Synchronous Reaction Semantics in SOS form

Marc Pouzet

ENS Paris
Marc.Pouzet@ens.fr

MPRI, October 3, 2017

Reaction semantics

- Define what is a valid synchronous reaction without considering how it is computed (its internal scheduling).
- Useful to study the semantics of well behave programs. Prove properties (e.g, equivalence) between two programs.
- In SOS form (Structural Operational Semantics [10]).
- Originaly introduced by Berry & Gonthier for Esterel [1]
- Defined here for a data-flow language.

A Basic Language

$$
\begin{array}{lcl}
D & ::= & D \,\texttt{and}\, D \mid x = e \mid \texttt{local}\ x\ \texttt{in}\ D \\
e & ::= & op(e_1, ..., e_n) \mid (e, e) \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid i \mid x \\
 & & \quad \mid \texttt{pre}\ i\ e \mid \texttt{init}\ i \\
i & ::= & \texttt{true} \mid \texttt{false} \mid ... \\
op^1 & ::= & \texttt{not} \mid ...
\end{array}
$$

Equations D; expressions e; immediate values i; operators op.

Notation shortcut:

e_1 `->` $e_2 =$ `if init` *true* `then` e_1 `else` e_2

Reaction semantics

Values:	v	$::= \quad i$
Environnement:	R	$::= \quad [v_1/x_1, ..., v_n/x_n] \ \ (\forall k, l, k \neq l \Rightarrow x_k \neq x_l)$
Composition:	R_1, R_2 tq $Dom(R_1) \cap Dom(R_2) = \emptyset$	
Reaction:	$R \vdash e_1 \xrightarrow{v} e_2 \quad R \vdash D \xrightarrow{R'} D'$	
Run:	$R.h \vdash D : R'.h'$	
History:	h	$::= \quad \epsilon \mid R.h$

If h is a sequence of length greater than n, we note $h(n)$ its n-th element.

The synchronous reaction:

- $R \vdash e_1 \xrightarrow{v} e_1'$ means that, under the local environment R, the expression e_1 produces the value v and rewrites to e_1'.
- $R \vdash D \xrightarrow{R'} D'$ means that, equation D produces R' and rewrites to D'.

Implicit rules:

Let $FV(D)$ be the list of free variables from D.

- Rules are considered modulo α-conversion (renaming).

$$\texttt{local}\ x\ \texttt{in}\ D \quad \equiv \quad \texttt{local}\ y\ \texttt{in}\ D[x/y] \text{ if } y \notin FV(D)$$

- Equations are considered modulo move of local declarations:

$$\begin{aligned}
\texttt{local}\ x\ \texttt{in}\ (D_1\ \texttt{and}\ D_2) &\equiv (\texttt{local}\ x\ \texttt{in}\ D_1)\ \texttt{and}\ D_2 \\
&\quad \text{if } x \notin FV(D_2) \\
\texttt{local}\ x\ \texttt{in}\ (D_1\ \texttt{and}\ D_2) &\equiv D_1\ \texttt{and}\ (\texttt{local}\ x\ \texttt{in}\ D_2) \\
&\quad \text{if } x \notin FV(D_1) \\
\texttt{local}\ x\ \texttt{in}\ \texttt{local}\ y\ \texttt{in}\ D &\equiv \texttt{local}\ y\ \texttt{in}\ \texttt{local}\ x\ \texttt{in}\ D \\
&\quad \text{if } x \neq y
\end{aligned}$$

Reaction rules:

$$\text{(Silent)}\quad \epsilon \vdash D : \epsilon$$

$$\text{(Sequence)}\quad \frac{R \vdash D \xrightarrow{R'} D' \qquad h \vdash D' : h'}{R.h \vdash D : R'.h'}$$

$$\text{(var)}\quad R[v/x] \vdash x \xrightarrow{v} x$$

$$\text{(const)}\quad R \vdash i \xrightarrow{i} i$$

$$\text{(Pre)}\quad \frac{R \vdash e_1 \xrightarrow{w} e_2}{R \vdash \texttt{pre}\ v\ e_1 \xrightarrow{v} \texttt{pre}\ w\ e_2}$$

$$\text{(op)}\quad \frac{\forall k \in [1..n],\ R \vdash e_k \xrightarrow{v_k} e'_k \qquad v = op(v_1, ..., v_n)}{R \vdash op(e_1, ..., e_n) \xrightarrow{v} op(e'_1, ..., e'_n)}$$

(If1)

$$\frac{R \vdash e_1 \xrightarrow{true} e_1' \qquad R \vdash e_2 \xrightarrow{v_2} e_2' \qquad R \vdash e_3 \xrightarrow{v_3} e_3'}{R \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \xrightarrow{v_2} \texttt{if } e_1' \texttt{ then } e_2' \texttt{ else } e_3'}$$

(If2)

$$\frac{R \vdash e_1 \xrightarrow{false} e_1' \qquad R \vdash e_2 \xrightarrow{v_2} e_2' \qquad R \vdash e_3 \xrightarrow{v_3} e_3'}{R \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \xrightarrow{v_3} \texttt{if } e_1' \texttt{ then } e_2' \texttt{ else } e_3'}$$

(Eq)

$$\frac{R(x) = v \qquad R \vdash e \xrightarrow{v} e'}{R \vdash x = e \xrightarrow{[v/x]} x = e'}$$

(Local)

$$\frac{R, [v/x] \vdash D \xrightarrow{R', [v/x]} D'}{R \vdash \texttt{local } x \texttt{ in } D \xrightarrow{R'} \texttt{local } x \texttt{ in } D'}$$

(Init)

$$R \vdash \texttt{init } i \xrightarrow{i} \texttt{init } false$$

(And)

$$\frac{R \vdash D_1 \xrightarrow{R_1} D_1' \qquad R \vdash D_2 \xrightarrow{R_2} D_2'}{R \vdash D_1 \texttt{ and } D_2 \xrightarrow{R_1, R_2} D_1' \texttt{ and } D_2'}$$

An Example

This semantics abstracts the way micro scheduling is done.

Example:

$$\texttt{local}\ z\ \texttt{in}\ x = z + y\ \texttt{and}\ z = (\texttt{pre}\ 0\ x) + 1$$

Given
$h = h_0.h_1.... = [1/y, 2/x].[2/y, 5/x].[3/y, 9/x].[4/y, 14/y]....$

$$\frac{\dfrac{\dfrac{h_0 \vdash z \xrightarrow{1} z \qquad h_0 \vdash y \xrightarrow{1} y}{h_0 \vdash z + y \xrightarrow{2} z + y}}{h_0, [1/z] \vdash x = z + y \overset{[2/x]}{\rightarrow} x = z + y} \quad (1)}{h_0, [1/z] \vdash \begin{array}{c} x = z + y \\ \texttt{and} \\ z = (\texttt{pre}\ 0\ x) + 1 \end{array} \overset{[2/x,1/z]}{\rightarrow} \begin{array}{c} x = z + y \\ \texttt{and} \\ z = (\texttt{pre}\ 2\ x) + 1 \end{array}}$$

For (1):

$$\dfrac{\dfrac{h_0, [1/z] \vdash x \xrightarrow{2} x}{h_0, [1/z] \vdash \texttt{pre } 0\ x + 1 \xrightarrow{1} (\texttt{pre } 2\ x) + 1}}{h_0, [1/z] \vdash z = (\texttt{pre } 0\ x) + 1 \xrightarrow{[1/z]} z = (\texttt{pre } 2\ x) + 1}$$

This semantics is magical and not operational!
It does not say how the reaction is made but what is a good reaction. This is why it has been also called logical semantics.

Rmk: It is also possible to define a micro-step operational semantics.

Interest of this semantics:

It is useful to compare programs/expressions and to prove that a compiler is correct.

Some program may have several possible reaction (non determinacy):

$$\frac{R, [\texttt{true}/x] \vdash x \xrightarrow{\texttt{true}} x}{R \vdash x = x \xrightarrow{[\texttt{true}/x]} x = x} \qquad \frac{R, [\texttt{false}/x] \vdash x \xrightarrow{\texttt{false}} x}{R \vdash x = x \xrightarrow{[\texttt{false}/x]} x = x}$$

Some program may have no reaction:

$$\frac{R, [?/x] \vdash x + 1 \xrightarrow{?} x + 1}{R \vdash x = x + 1 \xrightarrow{?}}$$

If a program is wrongly typed, it has no reaction. E.g., $1 +$ true.

Instead of a semantics that give a meaning to too many program, even non deterministic ones, define a deterministic semantics [13].

Some program are deterministic but are weird: [1]

$$\frac{R, [\texttt{true}/\mathit{tobe}] \vdash \mathit{tobe} \xrightarrow{\texttt{true}} \mathit{tobe} \qquad R, [\texttt{true}/\mathit{tobe}] \vdash \texttt{not}\ \mathit{tobe} \xrightarrow{\texttt{false}} \texttt{not}\ \mathit{tobe}}{R \vdash \mathit{tobe} = \mathit{tobe} \vee \texttt{not}\ \mathit{tobe} \xrightarrow{[\texttt{true}/\mathit{tobe}]} \mathit{tobe} = \mathit{tobe} \vee \texttt{not}\ \mathit{tobe}}$$

The only solution is tobe = true.

Should the semantics give meaning to weird programs as the one above?

What is the good criteriom to say that a program is reasonnable or not?

[1] This is the so-called "Hamlet" example due to Gérard Berry.

Causality

Causality: analyse the cause and consequences of an event. Track inconsistencies, critical races between event like, e.g., "this event is present if it is absent", or conversely.

Reproduce logical reasonning where computation time is abstracted.

Yet, the Hamlet program, if wired electricaly it is not "constructive", that is, the value of every output signal a consequence of known facts about the value of input signal.

What would result from the electrical implementation of the circuit? Certainly not generating 1 from no input.

Causality

Shiple and Berry defined constructive causality of an Esterel program as proving a boolean formula in constructive logic: only propagate known fact [12].

This observation has been fundamental! The intuition was that it coincides with synchronous circuits whose values stabilize in bounded time. The proof has been done only recently, by Mendler [8].

Read "The constructive semantics of Esterel" [3].

Find statically checkable conditions to say that a program has a unique possible reaction and is "reasonnable".

The Simple Lustre-like Solution:

All feedback loops must cross an explicit delay (`pre`).

We shall focus on that topic later.

Function definitions

- In **Lustre**, there are two distinct name spaces.
- One for functions; the other for sequences of values.

$$
\begin{array}{lcl}
d & ::= & \texttt{node}\, f(x) = e\ \texttt{with}\ D \\
prog & ::= & d\ \ldots\ d \\
D & ::= & \ldots \mid x = f(e)
\end{array}
$$

Semantics for functions

Values: $v ::= i$

Environment: $R ::= [v_1/x_1, ..., v_n/x_n]$

$G ::= [\lambda y_1.e_1 \texttt{ with } D_1/f_1, ..., \lambda y_m.e_m \texttt{ with } D_m/f_m]$
with $i \neq j \Rightarrow f_i \neq f_j$

Reaction: $G, R \vdash e_1 \xrightarrow{v} e_2$

$G, R \vdash D \xrightarrow{R'} D'$

Run: $G, R.h \vdash D : R'.h'$

- G is a global environment of function definitions.
- A node definition $d_1 = \texttt{node}\, f_1(x_1) = e_1 \texttt{ with } D_1$ defines $[\lambda y_1.e_1 \texttt{ with } D_1/f_1]$.
- A collection of declarations $d_1, ..., d_n$ defines a global environment G with:

$$G ::= [\lambda y_1.e_1 \texttt{ with } D_1/f_1, ..., \lambda y_n.e_n \texttt{ with } D_n/f_n]$$

Function Call

Previous rules are unchanged. We now write $G, R \vdash e \xrightarrow{v} e'$ instead of $R \vdash e \xrightarrow{v} e'$.

The function call $f(e)$ is replaced by the body of f put in parallel with the computation of e.

(Call)

$$\frac{G(f) = \lambda y.e_2 \texttt{ with } D \qquad G, R \vdash \texttt{local } y \texttt{ in } (x = e_2 \texttt{ and } y = e_1 \texttt{ and } D) \xrightarrow{R'} D'}{G, R \vdash x = f(e_1) \xrightarrow{R'} D'}$$

Slow and Fast Processes

We keep the same parallel composition with processes progressing in lock-step. Add an explicit silent (or absent) value (*abs*).
This approach is reminiscent to that of Milner for encoding CCS in SCCS [9].

Values:	v	$::= \quad i \mid abs$
Environnement:	R	$::= \quad [v_1/x_1, ..., v_n/x_n]$ with $i \neq j \Rightarrow x_i \neq x_j$
Reaction:	$G, R \vdash e_1 \xrightarrow{v} e_2$	
	$G, R \vdash D \xrightarrow{R'} D'$	
Run:	$R.h \vdash D : R'.h'$	

In the next slide, G is left implicit in all the rules, i.e., we write:

$$R \vdash e_1 \xrightarrow{v} e_1' \qquad R \vdash D \xrightarrow{R'} D'$$

(Const-abs)
$$R \vdash i \xrightarrow{abs} i$$

(Const)
$$R \vdash i \xrightarrow{i} i$$

(Eq)
$$R[v/x] \vdash x \xrightarrow{v} x$$

(Op-abs)
$$\frac{\forall j \in [1..n]\ R \vdash e_j \xrightarrow{abs} e'_j}{R \vdash op(e_1, ..., e_n) \xrightarrow{abs} op(e'_1, ..., e'_n)}$$

(Op)
$$\frac{\forall j \in [1..n]\ R \vdash e_j \xrightarrow{i_j} e'_j \qquad i = op(i_1, ..., i_n)}{R \vdash op(e_1, ..., e_n) \xrightarrow{i} op(e'_1, ..., e'_n)}$$

(Pre-abs)
$$\frac{R \vdash e \xrightarrow{abs} e'}{R \vdash \texttt{pre}\ i\ e \xrightarrow{abs} \texttt{pre}\ i\ e'}$$

(Pre)
$$\frac{R \vdash e \xrightarrow{j} e'}{R \vdash \texttt{pre}\ i\ e \xrightarrow{i} \texttt{pre}\ j\ e'}$$

(When-abs)

$$\frac{R \vdash e_1 \xrightarrow{abs} e_1' \qquad D \vdash e_2 \xrightarrow{abs} e_2'}{R \vdash e_1 \texttt{ when } e_2 \xrightarrow{abs} e_1' \texttt{ when } e_2'}$$

(When-true)

$$\frac{R \vdash e_1 \xrightarrow{i} e_1' \qquad D \vdash e_2 \xrightarrow{true} e_2'}{R \vdash e_1 \texttt{ when } e_2 \xrightarrow{i} e_1' \texttt{ when } e_2'}$$

(When-false)

$$\frac{R \vdash e_1 \xrightarrow{i} e_1' \qquad R \vdash e_2 \xrightarrow{false} e_2'}{R \vdash e_1 \texttt{ when } e_2 \xrightarrow{abs} e_1' \texttt{ when } e_2'}$$

(MERGE-ABS)

$$\frac{R \vdash e_1 \xrightarrow{abs} e_1' \qquad R \vdash e_2 \xrightarrow{abs} e_2' \qquad R \vdash e_3 \xrightarrow{abs} e_3'}{R \vdash \texttt{merge}\ e_1\ e_2\ e_3 \xrightarrow{abs} \texttt{merge}\ e_1'\ e_2'\ e_3'}$$

(MERGE-TRUE)

$$\frac{R \vdash e_1 \xrightarrow{true} e_1' \qquad R \vdash e_2 \xrightarrow{i} e_2' \qquad R \vdash e_3 \xrightarrow{abs} e_3'}{R \vdash \texttt{merge}\ e_1\ e_2\ e_3 \xrightarrow{i} \texttt{merge}\ e_1'\ e_2'\ e_3'}$$

(MERGE-FALSE)

$$\frac{R \vdash e_1 \xrightarrow{false} e_1' \qquad R \vdash e_2 \xrightarrow{abs} e_2' \qquad R \vdash e_3 \xrightarrow{i} e_3'}{R \vdash \texttt{merge}\ e_1\ e_2\ e_3 \xrightarrow{i} \texttt{merge}\ e_1'\ e_2'\ e_3'}$$

Synchronous execution

This semantics is partial, that is, some rules are lacking. E.g., there is no rule like:

$$\frac{R \vdash e_k \xrightarrow{i_k} e'_k \qquad \forall j \in [1..n]\ j \neq k\ R \vdash e_j \xrightarrow{abs} e'_j}{R \vdash op(e_1, ..., e_n) \xrightarrow{abs} op(e'_1, ..., e'_n)} \text{(OP)}$$

More generally, if one (or more) input is present while one (or more) is absent, there is no possible reaction.

This would need to store the present value leading possibly to an unbounded buffering.

The purpose of the clock calculus is to ensure, at compile time, that such situation do not arrive.

Determinism/Reactivity

Definition (Synchronized values)

Two reaction environment R_1 and R_2 are compatible, written $R_1 \sim R_2$ iff:

$$(Dom(R_1) = Dom(R_2)) \wedge (\forall x \in Dom(R_1).(R_1(x) = abs) \Leftrightarrow (R_2(x) = abs$$

By extension, $(h \sim h')$ iff $\forall i.h(i) \sim h'(i)$.

Definition (Determinim)

D is deterministic iff:

$$\forall h, h_1, h_2.(h_1 \sim h_2) \wedge (h, h_1 \vdash D : h_1) \wedge (h, h_2 \vdash D : h_2) \Rightarrow (h_1 = h_2)$$

Definition (Reactivity)

D is reactive iff $\forall R, \exists R', D'.\ R, R' \vdash D \xrightarrow{R'} D'$

Example: The equation $x = x$ is not deterministic.
Take h such that $h(i) = [0/x]$ and h' such that $h(i) = [1/x]$.

Higher-order I

The more general case with streams of (stream) functions.

$$\begin{array}{lll} D & ::= & D \,\texttt{and}\, D \mid x = e \mid x = e(e) \\ e & ::= & \ldots \mid \lambda x.e \;\texttt{with}\; D \end{array}$$

Values:	v	$::= \quad i \mid \lambda x.e \;\texttt{with}\; D$
Environment:	R	$::= \quad [v_1/x_1, \ldots, v_n/x_n]$
Reaction:	$R \vdash e_1 \xrightarrow{v} e_2$	
	$R \vdash D \xrightarrow{R'} D'$	
Run:	$R.h \vdash D : R'.h'$	

Higher-order II

- Previous rules stay unchanged.
- Add rules for function abstraction/application.

(App)

$$\frac{R \vdash e_1 \overset{\lambda y.e \texttt{ with } D}{\rightarrow} e_1' \qquad R \vdash x = e \texttt{ and } y = e_2 \texttt{ and } D \overset{R'}{\rightarrow} D'}{R \vdash x = e_1(e_2) \overset{R'}{\rightarrow} D'}$$

(Abs)

$$R \vdash \lambda x.e \texttt{ with } D \overset{\lambda x.e \texttt{ with } D}{\rightarrow} \lambda x.e \texttt{ with } D$$

(Local)

$$\frac{R, [v/x] \vdash D \overset{R,[v/x]}{\rightarrow} D' \qquad x \notin FV(v)}{R \vdash \texttt{local } x \texttt{ in } D \overset{R}{\rightarrow} \texttt{local } x \texttt{ in } D'}$$

Control Structures

This language is data-flow only: a set of equations D react forever. Two new situations can be considered:

- Activate an equation only when a boolean condition is true.
- Preempt an equation according to a boolean condition.
- Re-initialize an equation according to a boolean condition.

These type of constructs is the basis of Esterel.

Read the paper by Berry and Gonthier [1] and Berry [2].

Tardieu [13] gives an alternative presentation of the semantics for causally correct programs only.

Colaco et al. [6] define the reaction semantics of a data-flow language with hierarchical state machines (SCADE 6).

Control-structures can be compiled into a data-flow programs [11, 7].

Modular Reset

We extend the first language with a construction which reset an expresion e or equation D according to a boolean condition.

$$D \quad ::= \quad \ldots \mid \texttt{reset}\, D\, \texttt{every}\, e$$

The new predicate for a synchronous reaction becomes:

Reaction: $R \vdash_k e_1 \xrightarrow{v} e_2 \quad R \vdash_k D \xrightarrow{R'} D'$

Run: $R.h \vdash_k D : R'.h'$

History: $h \quad ::= \quad \epsilon \mid R.h \quad$ with $k \in \{0, 1\}$

Intuition

When $k = 1$, the expression e or equation D is reset, that is, the initialization operation `init` v restarts with value `true`.
Otherwise, $k = 0$ (continue)

Reaction semantics

- $R \vdash_k e_1 \xrightarrow{v} e_1'$ means that, under the local environment R, the expression e_1 produces the value v and rewrites to e_1'.
- $R \vdash_k D \xrightarrow{R'} D'$ means that, equation D produces R' and rewrites to D'.
- $k = 1$ means e or D is reset.

(Init-false)
$$H \vdash_0 \texttt{init}\ v \xrightarrow{v} \texttt{init}\ \texttt{false}$$

(Init-true)
$$H \vdash_1 \texttt{init}\ v \xrightarrow{\texttt{true}} \texttt{init}\ \texttt{false}$$

(Reset)
$$\frac{H \vdash_k e \xrightarrow{v} e' \qquad H \vdash_{k \vee v} D \xrightarrow{R'} D'}{H \vdash_k \texttt{reset}\ D\ \texttt{every}\ e \xrightarrow{R'} \texttt{reset}\ D'\ \texttt{every}\ e'}$$

with $k \vee \texttt{true} = 1$, $k \vee \texttt{false} = k$.
Other rules are left unchanged (add the subscript k everywhere)

Questions

- Add a construction to activate a computation only when a condition is true;
- Define the semantics for hierachical automata.
- How to model processes that terminate, e.g., do D `until` e ?
- How to model a process that perform a division par zero ?

The industrial language SCADE 6 [2] have all the above extensions.

- How to model a mix of imperative features and synchronous composition as found in Esterel? Read [3].

An alternative to define the semantics is to manipulate streams and interpret a synchronous program as a stream transformer.

- Read [4].
- Read [5] which shows the correspondance between co-induction and co-iteration for synchronous functions.

[2] `http://www.esterel-technologies.com/products/scade-suite/`


G. Berry and G. Gonthier.

The Esterel synchronous programming language, design, semantics, implementation.

Science of Computer Programming, 19(2):87–152, 1992.


Gérard Berry.

Preemption in concurrent systems.

In *FSTTCS*, pages 72–93, 1993.


Gérard Berry.

The constructive semantics of pure esterel.

Draft book. Available at: `http://www-sop.inria.fr/members/Gerard.Berry/Papers/EsterelConstructiveBook.pdf`, 1999.


Sylvain Boulmé and Grégoire Hamon.

Certifying Synchrony for Free.

In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 2250, La Havana, Cuba, December 2001. Lecture Notes in Artificial Intelligence, Springer-Verlag.

Short version of *A clocked denotational semantics for Lucid-Synchrone in Coq*, available as a Technical Report (LIP6), at `www.di.ens.fr/~pouzet/bib/bib.html`.

Paul Caspi and Marc Pouzet.

A Co-iterative Characterization of Synchronous Stream Functions.

In *Coalgebraic Methods in Computer Science (CMCS'98)*, Electronic Notes in Theoretical Computer Science, March 1998.

Extended version available as a VERIMAG tech. report no. 97–07 at `www.di.ens.fr/~pouzet/bib/bib.html`.

Jean-Louis Colaço, Grégoire Hamon, and Marc Pouzet.

Mixing Signals and Modes in Synchronous Data-flow Systems.

In *ACM International Conference on Embedded Software (EMSOFT'06)*, Seoul, South Korea, October 2006.

Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet.

A Conservative Extension of Synchronous Data-flow with State Machines.

In *ACM International Conference on Embedded Software (EMSOFT'05)*, Jersey city, New Jersey, USA, September 2005.

Michael Mendler, Thomas R. Shiple, and Gérard Berry.

Constructive boolean circuits and the exactness of timed ternary simulation.

Form. Methods Syst. Des., 40(3):283–329, June 2012.

Robin Milner.

Communication and Concurrency.

Prentice Hall, 1989.

Gordon Plotkin.

A structural approach to operational semantics.

Technical report, University of Aarhus, Denmark, 1981.

Dumitru Potop-Butucaru, Stephen A. Edwards, and Gérard Berry.

Compiling Esterel.

Springer, 2010.

Thomas R. Shiple and Gérard Berry.

Constructive analysis of cyclic circuits.

In *Proceedings of the International Design and Test Conference ITDC 96*, Paris, France, 1996.

Olivier Tardieu.

A Deterministic Logical Semantics for Esterel.

In *SOS Workshop*, London, United Kingdom, August 2004.