# Goto and Concurrency Introducing Safe Jumps in ESTEREL

Olivier Tardieu [1]

*INRIA Sophia Antipolis, France*

**Abstract**

ESTEREL is a design language for the specification of real time embedded systems. Based on the synchronous concurrency paradigm, its semantics describes execution as a succession of instants of computation. In this work, we consider the introduction of a new `gotopause` instruction in the language, which acts as a non-instantaneous jump instruction compatible with concurrency. It allows the programmer to activate state control points anywhere in the program, from where the execution is resumed in the next instant. In order to provide the formal semantics of the extended language, we first define a state semantics of ESTEREL, which we prove observationally equivalent to the original logical behavioral semantics. Including `gotopause` in the state semantics is then straightforward. We sketch two key applications of our new primitive: a direct encoding of automata and a quasi-linear rewriting of programs eliminating schizophrenic behaviors.

*Key words:* synchronous languages, program transformations.

## 1 Introduction

ESTEREL [4,5,6,7,8] is a high-level control-oriented synchronous reactive language (Section 2). Sophisticated control-flow patterns can be built through sequential and parallel compositions of behaviors, tests, loops and preemption mechanisms. These are all structural statements. No jump instruction is available or easily encoded in ESTEREL. This has important drawbacks, such as making flat automata encoding unnaturally difficult [1]. Therefore, the opportunity of adding a goto-like construct to the language is a subject of debate.

On one hand, such an extension would produce a more expressive language, allowing more compact specifications, with an enhanced support for automata. Moreover, most compilers [4,9,11,16] for ESTEREL are based on intermediate formats, languages or representations that involve jumps so that the implementation of this instruction should be straightforward.

On the other hand, gotos are widely regarded as a bad idea [10], especially in a concurrent framework such as ESTEREL, where they can easily break the semantics. In addition, specific correctness issues such as instantaneous loop detection [18] could become much worse because of gotos.

[1] Email: `olivier.tardieu@sophia.inria.fr`

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*
*URL:* `www.elsevier.nl/locate/entcs`

```
1: pause;
action 1;
present A then gotopause 2 end;
present B then gotopause 3 end;
gotopause 1;

2: pause;
action 2;
gotopause 1;

3: pause;
action 3;
gotopause 2
```

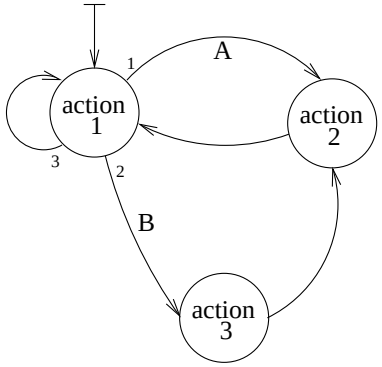


Fig. 1. Automata in Esterel$^{sp}$

Thus, to the best of our knowledge, no successful attempt of extension in this direction has been reported that provides both the necessary formal background and convincing practical applications.

In this paper, we detail what we believe is the right way to extend Esterel. We add to the language an instruction we call `gotopause`. Both `pause` and `gotopause` instructions are now labeled. When the control reaches some "`gotopause` *label*", it stops for the current instant, as if it had reached a regular `pause` instruction. However, when the execution is resumed in the following instant, instead of restarting from the "`gotopause` *label*" location, it starts from the corresponding "*label*: `pause`" location. This allows to branch non-instantly to a remote section of the program.

Thanks to non-instantaneity, the semantics remains simple enough to be defined and proved to be adequate, as well as understood and used.

Because of the non-locality of branching, such an extension usually requires some kind of continuation-passing style semantics. In the case of Esterel, we first have to reformulate the standard logical behavioral semantics [4,18] in the form of a state semantics that we prove observationally equivalent (Section 3). Then, we introduce and formalize `gotopause` and the semantics of the extended language, which we call Esterel$^{sp}$ (Section 4).

Automata made of non-instantaneous transitions are now easily encoded with conditional jumps, as shown by the example of Figure 1. More importantly, quasi-linear reincarnation [4,14,17] can be achieved by a simple preprocessing in Esterel$^{sp}$ (Section 5).

## 2 The Pure Esterel Kernel Language

Esterel [4,5,6,7,8] is an imperative synchronous programming language dedicated to reactive systems [12,13]. Pure Esterel is the fragment of the full Esterel language where data variables and data-handling primitives are abstracted away. As our only concern is with control-flow primitives, we concen-

| | |
|---|---|
| `nothing` | does nothing |
| `pause` | retains the control until next tick |
| `signal` $S$ `in` $p$ `end` | declares signal $S$ in $p$ |
| `emit` $S$ | emits $S$ (i.e. $S$ is present) |
| `present` $S$ `then` $p$ `else` $q$ `end` | if $S$ is present then does $p$ else does $q$ |
| `trap` $T$ `in` $p$ `end` | declares and catches exception $T$ in $p$ |
| `exit` $T$ | raises exception $T$ which propagates upward |
| $p$ `;` $q$ | $\begin{cases} \text{first starts } p \text{ then } q \text{ if/when } p \text{ terminates} \\ \text{terminates if/when } q \text{ does} \end{cases}$ |
| `[`$p$ `||` $q$`]` | $\begin{cases} \text{starts } p \text{ in parallel with } q \\ \text{terminates when both } p \text{ and } q \text{ are done} \end{cases}$ |
| `loop` $p$ `end` | repeats $p$ forever |

Fig. 2. Pure ESTEREL

trate in this paper on Pure ESTEREL. Moreover, without loss of generality, we focus on the Pure ESTEREL kernel language as defined by Berry in [4], which retains just enough of the language syntax to attain its full expressive power. Finally, for lack of space, we do not consider the statement `suspend` in the sequel. It raises no particular problem.

Figure 2 describes the grammar of this language, as well as the intuitive behavior of its constructs. The non-terminals $p$ and $q$ denote *statements* (i.e. *programs*), $S$ *signals*, and $T$ *exceptions*. An ESTEREL program runs in steps called *reactions* in response to the *ticks* of a *global clock*. Each reaction takes one *instant*. When the clock ticks, a reaction occurs. It may either finish the execution instantly or delay (part of) it till the next tick, because of `pause` instructions.

"`emit A; pause; emit B; emit C; pause; emit D`" emits the signal A in the first instant of its execution, then emits B and C in the second instant, then emits D and terminates in the third instant. It takes three instants to complete, or in other words, proceeds by three reactions.

Sequences, tests and (infinite) loops are the usual control-flow operators. Execution propagates in parallel branches in a deterministic synchronous way. "`emit A;[pause;emit B;pause;emit D||emit C;pause;emit E];emit F`" emits A and C, then B and E, and finally D and F.

Instantly broadcast signals and exceptions provide ways of interaction with the environment (through free identifiers) as well as local communication channels (through identifiers lexically scoped by `signal` and `trap` instructions).

An ESTEREL program $p$ has a tree structure. Three nodes have more than one child: the *test* (of presence), *sequence* and *parallel* nodes. Thus, two disjoint sub-terms $q$ and $r$ of $p$ are connected through one of these three operators. We note "$q//r$" and say that $q$ and $r$ are *compatible* if $q$ and $r$ are non-disjoint or composed in parallel; we note "$q\#r$" and say that $q$ and $r$ are *exclusive* otherwise. For example, in "`p;[q||r]`", $p$ and $q$ are exclusive, $p$ and $r$ are exclusive, $q$ and $r$ are compatible.

## 3 From the Logical Semantics to a State Semantics

We consider a family of semantics for ESTEREL. Reactions of a term $p$ are specified in a structural operational style [15] by a labeled transition system:

$$p \xrightarrow[E]{E',k} p'$$

The terms $p$ and $p'$ will either be programs or states (cf. Section 3.4). We call *domain* of the semantics the set of terms $p$ it applies to. The sets $E$ of *present signals* and $E'$ of *emitted signals* encode [2] the I/Os of the reaction. The integer $k$ is the *completion code* of the reaction and the term $p'$ its *residual*:

- If $k = 1$ then this reaction does not complete the execution of $p$.
  It has to be continued by the execution of $p'$ in the next instant.
- If $k \neq 1$ then this reaction ends the execution of $p$ ($p'$ is never executed):
  · $k = 0$ if the execution completes normally,
  · $k = k_T \geq 2$ if exception $T$ escapes from $p$, aborting the execution [3].

In general, a term $p$ may admit zero, one or many possible reactions and thus executions. An execution is a potentially infinite chain of reactions such that all completion codes but the last one are equal to 1:

$$p \xrightarrow[E_1]{E'_1,1} p_1 \xrightarrow[E_2]{E'_2,1} \ldots \xrightarrow[E_n]{E'_n,k\neq 1} p_n \quad \text{or} \quad p \xrightarrow[E_1]{E'_1,1} p_1 \xrightarrow[E_2]{E'_2,1} \ldots \xrightarrow[E_n]{E'_n,1} \ldots$$

### *3.1 Bisimulation and Observational Equivalence*

In order to compare various semantics, we need mathematical tools: bisimulation and observational equivalence [3]. Because the residual $p'$ of a reaction is irrelevant if $k$ is not 1 (i.e. not further executed), we consider an ad hoc definition of *1-bisimulation* in which we require $p' \sim q'$ only if $k$ is 1.

**Definition 3.1** *1-bisimulation.* Let $\rightarrow$ and $\mapsto$ be two semantics of respective domains $\mathcal{P}$ and $\mathcal{Q}$. A 1-bisimulation between $\rightarrow$ and $\mapsto$ is a relation $\sim$ of domain $\mathcal{P} \times \mathcal{Q}$ such that:

- for all $p \in \mathcal{P}$ there exists $q \in \mathcal{Q}$ such that $p \sim q$
- for all $q \in \mathcal{Q}$ there exists $p \in \mathcal{P}$ such that $p \sim q$
- if $p \sim q$ and $p \xrightarrow[E]{E',k} p'$ then there exists $q'$ such that $\begin{cases} q \xmapsto[E]{E',k} q' \\ k = 1 \Rightarrow p' \sim q' \end{cases}$
- if $p \sim q$ and $q \xmapsto[E]{E',k} q'$ then there exists $p'$ such that $\begin{cases} p \xrightarrow[E]{E',k} p' \\ k = 1 \Rightarrow p' \sim q' \end{cases}$

[2] The sets $I$ and $O$ of input and output signals are such that $E = I \cup O$ and $E' = O$. Program $p$ reacts to inputs $I$ with outputs $O$, completion code $k$ and residual $p'$ iff $p \xrightarrow[I\cup O]{O,k} p'$.

[3] Exceptions are numbered according to priorities [4,18], so that if $p$ terminates with completion code $k$ and $q$ with code $l$ then $[p \,||\, q]$ terminates with code $\max(k, l)$.

**Definition 3.2** *Observational Equivalence.* If there exists a 1-bisimulation between two semantics, we say these semantics are observationally equivalent.

In particular, if $\rightarrow$ and $\mapsto$ are observationally equivalent then:

- if $p_0 \xrightarrow[E_1]{E'_1,1} p_1 \xrightarrow[E_2]{E'_2,1} \ldots \xrightarrow[E_n]{E'_n,k} p_n$ then $\exists q_0 : q_0 \overset{E'_1,1}{\underset{E_1}{\longmapsto}} q_1 \overset{E'_2,1}{\underset{E_2}{\longmapsto}} \ldots \overset{E'_n,k}{\underset{E_n}{\longmapsto}} q_n$
- if $\forall n \geq 0, p_n \xrightarrow[E_{n+1}]{E'_{n+1},1} p_{n+1}$ then $\exists (q_n)_{n\geq 0} : \forall n \geq 0, q_n \xrightarrow[E_{n+1}]{E'_{n+1},1} q_{n+1}$

### *3.2 Logical Behavioral Semantics*

In Figure 3, we define a class of logical behavioral semantics for ESTEREL, parametrized by the auxiliary functions $\delta^k$. We note $\rightarrow$ the usual logical behavioral semantics [18] obtained by setting $\delta^k$ equal to the identity for all $k$.

$$(1)\ \texttt{nothing} \xrightarrow[E]{\emptyset,0} \texttt{nothing} \qquad \frac{S \in E \quad p \xrightarrow[E]{E',k} p'}{\texttt{present}\ S\ \texttt{then}\ p\ \texttt{else}\ q\ \texttt{end} \xrightarrow[E]{E',k} p'} \quad (7)$$

$$(2)\ \texttt{pause} \xrightarrow[E]{\emptyset,1} \texttt{nothing} \qquad \frac{S \notin E \quad q \xrightarrow[E]{F',l} q'}{\texttt{present}\ S\ \texttt{then}\ p\ \texttt{else}\ q\ \texttt{end} \xrightarrow[E]{F',l} q'} \quad (8)$$

$$(3)\ \texttt{exit}\ T \xrightarrow[E]{\emptyset,k_T} \texttt{nothing} \qquad \frac{p \xrightarrow[E]{E',k} p' \quad k = 0 \text{ or } k = k_T}{\texttt{trap}\ T\ \texttt{in}\ p\ \texttt{end} \xrightarrow[E]{E',0} \texttt{nothing}} \quad (9)$$

$$(4)\frac{S \in E}{\texttt{emit}\ S \xrightarrow[E]{\{S\},0} \texttt{nothing}} \qquad \frac{p \xrightarrow[E]{E',k} p' \quad k > 0 \text{ and } k \neq k_T}{\texttt{trap}\ T\ \texttt{in}\ p\ \texttt{end} \xrightarrow[E]{E',k} \delta^k(\texttt{trap}\ T\ \texttt{in}\ p'\ \texttt{end})}(a)$$

$$(5)\ \frac{p \xrightarrow[E]{E',k} p' \quad k \neq 0}{p\texttt{;}\ q \xrightarrow[E]{E',k} \delta^k(p'\texttt{;}\ q)} \qquad \frac{p \xrightarrow[E]{E',k} p' \quad q \xrightarrow[E]{F',l} q' \quad m = \max(k,l)}{\texttt{[}p\ \texttt{||}\ q\texttt{]} \xrightarrow[E]{E'\cup F',m} \delta^m(\texttt{[}p'\ \texttt{||}\ q'\texttt{]})} \quad (b)$$

$$(6)\ \frac{p \xrightarrow[E]{E',0} p' \quad q \xrightarrow[E]{F',l} q'}{p\texttt{;}\ q \xrightarrow[E]{E'\cup F',l} q'} \qquad \frac{p \xrightarrow[E]{E',k} p' \quad k \neq 0}{\texttt{loop}\ p\ \texttt{end} \xrightarrow[E]{E',k} \delta^k(p'\texttt{;}\ \texttt{loop}\ p\ \texttt{end})} \quad (c)$$

$$(d)\frac{p \xrightarrow[E\cup\{S\}]{E',k} p' \quad S \in E'}{\texttt{signal}\ S\ \texttt{in}\ p\ \texttt{end} \xrightarrow[E]{E'\setminus\{S\},k} \delta^k(\texttt{signal}\ S\ \texttt{in}\ p'\ \texttt{end})}$$

$$(e)\ \frac{p \xrightarrow[E\setminus\{S\}]{E',k} p' \quad S \notin E'}{\texttt{signal}\ S\ \texttt{in}\ p\ \texttt{end} \xrightarrow[E]{E',k} \delta^k(\texttt{signal}\ S\ \texttt{in}\ p'\ \texttt{end})}$$

Fig. 3. Logical Behavioral Semantics

We note $\mapsto$ the semantics corresponding to the alternate definition of $\delta^k$:

$$\delta^k(p) = \begin{cases} \texttt{nothing} & \text{if } k \neq 1 \\ p & \text{if } k = 1 \end{cases}$$

While matching the original semantics in term of observational equivalence, our alternate version of the semantics enjoys an extra normalization property: it maps completed or aborted executions to `nothing`. For example,

$$\texttt{[nothing || nothing]} \xrightarrow[E]{\emptyset, 0} \texttt{[nothing || nothing]}$$
$$\texttt{[nothing || nothing]} \overset{\emptyset, 0}{\underset{E}{\longmapsto}} \texttt{nothing}$$

**Theorem 3.3** *Equivalence.* $\rightarrow$ *and* $\mapsto$ *are observationally equivalent.*

**Proof.** By defining a 1-bisimulation between $\rightarrow$ and $\mapsto$. □

**Theorem 3.4** *Normalization. If* $p \overset{E', k}{\underset{E}{\longmapsto}} p'$ *and* $k \neq 1$ *then* $p'$ *is* `nothing`.

**Proof.** By induction on the proof of $p \overset{E', k}{\underset{E}{\longmapsto}} p'$. □

These logical semantics do not lead to intuitive behaviors and efficient algorithms for computing reactions, thus the need for *constructive semantics* [4,19]. Intuitively, it consists in restricting the semantics to programs with deterministic *causal* (i.e non-speculative) executions. As with Esterel, such refinements are possible in Esterel$^{sp}$ but not mandatory (cf. Section 4.4).

### *3.3 Labeled Semantics*

We suppose that `pause` instructions are labeled by integers, which we note $\texttt{pause}^l$ or $l : \texttt{pause}$. We do not require labels to be unique yet. We call $\mathcal{L}(p)$ the *set of labels of* $p$, for example $\mathcal{L}(\texttt{pause}^1\texttt{;emit S;pause}^2\texttt{;pause}^1) = \{1, 2\}$.

In Figure 4, we introduce a labeled semantics $\circ\!\!\rightarrow$ for Esterel by adding a set of labels $L$ to the previous semantics ($\mapsto$) as an extra component:

$$p \overset{E', k, L}{\underset{E}{\circ\!\!\longrightarrow}} p'$$

This set collects the labels of the *active pauses* of the statement, that is to say the `pause` instructions that will retain the control at the end of the reaction. For example, "$\texttt{present S then pause}^1\texttt{ else pause}^2\texttt{ end}$" in the presence of S produces the set $\{1\}$. Provided that the labeling is non-ambiguous, $L$ carries enough information to characterize $p'$ (cf. Section 3.4).

If "`pause`" and "`exit T`" are reached concurrently then this `pause` shall not be marked as active. The reaction of "$\texttt{trap T in pause}^1\texttt{||exit T end}$" shall lead to an empty set $L$ for example. Hence, in addition to $\delta^k$, we define functions $\gamma^k$ to be used in Rule $(b)$ of the labeled semantics as follows:

$$\gamma^k(L) = \begin{cases} \emptyset & \text{if } k \neq 1 \\ L & \text{if } k = 1 \end{cases}$$

$$(1)\quad \texttt{nothing} \xrightarrow[E]{\emptyset,0,\emptyset} \texttt{nothing} \qquad \frac{S \in E \quad p \xrightarrow[E]{E',k,L} p'}{\texttt{present } S \texttt{ then } p \texttt{ else } q \texttt{ end} \xrightarrow[E]{E',k,L} p'} \quad (7)$$

$$(2)\quad \texttt{pause}^l \xrightarrow[E]{\emptyset,1,\{l\}} \texttt{nothing} \qquad \frac{S \notin E \quad q \xrightarrow[E]{F',l,L} q'}{\texttt{present } S \texttt{ then } p \texttt{ else } q \texttt{ end} \xrightarrow[E]{F',l,L} q'} \quad (8)$$

$$(3)\quad \texttt{exit } T \xrightarrow[E]{\emptyset,k_T,\emptyset} \texttt{nothing} \qquad \frac{p \xrightarrow[E]{E',k,L} p' \quad k = 0 \text{ or } k = k_T}{\texttt{trap } T \texttt{ in } p \texttt{ end} \xrightarrow[E]{E',0,L} \texttt{nothing}} \quad (9)$$

$$(4)\quad \frac{S \in E}{\texttt{emit } S \xrightarrow[E]{\{S\},0,\emptyset} \texttt{nothing}} \qquad \frac{p \xrightarrow[E]{E',k,L} p' \quad k > 0 \text{ and } k \neq k_T}{\texttt{trap } T \texttt{ in } p \texttt{ end} \xrightarrow[E]{E',k,L} \delta^k(\texttt{trap } T \texttt{ in } p' \texttt{ end})} \quad (a)$$

$$(5)\quad \frac{p \xrightarrow[E]{E',k,L} p' \quad k \neq 0}{p\texttt{; } q \xrightarrow[E]{E',k,L} \delta^k(p'\texttt{; } q)} \qquad \frac{p \xrightarrow[E]{E',k,L} p' \quad q \xrightarrow[E]{F',l,L'} q' \quad m = \max(k,l)}{\texttt{[}p \texttt{ || } q\texttt{]} \xrightarrow[E]{E' \cup F',m,\gamma^m(L \cup L')} \delta^m(\texttt{[}p' \texttt{ || } q'\texttt{]})} \quad (b)$$

$$(6)\quad \frac{p \xrightarrow[E]{E',0,L} p' \quad q \xrightarrow[E]{F',l,L'} q'}{p\texttt{; } q \xrightarrow[E]{E' \cup F',l,L \cup L'} q'} \qquad \frac{p \xrightarrow[E]{E',k,L} p' \quad k \neq 0}{\texttt{loop } p \texttt{ end} \xrightarrow[E]{E',k,L} \delta^k(p'\texttt{; loop } p \texttt{ end})} \quad (c)$$

$$(d)\quad \frac{p \xrightarrow[E \cup \{S\}]{E',k,L} p' \quad S \in E'}{\texttt{signal } S \texttt{ in } p \texttt{ end} \xrightarrow[E]{E' \setminus \{S\},k,L} \delta^k(\texttt{signal } S \texttt{ in } p' \texttt{ end})}$$

$$(e)\quad \frac{p \xrightarrow[E \setminus \{S\}]{E',k,L} p' \quad S \notin E'}{\texttt{signal } S \texttt{ in } p \texttt{ end} \xrightarrow[E]{E',k,L} \delta^k(\texttt{signal } S \texttt{ in } p' \texttt{ end})}$$

Fig. 4. Labeled Semantics

**Theorem 3.5** *Equivalence.* $\forall p, E, E', k, p' \left[ p \xrightarrow[E]{E',k} p' \Leftrightarrow \exists L : p \xrightarrow[E]{E',k,L} p' \right]$.

**Proof.** Except from the recursive computation of $L$, this new semantics does not differ from the previous one. No hypothesis relies on $L$ value. □

**Theorem 3.6** *Completion. If* $p \xrightarrow[E]{E',k,L} p'$ *then* $L \subset \mathcal{L}(p)$ *and* $k \neq 1 \Leftrightarrow L = \emptyset$. *A reaction completes (k=0) or aborts (k=*$k_T$*) an execution iff no label is set.*

**Proof.** By induction on the proof of $p \xrightarrow[E]{E',k,L} p'$. □

**Corollary 3.7** *In Rule* (6) *of the labeled semantics, L is always empty. Thus, the merging of two non-empty sets of labels can only occur in Rule* (*b*).

$$
\begin{array}{lcl}
\hat{p} ::= \widehat{\texttt{pause}}^{\,l} & \xrightarrow{\epsilon} & \texttt{nothing} \\
\quad \hat{p};\ q & \xrightarrow{\epsilon} & \epsilon(\hat{p});\ q \\
\quad p;\ \hat{q} & \xrightarrow{\epsilon} & \epsilon(\hat{q}) \\
\quad \texttt{present}\ S\ \texttt{then}\ \hat{p}\ \texttt{else}\ q\ \texttt{end} & \xrightarrow{\epsilon} & \epsilon(\hat{p}) \\
\quad \texttt{present}\ S\ \texttt{then}\ p\ \texttt{else}\ \hat{q}\ \texttt{end} & \xrightarrow{\epsilon} & \epsilon(\hat{q}) \\
\quad \texttt{trap}\ T\ \texttt{in}\ \hat{p}\ \texttt{end} & \xrightarrow{\epsilon} & \texttt{trap}\ T\ \texttt{in}\ \epsilon(\hat{p})\ \texttt{end} \\
\quad \hat{p} \mid\mid \hat{q} & \xrightarrow{\epsilon} & \epsilon(\hat{p}) \mid\mid \epsilon(\hat{q}) \\
\quad \hat{p} \mid\mid q & \xrightarrow{\epsilon} & \epsilon(\hat{p}) \mid\mid \texttt{nothing} \\
\quad p \mid\mid \hat{q} & \xrightarrow{\epsilon} & \texttt{nothing} \mid\mid \epsilon(\hat{q}) \\
\quad \texttt{loop}\ \hat{p}\ \texttt{end} & \xrightarrow{\epsilon} & \epsilon(\hat{p});\ \texttt{loop}\ p\ \texttt{end} \\
\quad \texttt{signal}\ S\ \texttt{in}\ \hat{p}\ \texttt{end} & \xrightarrow{\epsilon} & \texttt{signal}\ S\ \texttt{in}\ \epsilon(\hat{p})\ \texttt{end}
\end{array}
$$

Fig. 5. Active States and their Expansion

### *3.4 States*

By adding hats on top of some of the `pause` instructions of a statement $p$, we obtain what we call a *state* of $p$. For example, "$\texttt{pause}^1\texttt{; emit S; }\widehat{\texttt{pause}}^{\,2}$" is a state of "$\texttt{pause}^1\texttt{; emit S; pause}^2$". Intuitively, a state represents some possible point [4] in the execution of a program. Similar introductions of states and state expansions are found in [4,14].

We say that a term is *well labeled* iff the labels of its `pause` instructions are pairwise distinct. From the combination of the well-labeled statement $p$ and the set of labels $L$, we build the state $p^L$ by selecting the `pause` instructions that have a label in $L$. For example, "$(\texttt{pause}^1\texttt{; emit S; pause}^2)^{\{2,3\}}$" is the state "$\texttt{pause}^1\texttt{; emit S; }\widehat{\texttt{pause}}^{\,2}$". In the sequel, we will use either representation, as convenient. From now on, we note $p^L$ only if $p$ is well labeled. On the other hand, as in previous example, $L$ is not supposed to be a subset of $\mathcal{L}(p)$.

We remark that $p$ is both a statement and a state (of the statement $p$ itself), further referred to as the *inactive state* of $p$, as no `pause` is selected. We say that $p^L$ is a *valid state* of $p$ iff $p^L$ is either the inactive state or some *active state* $\hat{p}$ of $p$, that is to say a state that conforms to the grammar of Figure 5. In particular, an active state has at least one `pause` selected.

Intuitively, invalid states are states that cannot be reached in the execution of the program [5]. For example, "$\widehat{\texttt{pause}}^{\,1}\texttt{; }\widehat{\texttt{pause}}^{\,2}$" is not a valid state. A state is valid iff `pause` instructions are not selected in both branches of a test or both parts of a sequence, i.e. iff selected `pause` instructions are pairwise compatible.

### *3.5 State Expansion*

In Figure 5, we also define a *state expansion function* $\epsilon : p \mapsto \epsilon(p)$. It derives a

[4] We will use states to represent starting and ending points of reactions. However, micro-steps within a reaction cannot be represented by states.

[5] Valid states are not always reachable! $\texttt{[}\widehat{\texttt{pause}}^{\,1}\texttt{||pause}^2\texttt{]}$ is both valid and unreachable.

statement from an active state. Let $\epsilon(p)$ be `nothing` for inactive states. This extends $\epsilon$ to valid states.

The expansion retains labels. We observe that even if $p^L$ is a valid state (of the well-labeled term $p$), the labeled term $\epsilon(p^L)$ is not necessarily well labeled, as loop unrolling may occur. For example,

$$\epsilon(\texttt{loop pause}^1\texttt{;pause}^2\ \texttt{end}) = \texttt{nothing;pause}^2\texttt{;loop pause}^1\texttt{;pause}^2\ \texttt{end}$$

**Theorem 3.8** *Stability. If $p^L$ is valid and $\epsilon(p^L) \circ\xrightarrow[E]{E', k, L'} p'$ then $p^{L'}$ is valid.*

**Proof.** If $k \in L'$, $l \in L'$ then there exists two compatible occurrences of $\texttt{pause}^k$ and $\texttt{pause}^l$ in $\epsilon(p^L)$ by Corollary 3.7. Thus, $\texttt{pause}^k$ and $\texttt{pause}^l$ are compatible in $p$, as the expansion does not introduce parallel operators. □

**Theorem 3.9** *Expansion. If $p$ is well labeled and $p \circ\xrightarrow[E]{E', k, L} p'$ then $\epsilon(p^L) = p'$.*

**Proof.** Obvious if $k \neq 1$ otherwise by induction on the proof of $p \circ\xrightarrow[E]{E', k, L} p'$.□

This proves that $p'$ can be rebuild from $L$ (and $p$). The result of the reaction is equivalently characterized by either the residual $p'$ (as defined by the logical behavioral semantics) or the state $p^L$ we have just introduced. This is the key that enables the definition of a state semantics for Esterel in the following section.

### *3.6 State Behavioral Semantics*

We define our state behavioral semantics $\hookrightarrow$ of Esterel as follows:

$$\text{for all } p^L \text{ valid}, E, E', k, L', \text{ we note } p^L \xhookrightarrow[E]{E', k} p^{L'} \text{ iff } \exists p' : \epsilon(p^L) \circ\xrightarrow[E]{E', k, L'} p'.$$

One reaction of the well-labeled term $p$ in the valid state $p^L$ produces the valid state $p^{L'}$ (Theorem 3.8) iff $L'$ is the set of active labels computed by the labeled semantics for the term $\epsilon(p^L)$ (being well labeled or not).

**Theorem 3.10** *Equivalence. $\mapsto$ and $\hookrightarrow$ are observationally equivalent.*

**Proof.** Let $\sim$ be the relation: $p^L \sim q$ iff $\epsilon(p^L) = q$ or $\epsilon(p^L) = \texttt{nothing};q$. This relation is a 1-bisimulation between $\hookrightarrow$ and $\mapsto$. In particular,

$$p^L \xhookrightarrow[E_1]{E'_1, 1} p^{L_1} \xhookrightarrow[E_2]{E'_2, 1} \ldots \xhookrightarrow[E_n]{E'_n, k} p^{L_n} \Leftrightarrow \epsilon(p^L) \xmapsto[E_1]{E'_1, 1} \epsilon(p^{L_1}) \xmapsto[E_2]{E'_2, 1} \ldots \xmapsto[E_n]{E'_n, k} \epsilon(p^{L_n})$$

□

State semantics for Esterel have already been proposed [4,14]. Similarly, our semantics describes the execution of a program in term of "moving hats":

$$\widehat{\texttt{pause}}\texttt{;[pause||pause]} \xhookrightarrow[E]{\emptyset, 1} \texttt{pause;[}\widehat{\texttt{pause}}\texttt{||}\widehat{\texttt{pause}}\texttt{]} \xhookrightarrow[E]{\emptyset, 0} \texttt{pause;[pause||pause]}$$

But this new definition remains very close to the logical behavioral semantics. The progression of the execution "within the instant" is still handled by the logical semantics, which makes the proof of Theorem 3.10 tractable.

Our state semantics is nevertheless different from the logical semantics in the way it chains reactions. Instead of resuming from the logical residual $p'$ of the previous reaction, the execution is restarted from the (expansion of) the state $p^{L'}$ that was also computed.

Our labeled semantics is a combination of a logical behavioral semantics, which takes care of the current instant of execution, and of a state computation (the set of labels), which prepares for the next instant of execution.

While up to now these two computations ($p'$ and $L'$) coincide in the labeled semantics (Theorem 3.9), the fact extra labels may be inserted into $L'$ without modifying $p'$ makes the addition of `gotopause` to the language possible.

## 4 Introducing `gotopause` in Esterel

We extend Esterel syntax with the instruction $\texttt{gotopause}^l$ (or $\texttt{gotopause}\ l$) for any integer label $l$. We note Esterel$^{sp}$ for the extended language. We would like this new construct to behave as follows:

$$\widehat{\texttt{pause}}^1;\texttt{gotopause}^2;\texttt{emit S};\texttt{pause}^2 \xrightarrow[E]{\emptyset,1} \texttt{pause}^1;\texttt{gotopause}^2;\texttt{emit S};\widehat{\texttt{pause}}^2$$

- the hat first moves from $\texttt{pause}^1$ to $\texttt{gotopause}^2$ since the term obtained by replacing `gotopause` by `pause` instructions with fresh labels reacts as follows:
$$\widehat{\texttt{pause}}^1;\texttt{pause}^3;\texttt{emit S};\texttt{pause}^2 \xrightarrow[E]{\emptyset,1} \texttt{pause}^1;\widehat{\texttt{pause}}^3;\texttt{emit S};\texttt{pause}^2$$
- the hat then jumps from $\texttt{pause}^3 \equiv \texttt{gotopause}^2$ to $\texttt{pause}^2$, as hats on top of $\texttt{gotopause}^l$ instructions are removed and moved to the corresponding $\texttt{pause}^l$ instructions at the end of the instant.

This is more or less what we formalize below. But this intuitive algorithm breaks on more complex programs. In the following example, whereas the initial state $p^{\{1\}}$ is valid, the derived state $p^{\{2,3\}}$ is not:

$$\widehat{\texttt{pause}}^1;[\texttt{gotopause}^2||\texttt{pause}^3];\texttt{pause}^2 \xrightarrow[E]{\emptyset,1} \texttt{pause}^1;[\texttt{gotopause}^2||\widehat{\texttt{pause}}^3];\widehat{\texttt{pause}}^2$$

While the execution of the program is supposed to be continued ($k = 1$), it cannot be since $\epsilon(\texttt{pause}^1;[\texttt{gotopause}^2||\widehat{\texttt{pause}}^3];\widehat{\texttt{pause}}^2)$ is undefined. Such a state does not make sense. Thus, "$\texttt{pause}^1;[\texttt{gotopause}^2||\texttt{pause}^3];\texttt{pause}^2$" cannot be considered to be a correct program. This is dealt with through a proper definition of *well-formedness*, which ensures that `gotopause` occurrences are compatible with Esterel concurrency.

### *4.1 Well-Formedness*

**Definition 4.1** *Well-formedness.* A well-labeled program $p$ is well formed iff:

$$\forall k, \forall l : \texttt{pause}^k \# \texttt{pause}^l \Rightarrow \begin{cases} \texttt{gotopause}^k \# \texttt{gotopause}^l \\ \texttt{gotopause}^k \# \texttt{pause}^l \\ \texttt{pause}^k \# \texttt{gotopause}^l \end{cases}$$

If $\texttt{pause}^k$ and $\texttt{pause}^l$ are exclusive then their respective "triggers" have to be exclusive, too. This purely static (syntactic) condition can be checked easily while building the abstract syntax tree of a program. Of course, every well-labeled ESTEREL program is a well-formed ESTEREL$^{sp}$ program.

### *4.2 Labeled Semantics*

We build the labeled semantics of ESTEREL$^{sp}$ by adding a rule for `gotopause` to the labeled semantics of ESTEREL (Figure 4):

$$\texttt{gotopause}^l \xrightarrow[E]{\emptyset, 1, \{l\}} \texttt{nothing}$$

States and state expansion retain their definitions in ESTEREL$^{sp}$. Thus, $(\texttt{gotopause}^1;\texttt{emit S};\texttt{pause}^1;\texttt{pause}^2)^{\{1\}} = \texttt{gotopause}^1;\texttt{emit S};\widehat{\texttt{pause}}^1;\texttt{pause}^2$ and $\epsilon(\texttt{gotopause}^1;\texttt{emit S};\widehat{\texttt{pause}}^1;\texttt{pause}^2) = \texttt{nothing};\texttt{pause}^2$, for example.

**Theorem 4.2** *If $p$ is well formed and $\epsilon(p^L) \xrightarrow[E]{E', k, L'} p'$ then $p^{L'}$ is valid.*

**Proof.** Similarly to Proof 3.8, if $k \in L'$, $l \in L'$ then there exists two compatible occurrences of $(\texttt{goto})\texttt{pause}^k$ and $(\texttt{goto})\texttt{pause}^l$ in $\epsilon(p^L)$ by Corollary 3.7 (which remains valid). Thus, by definition of $\epsilon$, there exists two compatible occurrences of $(\texttt{goto})\texttt{pause}^k$ and $(\texttt{goto})\texttt{pause}^l$ in $p$. As a consequence, by definition of well-formedness, $\texttt{pause}^k$ and $\texttt{pause}^l$ are compatible in $p$. □

### *4.3 State Semantics*

We now define the state semantics of ESTEREL$^{sp}$. For all $p$ well formed,

$$\text{for all } p^L \text{ valid}, E, E', k, L', \text{ we note } p^L \xrightarrow[E]{E', k} p^{L'} \text{ iff } \exists p' : \epsilon(p^L) \xrightarrow[E]{E', k, L'} p'.$$

Thanks to Theorem 4.2, this semantics is well defined on the domain of valid states over well-formed programs. Of course, in ESTEREL$^{sp}$, Theorem 3.9 no longer holds: $\epsilon(p^{L'})$ is not $p'$. This times we are really using $L'$.

This formalizes our naive semantics: (*i*) `gotopause` behaves just as `pause` during the reaction, (*ii*) as the definition of the expansion $\epsilon$ remains unchanged, both $\texttt{gotopause}^l$ and $\texttt{pause}^l$ potentially activate $\texttt{pause}^l$ by inserting $l$ into $L'$.

We remark that, for a given label $l$, there may be several occurrences of $\texttt{gotopause}^l$ in $p$ or there may be no $\texttt{pause}^l$ even if there is one $\texttt{gotopause}^l$. This is fine. Simultaneous jumps to the same target make sense. Although "goto nowhere" should probably be forbidden in practice, it is semantically harmless.

This state semantics of ESTEREL$^{sp}$ is *adequate*: its restriction to programs of the original ESTEREL language (i.e. the state semantics of Section 3.6) is observationally equivalent to the initial semantics of ESTEREL, by Theorems 3.3 and 3.10.

### *4.4 Constructive Semantics*

For lack of space we only briefly sketch the constructive semantics of ESTEREL$^{sp}$. ESTEREL constructive semantics is essentially a way to get around the issue of the overlapping between Rules $(d)$ and $(e)$ of the logical behavioral semantics [4,19], which are both applicable to some programs. For example,

```
signal S in present S then...
...emit S else pause end end
```

$$\begin{cases} \xrightarrow[\emptyset]{\emptyset,0} \texttt{signal S in nothing end} \\ \xrightarrow[\emptyset]{\emptyset,1} \texttt{signal S in nothing end} \end{cases}$$

It can be obtained by defining two exclusive predicates *Must* and *Cannot*, used for disambiguation. The condition "$S \in E'$" of Rule $(d)$ and "$S \notin E'$" of Rule $(e)$ are respectively replaced by "p must emit S" and "p cannot emit S".

Let *Must*(`gotopause`$^l$) and *Must*(`pause`$^l$) be defined as *Must*(`pause`) is in ESTEREL. Let *Cannot*(`gotopause`$^l$) and *Cannot*(`pause`$^l$) be *Cannot*(`pause`). An identical rewriting of Rules $(d)$ and $(e)$ of our labeled semantics of ESTEREL$^{sp}$ leads to an adequate constructive semantics of ESTEREL$^{sp}$.

Intuitively, as `gotopause` and `pause` instructions only differ via their non-instantaneous effects, causality (i.e. correctness with respect to the constructive semantics) is essentially unchanged. Of course, a causal ESTEREL program is a causal ESTEREL$^{sp}$ program. Then, as the constructive and logical behavioral semantics of causal programs match, the adequacy of this constructive semantics of ESTEREL$^{sp}$ is a consequence of the adequacy of its state semantics.

## 5 Schizophrenia and Reincarnation

A well known complexity of ESTEREL semantics is illustrated by the infinite loop of Figure 6. Intuitively, as the signal S is local to the loop, a fresh instance is accessed each time the loop is entered. Thus the signal O is never emitted. Formally, loop unrolling in the logical semantics produces two separate instances of S, so that the emission of S never reaches the test. Although there is a single object S in the source program, there are, in this case, two objects S involved in each reaction (starting from the second one). Such objects and programs are said to be *schizophrenic* [4,14,17].

```
loop
  signal S in
    present S then emit O end;
    pause;
    emit S
  end
end
```

$\xrightarrow[\emptyset]{\emptyset,1}$

```
signal S in nothing; emit S end;
loop
  signal S in
    present S then emit O end;
    pause;
    emit S
  end
end
```

Fig. 6. Schizophrenia

```
loop
  signal S1 in
    present S1 then emit O end;
    pause;
    emit S1
  end;
  signal S2 in
    present S2 then emit O end;
    pause;
    emit S2
  end;
end
```

```
loop
  signal S1 in
    present S1 then emit O end;
    gotopause 1;
    emit S1
  end;
  signal S2 in
    present S2 then emit O end;
    1: pause;
    emit S2
  end;
end
```

Fig. 7. Reincarnation in Esterel and Esterel$^{sp}$

For reasons beyond the scope of this paper, compiling schizophrenic programs is hard. It is usually achieved either directly through complex and error-prone compilation algorithms (such as those described in [4]) or in a two-step process: programs being first rewritten to get rid of schizophrenic behaviors, then more easily compiled, as in [16]. The first step is called *reincarnation*. It can be obtained by a recursive replication of loop bodies:

$$\texttt{loop}\ p\ \texttt{end} \Rightarrow \texttt{loop}\ p\texttt{;}p\ \texttt{end}$$

Applied to our example, this method produces[6] the first program of Figure 7. Each reaction involves one instance of S1 and one of S2. The resulting program is not schizophrenic. This algorithm however leads to a potentially exponential growth in code size in the presence of nested loops (not illustrated here), which is not acceptable. Improved schemes have been proposed [16,17] with various drawbacks such as the need of ad hoc intermediate languages for the representation of programs, complex rewriting rules, etc.

On the contrary, using the `gotopause` instruction of Esterel$^{sp}$, a simple and efficient source to source transformation is possible. For a piece of a well-labeled Esterel program $p$, we obtain the Esterel$^{sp}$ code $\overline{p}$ by replacing the `pause` instructions of $p$ by `gotopause` instructions, labels unchanged, and the loops by their bodies. For example, "$\overline{\texttt{loop pause}^1\texttt{;pause}^2\ \texttt{end}}$" is "$\texttt{gotopause}^1\texttt{;gotopause}^2$". We now consider the recursive rewriting of loops:

$$\texttt{loop}\ p\ \texttt{end} \Rightarrow \texttt{loop}\ \overline{p}\texttt{;}p\ \texttt{end}$$

The new rewriting generates the second program of Figure 7. With this preprocessing the worst-case growth is only quadratic, because loops are removed from $\overline{p}$. Moreover, because unreachable pieces of program are frequent (in *italic* in Figure 7) and may be erased (dead code), the growth is quasi-linear in practice, which is admittedly as efficient as reincarnation can get (as well as direct compilation of schizophrenic programs).

[6] For readability, we rename the two signals S into S1 and S2.

## 6 Conclusion and Perspectives

We have designed `gotopause` a new Esterel primitive and fully formalized the extended language Esterel$^{sp}$ by providing an adequate state behavioral semantics for it. While we focused in this report on a subset of Esterel, `gotopause` can of course be embedded in the complete language. The new construct was coded into our prototype (full) Esterel compiler within a few hours.

With `gotopause`, compiling Esterel becomes easier. The core compiler can concentrate on non-schizophrenic programs, relying on the preprocessing [7] of Section 5 to get rid of schizophrenic behaviors. Compiling to Esterel$^{sp}$ is also easier. We expect compilers for graphical formalisms built over Esterel such as SyncCharts [1,2] to take advantage of the enhanced support for automata in Esterel$^{sp}$.

We believe `gotopause` is both powerful and simple. It has a very intuitive behavior and, because its action is delayed (i.e not instantly different from a regular `pause` instruction), the intuition is not misleading. In particular, it cannot contribute to instantaneous loops or causal cycles. Thus, it is not only convenient for implementation purpose but can be made available to the end user.

## References

[1] André, C., *SyncCharts: A visual representation of reactive behaviors*, RR 95-52, I3S (1995).

[2] André, C., *Representation and analysis of reactive behaviors: A synchronous approach*, in: *Computational Engineering in Systems Applications*, 1996, pp. 19–29.

[3] Arnold, A. and I. Castellani, *An algebraic characterization of observational equivalence*, Theoretical Computer Science **156** (1996), pp. 289–299.

[4] Berry, G., *The constructive semantics of pure Esterel. Draft version 3* (1999), http://www-sop.inria.fr/meije/.

[5] Berry, G., *The Esterel v5 language primer* (2000), www-sop.inria.fr/meije/.

[6] Berry, G., *The foundations of Esterel*, in: *Proof, Language and Interaction: Essays in Honour of Robin Milner* (2000).

[7] Berry, G. and G. Gonthier, *The Esterel synchronous programming language: Design, semantics, implementation*, Science of Computer Programming **19** (1992), pp. 87–152.

[8] Boussinot, F. and R. de Simone, *The Esterel language*, Another Look at Real Time Programming, Proceedings of the IEEE **79** (1991), pp. 1293–1304.

[7] Preprocessing full Esterel requires further extensions to Esterel$^{sp}$.

[9] Closse, E., M. Poize, J. Pulou, P. Vernier and D. Weil, *Saxo-rt: Interpreting Esterel semantic on a sequential execution structure*, in: *SLAP'02*, Electronic Notes in Theoretical Computer Science **65** (2002).

[10] Dijkstra, E. W., *Letters to the editor: go to statement considered harmful*, Commun. ACM **11** (1968), pp. 147–148.

[11] Edwards, S., *Compiling Esterel into sequential code*, in: *CODES'99*, 1999.

[12] Edwards, S., "Languages for Digital Embedded Systems," Kluwer, 2000.

[13] Halbwachs, N., "Synchronous Programming of Reactive Systems," Kluwer, 1993.

[14] Mignard, F., "Compilation du langage Esterel en systèmes d'équations booléennes," Ph.D. thesis, Ecole des Mines de Paris (1994).

[15] Plotkin, G., *A structural approach to operational semantics*, Report DAIMI FN-19, Aahrus University (1981).

[16] Potop-Butucaru, D., "Optimizations for Faster Execution of Esterel Programs," Ph.D. thesis, Ecole des Mines de Paris (2002).

[17] Schneider, K. and M. Wenz, *A new method for compiling schizophrenic synchronous programs*, in: *CASES'01*, 2001, pp. 49–58.

[18] Tardieu, O. and R. de Simone, *Instantaneous termination in pure Esterel*, in: *SAS'03*, Lecture Notes in Computer Science **2694**, 2003, pp. 91–108.

[19] Toma, H., "Analysis and Sequential Optimization of Circuits Generated from the Synchronous Reactive Language Esterel," Ph.D. thesis, Ecole des Mines de Paris (1997).