Goto and Concurrency Introducing Safe Jumps in Esterel

Olivier Tardieu

INRIA, Sophia Antipolis, France olivier.tardieu@sophia.inria.fr fax: +33 4 92 38 79 98

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 **setpause** 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 **setpause** 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.

1 Introduction

Esterel [4–8] 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]. As a consequence, 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 [4,9,10,15] 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, 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 [17] could become much worse because of gotos.

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.



Fig. 1. Automata in Esterel^{sp}

In this paper, we detail what we believe is the right way to extend Esterel. We add to the language an instruction we call setpause. Both setpause and pause instructions are now labeled. When the control reaches setpause^{*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 setpause^{*label*} location, it starts from the corresponding pause^{*label*} 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 [4,17] in the form of a state semantics that we prove observationally equivalent (Section 3). Then, we introduce and formalize **setpause** and the semantics of the extended language, which we note 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, 13, 16] can be achieved by a simple preprocessing in Esterel^{sp} (Section 5).

2 The Pure Esterel Kernel Language

Esterel [4–8] is an imperative synchronous programming language dedicated to reactive systems [11, 12]. Pure Esterel is the fragment of the full Esterel language where data variables and data-handling primitives are discarded. As our only concern is with control-flow primitives, we concentrate 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.

p, q ::=	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	$\begin{cases} \text{first starts } p, \text{ then } q \text{ if/when } p \text{ terminates} \\ \text{terminates if/when } q \text{ does} \end{cases}$
	$[p \mid \mid q]$	$\begin{cases} \text{starts } p \text{ in parallel with } q \\ \text{terminates when both } p \text{ and } q \text{ are done} \end{cases}$
	$\verb"loop" p" end"$	repeats p forever

Fig. 2. Pure Esterel

Figure 2 describes the grammar of *programs* in this language, as well as the intuitive behavior of its constructs. 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 Logical Behavioral to State Behavioral Semantics

In the sequel, we consider a family of semantics for Esterel. Reactions of a term p are specified in a structural operational style [14] by a labeled transition system:

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

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 encode¹ 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,
 - $k = k_T$ if exception T escapes from p, aborting the execution².

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:

$$p \xrightarrow{E_1',1} p_1 \xrightarrow{E_2',1} \dots \xrightarrow{E_n',k \neq 1} p_n \quad \text{or} \quad p \xrightarrow{E_1',1} p_1 \xrightarrow{E_2',1} \dots \xrightarrow{E_n',1} \dots$$

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 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 $p \sim q$
- $\text{ if } p \sim q \text{ and } p \xrightarrow{E',k}{E} p' \text{ then there exists } p' = r \text{ such that } p' \sim q$ $\text{ if } p \sim q \text{ and } p \xrightarrow{E',k}{E} p' \text{ then there exists } q' \text{ such that } \begin{cases} q \mid \frac{E',k}{E} q' \\ k = 1 \Rightarrow p' \sim q' \end{cases}$ $\text{ if } p \sim q \text{ and } q \mid \frac{E',k}{E} q' \text{ then there exists } p' \text{ such that } \begin{cases} p \xrightarrow{E',k}{E} p' \\ k = 1 \Rightarrow p' \sim q' \end{cases}$

Definition 2. Observational Equivalence. If there exists a 1-bisimulation between two semantics then these two semantics are said to be observationally equivalent.

If \rightarrow and \mapsto are observationally equivalent then:

$$\begin{array}{c} - \text{ if } p \xrightarrow{E_1',1} p_1 \xrightarrow{E_2',1} \dots \xrightarrow{E_n',k} p_n \text{ then } \exists q: q \mid \stackrel{E_1',1}{E_1} q_1 \mid \stackrel{E_2',1}{E_2} \dots \mid \stackrel{E_n',k}{E_n} q_n \\ - \text{ if } p \xrightarrow{E_1',1} p_1 \xrightarrow{E_2',1} \dots \xrightarrow{E_n',1} \dots \text{ then } \exists q: q \mid \stackrel{E_1',1}{E_1} q_1 \mid \stackrel{E_2',1}{E_2} \dots \mid \stackrel{E_n',1}{E_n} \dots \end{array}$$

¹ The sets I and O of input and output signals are such that $E = I \cup O$ and E' = O. p reacts to inputs I with outputs O, completion code k and residual p' iff $p \frac{O, k}{I \cup O} p'$. ² Exceptions are numbered in the usual way [4], $k_T \ge 2$ being the code of exception T.

3.2 Logical Behavioral Semantics

(6)

In Figure 3, we define a class of logical behavioral semantics for Esterel, using the auxiliary function δ as a parameter. We note \rightarrow the usual Logical Behavioral Semantics [17] obtained by setting δ equal to the identity. We note \mapsto the semantics corresponding to the alternate definition of δ :

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

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

(2) pause
$$\frac{\emptyset,1}{E}$$
 nothing $S \notin E \quad q \stackrel{F',l}{\longrightarrow} q'$
present S then p else q end $\frac{F',l}{E} q'$ (8)

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

$$(4) \underbrace{S \in E}_{\substack{f \in \mathbb{Z} \\ f \in \mathbb{Z}}} \qquad \underbrace{p \xrightarrow{E', k}_{E} p' \quad k > 0 \text{ and } k \neq k_T}_{\substack{F}} (a)$$

(1) emit
$$S \xrightarrow{\{S\},0}_{E}$$
 nothing trap T in p end $\xrightarrow{E',k}_{E} \delta^{k}$ (trap T in p' end)
 $p \xrightarrow{E',k}_{E} p' \quad k \neq 0$ $p \xrightarrow{E',k}_{E} p' \quad q \xrightarrow{F',l}_{E} q' \quad m = \max(k,l)$

(5)
$$\frac{P}{p; q} \xrightarrow{E', k}{E} \delta^{k}(p'; q) \xrightarrow{P', k} (p') = \frac{P}{p; q} \xrightarrow{E', k}{E} \delta^{k}(p'; q) \xrightarrow{P', k} (p') \xrightarrow{E', k} \delta^{m}([p' || q'])$$
(b)

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

$$\begin{array}{c} p \xrightarrow{E', k} p' \quad S \in E' \\ \hline (d) & \hline \\ \text{signal } S \text{ in } p \text{ end } \xrightarrow{E' \setminus \{S\}, k} \delta^k(\text{signal } S \text{ in } p' \text{ end}) \\ \hline \\ (e) & \hline \\ \hline \\ \text{signal } S \text{ in } p \text{ end } \xrightarrow{E', k} p' \quad S \notin E' \\ \hline \\ \hline \\ \text{signal } S \text{ in } p \text{ end } \xrightarrow{E', k} \delta^k(\text{signal } S \text{ in } p' \text{ end}) \end{array}$$

Fig. 3. Logical Behavioral Semantics

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. **Theorem 1.** Equivalence. \rightarrow and \mapsto are observationally equivalent.

Proof (1). By defining a 1-bisimulation between \rightarrow and \mapsto .

Theorem 2. Normalization. If $p \mid \frac{E', k}{E} p'$ and $k \neq 1$ then p' is nothing. For example, $\begin{cases} \text{[nothing || nothing]} & \frac{\emptyset, 0}{E} \text{ [nothing || nothing]} \\ \text{[nothing || nothing]} & \frac{|\emptyset, 0}{E} \text{ nothing} \end{cases}$

Proof (2). By induction on the proof of $p \mid \stackrel{E', k}{\xrightarrow{E}} p'$.

These logical semantics do not lead to intuitive behaviors and efficient algorithms for computing reactions, thus the need for *constructive semantics* [4, 18]. Intuitively, it consists in restricting the semantics to programs with deterministic *causal* executions, i.e. non-speculative in some sense. As with Esterel, such refinements are possible in Esterel^{sp} but not mandatory (cf. Section 4.4).

3.3 Labeled Semantics

In the sequel, we suppose that the **pause** instructions of Esterel terms are labeled by integers. We do not require labels to be unique yet. We note $\mathcal{L}(p)$ the set of labels of p, for example $\mathcal{L}(\text{pause}^1; \text{ emit } S; \text{ pause}^2; \text{ pause}^1) = \{1, 2\}.$

In Figure 4, we introduce a labeled semantics \hookrightarrow for Esterel by adding a set of labels L to the previous semantics (\mapsto) as an extra component. It 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, "present S then pause¹ else pause² end" in the presence of S produces the set {1}. In addition to function δ , we define γ to be used in Rule (b).

$$p \circ \xrightarrow{E', k, L}{E} p' \qquad \qquad \gamma^k(L) = \begin{cases} \emptyset & \text{if } k \neq 1\\ L & \text{if } k = 1 \end{cases}$$

In fact, as we will show below (cf. Section 3.4), provided that the labeling is non-ambiguous, the set L carries enough information to characterize the residual p' of the reaction.

Remark Rule (b) for parallel statements. It specifies that if "pause" and "exit T" are reached concurrently then this pause is **not** active. For example, the execution of "trap T in pause¹ || exit T end" leads to an empty set L.

Theorem 3. Equivalence.
$$\forall p, E, E', k, p' \left[p \mid \frac{E', k}{E} p' \Leftrightarrow \exists L : p \circ \frac{E', k, L}{E} p' \right].$$

Proof (3). Except from the recursive computation of L, this new semantics does not differ from the previous one. In particular, no hypothesis relies on L value.

Theorem 4. Completion. If
$$p \stackrel{E', k, L}{\longrightarrow} p'$$
 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 (4). By induction on the proof of $p \circ \frac{E', k, L}{E} p'$.

Corollary 1. 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).

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

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

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

$$(4) \quad \frac{S \in E}{\operatorname{emit} S \circ \frac{\{S\}, 0, \emptyset}{E} \operatorname{nothing}} \quad \frac{p \cdot \frac{p}{E} \cdot p - w > 0 \operatorname{dial} w > w_{1}}{\operatorname{trap} T \operatorname{in} p \operatorname{end} \circ \frac{E', k, L}{E} \delta^{k}(\operatorname{trap} T \operatorname{in} p' \operatorname{end})} \quad (a)$$

$$(5) \quad \frac{p \circ \frac{E', k, L}{E} p'}{p; q \circ \frac{E', k, L}{E} \delta^{k}(p'; q)} \quad \frac{p \circ \frac{E', k, L}{E} p'}{[p \mid\mid q] \circ \frac{E' \cup F', m, \gamma^{m}(L \cup L')}{E} \delta^{m}([p' \mid\mid q'])} \quad (b)$$

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

$$(d) \frac{p \circ \frac{E', k, L}{E \cup \{S\}} p' \quad S \in E'}{(d) \frac{E' \setminus \{S\}, k, L}{E} \circ k(p) = 0}$$

Fig. 4. Labeled Semantics

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, "pause¹; emit S; pause²" is a state of "pause¹; emit S; pause²". Intuitively, a state represents some possible

point³ in the execution of a program. Similar introductions of states and state expansions are found in [4, 13].

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, "(**pause**¹; **emit** S; **pause**²)^{2,3}" is the state "**pause**¹; **emit** S; **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 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.

$\hat{p} ::=$	pause	$\stackrel{\epsilon}{\longrightarrow}$	nothing
	$\hat{p};\;q$	$\stackrel{\epsilon}{\longrightarrow}$	$\epsilon(\hat{p}); \; q$
	$p;~\hat{q}$	$\xrightarrow{\epsilon}$	$\epsilon(\hat{q})$
	present S then \hat{p} else q end	$\xrightarrow{\epsilon}$	$\epsilon(\hat{p})$
	present S then p else \hat{q} end	$\xrightarrow{\epsilon}$	$\epsilon(\hat{q})$
	$ t trap \; T \; t in \; \hat{p} \; t end$	$\xrightarrow{\epsilon}$	$ t trap \; T \; t in \; \epsilon(\hat{p}) \; t end$
	$\hat{p} \mid\mid \hat{q}$	$\xrightarrow{\epsilon}$	$\epsilon(\hat{p}) \mid\mid \epsilon(\hat{q})$
	$\hat{p} \mid\mid q$	$\xrightarrow{\epsilon}$	$\epsilon(\hat{p}) \mid\mid$ nothing
	$p \mid\mid \hat{q}$	$\xrightarrow{\epsilon}$	nothing $ ~\epsilon(\hat{q})$
	loop \hat{p} end	$\xrightarrow{\epsilon}$	$\epsilon(\hat{p}); \; \texttt{loop} \; p \; \texttt{end}$
	${\tt signal}\;S\;{\tt in}\;\hat{p}\;{\tt end}$	$\xrightarrow{\epsilon}$	signal S in $\epsilon(\hat{p})$ end

Fig. 5. Active States and their Expansion

Intuitively, invalid states are states that cannot be reached in the execution of the program⁴. For example, " $p\widehat{ause}^1$; $p\widehat{ause}^2$ " 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 ϵ . It derives a statement from an active state. Let $\epsilon(p)$ be nothing for inactive states. This extends ϵ to valid states.

³ We will use states to represent starting and ending points of reactions.

However, micro-steps within a reaction cannot be represented by states.

⁴ Valid states are not always reachable! $[\widehat{pause}^1 || pause^2]$ is both valid and unreachable.

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,

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

Theorem 5. Stability. If p^L is valid and $\epsilon(p^L) \circ \xrightarrow{E', k, L'}{E} p'$ then $p^{L'}$ is valid.

Proof (5). If $k \in L'$, $l \in L'$ then there exists two compatible occurrences of pause^k and pause^l in $\epsilon(p^L)$ by Corollary 1. Thus, pause^k and pause^l are compatible in p, as the expansion does not introduce parallel operators.

Theorem 6. Expansion. If p is well labeled and $p \circ \xrightarrow{E', k, L} p'$ then $\epsilon(p^L) = p'$.

Proof (6). Obvious if $k \neq 1$ else by induction on the proof of $p \circ \xrightarrow{E', k, L}{E} p'$.

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:

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

One reaction of the well-labeled term p in the valid state p^L produces the valid state $p^{L'}$ (Theorem 5) 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 7. Equivalence. \mapsto and \hookrightarrow are observationally equivalent.

Proof (7). Let ~ be the relation: $p^L \sim q$ iff $\epsilon(p^L) = q$ or $\epsilon(p^L) = \text{nothing}; q$. This relation is a 1-bisimulation between \hookrightarrow and \mapsto . In particular,

$$p^{L} \xrightarrow{E'_{1},1}_{E_{1}} p^{L_{1}} \xrightarrow{E'_{2},1}_{E_{2}} \dots \xrightarrow{E'_{n},k}_{E_{n}} p^{L_{n}} \Leftrightarrow \epsilon(p^{L}) \mid \xrightarrow{E'_{1},1}_{E_{1}} \epsilon(p^{L_{1}}) \mid \xrightarrow{E'_{2},1}_{E_{2}} \dots \mid \xrightarrow{E'_{n},k}_{E_{n}} \epsilon(p^{L_{n}})$$

State semantics for Esterel have already been proposed [4, 13]. As usual, our semantics describes the execution of a program in term of "moving hats":

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

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 7 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.

In other words, 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 6), the fact extra labels may be inserted into L' without modifying p' makes the addition of **setpause** to the language possible.

4 Introducing setpause in Esterel

We now extend Esterel syntax with the instruction $setpause^{l}$ for any integer label l. We note Esterel^{sp} for the extended language. We would like this new construct to behave as follows:

$$\widehat{\mathsf{pause}^1}; \texttt{setpause}^2; \texttt{emit} \; S; \texttt{pause}^2 \; \xleftarrow{\emptyset, 1}{E} \; \texttt{pause}^1; \texttt{setpause}^2; \texttt{emit} \; S; \widehat{\mathsf{pause}^2} \;$$

- the hat first moves from pause¹ to setpause² since the term obtained by replacing setpause by pause instructions with fresh labels reacts as follows:

$$\widehat{\texttt{pause}^1}; \texttt{pause}^3; \texttt{emit}\ S; \texttt{pause}^2 \ \underbrace{\overset{\emptyset,1}{\underset{E}{\longrightarrow}}} \ \texttt{pause}^1; \widehat{\texttt{pause}^3}; \texttt{emit}\ S; \texttt{pause}^2$$

the hat then jumps from pause³ ≡ setpause² to pause², as hats on top of setpause^l instructions are removed and moved to the corresponding 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**!

$$\widehat{\mathsf{pause}^1}; [\mathtt{setpause}^2 || \mathtt{pause}^3]; \mathtt{pause}^2 \xrightarrow[E]{\emptyset,1}{E} \mathtt{pause}^1; [\mathtt{setpause}^2 || \widehat{\mathtt{pause}^3}]; \widehat{\mathtt{pause}^2}$$

While the execution of the program is supposed to be continued (k = 1), it cannot be since $\epsilon(\texttt{pause}^1; [\texttt{setpause}^2 | | \texttt{pause}^3]; \texttt{pause}^2)$ is undefined. Such a state does not make sense with respect to Esterel (state) semantics. As a consequence, "<code>pause1</code>; [<code>setpause2</code> | [<code>pause3</code>]; <code>pause2</code>" cannot be considered to be a correct program. This is dealt with through a proper definition of *well-formedness*, which ensures that <code>setpause</code> occurrences are compatible with Esterel concurrency.

4.1 Well-Formedness

Definition 3. Well-formedness. A well-labeled program p is well formed iff:

 $\forall k, \forall l : \texttt{pause}^k \# \texttt{pause}^l \Rightarrow \left\{ egin{array}{c} \texttt{setpause}^k \# \texttt{setpause}^l \\ \texttt{setpause}^k \# & \texttt{pause}^l \\ \texttt{pause}^k \# \texttt{setpause}^l \end{array}
ight.$

If $pause^k$ and $pause^l$ 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 welllabeled 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 **setpause** to the labeled semantics of Esterel (Figure 4):

 $\mathtt{setpause}^l \circ \xrightarrow{\emptyset, 1, \{l\}} \mathtt{nothing}$

States and state expansion retain their definitions in Esterel^{sp}. For example, '(setpause¹; emit S; pause¹; pause²)^{1}, is 'setpause¹; emit S; pause¹; pause², and ϵ (setpause¹; emit S; pause²; pause²) = nothing; pause².

Theorem 8. Stability. If p is well formed and $\epsilon(p^L) \circ \frac{E', k, L'}{E} p'$ then $p^{L'}$ is valid.

Proof (8). Similarly to Proof 5, if $k \in L'$, $l \in L'$ then there exists two compatible occurrences of $(\texttt{set})\texttt{pause}^k$ and $(\texttt{set})\texttt{pause}^l$ in $\epsilon(p^L)$ by Corollary 1 (which remains valid). Thus, by definition of ϵ , there exists two compatible occurrences of $(\texttt{set})\texttt{pause}^k$ and $(\texttt{set})\texttt{pause}^l$ 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, p^L valid,

$$\forall E, E', k, L', \text{ we note } p^L \xrightarrow{E', k} p^{L'} \text{ iff } \exists p' : \epsilon(p^L) \circ \xrightarrow{E', k, L'} p'$$

Thanks to Theorem 8, this semantics is well defined on the domain of valid states over well-formed programs. Of course, in Esterel^{sp}, Theorem 6 no longer holds: $\epsilon(p^{L'})$ is not p'. This times we are really using L'.

This formalizes our naive semantics: (i) setpause behaves just as pause during the reaction, (ii) as the definition of the expansion ϵ remains unchanged, both setpause^l and pause^l potentially activate pause^l by inserting l into L'.

We remark that, for a given label l, there may be several occurrences of setpause^l in p or there may be no pause^l even if there is one setpause^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 1 and 7.

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, 18], which are both applicable to some programs. For example,

signal S in present S then	$\int \xrightarrow{\emptyset,0}{\emptyset}$ signal S in nothing end
emit S else pause end end	$\xrightarrow[]{\emptyset,1]{\emptyset}}$ signal S in nothing end

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's $Must(\texttt{setpause}^l)$ and $Must(\texttt{pause}^l)$ be defined as Must(pause) is in Esterel. Let's $Cannot(\texttt{setpause}^l)$ and $Cannot(\texttt{pause}^l)$ 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 **setpause** and **pause** instructions only differ via their noninstantaneous 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).

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

Fig. 6. Schizophrenia

```
loop
                                       loop
  signal S1 in
                                         signal S1 in
                                            present S1 then emit 0 end;
    present S1 then emit 0 end;
    pause;
                                            setpause<sup>1</sup>;
    emit S1
                                            emit S1
                                                                         %dead code
  end:
                                          end;
  signal S2 in
                                         signal S2 in
    present S2 then emit 0 end;
                                            present S2 then emit O end; % dead code
                                           pause<sup>1</sup>;
    pause;
                                            emit S2
    emit S2
  end:
                                         end:
                                       end
end
```

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

Such objects and programs are said to be *schizophrenic* [4, 13, 16]. For reasons beyond the scope of this paper, compiling schizophrenic programs is hard and expensive. It is usually achieved either directly through complex and error-prone compilation algorithms (such as thoses 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 [15]. The first step is called *reincarnation*.

Reincarnation can be obtained by a recursive replication of loop bodies:

$loop \ p \ end \Rightarrow loop \ p; p \ end$

Applied to our example, this method produces⁵ 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 [15, 16] 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 **setpause** 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 note \overline{p} the Esterel^{sp} code obtained by replacing the **pause** instructions of p by **setpause** instructions, labels unchanged, and the loops by their bodies. For example, 'loop pause¹; pause² end' is 'setpause¹; setpause²'. We now consider the recursive rewriting of loops:

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

It generates the second program of Figure 7. With this preprocessing, the worst-case growth is only quadratic. 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).

 $^{^5}$ For readability, we rename the two signals S into S1 and S2.

6 Conclusion and Perspectives

We have designed **setpause** 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, **setpause** 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 **setpause**, compiling Esterel becomes easier. The core compiler can concentrate on non-schizophrenic programs, relying on the preprocessing⁶ 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 **setpause** 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. C. André. SyncCharts: A visual representation of reactive behaviors. RR 95-52, I3S, 1995.
- C. André. Representation and analysis of reactive behaviors: A synchronous approach. In Computational Engineering in Systems Applications, pages 19–29, 1996.
- A. Arnold and I. Castellani. An algebraic characterization of observational equivalence. *Theoretical Computer Science*, 156:289–299, 1996.
- G. Berry. The constructive semantics of pure Esterel. Draft version 3. http://wwwsop.inria.fr/meije/, 1999.
- 5. G. Berry. The Esterel v5 language primer. http://www-sop.inria.fr/meije/, 2000.
- 6. G. Berry. The foundations of Esterel. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
- G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.
- F. Boussinot and R. de Simone. The Esterel language. Another Look at Real Time Programming, Proceedings of the IEEE, 79:1293–1304, 1991.
- E. Closse, M. Poize, J. Pulou, P. Vernier, and D. Weil. Saxo-rt: Interpreting Esterel semantic on a sequential execution structure. *Electronic Notes in Theoretical Computer Science*, 65, 2002.
- 10. S.A. Edwards. Compiling Esterel into sequential code. In CODES'99, 1999.
- 11. S.A. Edwards. Languages for Digital Embedded Systems. Kluwer, 2000.
- 12. N. Halbwachs. Synchronous Programming of Reactive Systems. Kluwer, 1993.
- F. Mignard. Compilation du langage Esterel en systèmes d'équations booléennes. PhD thesis, Ecole des Mines de Paris, 1994.

⁶ Preprocessing full Esterel requires further extensions to $Esterel^{sp}$.

- G. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aahrus University, 1981.
- 15. D. Potop-Butucaru. *Optimizations for Faster Execution of Esterel Programs*. PhD thesis, Ecole des Mines de Paris, 2002.
- K. Schneider and M. Wenz. A new method for compiling schizophrenic synchronous programs. In CASES'01, pages 49–58, 2001.
- 17. O. Tardieu and R. de Simone. Instantaneous termination in pure Esterel. In SAS'03, volume 2694 of Lecture Notes in Computer Science, pages 91–108, 2003.
- H. Toma. Analysis and Sequential Optimization of Circuits Generated from the Synchronous Reactive Language Esterel. PhD thesis, Ecole des Mines de Paris, 1997.