Causality in the Semantics of Esterel: Revisited

MohammadReza Mousavi
Department of Computer Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

We re-examine the challenges concerning causality in the semantics of Esterel and show that they pertain to the known issues in the semantics of Structured Operational Semantics with negative premises. We show that the solutions offered for the semantics of SOS also provide answers to the semantic challenges of Esterel and that they satisfy the intuitive requirements set by the language designers.

1 Introduction

Esterel [Ber99, PBEB07] is an imperative synchronous language used for the specification and programming of embedded systems. Esterel is based on the synchronous hypothesis, i.e., instantaneous reaction to signals and immediate propagation of signals in each time-instant. The combination of the imperative programming style and the synchronous hypothesis in Esterel has led to semantic challenges addressed in the literature [BG92, Ber99, Tin00, Tin01, TdS05, PB02]. In this paper, we present the main semantic challenge posed by Esterel, namely, the issue of causality. We show that it is reminiscent of the semantic challenges [Gro93, BG96, Gla04] in Structured Operational Semantics [AFV01] (esp. in the setting with negative premises; the same challenges were encountered before in logic programming [AB94]). We then show that using the known solutions for the latter simplifies the presentation of the semantics of the former substantially and leads to the desired intuitive properties set forth by the language designers.

The rest of this paper is organized as follows. In Section 2, we present a brief overview of the Esterel language and its intuitive semantics. Section 3 introduces Structured Operational Semantics and notions of semantics and well-definedness associated with SOS specifications. Section 4 connects these two worlds by first presenting an SOS specification for Esterel and then studying the notions of semantics and well-definedness for the given specification. There, we show that certain notions of semantics for SOS formalize the intuitive criteria given by the language designers. Section 5 concludes the paper and presents directions for future research.

2 Esterel and Its Semantics: A Cook's Tour

The abstract syntax of Esterel is given by the grammar in Figure 1.

A short introduction to the intuitive semantics of each of these constructs follows. In this grammar, `0` stands for the terminated process. Emitting signal s is denoted by `emit` s, which is instantaneously visible to all parts of the system (and may in turn cause more signals to be emitted). Reacting to present

$$\begin{array}{rcl} p,q & ::= & \texttt{0} \mid \texttt{emit}\ s \mid \texttt{pres}\ s\ \texttt{?}\ p \diamond q\ \texttt{end} \mid \\ & & p\ \texttt{;}\ q \mid p\ \texttt{||}\ q \mid \texttt{sign}\ s\ \texttt{in}\ p\ \texttt{end} \mid \\ & & \texttt{1} \mid \texttt{susp}\ p\ \texttt{when}\ s \mid \texttt{trap}\ t\ \texttt{in}\ p\ \texttt{end} \mid \texttt{exit}\ t \mid \texttt{loop}\ p\ \texttt{end} \end{array}$$

Figure 1: The Abstract Syntax of Esterel

This is a preliminary version of a paper
that will appear in Electronic Proceedings
in Theoretical Computer Science.

and absent signals is done via the if-then-else construct `pres` s `?` $p \diamond q$ `end`, where if s is currently present (emitted by some other part of the system), p is executed, otherwise if s is absent q is executed. The combination of synchronous assumption, i.e., instantaneous propagation of signals, and checking for absence/presence of signals leads to semantic complications, presented shortly. Parallel composition of p and q is denoted by p `||` q. Process `sign` s `in` p `end` encapsulates s in p, i.e., declares s local to p. Another way of reacting to signals is by using the suspend construct `susp` p `when` s, which initially acts as p, but after one synchronous round will stop p as soon as signal s is emitted (suspension may happen after a number of rounds). Process `1` stands for a process that passes one unit of time and then terminates. One can define traps (exit points, exception handlers) to which a program can jump to by `trap` t `in` p `end`. The actual jump (raising the exception) is performed by executing `exit` t. A program can engage in a loop by means of `loop` p `end` (it can either keep on executing in the loop or exit the loop using `exit` t).

An Esterel program is usually suffixed by a header declaring input and output signals. The syntax of this header is of the form `input` i;`output` o; and we assume that the set of input and output variables in a program is disjoint from the set of its local signals. Moreover, to unclutter the syntax, we assume fixed sets ι, ω and λ, respectively, of input, output and local variables. We pick typical members $i, i', i_0, \ldots \in \iota$, $o, o', o_0, \ldots \in \omega$ and $s, s', s_0 \in \lambda$. This way, one does not need to consider the input and out declaration anymore since input and output (and local) variables are recognized by their names. In some cases output and local variables can be treated uniformly, in which case we denote them by $x, x', x_0 \in \omega \cup \lambda$.

To study the semantic challenge concerning causality it suffices for us to look at the first two rows of our grammar. The other constructs, e.g., traps and time passing, are semantically interesting on their own but are treated satisfactorily in the literature and are orthogonal to the causality problems addressed here. Hence, in the remainder, we focus on the subset of Esterel given in the first two lines of our grammar and only in passing mention how to include time and traps in our presented semantics.

A causality relation between events s and s' (signals in this case), means that the presence and absence of s directly influences the presence or absence of s'. For example, consider the following Esterel program:

P0 `pres i ? emit s ◇ 0 end ; pres s ? 0 ◇ emit o end`

In the above program, there is a causality chain starting from the input variable i to the local variable s and from s to the output variable o, namely the presence of i determines the presence of s and eventually leads to the absence of o, while the absence of s (caused by the absence of i), determines the presence of o. Using the syntax of Esterel one can easily write programs with cyclic dependencies (e.g., s is present if and only if s is present) or even worse, cyclic dependencies of a paradoxical nature (e.g., s is present if and only if s is absent). To illustrate these issues in Esterel, consider the following simple programs, which are all due to [Ber99]. These programs are canonical examples of different issues concerning causality in Esterel programs.

P1 `pres s ? emit s ◇ 0 end`

Program P1 relies on the presence of s in order to emit signal s. The logical semantics of Esterel *rejects* this program on the ground that it has two "models". The first one is by assuming that s is present, which leads to a justification of this assumption by emitting s. The other one is by assuming that s is absent, which is supported by that `0` does not emit (denies emitting) signal s.

In each synchronous round, the "model" of an Esterel program is defined by a *global status*, which defines the status (presence/absence) of signals in this round. A global status of a program is called *coherent* when the presence/absence of signals are determined consistently by the emit statements in the program [Ber99]:

> The global status of a program is *logically coherent* iff at least one `emit` statement is executed for each signal assumed present and no `emit` statement is executed for each signal assumed absent.

For example, program P1 has two logically coherent global statuses, namely presence of *s* and absence of *s*, as motivated above. The basis for rejecting program P1 is called "logical determinism" and is defined as follows [Ber99].

> A program is logically deterministic if it has at most one logically coherent global status.

P2 `pres` s `?` 0 ◇ `emit` s `end`

Program P2 relies on the absence of *s* in order to emit signal *s*. According to the logical semantics of Esterel, the above-given program has no logically coherent global status. Assuming that *s* is absent leads to *s* being emitted and hence, incoherency. Likewise, assuming that *s* is present requires emission of *s*, which is only justified when *s* is absent.

The basis for rejecting program P2 is called "logical reactivity" and is defined as follows [Ber99].

> A program is logically reactive if it has at least one logically coherent global status.

The conjunction of logical determinism and logical reactivity is called logical coherency and is the main well-definedness criterion for the *logical semantics* of Esterel.

P3 `pres` s `?` `emit` s ◇ `emit` s `end`

The program above has only one logically coherent global status, namely that *s* is present. This global status is also coherent since assuming the presence of *s* leads to emitting it and moreover, it is not logically coherent to assume the absence of *s*, because it leads to its emission. Hence, as far as logical coherency is concerned this program is accepted and the logical semantics defines the semantics sketched above for this program.

However, the semantics of Esterel used for its compiler, called the *constructive semantics* [Ber99, PBEB07], has further constraints which lead to the rejection of the above program. In this paper, we consider the issue of causality in both variants of the semantics and hence, also study the issue of constructiveness defined below.

> A program is *constructive*, if for each signal, it either proves its presence (must emit the signal) or proves its absence (cannot emit it).

Program P3 is rejected by the above criterion since it can neither prove the emission of *s* (its only possible proof is cyclic since relies on the assumption that *s* is emitted), nor can it coherently prove its absence, since to prove the absence of *s* it should prove that neither of the two emit statements can be executed, thus it should prove that *s* can neither be present nor absent.

P4 `pres` s_0 `?` `emit` s_0 ◇ 0 `end` ||
`pres` s_0 `?` `pres` s_1 `?` 0 ◇ `emit` s_1 `end` ◇ 0 `end`

Note that P4 is logically coherent, since its only logically coherent global status is that both s_0 and s_1 are absent. To check its constructiveness, let us focus on the emission of s_0. It definitely does not have to emit s_0, since the only reason for emitting s_0 is the `emit` s_0 statement in the left-hand side of the parallel composition, which is guarded by the check on the presence of s_0. Hence, the only proof for emitting s_0 is cyclic. But it can potentially emit s_0 (because it contains an `emit` s_0

statement) and the only way to make sure that s_0 cannot be emitted is to prove that the guard for `emit` s_0 never becomes true, i.e., we need again to show that s_0 cannot be emitted, which is also a cyclic reasoning. Hence, we conclude that P4 is not constructive because it neither must emit s_0, nor it can deny its emission.

P5 `pres` s_0 `?` `emit` s_1 $\diamond$ `0` `end ; emit` s_0

The above program is logically coherent and its unique logically coherent global status is that both s_0 and s_1 are present. However, it is again rejected by the constructive semantics of Esterel. The reason is that in order to reach the emit statement for s_0, we should first make sure that the first statement has a well-defined semantics in this context, i.e., it either takes the if branch or the else branch and then terminates. However, giving a constructive proof for the transition of the conditional requires a constructive proof for the emission of s_0. This is another instance of the cyclic proof phenomenon rejected by the constructive semantics.

3 Structured Operational Semantics

Structural Operational Semantics (SOS) was originally proposed by Plotkin [Plo04] as a syntax-directed and compositional way of defining semantics. Gradually, SOS has gained popularity and by now has become a de facto standard in defining operational semantics. This popularity has called for a richer syntax for SOS deduction rules and thus, in some applications, SOS deduction rules lost their structural, i.e., inductive, nature. Some authors then decided to use the same acronym for Structured Operational Semantics [AFV01, GV92]. With the richer syntax of SOS rules, one can write deduction rules whose meaning is not clear any more.

Example 1 *Examples of cyclic rules are the deduction rules* (**r1**) *and* (**r2**) *given below.*

$$(\mathbf{r1})\frac{p \xrightarrow{s} p}{p \xrightarrow{s} p} \qquad (\mathbf{r2})\frac{p \xrightarrow{s}\!\!\!\!\!\!\rightarrow}{p \xrightarrow{s} p}$$

The reader may already note the curious similarity between program P1 and deduction rule (**r1**) *on one hand and program P2 and deduction rule* (**r2**) *on the other hand. Moreover, program P3 resembles the combination of* (**r1**) *and* (**r2**)*. These similarities materialize as formal definitions in the remainder of this paper.*

To formalize the syntax and semantics of SOS, we first formalize the concepts of formulae and (transition) formulae.

Definition 2 (Signature and (sub)terms) *We let V represent an infinite set of variables. A* signature Σ *is a set of function symbols (operators), each with a fixed arity. An operator with arity zero is called a* constant*. We define the set $\mathbb{T}(\Sigma)$ of* terms *over Σ as the smallest set satisfying the following constraints.*

- *A variable $x \in V$ is a term.*
- *If $f \in \Sigma$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.*

We write $t_1 \equiv t_2$ if t_1 and t_2 are syntactically equal. The function $vars : \mathbb{T}(\Sigma) \to 2^V$ gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma) \subseteq \mathbb{T}(\Sigma)$ is the set of closed terms*, i.e., terms that contain no variables. A* substitution σ *is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a* closing substitution*.*

A term s is considered a subterm *of itself; if s is a subterm of t_i, then s is also a subterm of $f(t_0,\ldots,t_i,\ldots,t_{n-1})$, for each $s,t_i \in \mathbb{T}(\Sigma)$, $0 \leq i < n$, and n-ary $f \in \Sigma$. The set of subterms of a term t are denoted by subterms(t).*

Next, we formalize the syntax of SOS in terms of Transition System Specifications.

Definition 3 (Transition System Specifications (TSS)) *A* transition system specification *is a triplet (Σ, L, D) where*

- *Σ is a signature.*
- *L is a set of labels. If $l \in L$, and $t,t' \in \mathbb{T}(\Sigma)$ we say that $t \xrightarrow{l} t'$ is a* positive formula *and $t \stackrel{l}{\not\rightarrow}$ (also denoted by $\neg t \xrightarrow{l}$) and $t \stackrel{l}{\not\rightarrow} t'$ are* negative formulae. *A formula, typically denoted by ϕ, ψ, ϕ', ϕ_i, $\ldots$ is either a negative formula or a positive one.*
- *D is a set of* deduction rules, *i.e., tuples of the form (Φ,ϕ) where Φ is a set of formulae and ϕ is a positive formula. We call the formulae contained in Φ the* premises *of the rule and ϕ the* conclusion.

We write vars(r) to denote the set of variables appearing in a deduction rule $(\mathbf{r})$. We say a formula is closed *if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way.*

A deduction rule (Φ,ϕ) is typically written as $\frac{\Phi}{\phi}$. For a deduction rule r, we write $conc(r)$ to denote its conclusion and $prem(r)$ to denote its premises. A set of positive closed formulae is called a *transition relation*. Given a transition relation T, L'-labeled transitions of closed term p, denoted by $T \downarrow (p,L')$ is the subset of T containing all formulae in T that have p as their source and some $l \in L'$ as their label. A TSS is supposed to define a transition relation but for the TSSs such as those given by deduction rules $(\mathbf{r0})$ and $(\mathbf{r1})$, it is not clear what the associated transition relation is. Several proposals are given in the literature, of which [Gla04] gives a comprehensive overview and comparison. In this paper, we shall use some of these proposals to define the semantics of Esterel. In order to facilitate the presentation of these proposals, we need two auxiliary definitions, namely contradiction and contingency, which are given below.

Definition 4 (Contradiction and Consistency) *Formula $t \xrightarrow{l} t'$ is said to* contradict *both $t \stackrel{l}{\not\rightarrow}$ and $t \stackrel{l}{\not\rightarrow} t'$, and vice versa. Φ is* consistent *w.r.t. Ψ, denoted by $\Phi \models \Psi$, when for each positive formula $\psi \in \Psi$, it holds that $\psi \in \Phi$ and for each negative formula $\psi \in \Psi$, there is no $\phi \in \Phi$ such that ϕ contradicts ψ.*

In the remainder, we only use negative formulae of the form $t \stackrel{l}{\not\rightarrow}$ in our specifications. We now have all the necessary ingredients to present different proposals for the semantics of TSSs. The first proposal is the following notion of supported model, which is a slight modification of the definition in [Gla04] (restricting it to particular sets of terms and labels).

Definition 5 (Supported Model) *Given A TSS, a transition relation T is a supported model for a set $P \subseteq \mathbb{C}(\Sigma)$ of closed terms and a set $L' \subseteq L$ of labels, when*

1. *for each $q,q' \in \mathbb{T}(\Sigma)$ and $l \in L$ if $q \xrightarrow{l} q' \in T$, then there exists a deduction rule $\frac{\Phi}{\phi}$ and a substitution σ such that $\sigma(\phi) = q \xrightarrow{l} q'$ and $T \models \Phi$, and*
2. *for each $p \in P$ and $l \in L'$, $p' \in \mathbb{T}(\Sigma)$, if there exists a deduction rule $\frac{\Phi}{\phi}$ and a substitution σ such that $\sigma(\phi) = p \xrightarrow{l} p'$ and $T \models \Phi$, then $p \xrightarrow{l} p' \in T$.*

A transition relation T is a supported model for a TSS when it is a supported model for $\mathbb{C}(\Sigma)$ and L.

Note that in the above definition and throughout the rest of the paper, we only consider the "immediate transitions" of p as its semantics. One can adapt the above definitions (and the subsequent ones) to consider the "transition system" associated with p as its semantics. For the subset of Esterel considered in this paper, these two notions lead to the same conclusion concerning the well-definedness and the semantics of a program.

Semantics 1 (Unique Supported Model Semantics) *Given a set $P \in \mathbb{C}(\Sigma)$ of closed terms and a set $L' \subseteq L$ of labels a TSS is* meaningful *w.r.t. P and L' when it has a unique supported model for P and L'; the* transition system *associated with P and L' is the unique supported model for P and L'. A TSS is meaningful when it has a unique supported model; the transition relation associated with a TSS is its unique supported model.*

To illustrate these concepts, we give a few simple TSSs and study their supported models.

Example 6 *Consider the deduction rules given in Example 1.*

Consider the TSS comprising only deduction rule (**r1**)*. This TSS is not meaningful (w.r.t. $\{p\}$ and $\{s\}$) according to Semantics 1 because it has two supported models, namely $\emptyset$ and $\{p \xrightarrow{s} p\}$.*

Also according to Semantics 1, the TSS comprising only deduction rule (**r2**) *is not meaningful (w.r.t. $\{p\}$ and $\{s\}$) either, because it has no supported model. Particularly, $T = \emptyset$ is not a supported model because it follows from the right-to-left implication of Definition 5 that $p \xrightarrow{s} p \in T$. $T = \{p \xrightarrow{s} p\}$ is not a supported model either since the only deduction rule providing a reason for $p \xrightarrow{s} p \in T$ is* (**r2**) *but it does not hold that $T \vDash p0 \overset{s}{\nrightarrow}$.*

The TSS comprising both (**r1**) *and* (**r2**) *is indeed meaningful and its associated transition relation is $T = \{p \xrightarrow{s} p\}$. Transition relation T is indeed a supported model since* (**r1**) *now provides a reason for $p \xrightarrow{s} p \in T$. Moreover $T' = \emptyset$ is not a supported model for this TSS because it then follows from* (**r2**) *and the right-to-left implication of Semantics 1 that $p \xrightarrow{s} p \in T'$.*

If one takes the transition system of a program as a formalization of its global state, then the TSS comprising of deduction rule (**r1**) *is rejected because it has no coherent global state and the TSS with only* (**r2**) *is rejected because it does equivocally define a coherent global state.*

This suggests that Semantics 1 provides a suitable formalization for logical coherency. Next, we give a formalization of constructiveness in terms of supported proofs and denials.

Definition 7 (Supported Proofs) *A TSS T provides a supported proof for a formula ϕ, denoted by $T \vdash_s \phi$, when there is a well-founded upwardly branching tree with formulae as nodes and of which*

- *the root is labelled by ϕ;*
- *if a node is labelled by a positive formula ψ and the nodes above it form the set K then $\frac{K}{\psi}$ is an instance of a deduction rule in T.*
- *if a node is labelled by a negative formula ψ, and the nodes above it form the set K, then for each instance of a deduction rule $\frac{K_i}{\psi_i}$ in T such that ψ_i contradicts ψ, there exists a formula $\psi_i' \in K$ contradicting a formula in K_i.*

Semantics 2 (S-Complete Semantics) *A TSS is s-complete for a set of closed terms P when for each formula ϕ with a $p \in P$ as its source, either ϕ or a formula contradicting it has a supported proof. A TSS is s-complete when it is s-complete for the set $\mathbb{C}(\Sigma)$ of all closed terms and its transition relation is the set of positive formulae, for which it provides supported proofs.*

The following theorem is taken from [Gla04], which shows that constructiveness is indeed stronger than logical coherency.

Theorem 8 *A program (TSS) is s-complete only if it is meaningful according to Semantics 1 (has a unique supported model) and its associated transition system (unique supported model) coincides with the set of all positive formulae with a supported proof.*

Another useful property of supported proofs is their consistency [Gla04], stated below.

Theorem 9 *The notion of supported proof is consistent, i.e., for each formula ϕ with a supported proof, its negation does not have a supported proof.*

Next, we re-examine the TSS of Example 6 using our new notion of semantics.

Example 10 *The two TSSs comprising only* (**r1**) *and only* (**r2**) *are both rejected by Semantics 2, as well, since neither $p \xrightarrow{s} p$, nor $p \xrightarrow{s}\!\!\!\!\!\rightarrow$ can be proven from either of them. (This is also an immediate consequence of Theorem 8.)*

In the case of the TSS comprising only (**r1**)*, any attempt to build a supported proof for $p \xrightarrow{s} p$ has the same formula as its premise. Moreover, $p \xrightarrow{s}\!\!\!\!\!\rightarrow$ cannot be proven because its proof tree should prove a negation of a premise of* (**r1**)*, i.e., again $p \xrightarrow{s}\!\!\!\!\!\rightarrow$. In other words, both $p \xrightarrow{s} p$ and $p \xrightarrow{s}\!\!\!\!\!\rightarrow$ only have cyclic, and thus unsupported, proofs.*

Similarly, in the case of the TSS comprising only (**r2**)*, neither $p \xrightarrow{s} p$, nor $p \xrightarrow{s}\!\!\!\!\!\rightarrow$ have a supported proof.*

Consider the TSS comprising both (**r1**) *and* (**r2**)*; it does have a unique supported model $T = \{p \xrightarrow{s} p\}$ but it is* not *s-complete and is thus rejected by Semantics 2. Any proof for $p \xrightarrow{s} p$ or its negation leads to a cycle, i.e., repeating the node below in the node above, and are thus not supported.*

Again drawing an analogy with Esterel programs, Semantics 2 requires the existence of a "constructive" (supported) proof for presence/absence of signals and thus rejects a program which uses both possibilities *for a signal in order to establish its own presence.*

4 Structured Operational Semantics for Esterel

Our semantic specification of Esterel is presented in Figures 2 and 3. The state of the SOS comprises the syntax of the program currently being executed (defined by the grammar in Figure 1). The semantics is supposed to define two predicate, $p\checkmark_{I,c}$, $p \uparrow^{I,c,s}$, respectively, where the former means that p terminates with input evaluation I and under context c (if p is part of program c), and the latter means that p emits signal s (in the present time-instant) under the same assumptions. (A predicate formula can be formally interpreted as a transition formula with a dummy right-hand-side; in our case one can take `0` to be the dummy target of all predicate formulae, i.e., read $p \uparrow^{I,c,s}$ and $p\checkmark_{I,c}$ as $p \uparrow^{I,c,s}$ `0` and $p\checkmark_{I,c}$`0`, respectively.) In addition to the two predicates, the semantics is supposed to define a transition relation of the form $p \xrightarrow{I,c,s} p'$, which denotes that program p emits signal s under input evaluation I and context c. Next, we briefly describe the deduction rules in Figures 2 and 3 and then show how they formalize the intuitive properties of Esterel programs discussed before. In all labels (of predicates and transitions) of Figures 2 and 3, $I \subseteq \{i^+, i^- \mid i \in \iota\}$ such that for each $i \in \iota$, either $i^- \in I$ or $i^+ \in \iota$ (but not both), $c \in \mathbb{C}(\Sigma)$, $i \in \iota$, $x \in \omega \cup \lambda$ and $s, s', s'' \in \lambda$.

In Figure 2, (**e0**) states that `emit` s can emit signal s under any arbitrary input evaluation and context. Deduction rules (**s0**), (**s1**) and (**s2**) describe when a sequential composition emits a signal, namely, when

$$(\mathbf{e0})\frac{}{\texttt{emit}\ x\uparrow^{I,c,x}} \qquad (\mathbf{s0})\frac{p\uparrow^{I,c,x}}{p\ ;\ q\uparrow^{I,c,x}} \qquad (\mathbf{s1})\frac{p\checkmark_{I,c} \quad q\uparrow^{I,c,x}}{p\ ;\ q\uparrow^{I,c,x}} \qquad (\mathbf{s2})\frac{p\xrightarrow{I,c,x'}p' \quad p'\checkmark_{I,c} \quad q\uparrow^{I,c,x}}{p\ ;\ q\uparrow^{I,c,x}}$$

$$(\mathbf{p0})\frac{p\uparrow^{I,c,x}}{p\ \texttt{||}\ q\uparrow^{I,c,x}} \qquad (\mathbf{p1})\frac{q\uparrow^{I,c,x}}{p\ \texttt{||}\ q\uparrow^{I,c,x}} \qquad (\mathbf{f0})\frac{c\uparrow^{I,c,s} \quad p\uparrow^{I,c,x}}{\texttt{pres}\ s\ \texttt{?}\ p\diamond q\ \texttt{end}\uparrow^{I,c,x}} \qquad (\mathbf{f1})\frac{\neg c\uparrow^{I,c,s} \quad q\uparrow^{I,c,x}}{\texttt{pres}\ s\ \texttt{?}\ p\diamond q\ \texttt{end}\uparrow^{I,c,x}}$$

$$(\mathbf{f2})\frac{i^{+}\in I \quad p\uparrow^{I,c,x}}{\texttt{pres}\ i\ \texttt{?}\ p\diamond q\ \texttt{end}\uparrow^{I,c,x}} \qquad (\mathbf{f3})\frac{i^{-}\in I \quad q\uparrow^{I,c,x}}{\texttt{pres}\ i\ \texttt{?}\ p\diamond q\ \texttt{end}\uparrow^{I,c,x}}$$

$$(\mathbf{en0})\frac{p[s''/s]\uparrow^{I,c,s'}}{\texttt{sign}\ s\ \texttt{in}\ p\ \texttt{end}\uparrow^{I,c,s'[s/s'']}} \quad s'' \text{ fresh in } p \text{ and } r$$

Figure 2: Structured Operational Semantics for Esterel (Part I: Signal Emission)

either the first component of the composition emits it, or when the first component terminates (possibly after a transition) and the second component emits the signal.

The notions of termination and transition are defined in Figure 3. A parallel composition emits a signal if one of its components emits the signal, which is captured by deduction rules (**p0**) and (**p1**). An if-then-else constructs emits a signal, if either, according to deduction rule (**f0**), the local signal in its condition is emitted and the if-branch emits the signal or, according to deduction rule (**f1**), the local signal in the condition cannot be emitted and the else-branch is taken. Deduction rules (**f2**) and (**f3**) take care of the case where the condition is an input signal. In such cases, the condition is checked against the given input evaluation. A program p with a local signal s can emit a signal s', if p with a fresh signal s'' substituted for s can emit s' (but if s' is s, then p should be able to emit s'').

In Figure 3, the concept of termination is defined through the predicate $\checkmark_{I,c}$, in a straightforward manner. Exceptions are deduction rules (**if4**) and (**if5**), which rely on (the impossibility of) the emission of the condition signal for proving termination. In Figure 3, the deduction rules specifying a transition relation are almost identical to their counterparts in Figure 2. The most notable exceptions are deduction rules (**seq0**) to (**seq4**) and (**par0**) to (**par3**), which should consider all possible combinations of simultaneous transitions and individual transitions with (non-)termination in order to record the right target for the transition.

One advantage of our approach to the semantics of Esterel presented in [Ber99, Tin00, Tin01] is that we can capture both the logical semantics and constructive semantics of Esterel using the same TSS (by using two generic notions of semantics for TSS already known in the literature). Another advantage is that it establishes a clear link between, respectively, the logical and the constructive approaches to Esterel semantics, on the one hand and the model- and proof-theoretic semantics of TSSs on the other hand.

Definition 11 (Logical Semantics of Esterel) *An Esterel program p is logically coherent if the above given TSS is meaningful according to Semantics 1 for subterms(p) and (predicates and) transitions labeled $\{\checkmark_{I,p}, \uparrow^{I,p,x}, \xrightarrow{I,p,x}\}$. The semantics of p is the set of above-mentioned predicates and transitions associated with subterms(p).*

Next, we show that Definition 11 indeed satisfies the intuition behind logical coherency by re-examining the examples introduced in Section 2.

$$(\textbf{nil})\frac{}{0\checkmark_{I,c}} \qquad (\textbf{em})\frac{}{\texttt{emit}\ x \xrightarrow{I,c,x} 0}$$

$$(\textbf{seq0})\frac{p \xrightarrow{I,c,x} p' \quad p'\checkmark_{I,c} \quad q \xrightarrow{I,c,x'} q'}{p\ ;\ q \xrightarrow{I,c,x} q'} \qquad (\textbf{seq1})\frac{p \xrightarrow{I,c,x} p' \quad p'\checkmark_{I,c} \quad q \xrightarrow{I,c,x'} q'}{p\ ;\ q \xrightarrow{I,c,x'} q'}$$

$$(\textbf{seq2})\frac{p\checkmark_{I,c} \quad q \xrightarrow{I,c,x} q'}{p\ ;\ q \xrightarrow{I,c,x} q'} \quad (\textbf{seq3})\frac{p \xrightarrow{I,c,x} p' \quad p'\checkmark_{I,c} \quad q\checkmark_{I,c}}{p\ ;\ q \xrightarrow{I,c,x} p'} \quad (\textbf{seq4})\frac{p\checkmark_{I,c} \quad q\checkmark_{I,c}}{p\ ;\ q\checkmark_{I,c}}$$

$$(\textbf{par0})\frac{p \xrightarrow{I,c,x} p' \quad q \xrightarrow{I,c,x'} q'}{p\ ||\ q \xrightarrow{I,c,x} p'\ ||\ q'} \quad (\textbf{par1})\frac{p \xrightarrow{I,c,x} p' \quad q \xrightarrow{I,c,x'} q'}{p\ ||\ q \xrightarrow{I,c,x'} p'\ ||\ q'}$$

$$(\textbf{par2})\frac{p\checkmark_{I,c} \quad q \xrightarrow{I,c,x} q'}{p\ ||\ q \xrightarrow{I,c,x} q'} \quad (\textbf{par3})\frac{p \xrightarrow{I,c,x} p' \quad q\checkmark_{I,c}}{p\ ||\ q \xrightarrow{I,c,x} p'} \quad (\textbf{par4})\frac{p\checkmark_{I,c} \quad q\checkmark_{I,c}}{p\ ||\ q\checkmark_{I,c}}$$

$$(\textbf{if0})\frac{c \uparrow^{I,c,s} \quad p \xrightarrow{I,c,x} p'}{\texttt{pres}\ s\ \texttt{?}\ p \diamond q\ \texttt{end} \xrightarrow{I,c,x} p'} \qquad (\textbf{if1})\frac{\neg c \uparrow^{I,c,s} \quad q \xrightarrow{I,c,x} q'}{\texttt{pres}\ s\ \texttt{?}\ p \diamond q\ \texttt{end} \xrightarrow{I,c,x} q'}$$

$$(\textbf{if2})\frac{i^+ \in I \quad p \xrightarrow{I,c,x} p'}{\texttt{pres}\ i\ \texttt{?}\ p \diamond q\ \texttt{end} \xrightarrow{I,c,x} p'} \qquad (\textbf{if3})\frac{i^- \in I \quad q \xrightarrow{I,c,x} q'}{\texttt{pres}\ i\ \texttt{?}\ p \diamond q\ \texttt{end} \xrightarrow{I,c,x} q'}$$

$$(\textbf{if4})\frac{c \uparrow^{I,c,s} \quad p\checkmark_{I,c}}{\texttt{pres}\ s\ \texttt{?}\ p \diamond q\ \texttt{end}\checkmark_{I,c}} \qquad (\textbf{if5})\frac{\neg c \uparrow^{I,c,s} \quad q\checkmark_{I,c}}{\texttt{pres}\ s\ \texttt{?}\ p \diamond q\ \texttt{end}\checkmark_{I,c}}$$

$$(\textbf{if6})\frac{i^- \in I \quad p\checkmark_{I,c}}{\texttt{pres}\ i\ \texttt{?}\ p \diamond q\ \texttt{end}\checkmark_{I,c}} \qquad (\textbf{if7})\frac{i^+ \in I \quad q\checkmark_{I,c}}{\texttt{pres}\ i\ \texttt{?}\ p \diamond q\ \texttt{end}\checkmark_{I,c}}$$

$$(\textbf{enc0})\frac{p[s''/s] \xrightarrow{c,s'} p'}{\texttt{sign}\ s\ \texttt{in}\ p\ \texttt{end} \xrightarrow{c,s'[s/s'']} \texttt{sign}\ s\ \texttt{in}\ p'[s/s'']\ \texttt{end}} \qquad (\textbf{enc1})\frac{p[s''/s]\checkmark_c}{\texttt{sign}\ s\ \texttt{in}\ p\ \texttt{end}\checkmark_{I,c}}$$

s'' fresh in p and r

Figure 3: Structured Operational Semantics for Esterel (Part II: Transition and Termination)

Example 12 *Consider program P0, recalled below.*

P0 `pres i ? emit s ◇ 0 end ; pres s ? 0 ◇ emit o end`

It is straightforward to check that the following is the semantics of P0:
$\{P0 \uparrow^{\{i^+\},P0,s}, P0 \uparrow^{\{i^-\},P0,o}, P0 \xrightarrow{\{i^+\},P0,i} 0, P0 \xrightarrow{\{i^-\},P0,o} 0,$ `emit s` $\xrightarrow{\{i^+\},P0,s} 0,$ `emit s` $\xrightarrow{\{i^-\},P0,s} 0,$ `emit o` $\xrightarrow{\{i^+\},P0,s} 0,$ `emit o` $\xrightarrow{\{i^-\},P0,s} 0, 0\checkmark_{\{i^+\},P0}, 0\checkmark_{\{i^-\},P0}\}$. [1]
Consider program P1 quoted below.

P1 `pres s ? emit s ◇ 0 end`

It has two supported models, namely $\{P1 \uparrow^{\emptyset,P1,s}, P1 \xrightarrow{\emptyset,P1,s} 0,$ `emit s` $\uparrow^{\emptyset,P1,s},$ `emit s` $\xrightarrow{\emptyset,P1,s} 0, 0\checkmark_{\emptyset,P1}\}$ *and* $\{$`emit s` $\xrightarrow{\emptyset,P1,s} 0,$ `emit s` $\uparrow^{\emptyset,P1,s}, 0\checkmark_{\emptyset,P1}\}$. *Hence, P1 is not meaningful according to Semantics 1.*
Consider program P2 recalled below.

P2 `pres s ? 0 ◇ emit s end`

Program P2 does not have any supported model: Assume, towards a contradiction, that $P2 \uparrow^{\emptyset,P2,s}$ *is in the purported supported model of T. It then follows from item 1 in Definition 5 that there exists a deduction rule whose conclusion can match* $P2 \uparrow^{\emptyset,P2,s}$ *and whose premises are consistent with T. The only candidates are* (**f0**) *and* (**f1**)*; we analyze both cases below and show that they both lead to a contradiction.*

(**f0**) *The premises of the instance of* (**f0**) *are* $P2 \uparrow^{\emptyset,P2,s}$ *and* $0 \uparrow^{\emptyset,P2,s}$. *It follows from item 1 of Definition 5 that both predicates should be in T and hence, item 1 again applies to both predicates and in particular to* $0 \uparrow^{\emptyset,P2,s}$. *Hence, there should exist a deduction rule whose conclusions matches with the above predicate. A simple syntactic check on the deduction rules of Figures 2 and 3 reveals that none of the conclusions can be unified with the above predicate and hence a contradiction follows.*

(**f1**) *The premises of the instance of* (**f0**) *are* $\neg P2 \uparrow^{\emptyset,P2,s}$ *and* `emit s` $\uparrow^{\emptyset,P2,s}$, *both of which should be in T. Again item 1 of Definition 5 applies and thus,* $\neg P2 \uparrow^{\emptyset,P2,s}$ *should be consistent with T, or in other words,* $P2 \uparrow^{\emptyset,P2,s} \notin T$, *which contradicts our initial assumption.*

The next program to consider is P3, quoted below.

P3 `pres s ? emit s ◇ emit s end`

Program P3 is indeed meaningful and has the following unique supported model.
$\{P3 \uparrow^{\emptyset,P3,s}, P3 \xrightarrow{\emptyset,P3,s} 0,$ `emit s` $\uparrow^{\emptyset,P3,s},$ `emit s` $\xrightarrow{\emptyset,P3,s} 0, 0\checkmark_{\emptyset,P3}\}$.
Note that $P3 \uparrow^{\emptyset,P3,s}$ *(and/or the transition of P3) cannot be removed from the supported model; to see this, it follows from item 2 of Definition 5 and deduction rule* (**e0**) *that* `emit s` $\uparrow^{\emptyset,P3,s} \in T$, *and following the same reasoning and deduction rule* (**f1**), *we have that* $P3 \uparrow^{\emptyset,P3,s} \in T$.

Program P4 is considered logically coherent but not constructive by the language designers. Next, we show that this intuition is indeed supported by our formal definitions.

P4
`pres` s_0 `? emit` s_0 `◇ 0 end || pres` s_0 `? pres` s_1 `? 0 ◇ emit` s_1 `end ◇ 0 end`

[1] For each program, we choose ι, ω and λ, respectively, to comprise only the input, output and local variables mentioned in the program at hand. This allows us to focus only on the possibly relevant part of I when considering supported models.

Program P4 has a unique supported model, given below.

$\{\texttt{emit}\ s_0 \uparrow^{\emptyset,P3,s_0}, \texttt{emit}\ s_0 \xrightarrow{\emptyset,P3,s_0} 0, \texttt{emit}\ s_1 \uparrow^{\emptyset,P3,s_1}, \texttt{emit}\ s_1 \xrightarrow{\emptyset,P3,s_1} 0, 0\checkmark_{\emptyset,P3}\}$.

Note that neither emission of s_0, nor s_1 cannot be present in a supported model. First, concerning s_1, suppose that s_1 can be emitted, then it follows from item 1 of Definition 5 that there should be a deduction rule supporting this emission. This can only be due to (**p1**) *and thus, the right-hand-side component of the parallel composition. This component, in turn can only emit s_1 (due to deduction rules* (**f0**) *and then* (**f1**)*) if s_0 is present and s_1 is absent under the same context. The latter contradicts our assumption. Similarly, suppose that the supported model contains a predicate (or transition) to the effect that s_0 can be emitted. We already know that no predicate for emitting s_1 can be in the supported model. Hence, it follows from successive application of item 2 of Definition 5 using deduction rules* (**f0**), (**f1**) *and* (**e0**) *that s_1 can be emitted under the same context, which is already shown to lead to contradiction.*

P5 $\texttt{pres}\ s_0\ \texttt{?}\ \texttt{emit}\ s_1 \diamond 0\ \texttt{end}\ \texttt{;}\ \texttt{emit}\ s_0$

Program P5 is also meaningful and has a unique supported model, given below.

$\{P5 \uparrow^{\emptyset,P5,s_0}, P5 \uparrow^{\emptyset,P5,s_1}, P5 \xrightarrow{\emptyset,P5,s_0} 0, P5 \xrightarrow{\emptyset,P5,s_1} 0,$

$\texttt{pres}\ s_0\ \texttt{?}\ \texttt{emit}\ s_1 \diamond 0\ \texttt{end} \uparrow^{\emptyset,P5,s_1}, \texttt{pres}\ s_0\ \texttt{?}\ \texttt{emit}\ s_1 \diamond 0\ \texttt{end} \xrightarrow{\emptyset,P5,s_1} 0,$

$\texttt{emit}\ s_0 \uparrow^{\emptyset,P5,s_0}, \texttt{emit}\ s_0 \xrightarrow{\emptyset,P3,s_0} 0, \texttt{emit}\ s_1 \uparrow^{\emptyset,P5,s_1}, \texttt{emit}\ s_1 \xrightarrow{\emptyset,P3,s_1} 0, 0\checkmark_{\emptyset,P5}\}$.

Note that none of the predicates or transitions concerning the emission of s_0 and s_1 can be omitted from the supported model. If the predicate (transition) concerning the emission of s_0 is omitted then the first component of sequential composition terminates and hence s_0 should be emitted due to the second component. Since s_0 should always be emitted, the emission of s_1 is guaranteed by the first component of sequential composition.

Definition 13 (Constructive Semantics of Esterel) *An Esterel program p is constructive if for each signal s and each input evaluation I either $p \uparrow^{I,p,s}$ and $p \xrightarrow{I,p,s} p'$ (for some p') or $\neg p \uparrow^{I,p,s}$ and $p \overset{I,c,s}{\nrightarrow}$ has a supported proof and moreover, either $p\checkmark_{I,p}$ or $\neg p\checkmark_{I,p}$ has a supported proof.*

To illustrate this semantics and identify its differences with the logical semantics, we reconsider those programs whom are considered non-constructive but logically coherent in Section 2.

Example 14 *Consider program P3. This program is both intuitively and formally shown to be logically coherent. Moreover, in Section 2, we introduced this program as a canonical example of a non-constructive program. Next, we show that it is also formally non-constructive since neither $P3 \uparrow^{\emptyset,P3,s}$ nor $\neg P3 \uparrow^{\emptyset,P3,s}$ have a supported proof (a similar reasoning shows that neither $P3 \xrightarrow{\emptyset,P3,s} p'$ for any p' nor $P3 \overset{\emptyset,P3,s}{\nrightarrow}$ have a supported proof). Suppose $P3 \uparrow^{\emptyset,P3,s}$ has a supported proof, then its proof is either due to* (**f0**) *or* (**f1**)*. In the former case, the nodes placed above our proof obligation are $P3 \uparrow^{I,P3,s}$ and $\texttt{emit}\ s \uparrow^{\emptyset,P3,s}$. While the latter has a supported proof (due to* (**e0**)*), the former was our original proof obligation, thus, it only remains to check the alternative option due to* (**f1**)*. The premises of* (**f1**) *are then $\neg P3 \uparrow^{I,P3,s}$ and $\texttt{emit}\ s \uparrow^{\emptyset,P3,s}$. Again the latter formula has a supported proof but the former is the negation of our proof obligation and thanks to Theorem 9, we know that if $\neg P3 \uparrow^{I,P3,s}$ has a supported proof then $P3 \uparrow^{I,P3,s}$ cannot have a supported proof. Similarly, if $\neg P3 \uparrow^{I,P3,s}$ has a supported proof, then a negation of a premise of all deduction rules that can match $P3 \uparrow^{I,P3,s}$ must have a supported proof. These two rules are again* (**f0**) *and* (**f1**)*. The negation of the common premise of these two rules, i.e., $\texttt{emit}\ s \uparrow^{\emptyset,P3,s}$ cannot have a supported proof (following Theorem 9, because the premise itself has a supported proof). Hence a negation of both $P3 \uparrow^{I,P3,s}$ and $\neg P3 \uparrow^{I,P3,s}$ should have supported proofs, which is again impossible due to Theorem 9.*

Program P4 is not constructive since neither $P4 \uparrow^{\emptyset,P4,s_0}$*, nor its negation have a supported proof. The only possible proof for the emission predicate can be due to* (**p0**) *or* (**p1**)*. The case for* (**p1**) *does not lead to a supported proof since the right-hand-side does not contain any emit statement for* s_0*. If the supported proof is due to* (**p0**)*, then it should hold that* $P4 \uparrow^{\emptyset,P4,s_0}$ *which was to be proven. The negation of the predicate, i.e.,* $P4 \uparrow^{\emptyset,P4,s_0}$ *does not have a supported proof, either. Since then a negation of a premise of* (**p0**) *and* (**p1**) *should have a supported proof. The negation of the only premise of* (**p0**) *is* `pres` s_0 `? emit` $s_0 \diamond$ `0 end` $\uparrow^{\emptyset,P4,s_0}$*, which in turn means that a negation of a premise of* (**f0**) *or* (**f1**) *must have a supported proof. Consider* (**f0**)*, its two premises are* $P4 \uparrow^{\emptyset,P4,s_0}$*, but we were seeking a proof of its negation and* `emit` $s_0 \uparrow^{\emptyset,P4,s_0}$*, whose negation cannot be proven.*

Program P5 is not constructive, either. We next show that neither $P5 \uparrow^{\emptyset,P5,s_0}$ *nor its negation are provable. The purported supported proof for predicate* $P5 \uparrow^{\emptyset,P5,s_0}$ *is due to one of the rules* (**s0**) *to* (**s2**)*. Next, we analyze each case and show that it leads to a contradiction.*

(**s0**) *Then, it should hold that* `pres` s_0 `? emit` $s_1 \diamond$ `0 end` $\uparrow^{\emptyset,P5,s_0}$*. This, in turn, can be either due to* (**f0**) *or* (**f1**)*. If the predicate is due to* (**f0**)*, then we should have a supported proof for* $P5 \uparrow^{\emptyset,P5,s_0}$*, which was to be proven. If the proof is due to* (**f1**)*, then* $\neg P5 \uparrow^{\emptyset,P5,s_0}$ *should have a supported proof, which is impossible due to Theorem 9.*

(**s1**) *Then, it should hold that* `pres` s_0 `? emit` $s_1 \diamond$ `0 end`$\checkmark_{\emptyset,P5}$*. This termination can be due to either* (**if4**) *or* (**if5**)*. None of these two are possible since otherwise, respectively,* $P5 \uparrow^{\emptyset,P5,s_0}$ *or* $\neg P5 \uparrow^{\emptyset,P5,s_0}$ *should have a supported proof.*

(**s2**) *Then, it should hold that* `pres` s_0 `? emit` $s_1 \diamond$ `0 end`$\checkmark_{\emptyset,P5,s'} p'$ *for some* s' *and* p'*. This transition is due to either* (**if0**) *or* (**if1**)*. Again, both cases lead to a contradiction due to a similar reasoning as in item* (**s0**)*.*

As a side note, the common intuition and the similarities between deduction rules of Figures 2 and 3 may suggest that we can replace deduction rules of Figure 2 with the following rule (or even do without the emission predicates and make the same changes in the deduction rule for if-then-else statements in Figure 3):

$$(\textbf{emit})\frac{p \xrightarrow{I,c,x} p'}{p \uparrow^{I,c,x}}$$

This change leads to a much more restrictive semantics, which is unable to provide supported proofs for transitions of perfectly acceptable programs such as the following:

P6 `pres s ? emit o ◇ 0 end || emit s`

To see this, the reader may try to prove that P6 can emit signal *o* using deduction rule (**par0**). The proof of the premise of (**par0**) then should rely on (**if0**) and hence due to deduction rule (**emit**), we need to prove that *s* can be emitted (for the if-then-else to be able to take a transition). In turn, this can only be due to (**par1**). But to apply (**par1**), we need to know that the left-hand-side component can take a transition (in order to record its target), which is what we wanted to prove initially. This cycle is broken in our semantics, by deduction rule (**p1**) which only considers one of the two components to infer the emission of s_0 (without trying to record the target of the transition). The following proof illustrates why this program is indeed constructive.

$$\dfrac{\dfrac{\dfrac{}{\texttt{emit s} \uparrow^{\emptyset,P6,s}}}{P6 \uparrow^{\emptyset,P6,s}} \quad \dfrac{}{\texttt{emit o} \xrightarrow{\emptyset,P6,o} \texttt{0}}}{\texttt{pres s ? emit o} \diamond \texttt{0 end} \xrightarrow{\emptyset,P6,o} \texttt{0}} \quad \dfrac{}{\texttt{emit s} \xrightarrow{\emptyset,P6,s} \texttt{0}}}{P6 \xrightarrow{\emptyset,P6,o} \texttt{0 || 0}}$$

5 Conclusions and Future Work

In this paper, we presented a link between the intuitive notions of logical coherency and constructiveness in the semantics of Esterel on the one hand, and the formal notions of supported models and supported proofs in the semantics of Structured Operational Semantics, on the other hand. By means of several canonical examples from the literature, we showed that our formal definitions indeed capture the intuitive criteria put forward by the language designers.

Several formalizations of these two intuitive criteria exist in the literature. For example [Ber99, PBEB07] present three formalizations of constructive semantics of Esterel. In [Tin00, Tin01] another formalization of constructive semantics of Esterel is presented and is proven to coincide with one of the notions in [Ber99]. A rigorous comparison between all these notions and the ones presented in this paper remains as a topic for future research.

In the semantics presented in this paper, we abstracted from the issues of exceptions (traps), loops and time. We expect that one can include these aspects without any substantial change in the semantics presented in this paper using the modular semantics approach of [Mos04, MN08]. This remains as another interesting exercise for the future.

References

[AB94] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming (JLAP)*, 19/20:9–71, 1994.

[AFV01] Luca Aceto, Willem Jan (Wan) Fokkink, and Chris Verhoef. Structural operational semantics. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra, Chapter 3*, pages 197–292. Elsevier Science, Dordrecht, The Netherlands, 2001.

[Ber99] Gérard Berry. *The Constructive Semantics of Pure Esterel*. 1999. Draft version, available from: `ftp://ftp-sop.inria.fr/meije/esterel/papers/constructiveness3.ps.gz`.

[BG92] Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming (SCP)*, 19(2):87–152, 1992.

[BG96] Roland N. Bol and Jan Friso Groote. The meaning of negative premises in transition system specifications. *Journal of the ACM (JACM)*, 43(5):863–914, September 1996.

[Gla04] Robert Jan (Rob) van Glabbeek. The meaning of negative premises in transition system specifications II. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:229–258, 2004.

[Gro93] Jan Friso Groote. Transition system specifications with negative premises. *Theoretical Computer Science (TCS)*, 118(2):263–299, 1993.

[GV92] Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation (I&C)*, 100(2):202–260, October 1992.

[MN08] Peter D. Mosses and Mark J. New. Implicit propagation in structural operational semantics. In *Proceedings of the 5th Workshop on Structural Operational Semantics (SOS'08)*, pages 78–92, 2008.

[Mos04] Peter D. Mosses. Modular structural operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:195–228, 2004.

[PB02] Dumitru Potop-Butucaru. *Optimizations for Faster Simulation of Esterel Programs*. PhD thesis, École des Mines de Paris, CMA, Paris, France, 2002.

[PBEB07] Dumitru Potop-Butucaru, Stephen A. Edwards, and Gérard Berry. *Compiling Esterel*. Springer-Verlag, 2007.

[Plo04] Gordon D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60:3–15, 2004.

[TdS05] Olivier Tardieu and Robert de Simone. Loops in Esterel. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 4:708–750, 2005.

[Tin00] Simone Tini. *Structural Operational Semantics for Synchronous Languages*. PhD thesis, Dipartimento di Informatica, Università degli Studi di Pisa, Pisa, Italy, 2000.

[Tin01] Simone Tini. An axiomatic semantics for Esterel. *Theoretical Computer Science (TCS)*, 269(1-2):231–282, 2001.