Statecharts Via Process Algebra*

Gerald Lüttgen[1], Michael von der Beeck[2], and Rance Cleaveland[3]

[1] Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23681-2199, USA, luettgen@icase.edu
[2] Department of Computer Science, Munich University of Technology, Arcisstr. 21, D-80290 München, Germany, beeck@in.tum.de
[3] Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA, rance@cs.sunysb.edu

Abstract. *Statecharts* is a visual language for specifying the behavior of *reactive systems*. The language extends *finite-state machines* with concepts of *hierarchy*, *concurrency*, and *priority*. Despite its popularity as a design notation for *embedded systems*, precisely defining its semantics has proved extremely challenging. In this paper, we present a simple *process algebra*, called *Statecharts Process Language* (SPL), which is expressive enough for encoding Statecharts in a structure-preserving and semantics-preserving manner. We also establish that the behavioral equivalence *bisimulation*, when applied to SPL, preserves Statecharts semantics.

1 Introduction

Statecharts is a visual language for specifying the behavior of *reactive systems* [7]. The language extends the notation of *finite-state machines* with concepts of (i) *hierarchy*, so that one may speak of a state as having sub-states, (ii) *concurrency*, thereby allowing the definition of systems having simultaneously active subsystems, and (iii) *priority*, so that one may express that certain system activities have precedence over others. Statecharts has become popular among engineers as a design notation for *embedded systems*, and commercially available tools provide support for it [10]. Nevertheless, precisely defining its semantics has proved extremely challenging, with a variety of proposals [6, 8, 9, 17, 18, 19, 22, 23] being offered for several dialects [26] of the language. The semantic subtlety of Statecharts arises from the language's capability for defining transitions whose enabledness disables other transitions. A Statechart may react to an *event* by engaging in an enabled transition, thereby performing a so-called *micro step*, which may generate new events that may in turn trigger new transitions while disabling others. When this chain reaction

* This work was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the first author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199, USA. The third author was supported by NSF grants CCR-9257963, CCR-9505662, CCR-9804091, and INT-9603441, AFOSR grant F49620-95-1-0508, and ARO grant P-38682-MA.

comes to a halt, one execution step – also referred to as a *macro step* – is complete. At a technical level, the difficulty for defining an operational semantics capturing the "macro-step" behavior of Statecharts arises from the fact that such a semantics should exhibit the following desirable properties: (i) the *synchrony hypothesis* [2], which guarantees that a reaction to an external event terminates before the next event enters the system, (ii) *compositionality*, which ensures that the semantics of a Statechart is defined in terms of the semantics of its components, and (iii) *causality*, which demands that the participation of each transition in a macro step must be causally justified. Huizing and Gerth showed that an operational semantics in which transitions are labeled purely by sets of events – i.e., the "observations" a user would make – cannot be given, if one wishes all three properties to hold [15]. In fact, the traditional semantics of Statecharts – as defined by Pnueli and Shalev [22] – satisfies the synchrony hypothesis and causality, but is not compositional. Other approaches, e.g. [17], have achieved all three goals, but at the expense of including complex information regarding causality in transition labels.

While not as well-established in practice, *process algebras* [1, 12, 21] offer many of the semantic advantages that have proved elusive in Statecharts. In general, these theories are operational, and place heavy emphasis on issues of compositionality through the study of *congruence relations*. Many of the behavioral aspects of Statecharts have also been studied for process algebras. For example, the synchrony hypothesis is related to the *maximal progress assumption* developed in *timed* process algebras [11, 28]. In these algebras, event transitions and "clock" transitions are distinguished, with only the latter representing the advance of time. Maximal progress then ensures that time may proceed only if the system under consideration cannot engage in internal computation. Clocks may therefore be viewed as "bundling" sequences of event transitions, which may be thought of as analogous to "micro steps," into a single "time step," which may be seen as a "macro step." The concept of priority has also been studied in process-algebraic settings [4], and the Statecharts hierarchy operator is related to the *disabling* operator of LOTOS [3].

In this paper, we present a new, *process-algebraic* semantics of Statecharts. Our approach synthesizes the observations above; specifically, we present a new process algebra, called *Statecharts Process Language* (SPL), and we show that it is expressive enough for embedding several Statecharts variants. SPL is inspired by Hennessy and Regan's *Timed Process Language* (TPL) [11] which extends Milner's CCS [21] by the concept of an abstract, global clock. Our algebra replaces the handshake communication of TPL by a *multi-event communication*, and introduces a mechanism to specify *priority* among transitions as well as a *hierarchy operator* [25]. The operational semantics of SPL uses SOS rules to define a transition relation whose elements are labeled with simple sets of events; then, using traditional process-algebraic results we show that SPL has a compositional semantic theory based on *bisimulation* [21]. We connect SPL with Statecharts by embedding the variant of the language considered by Maggiolo-Schettini et al. in [17]. More precisely, we define a compositional translation from Statecharts to SPL that preserves the macro-step semantics of the former. This result depends crucially on our treatment of the SPL macro-step transition relation as a *derived* one: the standard SPL transition relation becomes in essence a micro-step semantics. Thus, while our macro-step semantics cannot be compositional (cf. the result of Huizing and Gerth [15]), we obtain a compositional theory, in the form of a semantic congruence, at a lower, micro-step level. In addition to the usual benefits conferred by compositional reasoning, this semantics has a practical advantage: given the unavoidable complexity of inferring macro steps, actual users of Statecharts would benefit from a finer-grained semantics that helps them understand how the macro steps of their systems are arrived at.

2 Statecharts

Statecharts is a specification language for *reactive systems*, i.e., concurrent systems which are characterized by their ongoing interaction with their *environment*. They subsume finite state machines whose transitions are labeled by pairs of events, where the first component is referred to as *trigger* and may include *negated events*, and the second component is referred to as *action*. Intuitively, if the environment offers the events in the trigger, but not the negated ones, then the transition is triggered; it fires, thereby producing the events in the label's action. Concurrency is achieved by allowing Statecharts to be composed from more simple ones running in parallel, which may communicate via *broadcasting* events. Elementary, or *basic states* in Statecharts may also be hierarchically refined by injecting other Statecharts.

As an example, consider the Statechart depicted to the right. It consists of a so-called *and-state*, labeled by n_9, which denotes the parallel composition of the two Statecharts labeled by n_3 and n_8. Actually, n_3 and n_8 are the names of *or-states*, describing sequential state machines. The first consists of two states n_1 and n_2 that are connected via transition t_1 with label $\neg a/b$. The label specifies that t_1 is triggered by $\neg a$, i.e., by the absence of event a, and produces event b. States n_1 and n_2 are not refined further and, therefore, are referred to as basic states. Or-state n_8 is refined by or-state n_6 and basic state n_7, connected via a transition labeled by b/a. Or-state n_6 is further refined by basic states n_4 and n_5, and transition t_2 labeled by b/c. The variant of Statecharts considered here does not include *interlevel transitions* – i.e., transitions crossing borderlines of states – and *state references* – i.e., triggers of the form in_n, where n is a state name. Moreover, state hierarchy does not impose implicit priorities on transitions. The impact of altering our approach to accommodate these concepts is discussed in Sec. 6.

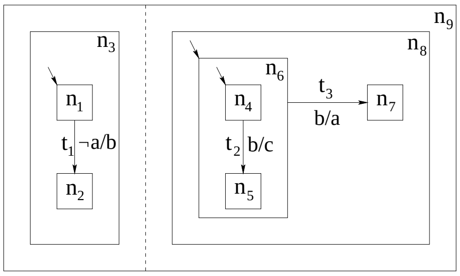


Fig. 1. Example Statechart

Table 1. States and transitions of Statecharts terms

$\mathsf{states}([n]) := \{n\}$	$\mathsf{states}([n : \boldsymbol{s}; l; T]) := \{n\} \cup \bigcup\{\mathsf{states}(s_i) \mid 1 \leq i \leq k\}$
	$\mathsf{states}([n : \boldsymbol{s}]) := \{n\} \cup \bigcup\{\mathsf{states}(s_i) \mid 1 \leq i \leq k\}$
$\mathsf{trans}([n]) := \emptyset$	$\mathsf{trans}([n : \boldsymbol{s}; l; T]) := T \cup \bigcup\{\mathsf{trans}(s_i) \mid 1 \leq i \leq k\}$
	$\mathsf{trans}([n : \boldsymbol{s}]) := \bigcup\{\mathsf{trans}(s_i) \mid 1 \leq i \leq k\}$

For our purposes, it is convenient to represent Statecharts not visually but by terms. This is also done in related work [16, 17, 24]; we closely follow [17]. Formally, let $\mathcal{N}$ be a countable set of names for Statecharts states, $\mathcal{T}$ be a countable set of names for Statecharts transitions, and Π be a countable set of Statecharts events. Moreover, we associate with every event $e \in \Pi$ its negated counterpart $\neg e$. We also lift negation to negated events by defining $\neg\neg e := e$. Finally, we write $\neg E$ for $\{\neg e \mid e \in E\}$. Then, the set of Statecharts terms is defined to be the least set satisfying the following rules.

1. **Basic state:** If $n \in \mathcal{N}$, then $s = [n]$ is a Statecharts term.
2. **Or-state:** If $n \in \mathcal{N}$, $s_1, \ldots, s_k$ are Statecharts terms, $k > 0$, $\rho = \{1, \ldots, k\}$, $T \subseteq \mathcal{T} \times \rho \times 2^{\Pi \cup \neg \Pi} \times 2^{\Pi} \times \rho$, and $1 \leq l \leq k$, then $s = [n : (s_1, \ldots, s_k); l; T]$ is a Statecharts term. Here, $s_1, \ldots, s_k$ are the sub-states of s, and T is the set of transitions between these states. Statechart s_1 is the default state of s, while s_l is the currently active state.
3. **And-state:** If $n \in \mathcal{N}$ and if $s_1, \ldots, s_k$ are Statecharts terms for $k > 0$, then $s = [n : (s_1, \ldots, s_k)]$ is a Statecharts term.

We refer to n as the *root* of s and write $\mathsf{root}(s) := n$. If $t = \langle \hat{t}, i, E, A, j \rangle \in T$ is a transition of or-state $[n : (s_1, \ldots, s_k); l; T]$, then we define $\mathsf{name}(t) := \hat{t}$, $\mathsf{out}(t) := s_i$, $\mathsf{ev}(t) := E$, $\mathsf{act}(t) := A$, and $\mathsf{in}(t) := s_j$. We write SC for the set of Statecharts terms, in which (i) all state names and transition names are mutually disjoint, (ii) no transition t produces an event that contradicts its trigger, i.e., $\mathsf{ev}(t) \cap \neg\mathsf{act}(t) = \emptyset$, and (iii) no transition t produces an event that is included in its trigger, i.e., $\mathsf{ev}(t) \cap \mathsf{act}(t) = \emptyset$. As a consequence of (i), states and transitions in Statecharts terms are uniquely referred to by their names. Therefore, we may identify a Statecharts state s and transition t with its name $\mathsf{root}(s)$ and $\mathsf{name}(t)$, respectively. The sets $\mathsf{states}(s)$ and $\mathsf{trans}(s)$ of all states and transitions of s are inductively defined on the structure of s, as depicted in Table 1, where $\boldsymbol{s} = (s_1, \ldots, s_k)$. Finally, let us return to our example Statechart in Fig. 1, and present it as a Statecharts term $s_9 \in \mathsf{SC}$. We choose $\Pi := \{a, b, c\}$, $\mathcal{N} := \{n_1, n_2, \ldots, n_9\}$, and $\mathcal{T} := \{t_1, t_2, t_3\}$.

$$
\begin{array}{lll}
s_9 := [n_9 : (s_3, s_8)] & s_3 := [n_3 : (s_1, s_2); 1; \{\langle t_1, 1, \{\neg a\}, \{b\}, 2\rangle\}] & s_1 := [n_1] \\
s_2 := [n_2] & s_8 := [n_8 : (s_6, s_7); 1; \{\langle t_3, 6, \{b\}, \{a\}, 7\rangle\}] & s_7 := [n_7] \\
s_4 := [n_4] & s_6 := [n_6 : (s_4, s_5); 1; \{\langle t_2, 4, \{b\}, \{c\}, 5\rangle\}] & s_5 := [n_5]
\end{array}
$$

In the remainder of this section, we formally present the semantics of Statecharts terms as is defined in [17], which is a slight variant of the "traditional" semantics proposed by Pnueli and Shalev [22]. More precisely, this semantics differs from [22] in that it does not allow the step-construction function, which we present below, to fail. The semantics of a Statecharts term s is a transition system, whose states and transitions are referred to as configurations and macro steps, respectively. Configurations of s are usually sets $\mathsf{conf}(s)$ of names of states which are currently active [22]. We define $\mathsf{conf}(s)$ along the structure of s: (i) $\mathsf{conf}([n]) := \{n\}$, (ii) $\mathsf{conf}([n : (s_1, \ldots, s_k); l; T]) := \{n\} \cup \mathsf{conf}(s_l)$, and (iii) $\mathsf{conf}([n : (s_1, \ldots, s_k)]) := \{n\} \cup \bigcup\{\mathsf{conf}(s_i) \,|\, 1 \leq i \leq k\}$. However, for our purposes it is more convenient to use Statecharts terms for configurations, as every or-state contains a reference to its active sub-state. Consequently, the *default configuration* $\mathsf{default}(s)$ of Statecharts term s may be defined inductively as follows: (i) $\mathsf{default}([n]) := [n]$, (ii) $\mathsf{default}([n : (s_1, \ldots, s_k); l; T]) := [n : (\mathsf{default}(s_1), \ldots, \mathsf{default}(s_k)); 1; T]$, and (iii) $\mathsf{default}([n : (s_1, \ldots, s_k)]) := [n : (\mathsf{default}(s_1), \ldots, \mathsf{default}(s_k))]$. As mentioned before, a Statechart reacts to the arrival of some external events by triggering enabled micro steps, possibly in a chain-reaction–like manner, thereby performing a macro step. More precisely, a macro step comprises a maximal set of micro steps, or transitions, that are *triggered* by events offered by the environment or generated by other micro steps, that are mutually *consistent*, *compatible*, and *relevant*, and that obey *causality*. The Statecharts principle of *global consistency*, which prohibits an event to be present and absent in the same macro step, is subsumed by *triggered* and *compatible*. In the following, we formally introduce the above notions.

Table 2. Step-construction function

```
function step-construction(s, E);  var T := ∅;
   while T ⊂ enabled(s, E, T) do choose t ∈ enabled(s, E, T) \ T;  T := T ∪ {t} od;
return T
```

A transition $t \in \mathsf{trans}(s)$ is *consistent* with all transitions in $T \subseteq \mathsf{trans}(s)$, in signs $t \in \mathsf{consistent}(s, T)$, if t is not in the same parallel component as any transition in T. Formally, $\mathsf{consistent}(s, T) := \{t \in \mathsf{trans}(s) \,|\, \forall t' \in T.\, t \bot_s t'\}$. Here, we write $t \bot_s t'$, if $t = t'$, or if there exists an and-state $[n : (s_1, \ldots, s_k)]$ in s, i.e., $n \in \mathsf{states}(s)$, such that $t \in \mathsf{trans}(s_i)$ and $t' \in \mathsf{trans}(s_j)$ for some $1 \leq i, j \leq k$ satisfying $i \neq j$. A transition $t \in \mathsf{trans}(s)$ is *compatible* to all transitions in $T \subseteq \mathsf{trans}(s)$, in signs $t \in \mathsf{compatible}(s, T)$, if no event produced by t appears negated in a trigger of a transition in T. Formally, $\mathsf{compatible}(s, T) := \{t \in \mathsf{trans}(s) \,|\, \forall t' \in T.\, \mathsf{act}(t) \cap \neg \mathsf{ev}(t') = \emptyset\}$. A transition $t \in \mathsf{trans}(s)$ is *relevant* for s, in signs $t \in \mathsf{relevant}(s)$, if the root of the source state of t is in the configuration of s. Formally, $\mathsf{relevant}(s) := \{t \in \mathsf{trans}(s) \,|\, \mathsf{root}(\mathsf{out}(t)) \in \mathsf{conf}(s)\}$. A transition $t \in \mathsf{trans}(s)$ is *triggered* by a set E of events, in signs $t \in \mathsf{triggered}(s, E)$, if the positive, but not the negative, trigger events of t are in E. Formally, $\mathsf{triggered}(s, E) := \{t \in \mathsf{trans}(s) \,|\, \mathsf{ev}(t) \cap \Pi \subseteq E$ and $\neg(\mathsf{ev}(t) \cap \neg \Pi) \cap E = \emptyset\}$. Finally, t is *enabled* in s regarding a set E of events and a set T of transitions, if $t \in \mathsf{enabled}(s, E, T)$, where $\mathsf{enabled}(s, E, T) := \mathsf{relevant}(s) \cap \mathsf{consistent}(s, T) \cap \mathsf{triggered}(s, E \cup \bigcup_{t \in T} \mathsf{act}(t)) \cap \mathsf{compatible}(s, T)$. Unfortunately, this formalism is still not rich enough to *causally* justify the triggering of each transition. The principle of *causality* may be introduced by computing macro steps, i.e., sets of transition names, using the nondeterministic *step-construction function* presented in Table 2. This function is adopted from [17], where also its soundness and completeness relative to the classical approach via the notion of *inseparability* of transitions [22] are stated. Note that the maximality of each macro step implements the synchrony hypothesis of Statecharts. The set of all macro steps that can be constructed using function *step-construction*, relative to a Statecharts term s and a set E of environment events, is denoted by $\mathsf{step}(s, E) \subseteq 2^{\mathcal{T}}$.

Table 3. Function update

$$\mathsf{update}([n], T') := [n] \quad \mathsf{update}([n : \boldsymbol{s}], T') := [n : (\mathsf{update}(s_1, T_1), \ldots, \mathsf{update}(s_k, T_k))]$$

$$\mathsf{update}([n : \boldsymbol{s}; l; T], T') := \begin{cases} [n : \boldsymbol{s}; l; T] & \text{if } T' = \emptyset \\ [n : (s_1, \ldots, \mathsf{update}(s_l, T'), \ldots, s_k); l; T] & \text{if } \emptyset \neq T' \subseteq \mathsf{trans}(s_l) \\ [n : (s_1, \ldots, \mathsf{default}(s_m), \ldots, s_k); m; T] & \text{if } \emptyset \neq T' = \{\langle t', l, E, A, m \rangle\} \subseteq T \\ [n] & \text{otherwise} \end{cases}$$

For a set $T \in \mathsf{step}(s, E)$, Statecharts term s may evolve in a macro step to term $s' := \mathsf{update}(s, T)$ when triggered by the environment events in E and, thereby, produce the events $A := \bigcup\{\mathsf{act}(t) \,|\, t \in T\}$. We denote this macro step by $s \overset{E}{\underset{A}{\Longrightarrow}} s'$. The function update is defined in Table 3, where $\boldsymbol{s} := (s_1, \ldots, s_k)$ and $T_i := T' \cap \mathsf{trans}(s_i)$, for $1 \leq i \leq k$. Observe that at most one transition of T may be enabled at the top-level of an or-state; thus, the "otherwise" case in Table 3 cannot occur in our context. Intuitively, $\mathsf{update}(s, T)$, when $T \subseteq \mathsf{trans}(s)$, re-defines the active states of s, when the transitions in T are executed.

3 Process-Algebraic Framework

Our process-algebraic framework is inspired by *timed process calculi*, such as Hennessy and Regan's TPL [11]. The *Statecharts Process Language* (SPL), which we intend to develop, includes a special action σ denoting the ticking of a global clock. SPL's semantic framework is based on a notion of transition system that involves two kinds of transitions, *action* transitions and *clock* transitions, modeling two different mechanisms of communication and synchronization in *concurrent* systems. The role of actions correspond to the one of events in Statecharts. A clock represents the progress of time, which manifests itself in a recurrent global synchronization event, the clock transition, in which all process components are forced to take part. However, action and clock transitions are not orthogonal concepts but are connected via the *maximal progress assumption* [11, 28]. Maximal progress implies that progress of time is determined by the *completion of internal computations* and, thus, mimics Statecharts' synchrony hypothesis. The key idea for embedding Statecharts terms in a timed process algebra is to represent a macro step as a sequence of micro steps that is enclosed by clock transitions, signaling the beginning and the end of the macro step, respectively. This sequence implicitly encodes causality and leads to a compositional Statecharts semantics. Unfortunately, existing timed process algebras are – in their original form – not suitable for embedding Statecharts. The reason is that Statecharts transitions may be labeled by *multiple* events and that some events may appear *negated*. The former feature implies that, in contrast to standard process algebras [1, 12, 21], processes may be forced to synchronize on more than one event simultaneously, and the latter feature is similar to mechanisms for handling priority [4]. Our framework must also include an operator similar to the *disabling operator* of LOTOS [3] for resembling state hierarchy [25].

Formally, let Λ be a countable set of *events* or *ports*, and let $\sigma \notin \Lambda$ be the distinguished *clock event* or *clock tick*. We define *input actions* to be of the form $\langle E, N \rangle$, where $E, N \subseteq \Lambda$, and *output actions* E to be subsets of Λ. In case of the input action $\langle \emptyset, \emptyset \rangle$, we speak of an *unobservable* or *internal* action, which is also denoted by $\bullet$. We let $\mathcal{A}$ stand for the set of all input actions. In contrast to CCS [21], the syntax of SPL includes two different operators for dealing with input and output actions, respectively. The *prefix operator* "$\langle E, N \rangle$." only permits prefixing with respect to input actions, which are instantly consumed in a single step. Output actions E are signaled to the environment of a process by attaching them to the process via the *signal* operator "$[E]\sigma(\cdot)$." They remain visible until the next clock tick σ occurs. The syntax of SPL is given by the following BNF

$$P \ ::= \ \mathbf{0} \mid X \mid \langle E, N \rangle.P \mid [E]\sigma(P) \mid P + P \mid P \rhd P \mid P \rhd_\sigma P \mid P|P \mid P \setminus L$$

where $L \subseteq \Lambda$ is a *restriction set*, and X is a *process variable* taken from some countable domain $\mathcal{V}$. We also allow the definition of *equations* $X \stackrel{\text{def}}{=} P$, where variable X is assigned to term P. If X occurs as a subterm of P, we say that X is *recursively* defined. We adopt the usual definitions for *open* and *closed* terms and *guarded* recursion, and refer to the closed and guarded terms as *processes* [21]. Moreover, we let $\mathcal{P}$, ranged over by P and Q, denote the set of all processes. Finally, the operators $\rhd$ and $\rhd_\sigma$, called *disabling* and *enabling* operator, respectively, allow us to model state hierarchy, as is illustrated below.

The operational semantics of an SPL process $P \in \mathcal{P}$ is given by a *labeled transition system* $\langle \mathcal{P}, \mathcal{A} \cup \{\sigma\}, \longrightarrow, P \rangle$, where $\mathcal{P}$ is the set of states, $\mathcal{A} \cup \{\sigma\}$ the alphabet, $\longrightarrow$ the transition relation, and P the start state. We refer to transitions with labels in $\mathcal{A}$ as *action transitions* and to those with label σ as *clock transitions*. For the sake of simplicity, we write (i) $P \xrightarrow[N]{E} P'$ instead of $\langle P, \langle E, N \rangle, P' \rangle \in \longrightarrow$ and (ii) $P \xrightarrow{\sigma} P'$ instead of $\langle P, \sigma, P' \rangle \in \longrightarrow$. We say that *$P$ may engage in a transition labeled by $\langle E, N \rangle$ or σ, respectively, and thereafter behave like process P'*. The transition relation is defined in Tables 4 and 5 using operational rules. In contrast to CCS [21], our framework does not provide a concept of output action transitions, such that "matching" input and output action transitions synchronize with each other and, thereby, simultaneously change states. Instead, output actions are attached to SPL processes via the signal operator. In order to present our communication mechanism, we need to introduce *initial output action sets*, $\overline{\mathbb{I}}(P)$, for $P \in \mathcal{P}$. These are defined as the *least* sets satisfying the equations in Table 4 (upper part). Intuitively, $\overline{\mathbb{I}}(P)$ collects all events which are initially offered by P.

Table 4. Initial output action sets & operational semantics (action transitions)

$$
\begin{array}{lll}
\overline{\mathbb{I}}([E]\sigma(P)) = E & \overline{\mathbb{I}}(P+Q) = \overline{\mathbb{I}}(P) \cup \overline{\mathbb{I}}(Q) & \overline{\mathbb{I}}(X) \quad\;\; = \overline{\mathbb{I}}(P) \quad \text{if } X \stackrel{\text{def}}{=} P \\
 & \overline{\mathbb{I}}(P\,|\,Q) \;\;\, = \overline{\mathbb{I}}(P) \cup \overline{\mathbb{I}}(Q) & \overline{\mathbb{I}}(P \setminus L) \;\;\, = \overline{\mathbb{I}}(P) \setminus L \\
 & \overline{\mathbb{I}}(P \rhd Q) = \overline{\mathbb{I}}(P) \cup \overline{\mathbb{I}}(Q) & \overline{\mathbb{I}}(P \rhd_\sigma Q) = \overline{\mathbb{I}}(P)
\end{array}
$$

$$
\mathsf{Act}\ \frac{-}{\langle E,N\rangle.P \xrightarrow[N]{E} P}
$$

$$
\mathsf{Rec}\ \frac{P \xrightarrow[N]{E} P'}{X \xrightarrow[N]{E} P'}\ X \stackrel{\text{def}}{=} P
\qquad
\mathsf{Sum1}\ \frac{P \xrightarrow[N]{E} P'}{P+Q \xrightarrow[N]{E} P'}
\qquad
\mathsf{Par1}\ \frac{P \xrightarrow[N]{E} P'}{P\,|\,Q \xrightarrow[N]{E\setminus\overline{\mathbb{I}}(Q)} P'\,|\,Q}\ N \cap \overline{\mathbb{I}}(Q) = \emptyset
$$

$$
\mathsf{En}\ \frac{P \xrightarrow[N]{E} P'}{P \rhd_\sigma Q \xrightarrow[N]{E} P' \rhd_\sigma Q}
\qquad
\mathsf{Sum2}\ \frac{Q \xrightarrow[N]{E} Q'}{P+Q \xrightarrow[N]{E} Q'}
\qquad
\mathsf{Par2}\ \frac{Q \xrightarrow[N]{E} Q'}{P\,|\,Q \xrightarrow[N]{E\setminus\overline{\mathbb{I}}(P)} P\,|\,Q'}\ N \cap \overline{\mathbb{I}}(P) = \emptyset
$$

$$
\mathsf{Dis1}\ \frac{P \xrightarrow[N]{E} P'}{P \rhd Q \xrightarrow[N]{E} P' \rhd_\sigma Q}
\qquad
\mathsf{Dis2}\ \frac{Q \xrightarrow[N]{E} Q'}{P \rhd Q \xrightarrow[N]{E} Q'}
\qquad
\mathsf{Res}\ \frac{P \xrightarrow[N]{E} P'}{P \setminus L \xrightarrow[N\setminus L]{E} P' \setminus L}\ E \cap L = \emptyset
$$

The semantics for action transitions, depicted in Table 4 (lower part), is set up such that $P \xrightarrow[N]{E} P'$ means: P can evolve to P', if the environment offers communications on all ports in E, but none on any port in N. More precisely, process $\langle E,N\rangle.P$ may engage in input action $\langle E,N\rangle$ and then behave like P. The *summation operator* $+$ denotes *nondeterministic choice*, i.e., process $P+Q$ may either behave like P or Q. Process $P\,|\,Q$ stands for the *parallel composition* of P and Q according to an interleaving semantics with synchronization on common ports. Rule Par1 describes the interaction of process P with its environment Q. If P can engage in a transition labeled by $\langle E,N\rangle$ to P', then P and Q synchronize on the events in $E \cap \overline{\mathbb{I}}(Q)$, provided that Q does not offer a communication on a port in N, i.e., $N \cap \overline{\mathbb{I}}(Q) = \emptyset$ holds. In this case, $P\,|\,Q$ can engage in a transition labeled by $\langle E \setminus \overline{\mathbb{I}}(Q), N\rangle$ to $P'\,|\,Q$. Rule Par2 deals with the symmetric case, where the roles of P and Q are interchanged. The semantics of the *disabling* and *enabling operators* are tightly connected. Process $P \rhd Q$ may behave as Q, thereby permanently disabling P, or as $P \rhd_\sigma Q$. In the latter case only P may proceed, and Q is disabled until the next clock tick arrives. This allows for modeling Statecharts or-states, where process P is on a lower level than Q. The *restriction operator* $\setminus L$ encapsulates all ports in L. Rule Res states that process $P \setminus L$ can only engage in an action transition labeled by $\langle E, N\rangle$, if there is no event in E, which is restricted by L. Moreover, the events in L may be eliminated from N. Finally, process variable X, where $X \stackrel{\text{def}}{=} P$, is identified with a process that behaves as a distinguished solution of the equation $X = P$.

Table 5. Operational semantics (clock transitions)

$$\mathsf{tNil}\ \frac{-}{\mathbf{0} \stackrel{\sigma}{\longrightarrow} \mathbf{0}} \qquad \mathsf{tAct}\ \frac{-}{\langle E,N\rangle.P \stackrel{\sigma}{\longrightarrow} \langle E,N\rangle.P}\ \langle E,N\rangle \neq \bullet \qquad \mathsf{tOut}\ \frac{-}{[E]\sigma(P) \stackrel{\sigma}{\longrightarrow} P}$$

$$\mathsf{tPar}\ \frac{P \stackrel{\sigma}{\longrightarrow} P' \quad Q \stackrel{\sigma}{\longrightarrow} Q'}{P \,|\, Q \stackrel{\sigma}{\longrightarrow} P' \,|\, Q'}\ \bullet \notin \mathrm{I}(P \,|\, Q) \qquad \mathsf{tSum}\ \frac{P \stackrel{\sigma}{\longrightarrow} P' \quad Q \stackrel{\sigma}{\longrightarrow} Q'}{P + Q \stackrel{\sigma}{\longrightarrow} P' + Q'}$$

$$\mathsf{tDis}\ \frac{P \stackrel{\sigma}{\longrightarrow} P' \quad Q \stackrel{\sigma}{\longrightarrow} Q'}{P \rhd Q \stackrel{\sigma}{\longrightarrow} P' \rhd Q'} \qquad \mathsf{tEn}\ \frac{P \stackrel{\sigma}{\longrightarrow} P'}{P \rhd_\sigma Q \stackrel{\sigma}{\longrightarrow} P' \rhd Q}$$

$$\mathsf{tRes}\ \frac{P \stackrel{\sigma}{\longrightarrow} P'}{P \setminus L \stackrel{\sigma}{\longrightarrow} P' \setminus L}\ \bullet \notin \mathrm{I}(P \setminus L) \qquad \mathsf{tRec}\ \frac{P \stackrel{\sigma}{\longrightarrow} P'}{X \stackrel{\sigma}{\longrightarrow} P'}\ X \stackrel{\text{def}}{=} P$$

The operational rules for clock transitions deal with the maximal progress assumption, i.e., if $\bullet \in \mathrm{I}(P) := \{\langle E, N\rangle \,|\, \exists P'.\, P \xrightarrow[N]{E} P'\}$, then a clock tick σ is inhibited. The reason that transitions other than labeled by $\bullet$ do not have pre-emptive power is that these only indicate the *potential* of progress, whereas $\bullet$ denotes *real* progress in our framework. Rule tNil states that inaction process $\mathbf{0}$ can idle forever. Similarly, process $\langle E, N\rangle.P$ may idle for clock σ, whenever $\langle E, N\rangle \neq \bullet$. The *signal operator* in $[E]\sigma(P)$, which offers communications on the ports in E to its environment, disappears as soon as the next clock tick arrives and, thereby, enables P. Time has to proceed equally on both sides of summation, parallel composition, and disabling, i.e., $P + Q$, $P \,|\, Q$, and $P \rhd Q$ can engage in a clock transition if and only if both P and Q can. The side condition of Rule tPar implements maximal progress and states that there is no pending communication between P and Q. The reason for the side condition in Rule tRes is that the restriction operator may turn observable input actions into the internal, unobservable input action $\bullet$ (cf. Rule tRes) and, thereby, may pre-empt the considered clock transition. Finally, Rule tEn states that a clock tick switches the enabling operator to the disabling operator.

The operational semantics for SPL possesses several pleasant algebraic properties which are known from various timed process algebras [11, 28], such as (i) the *idling* property, i.e., $\bullet \notin \mathrm{I}(P)$ implies $\exists P'.\, P \xrightarrow{\sigma} P'$, for all $P \in \mathcal{P}$, (ii) the *maximal progress* property, i.e., $\exists P'.\, P \xrightarrow{\sigma} P'$ implies $\bullet \notin \mathrm{I}(P)$, for all $P \in \mathcal{P}$, and (iii) the *time determinacy* property, i.e., $P \xrightarrow{\sigma} P'$ and $P \xrightarrow{\sigma} P''$ implies $P' = P''$, for all $P, P', P'' \in \mathcal{P}$. Moreover, the summation and parallel operators are *associative* and *commutative*. The well-known *behavioral equivalence* bisimulation [21] may be adapted to cater for SPL as follows. Other work can be used for establishing that it is a well-defined *congruence* for SPL [27].

Definition 1 (Bisimulation). Bisimulation equivalence, $\sim\, \subseteq \mathcal{P} \times \mathcal{P}$, *is the largest symmetric relation such that for* $P \sim Q$ *the following conditions hold.*

1. $\overline{\mathbb{I}}(P) \subseteq \overline{\mathbb{I}}(Q)$ *2. If* $P \xrightarrow[N]{E} P'$ *then* $\exists Q' \in \mathcal{P}.\, Q \xrightarrow[N]{E} Q'$ *and* $P' \sim Q'$.

4 Embedding of Statecharts

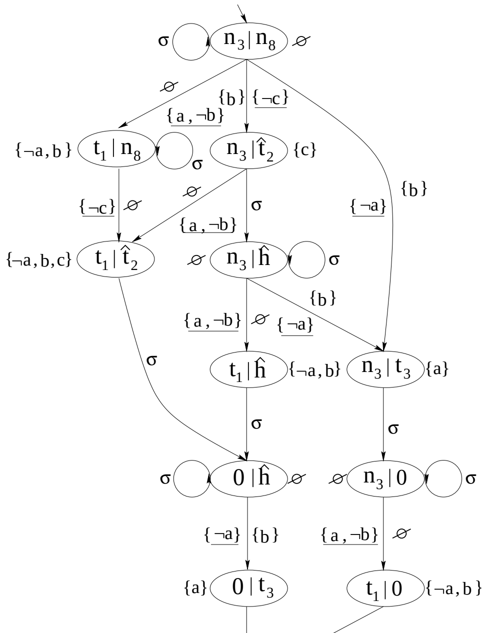
In this section we present an embedding of Statecharts in SPL, which is a mapping $[\![\cdot]\!]$ from Statecharts terms to processes defined by (mutually recursive) equations. Although SPL's semantics is defined on a "micro-step level," SPL allows us to encode the synchrony hypothesis of Statecharts by using maximal progress. More precisely, a macro step in Statecharts semantics corresponds to a sequence of SPL action transitions which is enclosed by clock transitions. These sequences implicitly contain the causal order inherent in a Statecharts macro step. Formally, we choose $\Pi \cup \neg\Pi$ for the set Λ of ports and $\mathcal{N} \cup \{\hat{n} \,|\, n \in \mathcal{N}\} \cup \mathcal{T}$ for the set $\mathcal{V}$ of process variables. We define the embedding $[\![\cdot]\!]$ inductively along the structure of Statecharts terms, where $\sum$ is the indexed version of $+$ satisfying $\sum_{i\in\emptyset} P_i := \mathbf{0}$.

1. If $s = [n]$, then $[\![s]\!] := n$ where $n \overset{\text{def}}{=} \hat{n} \overset{\text{def}}{=} \mathbf{0}$.
2. If $s = [n : (s_1, \ldots, s_k); l; T]$ and $n_i = \mathsf{root}(s_i)$, for $1 \le i \le k$, then $[\![s]\!] := n$, where $n \overset{\text{def}}{=} \hat{n}_l$ and $\hat{n}_i \overset{\text{def}}{=} n_i \rhd \sum\{\{\![t]\!\} \,|\, t \in T \text{ and } \mathsf{out}(t) = s_i\}$, together with the equations of $[\![s_1]\!], \ldots, [\![s_k]\!]$. Please see below for the translation $\{\![t]\!\}$ of t.
3. If $s = [n : (s_1, \ldots, s_k)]$, then $[\![s]\!] := n$ and $n \overset{\text{def}}{=} \hat{n} \overset{\text{def}}{=} \mathsf{root}(s_1) \,|\, \cdots \,|\, \mathsf{root}(s_k)$, together with the equations of $[\![s_1]\!], \ldots, [\![s_k]\!]$.

Semantically, a basic state corresponds to inaction process $\mathbf{0}$, whereas an or-state can either behave according to the embedding of the currently active state s_l, or it may exit s_l by engaging in a transition $t \in T$ with $\mathsf{out}(t) = s_l$. Observe that an or-state is mapped using the disabling operator. The translation of an and-state maps its component states to the parallel composition of the processes resulting from the translations of each of these states. The interesting part of the definition of $[\![\cdot]\!]$ is the translation $\{\![t]\!\}$ of a transition $\langle t, i, E, A, j\rangle$. In the following, E' stands for $E \cap \Pi$ and N' for $\neg(E \cap \neg\Pi) \cup \neg A$. We define $\{\![t]\!\} := \langle E', N'\rangle.t$ where $t \overset{\text{def}}{=} [A \cup (E \cap \neg\Pi)]\sigma(\hat{n}_j)$. The translation splits a transition $\langle t, i, E, A, j\rangle$ in two parts, one handling its trigger E and one executing its action A. In order for t to trigger, all positive events in E must be offered by the environment, and all negative events in E must be absent. However, there is one more thing we have to obey: *global consistency*. Especially, we must ensure that there is no previous transition t' in the same macro step, which has fired because of the absence of an event in A. Therefore, to prevent t from triggering, we include a distinguished event $\neg e$, where $e \in A$, in the set N' of $\{\![t]\!\}$, and we make sure that $\neg e$ is offered when t' triggers. Hence, $\{\![t]\!\}$ can evolve via a SPL transition labeled by $\langle E', N' \rangle$ to process t, whenever the trigger of t is satisfied according to Statecharts semantics and whenever global consistency is preserved. Process t signals that transition t has fired by offering the events in A as well as the already mentioned negated events $\neg e$ for $\neg e \in E \cap \neg\Pi$. These events are offered until the current macro step is completed, i.e., until a clock transition is executed. Thus, SPL's two-level semantics of action and clock transitions allows for broadcasting events using SPL's synchronization mechanism and SPL's maximal progress assumption.

Table 6. Embedding of the Example Statechart

$$
\begin{aligned}
[\![s_9]\!] &= n_9 \stackrel{\text{def}}{=} n_3 \mid n_8 \\
[\![s_3]\!] &= n_3 \stackrel{\text{def}}{=} \hat{n}_1 \\
&\quad\ \hat{n}_1 \stackrel{\text{def}}{=} n_1 \rhd \langle \emptyset, \{a, \neg b\} \rangle.t_1 \\
&\quad\ t_1 \stackrel{\text{def}}{=} [\{b, \neg a\}]\sigma(\hat{n}_2) \\
[\![s_1]\!] &= n_1 \stackrel{\text{def}}{=} \hat{n}_1 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_2]\!] &= n_2 \stackrel{\text{def}}{=} \hat{n}_2 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_8]\!] &= n_8 \stackrel{\text{def}}{=} \hat{n}_6 \\
&\quad\ \hat{n}_6 \stackrel{\text{def}}{=} n_6 \rhd \langle \{b\}, \{\neg a\} \rangle.t_3 \\
&\quad\ t_3 \stackrel{\text{def}}{=} [\{a\}]\sigma(\hat{n}_7) \\
[\![s_6]\!] &= n_6 \stackrel{\text{def}}{=} \hat{n}_4 \\
&\quad\ \hat{n}_4 \stackrel{\text{def}}{=} n_4 \rhd \langle \{b\}, \{\neg c\} \rangle.t_2 \\
&\quad\ t_2 \stackrel{\text{def}}{=} [\{c\}]\sigma(\hat{n}_5) \\
[\![s_4]\!] &= n_4 \stackrel{\text{def}}{=} \hat{n}_4 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_5]\!] &= n_5 \stackrel{\text{def}}{=} \hat{n}_5 \stackrel{\text{def}}{=} \mathbf{0} \\
[\![s_7]\!] &= n_7 \stackrel{\text{def}}{=} \hat{n}_7 \stackrel{\text{def}}{=} \mathbf{0}
\end{aligned}
$$



We now return to our introductory example by presenting its formal translation to SPL in Table 6, left-hand side. The embedding's operational semantics is depicted on the right-hand side of Table 6, where $\hat{t}_2 \stackrel{\text{def}}{=} t_2 \rhd_\sigma \langle \{b\}, \{\neg a\} \rangle.t_3$, and $\hat{h} \stackrel{\text{def}}{=} \mathbf{0} \rhd \langle \{b\}, \{\neg a\} \rangle.t_3$. Moreover, the initial output action set $\overline{\mathbb{I}}(P)$, for some $P \in \mathcal{P}$, is denoted next to the ellipse symbolizing state P, and the sets N' appearing in the label of transitions are underlined in order to distinguish them from the sets E'. Let us have a closer look at the leftmost path of the transition system, passing the states $(n_3 \,|\, n_8)$, $(t_1 \,|\, n_8)$, $(t_1 \,|\, \hat{t}_2)$, $(\mathbf{0} \,|\, \hat{h})$, $(\mathbf{0} \,|\, t_3)$, and $(\mathbf{0} \,|\, \mathbf{0})$. The first three states are separated from the last three states by a clock transition. Hence, the considered sequence corresponds to two "potential" macro steps. We say "potential," since macro steps only emerge when composing our Statecharts embedding with an environment which triggers macro steps. The events needed to trigger the transitions and the actions produced by them can be extracted from a macro-step sequence as follows. For obtaining the trigger, consider all transition labels $\langle E, N \rangle$ occurring in the sequence, add up all events in components E, and include the negations of all positive events in components N. Regarding the generated actions, consider the set of positive events in the initial output action sets of the states preceding the clock transition which signals the end of the macro step. Thus, the first potential macro step of the example sequence is triggered by $\neg a$ and produces events b and c, whereas the second is triggered by b and produces a. The state names along a sequence also indicate, which transitions have fired. More precisely, whenever a state includes a variable $t \in \mathcal{T}$ at its top-level, transition t participates in the current macro step. Thus, for the first potential macro step transitions t_1 and t_2 are chosen, whereas the second consists of transition t_3 only. Note that t_3 is not enabled in states $(t_1 \,|\, n_8)$ or $(t_1 \,|\, \hat{t}_2)$, since event $\neg a$ is in their initial output action sets and $a \in \mathsf{act}(t_3)$. Hence, our embedding respects global consistency, which prohibits t_1 and t_3 to occur in the same macro step.

5 Semantic Correspondence

For formalizing the semantic relation between Statecharts terms and their SPL embeddings, we define a notion of SPL *macro steps* by combining several transitions to a single step, as outlined in the previous section. We write $P \overset{E}{\underset{A}{\Longrightarrow}} P'$ if $\exists P'' \in \mathcal{P}.\ (\mathsf{Env}_E \,|\, P) \setminus \Lambda \overset{\emptyset}{\underset{\emptyset}{\longrightarrow}}{}^{*} (\mathsf{Env}_E \,|\, P'') \setminus \Lambda \overset{\sigma}{\longrightarrow} (\mathbf{0} \,|\, P') \setminus \Lambda$ and $\overline{\mathbb{I}}(P'') = A$, where $\mathsf{Env}_E \overset{\text{def}}{=} [E]\sigma(\mathbf{0})$. Intuitively, P is placed in context $(\mathsf{Env}_E \,|\, \cdot) \setminus \Lambda$, where Env_E models a single-step environment which offers the events in E until clock tick σ occurs. The following relation, which we refer to as *step correspondence*, provides the formal foundation for relating Statecharts and SPL macro steps.

Definition 2 (Step Correspondence). *A relation $\mathcal{R} \subseteq \mathit{SC} \times \mathcal{P}$ is a* step correspondence *if for all $\langle s, P \rangle \in \mathcal{R}$ and $E, A \subseteq \Pi$ the following conditions hold:*

1. *$\forall s' \in \mathit{SC}.\ s \overset{E}{\underset{A}{\Longrightarrow}} s'$ implies $\exists P' \in \mathcal{P}.\ P \overset{E}{\underset{A}{\Longrightarrow}} P'$ and $\langle s', P' \rangle \in \mathcal{R}$.*
2. *$\forall P' \in \mathcal{P}.\ P \overset{E}{\underset{A}{\Longrightarrow}} P'$ implies $\exists s' \in \mathit{SC}.\ s \overset{E}{\underset{A}{\Longrightarrow}} s'$ and $\langle s', P' \rangle \in \mathcal{R}$.*

s is step-correspondent *to P, if $\langle s, P \rangle \in \mathcal{R}$ for some step correspondence $\mathcal{R}$.*

Theorem 1 (Embedding). *Every $s \in \mathit{SC}$ is step-correspondent to $[\![s]\!]$.*

We close this section by returning to the behavioral relation $\sim$.

Theorem 2 (Preservation). *Let $P, Q \in \mathcal{P}$ such that $P \sim Q$, and suppose that $P \overset{E}{\underset{A}{\Longrightarrow}} P'$. Then $\exists Q' \in \mathcal{P}.\ Q \overset{E}{\underset{A}{\Longrightarrow}} Q'$ and $P' \sim Q'$.*

Now, we can state our desired result, namely that the behavioral equivalence *bisimulation*, when applied to SPL, preserves Statecharts semantics.

Corollary 1. *Let $E, A \subseteq \Pi$, $s \in \mathit{SC}$, and $P \in \mathcal{P}$ such that $[\![s]\!] \sim P$. Then*

1. $\forall s' \in \mathit{SC}.\ s \overset{E}{\underset{A}{\Longrightarrow}} s'$ *implies* $\exists P' \in \mathcal{P}.\ P \overset{E}{\underset{A}{\Longrightarrow}} P'$ *and* $[\![s']\!] \sim P'$.
2. $\forall P' \in \mathcal{P}.\ P \overset{E}{\underset{A}{\Longrightarrow}} P'$ *implies* $\exists s' \in \mathit{SC}.\ s \overset{E}{\underset{A}{\Longrightarrow}} s'$ *and* $[\![s']\!] \sim P'$.

6 Adaptability to Other Statecharts Variants

For Statecharts a variety of different semantics has been introduced in the literature [26]. In this section, we show how our approach can be adapted to these variants and, thereby, testify to its flexibility.

In the Statecharts variant examined in this paper, two features are left out which are often adopted in other variants. One feature concerns *inter-level transitions*, i.e., transitions which cross the "borderlines" of Statecharts states and, thus, permit a style of "goto"-programming. Unfortunately, when allowing inter-level transitions the syntax of Statecharts terms cannot be defined compositionally and, consequently, nor its semantics. The second feature left out is usually referred to as *state reference*, which permits the triggering of a transition to depend on the fact whether a certain parallel component is in a certain state. Such state references can be encoded in SPL's communication scheme by introducing special events in_n, for $n \in \mathcal{N}$, which are signaled by a process if it is in state n.

Another issue concerns the sensing of *internal* and *external events*. Usually, internal events are sensed within a macro step, but external events are not. Hence, events are *instantaneous*, i.e., an event exists only for the duration of the macro step under consideration. This is reflected in our signal operator which stops signaling events as soon as the next clock tick arrives. In the semantics of Statemate [8], an event is only sensed in the macro step following the one in which it was generated. This behavior can be encoded in our embedding by splitting every state $t \in \mathcal{T}$ into two states that are connected via a clock transition. The specific sensing of events in Statemate greatly simplifies the development of a *compositional* semantics [6].

The Statecharts concept of *negated events* forces transitions to be triggered only when certain events are absent. However, when permitting negated events in a macro-step semantics, one has to guarantee that the *effect of a transition is not contradictory to its cause.* Regarding this issue, one may distinguish two concepts: *global consistency* and *local consistency*. The first one prohibits a transition containing a negative trigger event $\neg e$ to be executed, if a micro step within the same macro step produces e. In our embedding, this is enforced by offering $\neg e$, whenever a transition triggers due to the absence of e. Moreover, $\neg e$ is included in the set of events which need to be absent in all Statecharts transitions producing e. When leaving out these events $\neg e$ in our embedding, we obtain the weaker notion of local consistency, i.e., once an event e is signaled in a micro-step, no following micro step of the same macro step may fire if its trigger contains $\neg e$. Local consistency implicitly holds in our embedding, since an event is always signaled until the next macro step begins.

In addition to encoding priorities between transitions via negated events, one may introduce an implicit priority mechanism along *state hierarchy*, as is done, e.g., in Statemate [10] but not in the Statecharts variant [17] considered in this paper. More precisely, a transition leaving an or-state may be given priority over any transition within this state, i.e., or-states can then be viewed as *pre-emptive interrupt* operators. SPL can easily be extended to capture this behavior.

7 Related Work

Achieving a compositional semantics for Statecharts is known to be a difficult task. The problems involved were systematically analyzed and investigated by Huizing and Gerth in the early nineties in the more general context of real-time reactive systems [15], for which three criteria have found to be desirable: (i) *responsiveness*, which corresponds to the synchrony hypothesis of Statecharts, (ii) *modularity*, which refers to the aspect of compositionality, and (iii) *causality*. Huizing and Gerth proved that these properties cannot be combined in a single-leveled semantics. In our approach the three properties hold on different levels: compositionality holds on the micro-step level – the level of SPL action transitions – whereas responsiveness and causality are guaranteed on the macro-step level – the level where sequences of SPL action transitions between global synchronizations, caused by clock ticks σ, are bundled together.

Uselton and Smolka [24, 25] and Levi [16] also focused on achieving a compositional semantics for Statecharts by referring to process algebras. In contrast to our approach, Uselton and Smolka's notion of transition system involves labels of the form $\langle E, \prec \rangle$, where E is a set of events, and $\prec$ is a transitive, irreflexive order on E encoding causality. Unfortunately, their semantics does not correspond, as intended, to the semantics of Pnueli and Shalev [22], as pointed out in [16, 17]. Levi repaired this shortcoming by modifying the domains of the arguments of $\prec$ to sets of events and by allowing empty steps to be represented explicitly.

Maggiolo-Schettini et al. considered a hierarchy of equivalences for Statecharts and studied congruence properties with respect to Statecharts operators [17]. For this purpose, they defined a compositional, operational macro-step semantics of Statecharts, which slightly differs from the one of Pnueli and Shalev, since it does not allow the step-construction function to fail. Their semantics is also expressed in terms of labeled transition systems, where labels consist of four-tuples which include information about causal orderings, global consistency, and negated events. The framework of Maggiolo-Schettini et al. serves well for the purpose of studying certain algebraic properties of equivalences on Statecharts, such as fully-abstractness results and axiomatizations [14, 15].

Another popular design language with a visual appeal like Statecharts and, moreover, a solid algebraic foundation is *Argos* [18]. However, the semantics of Argos – defined via SOS-rules as labeled transition systems – significantly differs from classical Statecharts semantics. For example, Argos is deterministic, abstracts from "non-causal" Statecharts by semantically identifying them with a *failure* state, and allows a single parallel component to fire more than once within a macro step.

Interfacing Statemate [10] to verification tools is a main objective in [19, 20]. The former work formalizes Statemate semantics in Z, while the latter work translates a subset of Statemate to the model-checking tool Spin [13].

8 Conclusions and Future Work

This paper presented a process-algebraic approach to defining a compositional semantics for Statecharts. Our technique translates Statecharts terms to terms in SPL which allows one to encode a "micro-step" semantics of Statecharts. The macro-step semantics may then be given in terms of a derived transition relation. We demonstrated the utility of our technique by formally embedding the Statecharts semantics of [17], which is a slight variant of Pnueli and Shalev's semantics [22], in SPL. Our approach also allows for interfacing Statecharts to existing verification tools and for the possibility of lifting behavioral equivalences from process algebras to Statecharts. We illustrated the viability of this last point by showing that bisimulation equivalence, which is a congruence for SPL, preserves Statecharts macro-step semantics.

Regarding future work, we plan to continue our investigation of behavioral equivalences for Statecharts in general, and "weak" equivalences in particular, by studying them for SPL. It may also be interesting to characterize the "Statecharts sub-algebra" of SPL. Moreover, we intend to implement SPL and our embedding in the *Concurrency Workbench of North Carolina* (CWB-NC) [5].

We would like to thank Peter Kelb, Ingolf Krüger, Michael Mendler, and the anonymous referees for many valuable comments and suggestions.

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, vol. 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[2] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[3] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[4] R. Cleaveland, G. Lüttgen, and V. Natarajan. Priority in process algebra. In *Handbook of Process Algebra*. Elsevier, 1999.

[5] R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In *CAV '96*, vol. 1102 of *LNCS*, pages 394–397, 1996. Springer Verlag.

[6] W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In *Compositionality: The Significant Difference*, vol. 1536 of *LNCS*, pages 186–238, 1997. Springer Verlag.

[7] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[8] D. Harel and A. Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering*, 5(4):293–333, 1996.

[9] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the formal semantics of Statecharts. In *LICS '87*, pages 56–64, 1987. IEEE Computer Society Press.

[10] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw Hill, 1998.

[11] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[12] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[13] G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[14] J.J.M. Hooman, S. Ramesh, and W.-P. de Roever. A compositional axiomatization of Statecharts. *Theoretical Computer Science*, 101:289–335, 1992.

[15] C. Huizing. *Semantics of Reactive Systems: Comparison and Full Abstraction*. PhD thesis, Eindhoven University of Technology, 1991.

[16] F. Levi. *Verification of Temporal and Real-Time Properties of Statecharts*. PhD thesis, University of Pisa-Genova-Udine, 1997.

[17] A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *CONCUR '96*, vol. 1119 of *LNCS*, pages 687–702, 1996. Springer Verlag.

[18] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *CONCUR '92*, vol. 630 of *LNCS*, pages 550–564, 1992. Springer Verlag.

[19] E. Mikk, Y. Lakhnech, C. Petersohn, and M. Siegel. On formal semantics of Statecharts as supported by STATEMATE. In *Second BCS-FACS Northern Formal Methods Workshop*, 1997. Springer Verlag.

[20] E. Mikk, Y. Lakhnech, M. Siegel, and G.J. Holzmann. Verifying Statecharts with Spin. In *WIFT '98*, 1998. IEEE Computer Society Press.

[21] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[22] A. Pnueli and M. Shalev. What is in a step: On the semantics of Statecharts. In *TACS '91*, vol. 526 of *LNCS*, pages 244–264, 1991. Springer Verlag.

[23] P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts*. PhD thesis, Munich University of Technology, 1998.

[24] A.C. Uselton and S.A. Smolka. A compositional semantics for Statecharts using labeled transition systems. In *CONCUR '94*, vol. 836 of *LNCS*, pages 2–17, 1994. Springer Verlag.

[25] A.C. Uselton and S.A. Smolka. A process-algebraic semantics for Statecharts via state refinement. In *PROCOMET '94*. North Holland/Elsevier, 1994.

[26] M. v.d. Beeck. A comparison of Statecharts variants. In *FTRTFT '94*, vol. 863 of *LNCS*, pages 128–148, 1994. Springer Verlag.

[27] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2:274–302, 1995.

[28] W. Yi. CCS + time = an interleaving model for real time systems. In *ICALP '91*, vol. 510 of *LNCS*, pages 217–228, 1991. Springer Verlag.