A Process Language for Statecharts ⋆

Francesca Levi

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56100, Pisa, Italy

Abstract. We define a compositional labelled transition system semantics for statecharts via a translation into a new process language called $\mathcal{SP}$. The main novelty of the language is an operator of process refinement, which reflects the statecharts hierarchical structure. The translation agrees with Pnueli and Shalev semantics of statecharts. However, since the language is parametric in the set of basic actions and in some operations over actions, other semantics of statecharts can be obtained by suitably instantiating the actions corresponding to transitions and the operations over actions.

1 Introduction

Statecharts [4] are a visual synchronous formalism for the specification of reactive systems, which is based on the idea of enriching state-transition diagrams with notions of hierarchy, concurrency and broadcast communication. These features permit high level and succinct descriptions. The underlying assumption of synchronous languages [2, 11, 3] is that the output of the system is simultaneous with the input that causes it. This *synchrony hypothesis* is indeed an abstraction and relies on the idea that the environment is discrete and that the system is faster than the environment. The approach has the clear advantage to permit to reason in abstract time without reference to the physical reaction time. However, the definition of a formal semantics of statecharts meets some intrinsic difficulties due to the well-known causal paradoxes, such as events disabling their own causes. Since the definition of a formal semantics for statecharts is quite complicated, there is no widely accepted semantics and many approaches have been proposed in the last years [5, 8, 7, 16, 6, 10]. The semantics of Pnueli and Shalev [13] is a very convincing proposal for dealing with causality problems. The main drawback of this proposal is that the semantics is not compositional.

In this paper we propose a compositional labelled transition system semantics for statecharts via a translation into a process language called $\mathcal{SP}$. The main novelty of the language is an operator of process refinement for representing the statecharts hierarchical structure. Moreover, $\mathcal{SP}$ processes have locations as counterpart of statecharts boxes. The language is general, namely it is parametric in the set of basic actions and in some operations over actions and other process calculi such as SCCS [12] and CBS [14] can easily be expressed into $\mathcal{SP}$.

⋆ Research partially supported by Esprit BRA 8130 LOMAPS

We define the semantics of closed $\mathcal{SP}$ processes in the SOS style and we show a translation of statecharts into closed processes, which is correct with respect to the semantics of Pnueli and Shalev. The idea is to take a process where basic actions represent transitions and the corresponding communication and locations correspond to statechart boxes. The main problem is that the semantics of Pnueli and Shalev is not modular [7]. In other words, it is not possible to obtain the semantics of the whole statechart by considering the parts of the system as black boxes and by observing only their input-output behaviour. One way of achieving compositionality is to consider structured actions corresponding to transitions, where both the input-output and the causal ordering between input and output is represented. This is not surprising as already pointed out by many authors [16, 8, 6, 7]. However, it is imporant to stress that the translation is not ad hoc for the Pnueli and Shalev semantics, since other semantics of statecharts, such as the ones of [5, 10], can be obtained by taking different actions describing transitions and a different definition of parallel composition of actions.

The definition of a process algebra for statecharts provides a formal basis for verification, since it allows us to use various techniques proposed for process algebras such as equivalences, preorders and model checking. A process algebra for statecharts has been already presented in [17]. However, the translation of Uselton and Smolka does not give a labelled transition system semantics equivalent to the one of [13].

The paper is organized as follows. In section 2 statecharts are introduced. Section 3 presents the language $\mathcal{SP}$ and its operational semantics. Sections 4 shows how statecharts can be embedded into $\mathcal{SP}$. Finally section 5 discuss the relation of our semantics with the one of [17].

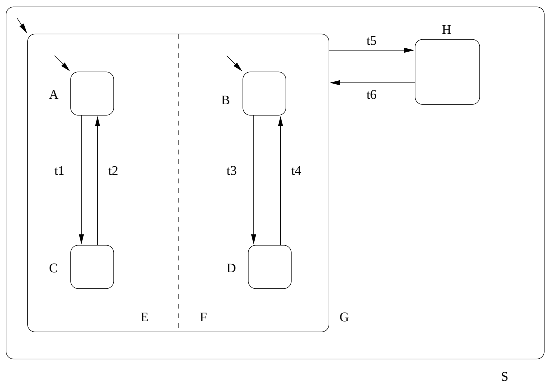

Fig. 1. A statechart

2 Introduction to Statecharts

In this section we introduce statecharts and briefly discuss the main features of the semantics proposed in [13].

A statechart specifies a reactive system, which continuously interacts with its environment by receiving and sending signals of a finite set Π of primitive events. We assume $nil \in \Pi$, where nil denotes the null event, and we denote by $\bar{\Pi} = \{\bar{e} \mid e \in \Pi\}$, where $\bar{e}$ is the negation of event e. For $\bar{e} \in \bar{\Pi}$, we denote by $\bar{\bar{e}} = e$.

In the following, we use the notation $i \in \{1, k\}$ for $i \in \{1, \ldots, k\}$.

Definition 1. A statechart over Π is defined as follows:

- a basic statechart $S : (\{S_1, \ldots, S_k\}, T, in, out, \chi, \delta)$ over Π, where
  - $\{S_1, \ldots, S_k\}$ is a set of boxes, such as $S_i \neq S$, for each $i \in \{1, k\}$. We define $boxes(S) = \{S\} \cup \{S_1, \ldots, S_k\}$.
  - T is a finite not empty set of transitions.
  - $out : T \to \{S_1, \ldots, S_k\}$, $in : T \to \{S_1, \ldots, S_k\}$ give the source and target states of transitions, respectively.
  - $\chi : T \to 2^{\Pi \cup \bar{\Pi}} \times 2^{\Pi}$ is the labelling function of transitions. We denote by $trigger(t)$ and $action(t)$ the first and second component of χ, respectively and we require $trigger(t), action(t) \neq \emptyset$ and $e, \bar{e} \notin trigger(t) \cup action(t)$.
  - $\delta : S \to \{S_1, \ldots S_k\}$ is the default function giving the default substate.

  If $k = 0$ then S is an *empty* statechart and $boxes(S) = \{S\}$.

- A refined statechart $\bigtriangledown(S, \{S_1, \ldots, S_k\})$ over Π, where
  - S is a basic non-empty statechart, where $boxes(S) = \{S\} \cup \{S_1, \ldots, S_k\}$.
  - $\{S_1, \ldots, S_k\}$ is a non-empty finite set of statecharts S_i over Π such that $\{S\} \cap boxes(S_i) = \emptyset$ and $boxes(S_i) \cap boxes(S_j) = \emptyset$, for all $i, j \in \{1, k\}$, such that $i \neq j$. Moreover, $boxes(\bigtriangledown(S, \{S_1, \ldots, S_k\})) = \{S\} \cup \bigcup_{i \in \{1,k\}} boxes(S_i)$.

- An AND statechart $S : \mathrm{AND}(S_1, S_2)$ over Π iff
  - S_1, S_2 are a non-empty statecharts over Π, and $boxes(S) = \{S\} \cup \bigcup_{i \in \{1,2\}} boxes(S_i)$, and $\{S\} \cap boxes(S_i) = \emptyset$ and $boxes(S_i) \cap boxes(S_j) = \emptyset$, for all $i, j \in \{1, 2\}$, such that $i \neq j$.

Consider the statechart of figure 1, where transitions are labelled as follows

$$t_1 : a/b, t_2 : a/b, t_3 : b/nil, t_4 : b/nil$$
$$t_5 : d/nil, t_6 : d/nil.$$

Statecharts E and F are basic statecharts, while $G : \mathrm{AND}(E, F)$. Statechart of figure 1 is a refined statechart $\bigtriangledown(S, \{G, H\})$, where S is a basic statechart with boxes G and H. As shown by the picture, state G is refined by the parallel composition of E and F, while state H is refined by the empty statechart. Transitions are depicted by arrows and are labelled by a pair *trigger/action*. Intuitively, the trigger gives conditions on the input for the transition to be permorfed, while the action gives the set of generated signals. For instance t_1 requires event a from the environment and produces event b. Note that the trigger may contain negated events, where $\bar{a}$ is interpreted as "a is absent". For a basic statechart a state is distinguished as the default state by an entering arrow without source, such as state A in statechart E.

A configuration (global state) of a statechart is given by a maximal and consistent subset of boxes, where two states are consistent iff they are in two parallel components. Given $S_1, S_2 \in boxes(S)$, we denote by $LCA(S_1, S_2)$ the statechart S', such that $S' \in boxes(S)$ and $S_1, S_2 \in boxes(S')$ and, for all $S'' \in boxes(S)$, such that $S_1, S_2 \in boxes(S'')$, $S' \in boxes(S'')$.

Definition 2. The set $C(S)$ of configurations of a statechart S is given by the subsets $c \in 2^{boxes(S)}$ such that, for each $S_1, S_2 \in c$, $LCA(S_1, S_2)$ is an AND statechart and c is maximal.

A configuration $c \in C(S)$ is called a *default* configuration iff, for each $S' \in boxes(S)$, such that either S' is a basic statechart or a refined statechart, $\delta(S') \in c$. We denote by $\Delta(S)$ the default configuration of S, namely the initial configuration.

We briefly discuss the semantics of [13]. The semantics relies on the assumption that both the system and the environment can be described as discrete processes and that the reaction of the system is instantaneous. The environment prompts the system by means of a sequence of primitive events sets and the statechart reacts to each set of inputs by performing a *step*, that is a maximal set of parallel transitions taking the statechart to a new configuration and producing a set of outputs. A step satisfies some basic requirements, which can be summarized as the *synchrony hypothesis* and the principles of *causality* and *global consistency*.

By the synchrony hypothesis all the events generated by transitions taken in a step are simultaneous with inputs. Consider the statechart of figure 1 and the configuration $\{S, G, E, F, A, B\}$. If the environment communicates event a, transitions t_1 and t_3 are taken in the same step, since t_1 generates event b, which enables t_3. The reached configuration is $\{S, G, E, F, C, D\}$. On the other hand, if the environment produces d, then $\{t_5\}$ is a step and the reached configuration is $\{S, H\}$. Note that, if the environment produces both a and d, then both $\{t_1, t_3\}$ and $\{t_5\}$ are steps and there is a non deterministic choice.

The principle of causality requires the existence of a causal ordering between transitions taken in a step. Consider the statechart G of figure 1, where transitions t_1 and t_3 are labelled by a/b and b/a, respectively. If the current configuration is $\{G, E, F, A, B\}$ and if the environment communicates neither a nor b, the reaction of the system is the empty step, since each transition depends on the other.

Global consistency requires that the causes of a transition are never disallowed. Consider statechart G, where transitions t_1 and t_3, are labelled by $\bar{a}/b$ and b/a, respectively. If the current configuration is $\{G, E, F, A, B\}$ and if the environment produces neither a nor b, then no step is possible. Actually, t_1 is enabled with respect to the input and produces b which enables t_3. However, t_3 produces a which disables t_1. This is a so called causal paradox. Note that in this case the reaction of the system is not defined.

All these requirements are satisfied if a step is defined as a subset of transitions that is the fixpoint of a relation $En_{c,I}$. The idea is that $t \in En_{c,I}(T)$ iff t is enabled in configuration c with respect to input I assuming that transitions in T are taken.

We denote by $T(S)$ the set of transitions of S, namely the union of sets T, such that $S' : (\{S_1, \ldots, S_k\}, T, in, out, \chi, \delta) \in boxes(S)$.

Definition 3. Let S be a statechart and $I \subseteq \Pi$.

- For $t_1, t_2 \in T(S)$, t_1 is *consistent* with t_2 iff $t_1 \in T(S_1), t_2 \in T(S_2)$, for $S_1, S_2 \in boxes(S)$, and $LCA(S_1, S_2)$ is an AND statechart.
- For $T \subseteq T(S)$, we define $con(T) = \{t \in T(S) \mid t$ is consistent with t' for all $t' \in T\}$.
- For $c \in C(S)$, we define $rel(c) = \{t \in T(S) \mid out(t) \in c\}$.
- We define $trig(I) = \{t \in T(S) \mid \forall e \in trigger(t), e \in I$ and $\forall \bar{e} \in trigger(t), e \notin I\}$.
- We define $En_{c,I}(T) = rel(c) \cap con(T) \cap trig(I \cup gen(T))$, where $gen(T) = \bigcup_{t \in T} action(t)$.

Definition 4. Given a configuration $c \in C(S)$ and a set of input signals $I \subseteq \Pi$, the set of transitions $T \subseteq T(S)$ is an *admissible step* with respect to c and I iff $En_{c,I}(T) = T$ and if T is *inseparable*, i.e. for each $T' \subset T$, $En_{c,I}(T') \cap (T - T') \neq \emptyset$.

The configuration $c' \in C(S)$ obtained from c via T is given by

$$c' = c - (\bigcup_{t \in T} boxes(out(t))) \cup (\bigcup_{t \in T} \Delta(in(t))).$$

Note that, when a step is performed, the reached configuration is the one containing the default configurations of the states that are the target of transitions in the step. The condition of inseparability guarantees the existence of a causal relation between transitions.

We use the notation $c \stackrel{I,O}{\Longrightarrow} c'$ to denote that there is a step from configuration c to configuration c' on input I, which produces output O.

It is important to stress that, in this approach, empty steps are admissible, while there are cases where the reaction is not defined, because of a causal paradox. These cases can be seen as run-time errors.

Different interpretations to causal paradoxes are given in the other semantics. In the original semantics of [5] a step is defined as a sequence of microsteps causally but not temporally related. The main difference is that a step is not required to be globally consistent. Consider statechart G, where transitions t_1 and t_3, are labelled by $\bar{a}/b$ and b/a, respectively. If the current configuration is $\{G, E, F, A, B\}$ and if the input is empty, then the step $\{t_1, t_3\}$ is admissible, since it is obtained by the sequence of microsteps $\{t_1\}, \{t_3\}$. Actually, t_1

is enabled with respect to the input and enables t_2. On the other hand, in the semantics of [10] a step is required to be globally consistent, but a different interpretation to negative conditions is given. For instance, in the example before $\{t_1\}$ is the only admissible step. The idea is that, even if t_3 is enabled, it is not performed since t_3 disables t_1. We believe that the semantics of Pnueli and Shalev gives the more natural interpretation to negative events so that they can be adequately used for expressing priorities and interrupts.

It is important to stress that a common feature of all the semantics for statecharts is the principle of causality. As pointed out in [7] this has the drawback that they cannot be modular.

3 The language $\mathcal{SP}$

In this section we present the process language $\mathcal{SP}$. We assume a set Act of basic actions, a set Loc of locations and a set Var of variables. Since a reactive system interacts continuously with the environment, basic actions tipically describe also the behaviour of the environment. On Act we assume an operation $\times : Act \times Act \to 2^{Act}$ of parallel composition and an operation $\# : Act \cup \{*\} \to 2^{Act}$. The idea is that actions $\#(a)$ are in conflict with action a, i.e. the environment cannot enable both a and a', if $a' \in \#(a)$. The action $\{*\}$ is used to give the semantics of the empty process. Moreover, we assume $\times$ to be associative, namely, for each $a_1, a_2, a_3 \in Act$, $a_1 \times (a_2 \times a_3) = (a_1 \times a_2) \times a_3$.

Definition 5. We define processes $\mathcal{SP}$ as follows:

$$\begin{array}{ll} \mathcal{B} ::= & nil \mid a.t \mid b_1 + b_2 \\ \mathcal{T} ::= & \ell :: b \mid rec\, x.t \mid t \bigtriangledown p \mid x \\ \mathcal{SP} ::= & \ell :: t \mid \ell :: p_1 \times p_2 \end{array}$$

where $a \in Act$, $x \in Var$, $\ell \in Loc$.

Process $\ell :: t \in \mathcal{SP}$ denotes the process t at location ℓ and process $\ell :: p_1 \times p_2$ denotes the parallel composition of processes p_1 and p_2. As usual nil denotes the empty process, $b_1 + b_2$ denotes the choice between b_1 and b_2, while $a.t$ denotes process t prefixed by action a. Process $rec\, x.t$ is the classical recursive process and process $t \bigtriangledown p$ denotes process t refined by p. The choice for locations is motivated by the corresponding definition of boxes in statecharts.

As usual we assume $variables(p) = \{x \in Var \mid x \text{ is a subterm of } p\}$ and $free(p) = \{x \in variables(p) \mid x \text{ does not occur inside the scope of } rec\, x\}$.

Assume $p \in \mathcal{SP}$, we define the set of *current locations*, $loc(p)$ as

- $loc(\ell :: t) = \{\ell\} \cup loc(t)$;
- $loc(\ell :: p_1 \times p_2) = \{\ell\} \cup loc(p_1) \cup loc(p_2)$;
- $loc(\ell :: b) = \{\ell\}$;
- $loc(rec\, x.t) = loc(t)$;
- $loc(t \bigtriangledown p) = loc(t) \cup loc(p)$.

Intuitively, for a process p describing a statechart, $loc(p)$ will be the configuration represented by the process.

The operational semantics of the process language is given as a labelled transition system $(\mathcal{CSP}, \mathcal{S}(Act), \mapsto)$, where $\mathcal{CSP}$ is the subset of closed $\mathcal{SP}$ and $\mathcal{S}(Act)$ is the set of *semantic* actions, defined as follows.

Definition 6. Let Act be a set of basic actions. We define $\mathcal{S}(Act)$ as the set of *semantic* actions such that, for each $a \in Act$, $\tau \in \{\epsilon, \bar{\epsilon}\}$, $a : \tau \in \mathcal{S}(Act)$.

Types ϵ and $\bar{\epsilon}$ denote *non-idle* and *idle* actions, respectively. Idle actions are used to represent empty steps of statecharts, while non-idle are used to represent non-empty steps. This distinction is essential for ensuring the maximality of steps. The idea is that idle actions are taken iff no non-idle action is enabled by the environment.

We extend the operation $\times$ to semantic actions $\mathcal{S}(Act)$ as follows.

Definition 7. Assume $\alpha_i = a_i : \tau_i \in \mathcal{S}(Act)$, for $i \in \{1, 2\}$. We define $\alpha_1 \times \alpha_2$ as the set $\{a : \tau \in \mathcal{S}(Act)\}$, such that

- $a \in a_1 \times a_2$;
- $\tau = \epsilon$ if $\tau_i = \epsilon$, for some $i \in \{1, 2\}$, and $\tau = \bar{\epsilon}$ otherwise.

For $\alpha = a : \tau \in \mathcal{S}(Act)$, we denote by $type(\alpha) = \tau$.

Assume $p \in \mathcal{CSP}$. The transition relation is obtained by the following rules:

$$\frac{t \overset{\alpha}{\mapsto} t'}{\ell :: t \overset{\alpha}{\mapsto} \ell :: t'}$$

$$\frac{\{p_i \overset{\alpha_i}{\mapsto} p_i'\}_{i \in \{1,2\}} \quad \alpha \in \alpha_1 \times \alpha_2}{\ell :: p_1 \times p_2 \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'} \; par$$

$$\ell :: a.t \overset{a:\epsilon}{\mapsto} t \qquad\qquad \ell :: a.t \overset{b:\bar{\epsilon}}{\mapsto} \ell :: a.t \quad b \in \#(a)$$

$$\ell :: nil \overset{b:\bar{\epsilon}}{\mapsto} \ell :: nil \quad b \in \#(*)$$

$$\frac{\begin{array}{c}\ell :: b_i \overset{\alpha}{\mapsto} t \\ type(\alpha) = \epsilon\end{array}}{\ell :: b_1 + b_2 \overset{\alpha}{\mapsto} t} \; choice_1 \qquad \frac{\begin{array}{c}\ell :: b_1 \overset{\alpha}{\mapsto} \ell :: b_1 \\ \ell :: b_2 \overset{\alpha}{\mapsto} \ell :: b_2 \\ type(\alpha) = \bar{\epsilon}\end{array}}{\ell :: b_1 + b_2 \overset{\alpha}{\mapsto} \ell :: b_1 + b_2} \; choice_2$$

$$\frac{t \overset{\alpha}{\mapsto} t' \; type(\alpha) = \epsilon}{t \bigtriangledown p \overset{\alpha}{\mapsto} t'} \; \bigtriangledown_1$$

$$\frac{p \overset{\alpha}{\mapsto} p' \; type(\alpha) = \epsilon}{t \triangledown p \overset{\alpha}{\mapsto} t \triangledown p'} \triangledown_2 \qquad \frac{\begin{array}{l} t \overset{\alpha}{\mapsto} t \\ p \overset{\alpha}{\mapsto} p \\ type(\alpha) = \bar{\epsilon} \end{array}}{t \triangledown p \overset{\alpha}{\mapsto} t \triangledown p} \triangledown_3$$

$$\frac{t[rec\ x.t/x] \overset{\alpha}{\mapsto} t' \; type(\alpha) = \epsilon}{rec\ x.t \overset{\alpha}{\mapsto} t'} rec_1$$

$$\frac{t[rec\ x.t/x] \overset{\alpha}{\mapsto} t[rec\ x.t/x] \; type(\alpha) = \bar{\epsilon}}{rec\ x.t \overset{\alpha}{\mapsto} rec\ x.t} rec_2$$

Let us explain the rules. The rule of parallel composition is the standard rule for synchronous languages and is based on the operation of parallel composition of actions. Note that an action of the parallel composition is idle iff it results from idle actions of both the components. Consider a process $\ell :: a.t \in \mathcal{T}$. If the environment enables action a then $\ell :: a.t$ evolves into t by performing an action $a : \epsilon$ (non-idle) and location ℓ is left. On the other hand, if an action a' in conflict with a is enabled by the environment then the reached process is $\ell :: a.t$ and the action is a' of type $\bar{\epsilon}$. Actions of type ϵ and type $\bar{\epsilon}$ give a different behaviour to the operators of refinement and of choice. Consider $\ell :: b_1 + b_2 \in \mathcal{T}$. Rule $choice_1$ correspond to the classical rule for choice and is applicable iff the action is non-idle. On the other hand, b_1 and b_2 must agree on idle actions, as required by rule $choice_2$. In a similar way the semantics of $t \triangledown p$ is defined. Actions $\bar{\epsilon}$ of $t \triangledown p$ must be actions $\bar{\epsilon}$ of both t and p. Note that when a non-idle action of p is performed, the process t is not left. This is the basic idea of process refinement. Rule rec_1 is standard, while rule rec_2 reflects the idea that, when an idle action is performed, the process does not change.

4 The translation

In this section we define a compositional mapping α from statecharts into closed processes, such that the labelled transition system semantics of $\alpha(S)$ is equivalent to Pnueli and Shalev semantics of S.

Given a statechart S over Π, we construct a process with basic actions Act corresponding to transitions and describing their input-output and locations Loc corresponding to the boxes of S. However, since the input-output is not sufficient for compositionality, also the causal ordering between events is recorded in basic actions.

Hence, we consider actions with the following syntax.

Definition 8. Let Π a finite set of primitive events. We define $\Sigma(\Pi)$ as the set of pairs $(\ell, \prec)$ where

1. $\ell \in 2^{\Pi \cup \bar{\Pi}}$ and ℓ is *consistent*, i.e. $e, \bar{e} \notin \ell$;
2. $\prec$ is an ordering over $2^\ell \times 2^\ell$ such that:
   (a) $\prec$ is an *irreflexive* ordering relation;

(b) for each $C, C' \in 2^{\ell}$ such that $C' \subseteq C$, if $C'' \prec C'$ then $C[C''/C'] \prec C$.

The ordering reflects causality, so that $\{e, e''\} \prec \{e'\}$ means that both e and e'' are necessary for event e' to be produced. On the other hand, $\{e\} \prec \{e'\}$ and $\{e''\} \prec \{e'\}$ means that both e and e'' are sufficient for event e' to be produced. Moreover, we require labels to be *consistent* for global consistency.

For $\sigma = (\ell, \prec) \in \Sigma(\Pi)$, we define $trigger(\sigma) = \{e \in \Pi \cup \bar{\Pi} \mid \nexists C \in 2^{\Pi \cup \bar{\Pi}}$ such that $C \prec \{e\}\}$ and $action(\sigma) = \ell \setminus trigger((\ell, \leq))$. The trigger of σ is the set of events required either to be present or absent in the input for enabling the step, while its action is the set of events produced by the step.

The operation $\times$ of parallel composition over actions $\Sigma(\Pi)$ is quite complex due to the communication between parallel components. Given two orderings $\prec_1, \prec_2$, we denote by $(\prec_1 \cup \prec_2)^+$, the ordering $\prec$, such that $\prec_1 \cup \prec_2 \subseteq \prec$ and such that, if $C \prec C'$ and $C' \prec C''$, then $C \prec C''$ and such that, for each C, C', if $C' \subseteq C$ and $C'' \prec C'$, then $C[C''/C'] \prec C$.

Definition 9. Given $\sigma_1 = (\ell_1, \prec_1) \in \Sigma(\Pi)$ and $\sigma_2 = (\ell_2, \prec_2) \in \Sigma(\Pi)$, we define $\sigma_1 \times \sigma_2 = \{\sigma = (\ell, \prec)\}$ such that

- $\ell = \ell_1 \cup \ell_2$ iff $\ell_1 \cup \ell_2$ is consistent;
- $\prec \subseteq (\prec_1 \cup \prec_2)^+$ such that:
  - $(\ell, \prec) \in \Sigma(\Pi)$;
  - $\prec$ is maximal.

The idea is that the parallel composition of actions $(\ell_1, \prec_1)$ and $(\ell_2, \prec_2)$ is given by the union of events $\ell_1 \cup \ell_2$ and by the ordering, that is the maximal subset of $(\prec_1 \cup \prec_2)^+$, which is irreflexive.

Consider $\sigma_1 = (\{a, b\}, \{a\} \prec_1 \{b\})$ and $\sigma_2 = (\{b, c\}, \{b\} \prec_2 \{c\})$. We have $\sigma_1 \times \sigma_2 = \{\sigma\}$ with $\sigma = (\{a, b, c\}, \{a\} \prec \{b\}, \{b\} \prec \{c\}, \{a\} \prec \{c\})$. The communication is realized by the transitive closure of the two orderings. Actually, $trigger(\sigma) = \{a\}$ and $action(\sigma) = \{b, c\}$, while $trigger(\sigma_1) = \{a\}$, $trigger(\sigma_2) = \{b\}$, so that b is not required from the environment.

Note that $(\prec_1 \cup \prec_2)^+$ may be reflexive. Consider $\ell_1 = (\{a, b\}, \{a\} \prec_1 \{b\})$ and $\ell_2 = (\{a, b\}, \{b\} \prec_2 \{a\})$. The parallel composition of σ_1 and σ_2 cannot be $\sigma = (\{a, b\}, \{a\} \prec \{b\}, \{b\} \prec \{a\})$ since $trigger(\sigma) = \emptyset$ and this contradicts the causality principle. We have $\sigma_1 \times \sigma_2 = \{\sigma_1, \sigma_2\}$ so that either event a or event b is needed from the environment.

Definition 10. For $\sigma \in \Sigma(\Pi)$, we define $\#(\sigma) = \{(E, \emptyset) \mid E \subseteq \Pi \cup \bar{\Pi}$, such that E is consistent and $\forall e \in \Pi$, either $e \in E$ or $\bar{e} \in E$ and there exists $e \in E$, such that $\bar{e} \in trigger(\sigma)\}$. Moreover, we define $\#(*) = \{(E, \emptyset) \mid E \subseteq \Pi \cup \bar{\Pi}$, such that E is consistent and $\forall e \in \Pi$, either $e \in E$ or $\bar{e} \in E\}$.

Intuitively, $\sigma' \in \#(\sigma)$ iff for each input $I \subseteq \Pi$, if I enables σ', then I does not enable σ. Consider an action $\sigma = (\{a, b, c\}, \{a\} \prec \{c\})$. We have $\#(\sigma) = (\{\bar{a}, b, c\}, \emptyset), (\{\bar{a}, b, \bar{c}\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{c}\}, \emptyset), (\{\bar{b}, a, c\}, \emptyset), (\{\bar{b}, \bar{a}, c\}, \emptyset), (\{\bar{b}, a, \bar{c}\}, \emptyset)\}$. Actually, if the environment does not produce both a and b the transition corresponding to σ cannot be performed, since $trigger(\sigma) = \{a, b\}$.

The translation $\alpha(S)$ is inductively defined over the structure of statechart S. We need the following definition for refinement.

Definition 11. Let $t \in \mathcal{T}$, $\{p_i\}_{i\in\{1,n\}} \in \mathcal{CSP}$ be closed processes and let $l_i \in Loc$. We define $ref(t, \{(l_1, p_1), \ldots, (l_n, p_n)\})$ as the term obtained by substituting in t each occurrence of a subterm $l_i :: t'$ with $l_i :: t' \bigtriangledown p_i$.

Moreover, for a transition $t \in T(S)$, we define $lab(t) = (trigger(t) \cup action(t), \{trigger(t) \prec \{e'\} \mid e' \in action(t) - \{nil \cup trigger(t)\}\})$.

Definition 12. The mapping α is inductively defined as follows.

- Let S be a basic empty statechart. Then $\alpha(S) = S :: S :: nil$.

- Let $S : (\{S_1, \ldots, S_n\}, T, in, out, \chi, \delta)$ be a basic non-empty statechart. For each $i \in \{1, n\}$, let $T_i = \{t \in T \mid out(t) = S_i\}$. Let $t_{S_i} = rec\ s_i.S_i :: (\sum_{t\in T_i} lab(t).in(t))$, if $T_i \neq \emptyset$ and $t_{S_i} = S_i :: nil$, otherwise.
  Process $\alpha(S)$ is obtained by repeating the following substitutions, where initially $\alpha(S) = S :: t_{\delta(S)}$:
  - If $\alpha(S)$ is closed then stop;
  - If $\alpha(S)$ is open, then for each $S_j \in free(\alpha(S))$, $\alpha(S) = \alpha(S)[t_{S_j}/S_j]$.

- Let S be a refined statechart $\bigtriangledown(S, \{S_1, \ldots, S_n\})$ and $\alpha(S) = S :: t_S$. Then $\alpha(\bigtriangledown(S, \{S_1, \ldots, S_n\}))$ is defined as

$$S :: ref((t_S, \{(S_1, \alpha(S_1)), \ldots, (S_n, \alpha(S_n))\}).$$

- Let $S : \mathrm{AND}(S_1, S_2)$. Then $\alpha(S) = S :: \alpha(S_1) \times \alpha(S_2)$.

The definitions of $lab(t)$ and of $\times$, $\#$ are given so that the semantics of $\alpha(S)$ agrees with the Pnueli and Shalev semantics of S. We need some preliminary concepts.

Assume $I \subseteq \Pi$ and $\ell \subseteq \Pi \cup \bar{\Pi}$. We say that I *triggers* ℓ ($I \rightsquigarrow \ell$) iff, for each $e \in \ell \cap \Pi$, $e \in I$, and for each $\bar{e} \in \ell \cap \bar{\Pi}$, $e \notin I$. Moreover, for $\alpha = \sigma : \tau$, we denote by $trigger(\alpha)$ and $action(\alpha)$ the sets $trigger(\sigma)$ and $action(\sigma)$, respectively.

Assume $p \in \mathcal{CSP}$. Process p is *regular* iff it has guarded recursion and for each variable x, x does not occur free in subterms $\ell :: p_1 \times p_2$ and in process p_1, such that $\ell :: t \bigtriangledown p_1$ is a subterm of p.

Definition 13. Assume $p \in \mathcal{CSP}$. We define $\mathcal{R}_p$ as the subset of $\mathcal{CSP}$ such that

- $p \in \mathcal{R}_p$;
- if $p' \in \mathcal{R}_p$ and, if $p' \overset{\alpha}{\mapsto} p''$, then $p'' \in \mathcal{R}_p$.

For a regular process $p \in \mathcal{CSP}$, $\mathcal{R}_p$ is finite, namely the process is finite-state. It is quite obvious that the $\mathcal{SP}$ processes corresponding to statecharts are closed and regular.

Theorem 14. *Let S be a statechart over Π.*

– *For each* $c \in C(S)$*, such that* c *is reachable from* $\Delta(S)$*, there exists* $p \in \mathcal{R}_{\alpha(S)}$*, such that* $loc(p) = c$*. Moreover, for each* $I \subseteq \Pi$*, if* $c \stackrel{I,O}{\Longrightarrow} c'$*, then* $p \stackrel{\alpha}{\mapsto} p'$*, where* $loc(p') = c'$ *and* $I \rightsquigarrow trigger(\alpha)$*,* $O - trigger(\alpha) = action(\alpha)$*;*
– *For each* $p \in \mathcal{R}_{\alpha(S)}$*, if* $p \stackrel{\alpha}{\mapsto} p'$*, then there exist* $c, c' \in C(S)$*, where* $loc(p) = c$*,* $loc(p') = c'$ *and* $c \stackrel{I,O}{\Longrightarrow} c'$*, for each* $I, O \subseteq \Pi$*, such that* $I \rightsquigarrow trigger(\alpha)$ *and* $O - trigger(\alpha) = action(\alpha)$*.*

This theorem can be proved by induction on the structure of the statechart. We omit the proof for lack of space. Consistency of steps is guaranteed by the definition of actions $\Sigma(\Pi)$, while maximality is guaranteed by the definition of $\#$ and by the semantics of idle actions (rules $choice_2$ and $\bigtriangledown_3$). The more complex case is that of parallel composition, since it is necessary to show that the condition of inseparability is captured by the definition of $\times$ over $\Sigma(\Pi)$. The idea is that the irreflexivity of the ordering ensures the existence of a causal ordering between transitions.

Note that by taking a different set of actions corresponding to transitions and different operations $\times$ and $\#$, we could have a process $\alpha(S)$, which has a different semantics, for instance the ones of [5, 10].

We explain the translation for the statechart of figure 1 over primitive events $\Pi = \{a, b, d, nil\}$. For basic statecharts E and F,

$$\alpha(E) = E :: rec\, x.A :: lab(t_1).rec\, y.C :: lab(t_2).x$$
$$\alpha(F) = F :: rec\, x.B :: lab(t_3).rec\, y.D :: lab(t_4).x.$$

Note that $loc(\alpha(E)) = \{E, A\}$ and $loc(\alpha(F)) = \{F, B\}$, which are the initial configurations of E and F respectively. It is quite obvious that the semantics of $\alpha(E)$ and of $\alpha(F)$ correspond to the semantics of E and F, respectively.

By rule rec_1,

$$\alpha(E) \stackrel{lab(t_1):\epsilon}{\mapsto} E :: rec\, y.C :: lab(t_2).t_1$$

where $t_1 = rec\, x.A :: lab(t_1).rec\, y.C :: lab(t_2).x$. Since $trigger(lab(t_1)) = \{a\}$ and $action(lab(t_1)) = \{b\}$, this trasition corresponds to the step $\{t_1\}$ from $\{E, A\}$ to $\{E, C\}$ for inputs containing a. On the other hand, by rule rec_2,

$$\alpha(E) \stackrel{\sigma:\bar{\epsilon}}{\mapsto} \alpha(E)$$

for $\sigma \in \#(lab(t_1))$, i.e. $\sigma \in \{(\{\bar{a}, b, d\}, \emptyset), (\{\bar{a}, b, \bar{d}\}.\emptyset), (\{\bar{a}, \bar{b}, d\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{d}\}, \emptyset)\}$. This transition corresponds to the empty step from $\{E, A\}$ with respect to inputs that do not contain a.

Since $G : \mathrm{AND}(E, F)$, $\alpha(G) = G :: \alpha(E) \times \alpha(F)$. Maximality and consistency of steps are obtained by the definition of $\times$ over actions. Let $p_1 = E :: rec\, y.C :: lab(t_2).t_1$ and $p_2 = F :: rec\, y.D :: lab(t_4).t_2$. By rule par,

$$\alpha(G) \stackrel{(\{a,b\}, \{a\} \prec \{b\}):\epsilon}{\mapsto} G :: p_1 \times p_2.$$

This transition corresponds to the step $\{t_1, t_3\}$ from the configuration $\{G, E, F, A, B\}$ to the configuration $\{G, E, F, C, D\}$, when the environment produces a.

Actually, $trigger((\{a,b\},\{a\} \prec \{b\})) = \{a\}$. Steps such that one component takes a transition and the other one not, are obtained by combining a non-idle arrow of one component with an idle arrow of the other. For instance, from $\alpha(E) \overset{(\{\bar{a},b,\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(E)$ and $\alpha(F) \overset{\{b\},\emptyset):\epsilon}{\mapsto} F :: rec\, y.D :: lab(t_4).t_2$, we have

$$G :: \alpha(E) \times \alpha(F) \overset{\{b,\bar{a},\bar{d}\},\emptyset):\epsilon}{\mapsto} G :: \alpha(E) \times F :: rec\, y.D :: lab(t_4).t_2.$$

This transition corresponds to the non-empty step $\{t_3\}$ when the environment produces b and does not produce a and d. Note that there are no steps, such that E takes a transition and F does not take a transition, since $action(t_1)$ triggers transition t_2. Finally, we have the following empty step $\alpha(G) \overset{\sigma:\bar{\epsilon}}{\mapsto} \alpha(G)$ for $\sigma \in \#(lab(t_1)) \cap \#(lab(t_3))$, i.e. $\sigma \in \{(\{\bar{a},\bar{b},d\},\emptyset), \{\bar{a},\bar{b},\bar{d}\},\emptyset)$.

Consider basic statechart S. We have

$$\alpha(S) = S :: rec\, x.G :: lab(t_5).rec\, y.H :: lab(t_6).x.$$

Hence, for the refinement of S, $\alpha(\bigtriangledown(S,\{G,H\}))$

$$S :: rec\, x.(G :: lab(t_5).rec\, y.(H :: lab(t_6).x \bigtriangledown \alpha(H))) \bigtriangledown \alpha(G)$$

where $\alpha(H) = H :: H :: nil$.

Since $\alpha(G) \overset{(\{a,b\},\{a\}\prec\{b\}):\epsilon}{\mapsto} G :: p_1 \times p_2$, by rule rec_1 and by rule $\bigtriangledown_2$,

$$\alpha(\bigtriangledown(S,\{G,H\})) \overset{(\{a,b\},\{a\}\prec\{b\}):\epsilon}{\mapsto} S :: t_4 \bigtriangledown G :: p_1 \times p_2$$

where $t_4 = G :: lab(t_5).rec\, y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H))$.

Since $loc(G :: p_1 \times p_2) = \{G,E,F,D,C\}$, $loc(S :: t_4 \bigtriangledown G :: p_1 \times p_2) = \{S,G,E,F,D,C\}$, this arrow corresponds to the step $\{t_1,t_3\}$ when the input contains event a.

Since $S :: t_4 \overset{lab(t_5):\epsilon}{\mapsto} S :: rec\, y.\ (H :: lab(t_6).t_6 \bigtriangledown \alpha(H))$, by rule $\bigtriangledown_1$,

$$\alpha(\bigtriangledown(S,\{G,H\})) \overset{lab(t_5):\epsilon}{\mapsto} S :: rec\, y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H)).$$

Since $loc(S :: rec\, y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H))) = \{S,H\}$, this corresponds to the step $\{t_5\}$ from configuration $\{S,G,E,F,A,B\}$ to configuration $\{S,H\}$, when the environment produces d.

Note that rules $\bigtriangledown_1$ and $\bigtriangledown_2$ can be applied, since the type of the action of $S :: t_4$ and $\alpha(G)$ is non-idle, respectively. On the other hand, empty steps from $\{S,G,E,F,A,B\}$ must be empty steps both of statechart G and of basic statechart S. Hence, we have an empty step iff the environment produces neither a, b nor d. This is reflected by rule $\bigtriangledown_3$, where we require that the two parts agree on idle actions.

We have $\alpha(G) \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(G)$ and $S :: t_4 \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} S :: t_4$, so that

$$S :: t_4 \bigtriangledown \alpha(G) \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} S :: t_4 \bigtriangledown \alpha(G).$$

By rule rec_2,

$$\alpha(\bigtriangledown(S,\{G,H\}) \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(\bigtriangledown(S,\{G,H\}).$$

Note that $\#(lab(t_1)) \cap \#(lab(t_3)) \cap \#(lab(t_5)) = (\{\bar{a},\bar{b},\bar{d}\},\emptyset)$.

5 Related works

In this section we discuss the differences between our process-algebraic semantics and the one of [17]. The process language has an operator of refinement similar to ours and the translation differs only in the actions describing transitions. However, the labelled transition system semantics of the language differs in many aspects.

The first difference arises in the definition of basic actions corresponding to transitions. In the approach of [17] transitions are represented by pairs $(\ell, \prec)$, where ℓ is a set of events and $\prec$ is an irreflexive and transitive ordering on ℓ. On the other hand, in our definition $\prec$ is an ordering over $2^{\Pi \cup \bar{\Pi}}$. This definition leads to a semantics different from the one of Pnueli and Shalev, when a transition has a set of events as trigger.

Consider statechart G of figure 1, where the labels of t_1 and t_3 are ac/b and b/c, respectively. The action corresponding to transition t_1 is $a_1 = (\{a, b, c\}, a \prec b, c \prec b)$, while that corresponding to t_2 is $a_2 = (\{b, c\}, b \prec c)$. The parallel composition of actions is defined as in our approach. However, if we combine a_1 and a_2 we obtain the set $\{(\{a, b, c\}, a \prec b, c \prec b), (\{a, b, c\}, a \prec b, b \prec c, a \prec c)\}$. The first action corresponds to take both transitions on input $\{a, c\}$. On the other hand, the action $(\{a, b, c\}, a \prec b, b \prec c, a \prec c)$ corresponds to take both transitions with respect to an input that contain $\{a\}$. In particular this is a step with respect to the input $\{a\}$. This is not correct, since event a is not suffient for enabling both transitions.

In our definition, we have considered a causal ordering between sets of events instead of single events. Hence, the action corresponding to t_1 is $a_1 = (\{a, b, c\}, \{a, c\} \prec \{b\})$, while the one corresponding to t_3 is $a_2 = (\{b, c\}, \{b\} \prec \{c\})$. The parallel composition of a_1 and a_2 is the set $\{a = (\{a, b, c\}, \{a, c\} \prec \{b\}), a' = (\{a, b, c\}, \{b\} \prec \{c\})\}$. Hence, we have $trigger(a) = \{a, c\}$ and $trigger(a') = \{a, b\}$ so that event a is not sufficient to trigger the step $\{t_1, t_3\}$.

Another drawback in the semantics of Uselton and Smolka regards the labelled transition system semantics of the process language. Actually, in their approach idle actions are not considered so that empty steps are not explicitely represented. Consider statechart G of figure 1 where t_1 and t_3 are labelled by $\bar{a}/b$ and b/a, respectively. The $\mathcal{SP}$ process corresponding to statechart G is given by $G :: p_1 \times p_2$, where $p_1 = E :: rec\, x.A :: lab(t_1).t_1$ and $p_2 = F :: rec\, y.B :: lab(t_3).t_2$ with $lab(t_1) = (\{\bar{a}, b\}, \{\bar{a}\} \prec \{b\})$ and $lab(t_2) = (\{b, a\}, \{b\} \prec \{a\})$ and t_1, t_2 the obvious processes. The process obtained by Uselton and Smolka is equivalent to this one and differs only for the notation. However, in this approach the only transition for the process corresponding to G is given by

$$G :: p_1 \times p_2 \overset{(\{b,a\}, b \prec a)}{\longmapsto} G :: p_1 \times F :: t_2.$$

Actually, the step $\{t_3\}$ with respect to inputs that contain b is the only non-empty step from the configuration $\{G, E, F, A, B\}$. With respect to the empty input a causal paradox occurs, while for inputs with $\{a\}$ the empty set of transitions is an admissible reaction. However, the difference between the empty step on input

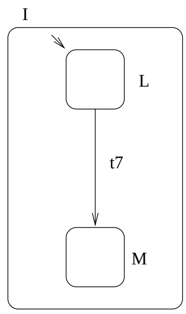

Fig. 2. The basic statechart I

$\{a\}$ and the run-time error on empty input is not reflected by the transition system semantics of Uselton and Smolka. This gives rise to non-empty steps, that are not admissible. The problem arises, when a non-empty step of the parallel composition is given by an empty step of one component and by a non-empty step of the other component. Since empty steps are not explicitly represented, Uselton and Smolka use an additional rule for parallel composition with negative premises. However, non-admissible steps and empty steps cannot be distinguished by this rule.

Consider for instance the statechart $H : \mathrm{AND}(G, I)$, where I is shown in figure 2 and t_7 is labelled by nil/c. The corresponding process is

$$p' = H :: G :: (p_1 \times p_2) \times p_3$$

where $p_3 = I :: rec\, z.L :: lab(t_7).M :: nil = \alpha(I)$. By applying the rule with negative premises, we would obtain the following transition

$$p' \overset{(\{\bar{b}, c, \bar{a}, nil\}, nil \prec c)}{\longmapsto} H :: G :: (p_1 \times p_2) \times I :: M :: nil$$

where $(\{\bar{b}, c, \bar{a}, nil\}, nil \prec c) \in (\{c, nil\}, nil \prec c) \times (\{\bar{a}, \bar{b}\}, \emptyset)$.
The action $(\{c, nil\}, nil \prec c)$ is the one of the non-empty step of I, $p_3 \overset{(\{c,nil\},nil \prec c)}{\longmapsto} I :: M :: nil$, while the action $(\{\bar{a}, \bar{b}\}, \emptyset)$ represents an empty step of G. The idea is that $(\{\bar{a}, \bar{b}\}, \emptyset)$ is in conflict with the action $(\{b, a\}, b \prec a)$, namely when the environment produces neither b nor a, the only non-empty step of G is not enabled. However, since $trigger((\{\bar{b}, c, \bar{a}, nil\}, nil \prec c)) = \{nil, \bar{a}, \bar{b}\}$, this transition corresponds to the step $\{t_7\}$ with respect to inputs that contain neither a nor b. This is not an admissible step in the Pnueli and Shalev semantics, because of the causal paradox.

Therefore, the explicit representation of empty steps in the labelled transition system semantics is necessary to obtain the correct semantics. Moreover, this approach allows us to define a single rule for parallel composition rather than the complex rules of [17]. Note that the same drawbacks are present also in the labelled transition system semantics presented in [16].

6 Conclusions

In this paper we have defined the process language $\mathcal{SP}$ and we have shown a translation of statecharts into closed processes of $\mathcal{SP}$ according to the semantics of Pnueli and Shalev. The main novelty of the process language $\mathcal{SP}$ is the operator of process refinement, which is needed to represent statecharts hierarchy. In [17] a process-algebraic version of statecharts has been proposed, with the aim of finding an axiomatization of statecharts equivalence. The operator of refinement is similar to ours. However, the labelled transition system semantics of the language does not represent *idle* actions so that it is not equivalent to the one of Pnueli and Shalev, as we have discussed. Moreover, our translation has the advantage to be general, since other semantics of statecharts can be obtained by modifying the operations over actions.

Our interest for a process-algebraic semantics for statecharts is motivated by the possibility of using the semantics for studying equivalences of statecharts and for proving temporal properties by model checking [15]. Actually, the translation allows us to construct in a compositional way the labelled transition system of a statechart. This is very useful since, if we replace a component with one that is equivalent with respect to labelled transition system isomorphism, we should not redo the proofs. Moreover, this proposal is a first step towards a compositional verification technique for statecharts, where the verification that a composed statechart satisfies a formula is reduced to the verification of derived formulas for its components. The main advantage is that, when changing a part of a process, only the verification concerning that part must be redone. Moreover, compositional verfication permits to reason with partial specifications and can be applied during the process of specifications development. A compositional proof system in the style of [1] for checking whether an $\mathcal{SP}$ process satisfies a μ-calculus formula has been developed in [9].

References

1. H.R. Andersen, C. Stirling, and G. Winskel. A Compositional Proof System for the Modal mu-Calculus. In *Proc. Ninth IEEE Symp. on Logic In Computer Science*, pages 144–153, 1994.
2. G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, (19):87–152, 1992.
3. P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. Lustre, a Declarative Language for Programming Synchronous Systems. In *Proccedings 14-th Symposium of Principles of Programming Languages*, pages 178–188, 1987.
4. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
5. D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. In *Proc. Second IEEE Symp. on Logic In Computer Science*, pages 54–64. IEEE Computer Society Press, 1987.
6. J.J.M. Hooman, R. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *Theoretical Computer Science*, 101:289–335, 1992.

7. C. Huizing and R. Gerth. Semantics of Reactive Systems in Abstract Time. volume 600 of *Lecture Notes in Computer Science*, pages 291–314. Springer-Verlag, Berlin, 1992.
8. C. Huizing, R. Gerth, and W.P. de Roever. Modelling Statecharts in a Fully Abstract Way. volume 299 of *Lecture Notes in Computer Science*, pages 271–294. Springer-Verlag, Berlin, 1988.
9. F. Levi. *Verification of Temporal and Real-Time Properties of Statecharts.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1997.
10. A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *Proc. of CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 687–702. Springer-Verlag, Berlin, 1996.
11. F. Maraninchi. Operational and Compositional Semantics of Synchronous Automaton Composition. In *Proc. of CONCUR 92*, volume 630 of *Lecture Notes in Computer Science*, pages 550–564, 1992.
12. R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
13. A. Pnueli and M. Shalev. What is in a Step. volume 525 of *Lecture Notes in Computer Science*, pages 244–464. Springer-Verlag, Berlin, 1991.
14. K.V.S. Prasad. A Calculus of Broadcast Systems. In *Proc. of TAPSOFT-CAAP'91*, volume 493 of *Lecture Notes in Computer Science*, pages 338–358, 1991.
15. C. Stirling. Modal and Temporal Logics. In S. Abramsky and D. Gabbay, editors, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.
16. A.C. Uselton and S.A. Smolka. A Compositional Semantics for Statecharts using Labelled Transition Systems. In *Proc. of CONCUR 94*, volume 836 of *Lecture Notes in Computer Science*, pages 2–17, 1994.
17. A.C. Uselton and S.A. Smolka. A Process Algebraic Semantics for Statecharts. In *Proceedings of IFIP Working Conference on Programming Concepts, Methods and Calculi*, 1994.