i \neq j \implies b_i \land b_j = \text{false}$

Boolean Outputs: beep, on/off for each segment of the display

Spec: Emit the commands on/off to the display and the command beep according to the buttons $b_1, b_2, b_3, b_4$. 
A note on exclusive inputs (I)

- The hypothesis has to be “physically” relevant
- Knowing that inputs are exclusive helps simplifying the code (see parallel composition)
Digital watch: the interface

Display management (emitting OnSegment1, ...)

Counters
Argos - Definition

• Basic objects: Deterministic and Reactive Boolean Mealy machines ($M_{dr}$)

• Compositions: parallel and hierarchic + encapsulation

• Trace semantics: in terms of flat $M_{dr}$
Basic Objects of the Language (1)

A signal alphabet: $\mathcal{A} = \{\alpha, \beta, \gamma, \ldots\}$

A Boolean Mealy machine: $M = (S, s_0, I, O, T)$

Inputs/Outputs: $I, O \subseteq \mathcal{A}$, Transitions: $T \subseteq S \times \mathcal{B}(I) \times 2^O \times S$

$\mathcal{B}(I)$: Boolean Expressions with variables in $I$.

When $I = \{\alpha, \beta, \gamma\}$, the expression $\alpha \land \beta$ means: $(\alpha \land \beta \land \gamma) \lor (\alpha \land \beta \land \lnot \gamma)$
Basic Objects of the Language (2)

**Determinism**: For all states: \( i \neq j \implies b_i \land b_j = \text{false} \)

**Reactivity**: For all states: \( \lor b_i = \text{true} \)

Remark: no explicit loops in the concrete syntax.
A note on signal emission (I)

Transition labels are of the form: \( c/o \in B(I) \times 2^O \)

Comments:

- In the **input conditions**, true and false are treated symmetrically.
- In the **output part**, true and false are treated asymmetrically.

In other words: one cannot write: \( \text{some condition}/\neg a \)

Consequences: no contradictions in parallel compositions
Traces of a Boolean Mealy machine (1)

A signal configuration $\Sigma$ is a total function from $A$ to $B = \{0, 1\}$.

A state/input/output trace is a sequence of states and signal configurations indexed by integers.

A trace of $M = (S, s_0, I, O, T)$ with $I, O \subseteq A$ is a sequence $(S_0, \Sigma_0), (S_1, \Sigma_1), \ldots$ such that:

- $S_0 = s_0$
- $\forall n \geq 0. \exists (s, c/o, s') \in T. \text{such that:}$
  
  $c(\Sigma_n) \land$
  
  $S_n = s \land S_{n+1} = s' \land$
  
  $\Sigma_n[O] = o$
Traces of a Boolean Mealy machine (2)

If \( M = (S, s_0, I, O, T) \) is reactive, each trace may be extended.
We are interested in infinite traces only.

If \( M = (S, s_0, I, O, T) \) is deterministic, there are several traces starting in a given state, but they are distinguished by inputs only.

i.e., if we choose an initial state \( S_0 \) and a sequence of inputs \( \Sigma_0[I], \Sigma_1[I], \ldots \), then the sequence of outputs is unique.
Parallel Composition without communication

Cartesian product with conjunction of guards, union of output sets.

\[ \neg a \land \neg b \]

\[ a \land \neg b / A \]

\[ a \land \neg b / A, B \]

\[ a \land b / A, B \]

\[ a \land b / A \]

\[ a \land b / B \]
A note on exclusive inputs (2)

Component A with input \( a \)
Component B with input \( b \)

In their parallel composition: inputs \( a \) and \( b \).
If inputs \( a \) and \( b \) are not exclusive: build transitions triggered by \( \neg a.\neg b, a.\neg b, a.\neg b, a.b \).
If inputs \( a \) and \( b \) are exclusive: build \( \neg a.\neg b, a.\neg b, \neg a.b \) only.
A note on signal emission (2)

recall we cannot write: some condition/$\neg a$.
It would mean that a component imposes $a = false$

Consequences:
There are no possible contradictions between parallel components.
In a parallel composition, $a = false$ is the default behavior.
As soon as one component imposes $a = true$, the signal is emitted.
Hierarchy without communication
Hierarchy without communication comments

- **Priorities**: the outermost transitions have priority
- Processes are entered in their *initial* state
  (no memory, hierarchy $\neq$ suspension)
- **Exclusivity of inputs and signal emission**: same remarks as for parallel composition.
Compositions without communication...

Good property: they preserve both determinism and reactivity

Not very interesting:

- Parallel composition describes independent processes
- Hierarchy expresses interrupts only
  (no exceptions, no normal terminations)
  (we do not have cross-level arrows)
Communication: the synchronous broadcast

- Processes may send signals to each other as the main program sends signals to its environment (we’ll need a scoping mechanism).
- Sending is non blocking
- Any number of processes may receive the signal (broadcast)
- The reaction to a broadcast happens in the same instant as the emission (no delay)
Parallel Composition with communication (I)
Parallel Composition with communication (2)

\[
\begin{aligned}
&\neg a \\
&\neg b \\
&\neg b \\
&\neg a \\
&\neg a \\
\end{aligned}
\]
Parallel Composition with communication (3)
Communication and encapsulation (scoping)

First remarks...

- Encapsulation = scoping (syntactic) + enforcing synchronization

- A Flexible way of “cutting” parts of the whole Cartesian product: a communication (or synchronization) mechanism is only a way of removing parts in what we obtain by simply putting two components together (in parallel) in a completely independent way.
Synchronous Broadcast vs rendez-vous

- **CCS**, models for asynchronous things:
  - Parallel composition = interleaving
  - Synchronization = rendez-vous (symmetrical, binary, blocking)

- **Argos** (and Esterel, Lustre...):
  - Parallel composition = full Cartesian product, with all combinations of signals.
  - Synchronization = synchronous broadcast (asymmetrical, n-ary, non blocking)
What about hierarchy and communication?

- Argos hierarchy is a kind of “asymmetrical” parallel composition: the controller (the main automaton) runs in parallel with the process that refines its current state: they may communicate.
Argos Parallel and Hierarchic compositions

\[ A \vdash a/b \]
\[ B \vdash b \]
\[ A' \]
\[ B' \]
\[ A'B' = AB' = B' \]
Argos Hierarchy
= Esterel weak preemption

Preemption: AB is killed, whatever its internal state be weak: it may react at the instant when it is killed (and emit X)
Hierarchy with communication (I)
back to Statecharts cross-level arrows
Esterel weak preemption + suicide

Remark: Communication needs weak preemption
Hierarchy with communication (2)

Esterel strong preemption

\[(I \land \neg \alpha)/X\]

\[\alpha\]

\[\text{XA} \rightarrow \text{Y}\]
A useful construct: inhibition $< \alpha >$
Argos -- More examples

- The instantaneous dialogue
- A suspension mechanism
Instantaneous dialogue: the problem rendez-vous with synchronous broadcast

cause and in C
Instantaneous dialogue: Argos solution (1)
Instantaneous dialogue: Argos solution (2)
A suspension Mechanism

\[ A \xrightarrow{\alpha} P \]

\[ \xrightarrow{\alpha} \]

\[ \xrightarrow{\alpha} \]

\[ \xrightarrow{\alpha} \]

\[ \xrightarrow{\alpha} \]

run \quad suspend/\alpha

resume

suspend

\[ \xrightarrow{\alpha} \]

\[ \xrightarrow{\alpha} \]

tick/\alpha
Argos -- Formal definition

- Abstract grammar
- Operational semantics:
  - Flattening the constructs and the semantic function without errors
  - The Equational view
  - Incorrect programs
- Properties
Language Structure [CONCUR’92]

The set of programs: \( \mathcal{E} \)

\[
\begin{align*}
E & ::= E \| E \\
   & \quad | \quad E^T \\
   & \quad | \quad E^{<\gamma>} \\
   & \quad | \quad R_M(R_1, \ldots, R_n) \\
R & ::= E \quad | \quad \text{NIL}
\end{align*}
\]
Semantics by Flattening (without errors)

Let $\mathcal{M}_{dr}$ be the set of deterministic and reactive Boolean Mealy machines.
Define the operations:

$\times : \mathcal{M}_{dr} \times \mathcal{M}_{dr} \longrightarrow \mathcal{M}_{dr}$ \hspace{1cm} \text{parallel composition}

$\triangleright : \mathcal{M}_d \times 2^{\mathcal{M}_{dr}} \longrightarrow \mathcal{M}_{dr}$ \hspace{1cm} \text{refinement}

$\setminus \Gamma : \mathcal{M}_{dr} \times 2^A \longrightarrow \mathcal{M}_{dr}$ \hspace{1cm} \text{encapsulation}

$\neg \gamma : \mathcal{M}_{dr} \times A \longrightarrow \mathcal{M}_{dr}$ \hspace{1cm} \text{inhibition}
Semantics by Flattening: operation ×
Semantics by Flattening: operation ▷
Semantics by Flattening: operation $\{b\}$ (1)
Semantics by Flattening: operation \( \setminus \{b\} \) (2)

Keep the transition \( c/e \) if and only if:
\[
(b \in e \implies c \land b \neq \text{false}) \land (b \not\in e \implies c \land \neg b \neq \text{false})
\]
+ hiding of \( b \).

\[
\begin{align*}
A1 & \quad \neg a \land \neg b \\
B1 & \quad \neg a \land b/B \\
A2B1 & \quad a \land \neg b/b \\
\end{align*}
\]

\[
\begin{align*}
A1 & \quad \neg a \\
B1 & \quad a/B \\
A2B2 & \quad a \land b/b, B \\
\end{align*}
\]
Semantics with errors [CONCUR’92]

\[ S : \mathcal{E} \longrightarrow \mathcal{M}_{dr} \text{ (Det. and React. Mealy machines)} : \]

\[
S(E_1 \parallel E_2) = S(E_1) \times S(E_2)
\]

\[
S(R_M(R_1, ..., R_n)) = M \triangleright (S(R_1), ..., S(R_n))
\]

\[
S(\overline{E^T}) = S(E) \setminus \Gamma
\]

\[
S(\text{NIL}) = (\{\text{NIL}\}, \text{NIL}, \emptyset, \emptyset, \{(\text{NIL, true, } \emptyset, \text{NIL})\})
\]

= recursive application of the flattening operations on the abstract tree

(this is not a compiling technique !)
The Equational view: idea

State: A1B1C1
input: a
status of local and output signals:
\[ b = a = \text{true} \]
\[ c = b \]
\[ C = c \]
Unique solution:
\[ a = b = c = C = \text{true} \]

General form of the equations:
emitted signal = conditions under which it is emitted (given a state)
The Equational view: instantaneous dialogue (I)

State: AC
Input: cause
Equations:
\[ Q = cause \]
\[ Y = Q \]
Solution: \( Y = Q = cause = true \)
Transition: AC \(\xrightarrow{cause} BC\)
The Equational view: instantaneous dialogue (2)

State: AC
Input: cause
Equations:
\[
Q = \text{cause} \land \neg Y \\
\lor \text{cause} \land Y \\
Y = Q
\]
Solution: \(Y = Q = \text{cause} = \text{true}\)

Transition: AC \(\xrightarrow{\text{cause}}\) BC

Same solution, via Boolean calculus
Incorrect Programs: problem and examples

The encapsulation operation is not internal in $\mathcal{M}_{dr}$.

or:

The system of equations does not necessarily have a unique solution.

Typical Examples (borrowed from Esterel):

\[(i.a/b \parallel i.b/a)^{a,b}: 2 \text{ solutions, non-determinism appears for input } i\]

\[(i.\neg a/b \parallel i.b/a)^{a,b}: \text{no solution, non-reactivity appears for input } i\]
Incorrect Programs: what to do?

Non-determinism: what would be the operational semantics of a non-deterministic choice? (flip a coin at execution time?)

more details: [ESOP’96]

Non-reactivity: even worse

(Note that: no transition defined for state A and input i is different from:

- a “null” transition A \( \xrightarrow{i/\emptyset} A \).

\implies Declare such programs to be incorrect because we definitely cannot give them a behavior
New semantic function with errors

\[ S : \mathcal{E} \rightarrow \mathcal{M}_{dr} \cup \{ \bot \} \]

\[ S(E^\Gamma) = \begin{cases} 
\text{let } R = S(E) \setminus \Gamma \\
\text{in if } R \in \mathcal{M}_{dr} \text{ then } R \text{ else } \bot
\end{cases} \]

\( \bot \) is absorbant in all other operations.
Equivalences: definitions

Let $\approx$ be the trace equivalence of deterministic and reactive Boolean Mealy machines (i.e., the equality of the sets of input/output traces.

Equivalence of Argos programs:

\[ P_1 \equiv P_2 \iff \left\{ \begin{array}{l} S(P_1) \neq \bot \land S(P_2) \neq \bot \land S(P_1) \approx S(P_2) \vee \\
S(P_1) = S(P_2) = \bot \end{array} \right. \]
Argos compositionality: “the” result

≡ is a congruence for the Argos operators.

\[
P \equiv P' \implies \begin{cases} 
\forall Q. \ P \parallel Q \equiv P' \parallel Q \\
\forall \Gamma. \ \overline{P}^\Gamma \equiv \overline{P'}^{\overline{\Gamma}} \\
\forall \gamma. \ \overline{P}^{\langle \gamma \rangle} \equiv \overline{P'}^{\langle \overline{\gamma} \rangle} \\
\forall M. \forall R_1, \ldots, R_n \ \overline{R_M(P, R_1, \ldots, R_n)} \equiv R_M(P', R_1, \ldots, R_n) 
\end{cases}
\]
Argos – Compositionality advantages

- “separate” compilation + minimization
- Multi-language programming
- Definition of macro-notations
Compilation into dataflow equations

[FTRTFT’96]

Main ideas:
--- simple encoding of automata (one Boolean var. per state)
--- encoding each automata with its context (the part of the abstract tree which is above it), summarized by two variables (alive, killed).
Related Work

• Various semantics of Statecharts (none of them is synchronous)
  problems with incorrect programs, the synchronous hypothesis is not exploited fully, some of the high-level constructs are macros in Argos, and specific operators in these languages.

• SyncCharts (Ch. André, Nice)
  A graphical version of Esterel, quite complex

• Safe State Machines (SSM in SCADE, EsterelTech.)
  quite close to Argos
Argos + Lustre
The Synchronous Dataflow Language Lustre

node example (i : int) -- input
returns (X : int; Y : int); -- outputs
var M : bool; -- local variable
let -- (unsorted) set of eq. for local and output variables

\[ M = \text{true} \rightarrow \begin{cases} \text{if (pre}(M) \text{ and (X>20)) or ((not pre}(M)) \text{ and (X<0))} \\ \text{then not pre}(M) \text{ else pre}(M); \end{cases} \]

\[ X = 0 \rightarrow \begin{cases} \text{if pre}(M) \text{ then pre}(X) + Y + I \text{ else pre}(X) - Y - I; \end{cases} \]

\[ Y = 0 \rightarrow \begin{cases} \text{if pre}(M) \text{ then } i + \text{pre}(Y) \text{ else } i - \text{pre}(Y); \end{cases} \]

tel

The variable-dependency graph is acyclic
There exists a topological sort : M \rightarrow X \rightarrow Y \rightarrow i
Compiled Code = endless loop

\[ \text{init} = 1 ; \]
while (1)
    \[
    \text{read (i) -- interfacing code : read from sensors}
    \]
        if (init) \{ M = 1 ; X = Y = 0 ; init = 0 ; \}
        else \text{ -- new values: } nY, \text{ then } nX, \text{ then } nM
            y1 = i+Y ; y2 = i-Y;  nY = (M ? y1 : y2) ;
            x1 = X+nY+1; x2 = X-nY-1;  nX = (M ? x1 : x2 ) ;
            m1 = ! M ;
            nM =((M && (nX>20)) ||(!M && (nX<0)))? m1 : M);
            M = nM ; X = nX ; Y = nY ; \text{ -- update memory}
    print (X, Y) -- interfacing code : write to actuators

Remark: IF is \textit{strict}
Optimizations: avoid introducing variables \textit{nM, nX, nY}
Quality of the Compiled Code

```plaintext
init
while (true)
    read inputs from sensors
    compute outputs
    write outputs to actuators
```

Execution Time $\text{WCET} =$
max. time between two successive samplings of the inputs
Compiled code from Lustre (without clocks):
the WCET corresponds to all the nodes being computed
Argos and Lustre: ideas

• Argos programs can be encoded into Lustre programs (structurally)

• Lustre programs can be compiled into flat deterministic and reactive Boolean Mealy machines (= basic Argos components)

• Argos+Lustre is easy to define (but not to use!) [ICCL’94]

• Implementation: link-editing mechanism at the equational level.
Argos and Lustre: a simple example

Argos refinement implies that Lustre programs are re-initialized.
Mode-Automata

F. Maraninchi, Y. Rémond (Phd 1998–2001)
The notion of a running mode

- Aircraft flying/taxi/take-off/landing
- Launch phases of Ariane 5
- Commands for a robot arm, regulation of the heater
- Electrical load and unload in the Airbus

A mode: build a configuration of the system parameters such that another mode can start

Sequential decomposition into exclusive periods
Running Modes of Regulation Systems

\[ \text{comm} = f_1 (\text{pre} (\text{comm}), \text{inputs}) \]

\[ \text{comm} = f_2 (\text{pre} (\text{comm}), \text{inputs}) \]

\[ \ldots \]

\( f_1 \) and \( f_2 \) are not necessarily \textit{combinational} (they may depend on \text{pre}(\text{inputs}) \) for instance.)
A Simple System with Modes

Timing Diagram of a Variable X
Usual description in Lustre

\[ X = 0 \rightarrow \text{if Model then} \]
\[ \text{pre}(X) + 1 \]
\[ \text{else if Mode2 then} \]
\[ \text{pre}(X) + 1/2 \]
\[ \text{else} \]
\[ \text{pre}(X) - 2 \]

\[ \text{comm} = \text{initcomm} \rightarrow \]
\[ \text{if } \square \text{ then} \]
\[ f1(\text{pre}(\text{comm}), \text{inputs}) \]
\[ \text{else if } \square \text{ then} \]
\[ f2(\text{pre}(\text{comm}), \text{inputs}) \]
\[ \text{else if } \square \text{ then } \ldots \]

— Poor readability
— The mode structure is duplicated if several variables have the same modes
— The mode conditions are states
We would like...

(1) A separate description of each mode

1. Initially: $X = 0$

2. Then $X$ is updated according to 3 modes:
   
   - A) $X = \text{pre}(X) + 1$
   - B) $X = \text{pre}(X) + 1/2$
   - C) $X = \text{pre}(X) - 2$

+ (2) The way modes are organized (automata, control structures, ...)

Diagram:

- A
- B
- C

$X = 0$ $X = 5$ $X = 10$
A Proposal: Mode-Automata [ESOP98]

\[ X : \text{int} : 0 \]

Mode-Automata: About Modes and States for Reactive Systems

F. Maraninchi and Y. Rémond -- ESOP'98
Mode-Automata vs. Argos+Lustre

\[ X = 0 \rightarrow \text{pre}(X) + 1 \]

\[ X = 0 \rightarrow \text{pre}(X) - 1 \]

\[ X = \text{pre}(X) - 1 \]

\[ X = \text{pre}(X) + 1 \]
Mode-Automata vs. Argos+Lustre

Argos + Lustre = start and stop processes
A process = an independent program with initial state
Processes may explicitly communicate values
multi-language approach is possible

Mode-Automata = commute between modes
A mode = only a transition function
Variables have a global scope and
Values are implicitly transmitted
multi-language approach is less easy
Composing Mode-Automata (I)

\[
\begin{align*}
X &= \text{pre}(X) + 1 \\
Y &= \text{pre}(Y) + X \\
Z &= \text{pre}(Z) + Y
\end{align*}
\]

\[
\begin{align*}
X &= \text{pre}(X) - 1 \\
Y &= \text{pre}(Y) - X \\
Z &= \text{pre}(Z) - Y
\end{align*}
\]

\[
\begin{align*}
X &= 0 \\
Y &= 0 \\
Z &= 0
\end{align*}
\]

\[
\begin{align*}
X &= \text{pre}(X) - 1 \\
Y &= \text{pre}(Y) + X \\
Z &= \text{pre}(Z) + Y
\end{align*}
\]

\[
\begin{align*}
X &= \text{pre}(X) + 1 \\
Y &= \text{pre}(Y) - X \\
Z &= \text{pre}(Z) - Y
\end{align*}
\]

\[
\begin{align*}
X &= 0 \\
Y &= 0 \\
Z &= 0
\end{align*}
\]
Composing Mode-Automata (2)
References (Verimag)

- A Proposal : Mode-automata [ESOP’98]
- Compiling Mode-Automata [CC’00]
- Analysis of Real-Time Systems with Modes [ECRTS’00]
- Examples taken from Robotics [CDC’00]
- Yann Rémond’s thesis (the language and the compiler) [oct 01]
Related (ongoing) work on mode-automata
LucidSynchrone, Marc Pouzet, Lip6

let node two_states min max = x where
rec automaton
    S1 -> -- state
       do x = last x + 1 -- the equation attached
          until x = max then S2 -- the outgoing transition
    | S2 ->
       do x = last x - 1
          until x = min then S1
    end
and last x = 0 -- init value
Related (ongoing) work on mode-automata
Reluc, J.-L. Colaço, EsterelTech

A support for imperative structures in Lustre/SCADE. Should be available in SCADE V6.