NASA/CR-1999-209713
ICASE Report No. 99-42

Statecharts via Process Algebra

Gerald Lüttgen
ICASE, Hampton, Virginia

Michael von der Beeck
Munich University of Technology, München, Germany

Rance Cleaveland
State University of New York at Stony Brook, Stony Brook, New York

October 1999

NASA/CR-1999-209713
ICASE Report No. 99-42

Statecharts via Process Algebra

Gerald Lüttgen
ICASE, Hampton, Virginia

Michael von der Beeck
Munich University of Technology, München, Germany

Rance Cleaveland
State University of New York at Stony Brook, Stony Brook, New York

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

October 1999

Available from the following:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

STATECHARTS VIA PROCESS ALGEBRA*

GERALD LÜTTGEN[†], MICHAEL VON DER BEECK[‡], AND RANCE CLEAVELAND[§]

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, a simple *process algebra*, called *Statecharts Process Language* (SPL), is presented, which is expressive enough for encoding Statecharts in a structure-preserving and semantics-preserving manner. It is established that the behavioral relation *bisimulation*, when applied to SPL, preserves Statecharts semantics.

Key words. bisimulation, compositionality, operational semantics, process algebra, Statecharts

Subject classification. Computer Science

1. Introduction. *Statecharts* is a visual language for specifying the behavior of *reactive systems* [7]. The language extends the traditional 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 the semantics of the language has proved extremely challenging, with a variety of proposals [8, 9, 18, 19, 21, 28, 29] being offered for several dialects [34] of the language. While the research results have yielded insight into different aspects of the notation, no definitive account has emerged. This has an obviously undesirable practical ramification; tool builders for Statecharts must resort to *ad hoc* decisions in their implementations of semantically-based tools, such as model checkers [16, 23], and this means that designs developed by engineers have a meaning that may vary from implementation to implementation.

The semantic subtlety of Statecharts arises from the language's capability for defining transitions whose enablement 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 comes to a halt, one execution step, a so-called *macro step*, is complete. Technically, 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

*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.

[†] ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-2199, USA, e-mail: luettgen@icase.edu.

[‡] Department of Computer Science, Munich University of Technology, Arcisstr. 21, D-80290 München, Germany, e-mail: beeck@in.tum.de.

[§] Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA, e-mail: rance@cs.sunysb.edu.

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 [28], satisfies the synchrony hypothesis and causality, but is not compositional. Other approaches [17, 18, 31] 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, 24] 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, such as *bisimulation* [24, 25]. Many of the behavioral aspects of Statecharts have also been studied in the setting of process algebra. For example, the synchrony hypothesis is related to the *maximal progress assumption* developed in *timed* process algebras [11, 35]. 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 traditional SOS-style presentations of process algebras capture a notion of "causality" *à la* Statecharts. 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 involves synthesizing 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], in that it extends Milner's CCS [24] 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 [32]. The operational semantics of SPL uses SOS rules [26] 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. We connect SPL with Statecharts by embedding the variant of the language considered by Maggiolo-Schettini et al. in [18]. More precisely, we define a compositional translation from Statecharts to SPL that preserves the macro-step semantics of the former. This result crucially depends 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 (see 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.

The remainder of this paper is organized as follows. The next section gives a brief introduction to Statecharts, while Section 3 defines the process algebra SPL. Sections 4 and 5 formalize our embedding of Statecharts semantics in SPL and present our main technical results, respectively. Section 6 shows the flexibility of our approach by examining its adaptability to other Statechart variants. Related work is discussed in Section 7. Finally, Section 8 gives our conclusions and directions for future research.

2. Statecharts.

Statecharts is a specification language for *reactive systems* [27], 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 and can be executed; it fires, thereby producing the events in the label's action. Concurrency is achieved by allowing complex 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. Concurrency and hierarchy are especially important concepts, since they allow for *bottom-up* and *top-down* specifications of systems.

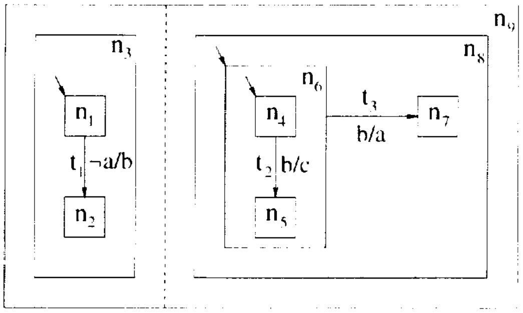


FIG. 2.1. *Example Statechart*

As an example, consider the Statechart depicted in Figure 2.1. 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 also 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.

It should be mentioned that the variant of Statecharts considered here does not include "features" present in some other variants. In particular, we prohibit *interlevel transitions*, i.e., transitions crossing borderlines of states, and triggers of the form in_n, where n is the name of a state. Moreover, state hierarchy does not impose implicit priorities to transitions, where transitions on higher levels of the hierarchy have precedence over transitions on lower levels; e.g., transition t_3 does not have priority over transition t_2 in our example. The impact of altering our approach to accommodate these concepts is discussed in Section 6.

2.1. Statecharts Terms.

For our purposes, it is convenient to represent Statecharts not visually but by terms. This is also done in related work [17, 18, 31], and our approach closely follows the one described in [18]. 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 $\neg\neg e =_{\mathrm{df}} e$. Finally, we write $\neg E$ for $\{\neg e \,|\, e \in E\}$, if $E \subseteq \Pi \cup \{\neg e \,|\, e \in \Pi\}$. 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 $n \in \mathcal{N}$, $s_1, \dots, s_k$ are Statecharts terms for $k > 0$, $T \subseteq \mathcal{T} \times \{1, \dots, k\} \times 2^{\Pi \cup \neg\Pi} \times 2^{\Pi} \times \{1, \dots, k\}$, and $1 \leq l \leq k$, then $s = [n : (s_1, \dots, s_k); l; T]$ is a Statecharts term. Intuitively, $s_1, \dots, s_k$ are the sub-states of s, and T is the set of transitions between these states. The Statechart s_1 is the default state of s, while s_l is the state that is currently active; initially, $l = 1$.
3. **And-state:** If $n \in \mathcal{N}$, and if $s_1, \dots, s_k$ are Statecharts terms for $k > 0$, then $s = [n : (s_1, \dots, s_k)]$ is a Statecharts term.

We refer to n as the *root* of s and write $\mathsf{root}(s) =_{\mathrm{df}} n$. If $\hat{t} = \langle t, i, E, A, j \rangle \in T$ is a transition of or-state $[n : (s_1, \dots, s_k); l; T]$, then we define $\mathsf{name}(\hat{t}) =_{\mathrm{df}} t$, $\mathsf{out}(\hat{t}) =_{\mathrm{df}} s_i$, $\mathsf{ev}(\hat{t}) =_{\mathrm{df}} E$, $\mathsf{act}(\hat{t}) =_{\mathrm{df}} A$, and $\mathsf{in}(\hat{t}) =_{\mathrm{df}} s_j$.

TABLE 2.1

States and transitions of Statecharts terms

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

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 name. For convenience, we often 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 2.1, where $\vec{s} = (s_1, \dots, s_k)$. Finally, let us return to our example Statechart in Figure 2.1 and present it as a Statecharts term $s_9 \in \mathsf{SC}$. For our framework, we choose $\Pi =_{\mathrm{df}} \{a, b, c\}$, $\mathcal{N} =_{\mathrm{df}} \{n_1, n_2, \dots, n_9\}$, and $\mathcal{T} =_{\mathrm{df}} \{t_1, t_2, t_3\}$.

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

2.2. Statecharts Semantics. In the remainder of this section, we formally present the semantics of Statecharts terms as is defined in [18], which is a slight variant of the "traditional" semantics proposed by Pnueli and Shalev [28]. More precisely, this semantics differs from [28] 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 $\mathsf{config}(s)$ of the names of those states which are currently active [28]. We define $\mathsf{config}(s)$ along the structure of s: (i) $\mathsf{config}([n]) =_{\mathrm{df}} \{n\}$, (ii) $\mathsf{config}([n : (s_1, \dots, s_k); l; T]) =_{\mathrm{df}} \{n\} \cup \mathsf{config}(s_l)$, and (iii) $\mathsf{config}([n : (s_1, \dots, s_k)]) =_{\mathrm{df}} \{n\} \cup \bigcup\{\mathsf{config}(s_i) \mid 1 \leq i \leq k\}$. 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) $\mathsf{default}([n]) =_{\mathrm{df}} [n]$, (ii) $\mathsf{default}([n : (s_1, \dots, s_k); l; T]) =_{\mathrm{df}} [n : (\mathsf{default}(s_1), \dots, \mathsf{default}(s_k)); 1; T]$, and (iii) $\mathsf{default}([n : (s_1, \dots, s_k)]) =_{\mathrm{df}} [n : (\mathsf{default}(s_1), \dots, \mathsf{default}(s_k))]$. 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

TABLE 2.2

Step-construction function

```
function $step\text{-}construction(s, E)$;  var $T := \emptyset$;
  while $T \subset \mathsf{enabled}(s, E, T)$ do choose $t \in \mathsf{enabled}(s, E, T) \setminus T$;  $T := T \cup \{t\}$ od;
return $T$
```

TABLE 2.3

Function update

$$\mathsf{update}([n], T') =_{\mathrm{df}} [n] \qquad \mathsf{update}([n : \vec{s}], T') =_{\mathrm{df}} [n : (\mathsf{update}(s_1, T_1), \ldots, \mathsf{update}(s_k, T_k))]$$

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

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 the notions of *triggered* and *compatible*.

A transition $t \in \mathsf{trans}(s)$ is *consistent* with $T \subseteq \mathsf{trans}(s)$, in signs $t \in \mathsf{consistent}(s, T)$, if t is not in the same parallel component as any transition in T. Formally,

$$\mathsf{consistent}(s, T) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \forall t' \in T.\, t \bot_s t'\}\,. \tag{2.1}$$

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,

$$\mathsf{compatible}(s, T) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \forall t' \in T.\, \mathsf{act}(t) \cap \neg\mathsf{ev}(t') = \emptyset\} \tag{2.2}$$

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,

$$\mathsf{relevant}(s) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \mathsf{root}(\mathsf{out}(t)) \in \mathsf{config}(s)\} \tag{2.3}$$

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,

$$\mathsf{triggered}(s, E) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \mid \mathsf{ev}(t) \cap \Pi \subseteq E \text{ and } \neg(\mathsf{ev}(t) \cap \neg\Pi) \cap E = \emptyset\} \tag{2.4}$$

Finally, a transition t is *enabled* in configuration s regarding a set E of events and a set T of transitions, if $t \in \mathsf{enabled}(s, E, T)$, where

$$\mathsf{enabled}(s, E, T) =_{\mathrm{df}} \mathsf{relevant}(s) \cap \mathsf{consistent}(s, T) \cap \mathsf{triggered}(s, E \cup \bigcup_{t \in T} \mathsf{act}(t)) \cap \mathsf{compatible}(s, T) \tag{2.5}$$

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 2.2. This function is adopted from [18], where also its soundness and completeness relative to the classical approach via the notion of ***inseparability*** of transitions [28] 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 $\mathsf{step}(s, E) \subseteq 2^{\mathcal{T}}$. For a set $T \in \mathsf{step}(s, E)$. Statecharts term s may evolve in a (single) macro step to term $s' =_{\mathrm{df}} \mathsf{update}(s, T)$ when triggered by the environment events in E and, thereby, produce the events in $A =_{\mathrm{df}} \bigcup\{\mathsf{act}(t) \,|\, t \in T\}$. We denote this macro step by $s \overset{E}{\underset{A}{\Longrightarrow}} s'$. The function update is defined in Table 2.3, where $\vec{s} =_{\mathrm{df}} (s_1 \ldots, s_k)$ and $T_i =_{\mathrm{df}} T' \cap \mathsf{trans}(s_i)$, 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 2.3 cannot occur in our context. Intuitively, $\mathsf{update}(s, T)$, for $T \subseteq \mathsf{trans}(s)$. re-defines the active states of s when the transitions in T are executed.

2.3. Compositional Characterization of enabled. We conclude this section about Statecharts with a compositional characterization of $\mathsf{enabled}$, which will be needed later in the paper. For this purpose, we augment $\mathsf{enabled}$ with a fourth argument $A \subseteq \Pi$ which contains the events that must not be generated by enabled transitions. Formally, we define $\mathsf{enabled} : \mathsf{SC} \times 2^{\Pi} \times 2^{\mathrm{II}} \times 2^{\mathcal{T}} \longrightarrow 2^{\mathcal{T}}$ by

$$\mathsf{enabled}(s, E, A, T) =_{\mathrm{df}} \mathsf{relevant}(s) \cap \mathsf{consistent}(s, T) \cap \mathsf{triggered}(s, E \cup \bigcup_{t \in T} \mathsf{act}(t)) \cap \mathsf{compatible}(s, A, T)$$

where $\mathsf{compatible}(s, A, T) =_{\mathrm{df}} \{t \in \mathsf{trans}(s) \,|\, \mathsf{act}(t) \cap (A \cup \bigcup_{t' \in T} \neg(\mathsf{ev}(t') \cap \neg\Pi)) = \emptyset\}$. It is easy to see that the new definition of $\mathsf{enabled}$ extends the old one as follows: $\mathsf{enabled}(s, E, T) = \mathsf{enabled}(s, E, \emptyset, T)$. The extended version of $\mathsf{enabled}$ may now be compositionally characterized as follows.

PROPOSITION 2.1. *Let $s \in SC$, $E, A \subseteq \Pi$, and $T' \subseteq \mathcal{T}$.*

1. *If $s = [n]$ is a basic state, then $\mathit{enabled}(s, E, A, T') = \emptyset$.*
2. *If $s = [n : (s_1, \ldots, s_k); l; T]$ is an or-state, then $\mathit{enabled}(s, E, A, T') =$*

$$\begin{cases} \mathit{enabled}(s_l, E, A, T') \cup & \\ \{t \in T \,|\, \mathit{out}(t) = s_l,\ t \in \mathit{triggered}(s_l, E) \cap \mathit{compatible}(s_l, A, T')\} & \textit{if } T' = \emptyset \\ \mathit{enabled}(s_l, E, A, T') & \textit{if } \emptyset \neq T' \subseteq \mathit{trans}(s_l) \\ \{t' \,|\, t' \in \mathit{triggered}(s_l, E) \cap \mathit{compatible}(s_l, A, T')\} & \textit{if } \emptyset \neq T' = \{t'\} \subseteq T.\ \mathit{out}(t') = s_l \\ \emptyset & \textit{otherwise} \end{cases}$$

3. *If $s = [n : (s_1, \ldots, s_k)]$ is an and-state, then $\mathit{enabled}(s, E, A, T') = \bigcup_{1 \leq i \leq k} \mathit{enabled}(s_i, E_i, A_i, T_i)$, where $E_i =_{df} E \cup \bigcup\{\mathit{act}(t) \,|\, t \in T_j,\ j \neq i\}$. $A_i =_{df} A \cup \bigcup\{\neg(\mathit{ev}(t) \cap \neg\Pi) \,|\, t \in T_j,\ j \neq i\}$. and $T_i =_{df} T' \cap \mathit{trans}(s_i)$. for $1 \leq i \leq k$.*

The proof of this proposition can be done by induction on the structure of s.

3. Process-Algebraic Framework. In this section, we present our process-algebraic framework which is inspired by *timed process calculi*, such as Hennessy and Regan's TPL [11]. Our language, which we refer to as *Statecharts Process Language* (SPL), 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 in process algebras corresponds 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 that can be specified independently from each other, but are connected via the *maximal progress assumption* [11, 35]. Maximal progress implies that progress of time is determined by the *completion of internal computations* and, thus, mimics the synchrony hypothesis of Statecharts. 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, thus, leads to a compositional semantics for Statecharts, whose practicality does not suffer from complicated transition labels including causal orders [17, 18, 31].

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 in their *negated* form. The former feature implies that – in contrast to standard process algebras [1, 12, 24] – processes may be forced to synchronize on more than one event simultaneously, and the latter feature is similar to mechanisms for handling priority [4]. Moreover, our framework must include an operator similar to the *disabling operator* of LOTOS [3] for resembling state hierarchy [32]. Our Statecharts Process Language combines these well-known concepts in a single process algebra, which is expressive and flexible enough for embedding several Statecharts variants, as we will show below.

3.1. Syntax. Formally, let Λ be a countable set of *events* or *ports*, and let $\sigma \notin \Lambda$ be the distinguished *clock event* or *clock tick*. Based on Λ, we define *input actions* in SPL 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 $\bullet$. Moreover, we let $\mathcal{A}$ stand for the set of all input actions. In contrast to CCS [24], 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 $\langle E, N\rangle$ 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

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

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* $X \overset{\text{def}}{=} P$, 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* [24]. The symbol $\mathcal{P}$ denotes the set of all processes and is ranged over by P and Q. Finally, the operators $\rhd$ and $\rhd_\sigma$ – called *disabling* and *enabling* operator, respectively – allow us to model state hierarchy.

3.2. Operational Semantics. 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, $\longrightarrow \subseteq \mathcal{P} \times (\mathcal{A} \cup \{\sigma\}) \times \mathcal{P}$ 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 $P \xrightarrow[N]{E} P'$ instead of $\langle P, \langle E, N\rangle, P'\rangle \in \longrightarrow$ and $P \xrightarrow{\sigma} P'$ 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 3.2 and 3.3 using operational rules. In contrast to CCS [24], 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 3.1. Intuitively, $\overline{\mathbb{I}}(P)$ collects all events which are initially offered by P.

TABLE 3.1
Initial output action sets

$\overline{\mathbb{I}}([E]\sigma(P)) = E$	$\overline{\mathbb{I}}(P+Q) = \overline{\mathbb{I}}(P) \cup \overline{\mathbb{I}}(Q)$	$\overline{\mathbb{I}}(X) = \overline{\mathbb{I}}(P)$	where $X \overset{\text{def}}{=} P$
	$\overline{\mathbb{I}}(P \mid Q) = \overline{\mathbb{I}}(P) \cup \overline{\mathbb{I}}(Q)$	$\overline{\mathbb{I}}(P \setminus L) = \overline{\mathbb{I}}(P) \setminus L$	
	$\overline{\mathbb{I}}(P \rhd Q) = \overline{\mathbb{I}}(P) \cup \overline{\mathbb{I}}(Q)$	$\overline{\mathbb{I}}(P \rhd_\sigma Q) = \overline{\mathbb{I}}(P)$	

TABLE 3.2
Operational semantics (action transitions)

Act $\dfrac{-}{\langle E,N\rangle.P \xrightarrow[N]{E} P}$	Rec $\dfrac{P \xrightarrow[N]{E} P'}{X \xrightarrow[N]{E} P'}\; X \overset{\text{def}}{=} P$	Sum1 $\dfrac{P \xrightarrow[N]{E} P'}{P + Q \xrightarrow[N]{E} P'}$	Par1 $\dfrac{P \xrightarrow[N]{E} P'}{P \mid Q \xrightarrow[N]{E \setminus \overline{\mathbb{I}}(Q)} P' \mid Q}\; N \cap \overline{\mathbb{I}}(Q) = \emptyset$
	En $\dfrac{P \xrightarrow[N]{E} P'}{P \rhd_\sigma Q \xrightarrow[N]{E} P' \rhd_\sigma Q}$	Sum2 $\dfrac{Q \xrightarrow[N]{E} Q'}{P + Q \xrightarrow[N]{E} Q'}$	Par2 $\dfrac{Q \xrightarrow[N]{E} Q'.}{P \mid Q \xrightarrow[N]{E \setminus \overline{\mathbb{I}}(P)} P \mid Q'}\; N \cap \overline{\mathbb{I}}(P) = \emptyset$
	Dis1 $\dfrac{P \xrightarrow[N]{E} P'}{P \rhd Q \xrightarrow[N]{E} P' \rhd_\sigma Q}$	Dis2 $\dfrac{Q \xrightarrow[N]{E} Q'}{P \rhd Q \xrightarrow[N]{E} Q'}$	Res $\dfrac{P \xrightarrow[N]{E} P'}{P \setminus L \xrightarrow[N \setminus L]{E} P' \setminus L}\; E \cap L = \emptyset$

The operational semantics for action transitions is set up such that $P \xrightarrow[N]{E} P'$ may be read as follows: P can evolve to P' whenever 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 $P \mid Q$ 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 an action 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, $P \mid Q$ can engage in an action transition labeled by $\langle E \setminus \overline{\mathbb{I}}(Q), N\rangle$ to $P' \mid Q$. 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 temporarily 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 disabling operator may also be thought of as a *non-pre-emptive interrupt* operator, where Q is the *interrupt handler* (see Section 6). The *restriction operator* $\setminus L$ encapsulates all ports in L and, thereby, allows the scoping of events. Accordingly, 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. Hence, the internal action $\bullet$ is produced from $\langle E, N\rangle$, if the environment offers every event in E and if all events in N are restricted. Finally, process variable X, where $X \overset{\text{def}}{=} P$, is identified with a process that behaves as a distinguished solution of the equation $X = P$.

TABLE 3.3
Operational semantics (clock transitions)

tAct $\dfrac{-}{\langle E,N\rangle.P \xrightarrow{\sigma} \langle E,N\rangle.P}\ \langle E,N\rangle \neq \bullet$	tOut $\dfrac{-}{[E]\sigma(P) \xrightarrow{\sigma} P}$	tSum $\dfrac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P + Q \xrightarrow{\sigma} P' + Q'}$
tPar $\dfrac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P \,\vert\, Q \xrightarrow{\sigma} P' \,\vert\, Q'}\ \bullet \notin \mathrm{I}(P \,\vert\, Q)$	tNil $\dfrac{-}{\mathbf{0} \xrightarrow{\sigma} \mathbf{0}}$	tDis $\dfrac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P \rhd Q \xrightarrow{\sigma} P' \rhd Q'}$
tRes $\dfrac{P \xrightarrow{\sigma} P'}{P \setminus L \xrightarrow{\sigma} P' \setminus L}\ \bullet \notin \mathrm{I}(P \setminus L)$	tRec $\dfrac{P \xrightarrow{\sigma} P'}{X \xrightarrow{\sigma} P'}\ X \overset{\text{def}}{=} P$	tEn $\dfrac{P \xrightarrow{\sigma} P'}{P \rhd_{\sigma} Q \xrightarrow{\sigma} P' \rhd Q}$

The operational rules for clock transitions deal with the maximal progress assumption, i.e., if $\bullet \in \mathrm{I}(P) =_{\text{df}} \{\langle E,N\rangle \,\vert\, \exists P'.\, P \xrightarrow[N]{E} P'\}$ then a clock tick σ is inhibited. The reason that transitions other than those 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 process $[E]\sigma(P)$, which offers communications on the ports in E to its environment, disappears as soon as the next clock tick arrives and, thereby, enables process P. Time has to proceed equally on both sides of summation, parallel composition, and disabling, i.e., $P + Q$, $P \,\vert\, Q$, 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 $\bullet$ (see Rule Res) and, thereby, may pre-empt the considered clock transition. Finally, Rule tEn states that a clock tick switches the enabling to the disabling operator. Rule tRec does not require extra explanation.

The operational semantics for SPL possesses several pleasant algebraic properties which are known from timed process algebras [11, 35], such as (i) the *idling* property, i.e., $\bullet \notin \mathrm{I}(P)$ implies $\exists P' \in \mathcal{P}.\, P \xrightarrow{\sigma} P'$, for all $P \in \mathcal{P}$, (ii) the *maximal progress* property, i.e., $\exists P' \in \mathcal{P}.\, P \xrightarrow{\sigma} P'$ 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*.

3.3. A Behavioral Equivalence. As shown above, the SPL operational semantics interprets processes as labeled transition systems. However, from a semantic point of view, several transition systems might describe the same observable system behavior. For coping with this situation, standard process algebras introduce *behavioral equivalences* which relate processes, or transition systems, that describe the same intuitive behavior. One popular behavioral equivalence is *bisimulation* [24] which may be adapted to cater for SPL as follows.

DEFINITION 3.1 (Bisimulation). Bisimulation equivalence, $\sim\, \subseteq \mathcal{P} \times \mathcal{P}$, *is the largest symmetric relation such that whenever* $P \sim Q$, *the following conditions hold.*

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

Note that SPL states – in contrast to traditional process algebras – also contain information in the form of initial output action sets. This special situation is taken care of by Condition (1). Traditional results in process algebra show that the above definition is well-formed and that bisimulation equivalence is indeed an equivalence. Other work [33] may be used to establish that $\sim$ is a congruence for SPL.

4. Embedding of Statecharts. In this section, we present an embedding of Statecharts terms in SPL, which is defined to be a mapping $\llbracket \cdot \rrbracket$ from Statecharts terms to SPL processes. Although the semantics of SPL is defined on a "micro-step level," our process algebra allows us to encode the synchrony hypothesis of Statecharts via maximal progress. More precisely, a macro step in Statecharts semantics corresponds to a sequence of SPL action transitions which is enclosed by clock transitions; such sequences implicitly contain the causal order inherent in a Statecharts macro step. This correspondence is the key for proving a one-to-one relationship between a Statechart and its embedding.

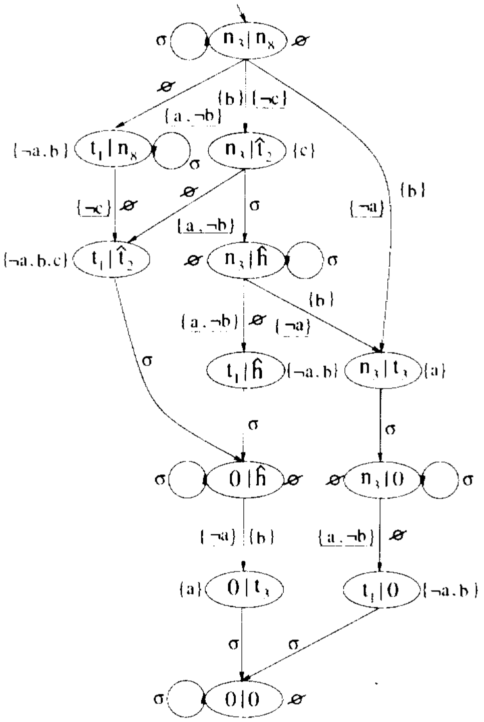
4.1. Formalization of the Embedding. We start off by instantiating the process algebra SPL. We choose $\Pi \cup \neg\Pi$ for the set of ports Λ and $\mathcal{N} \cup \{\hat{n} \,|\, n \in \mathcal{N}\} \cup \mathcal{T}$ for the set of process variables $\mathcal{V}$. The necessity for including negated events in Λ will become obvious later. We define the embedding $\llbracket \cdot \rrbracket : \mathsf{SC} \longrightarrow \mathcal{P}$ inductively along the structure of Statecharts terms, as follows, where $\sum$ is the indexed version of $+$ satisfying $\sum_{i \in \emptyset} P_i =_{\mathrm{df}} \mathbf{0}$.

1. If $s = [n]$, then $\llbracket s \rrbracket =_{\mathrm{df}} n$ where $n \overset{\mathrm{def}}{=} \hat{n} \overset{\mathrm{def}}{=} \mathbf{0}$.
2. If $s = [n : (s_1, \ldots, s_k); l; T]$ and $n_i = \mathsf{root}(s_i)$, for $1 \leq i \leq k$, then $\llbracket s \rrbracket =_{\mathrm{df}} n$, where $n \overset{\mathrm{def}}{=} \hat{n}_l$ and $\hat{n}_i \overset{\mathrm{def}}{=} n_i \rhd \sum\{\{\!\{t\}\!\} \,|\, t \in T \text{ and } \mathsf{root}(\mathsf{out}(t)) = n_i\}$, together with the equations produced by $\llbracket s_1 \rrbracket, \ldots, \llbracket s_k \rrbracket$. The translation $\{\!\{t\}\!\}$ of a transition t will be defined later.
3. If $s = [n : (s_1, \ldots, s_k)]$, then $\llbracket s \rrbracket =_{\mathrm{df}} n$ where $n \overset{\mathrm{def}}{=} \hat{n} \overset{\mathrm{def}}{=} \mathsf{root}(s_1) \,|\, \cdots \,|\, \mathsf{root}(s_k)$, together with the equations produced by $\llbracket s_1 \rrbracket, \ldots, \llbracket s_k \rrbracket$.

First, observe that the image of the embedding mapping is a process, defined via a process equation system, where the left-hand side of the equations are process variables taken from the names of states and transitions. A basic state semantically corresponds to the inaction process $\mathbf{0}$, whereas an or-state can either behave according to the process semantics of the embedding of the currently active state s_l, or it may leave 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 which semantically resembles state hierarchy. The translation of an and-state, which allows one to specify parallel composition, straightforwardly 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 $\llbracket \cdot \rrbracket$ is the translation $\{\!\{t\}\!\}$ of a transition $\langle t, i, E, A, j \rangle$. In the following, E' stands for $E \cap \Pi$, the set of positive events contained in E, and N' denotes the set $\neg(E \cap \neg\Pi) \cup \neg A$, which includes the negated negative events in E and the negation of the events in A. We define $\{\!\{t\}\!\} =_{\mathrm{df}} \langle E', N' \rangle.t$ where $t \overset{\mathrm{def}}{=} [A \cup (E \cap \neg\Pi)]\sigma(\hat{n}_j)$, i.e., the translation splits a Statecharts transition $\langle t, i, E, A, j \rangle$ in two parts, one handling its trigger E and one executing its action A. In order to execute its 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 when triggering a transition: *global consistency.* Especially, we must ensure that there is no transition in the same macro step, which fires because of the absence of an event in A. Therefore, we include $\neg e$, where $e \in A$, in the set N'. Events of the form $\neg e$ are offered by process t, whenever transition t triggers due to the absence of event e. Hence, $\{\!\{t\}\!\}$ can evolve via a SPL transition labeled by $\langle E', N' \rangle$ to process t, whenever the trigger of t is satisfied and whenever global consistency is guaranteed. Process t signals that transition t has

TABLE 4.1
Embedding of the Example Statechart

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



been triggered. Accordingly, it offers the events in A until the current macro step is completed, i.e., until a clock transition is executed. In order to ensure global consistency, process t also offers the events in $E \cap \neg\Pi$. It is worth noting that SPL's two-level semantics of action and clock transitions allows for broadcasting events using SPL's synchronization mechanism together with its maximal progress assumption.

We now return to our introductory example by presenting its formal translation to SPL in Table 4.1, left-hand side. The embedding's operational semantics is depicted on the right-hand side of Table 4.1, 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 $\mathbb{I}(P)$, 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, which passes 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 the transitions which 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.

4.2. Generalization of the Embedding. As a technical means for proving the main result of this paper which is stated in the next section, we generalize the embedding function to $\llbracket \cdot, \cdot \rrbracket : \mathsf{SC} \times 2^{\mathcal{T}} \longrightarrow \mathcal{P}$ in order capture micro steps. Intuitively, $\llbracket s, T \rrbracket$ identifies the SPL process which $\llbracket s \rrbracket$ reaches when it engages in the transitions in T. Formally $\llbracket s, T \rrbracket$ is defined inductively over the structure of s as follows.

1. If $s = [n]$, then $\llbracket s, T' \rrbracket =_{\mathrm{df}} n$.
2. If $s = [n : (s_1, \dots, s_k); l; T]$, then

$$\llbracket s, T' \rrbracket =_{\mathrm{df}} \begin{cases} \llbracket s_l, T' \rrbracket \rhd \ \sum\{\{\!\![t]\!\!\}\,|\, t \in T,\ \mathsf{out}(t) = s_l\} & \text{if } T' = \emptyset \\ \llbracket s_l, T' \rrbracket \rhd_\sigma \sum\{\{\!\![t]\!\!\}\,|\, t \in T,\ \mathsf{out}(t) = s_l\} & \text{if } \emptyset \neq T' \subseteq \mathsf{trans}(s_l) \\ t' & \text{if } \emptyset \neq T' = \{t'\} \subseteq T,\ \mathsf{out}(t') = s_l \\ \mathbf{0} & \text{otherwise} \end{cases}$$

3. If $s = [n : (s_1, \dots, s_k)]$, then $\llbracket s, T' \rrbracket =_{\mathrm{df}} \llbracket s_1, T_1 \rrbracket \,|\, \dots \,|\, \llbracket s_k, T_k \rrbracket$, where $T_i =_{\mathrm{df}} T' \cap \mathsf{trans}(s_i)$, $1 \leq i \leq k$.

In our proof context, T is a prefix of a sequence of transitions generated by the step-construction function, i.e, $\overline{\mathbb{I}}(\llbracket s, T \rrbracket) = \bigcup_{t \in T} \mathsf{act}(t)$ holds. The mapping $\llbracket \cdot, \cdot \rrbracket$ is a generalization of $\llbracket \cdot \rrbracket$ since $\llbracket s \rrbracket \doteq \llbracket s, \emptyset \rrbracket$, for all $s \in \mathsf{SC}$. Here, the symbol $\doteq$ stands for syntactic equality on processes up to "unfolding" of recursion. Formally, $\doteq$ is the largest congruence on $\mathcal{P}$ that contains syntactic equality and obeys the following property: $C \doteq Q$ and $C \overset{\mathrm{def}}{=} P$ implies $P \doteq Q$.

5. Semantic Correspondence. For formalizing our intuition of the semantic relation between Statecharts terms and their SPL embeddings, we define a notion of SPL *macro step* by combining several transitions to a single step, as outlined in Section 4.1. Accordingly, we write $P \overset{E}{\underset{A}{\Longrightarrow}} P'$ if there exists some $P'' \in \mathcal{P}$ such that $(\mathsf{Env}_E \,|\, P) \setminus \Lambda \overset{\emptyset}{\underset{\emptyset}{\longrightarrow}}{}^{*} (\mathsf{Env}_E \,|\, P'') \setminus \Lambda \overset{\sigma}{\longrightarrow} (\mathbf{0} \,|\, P') \setminus \Lambda$ and $\overline{\mathbb{I}}(P'') = A$, where $\mathsf{Env}_E \overset{\mathrm{def}}{=} [E]\sigma(\mathbf{0})$. Intuitively, P is placed in the context $(\mathsf{Env}_E \,|\, \cdot) \setminus \Lambda$, in which Env_E models a generic, single-step environment that offers the events in E until clock tick σ occurs.

5.1. Step Correspondence. The following relation, which we refer to as *step correspondence*, provides the formal foundation for relating Statecharts macro steps and SPL macro steps.

DEFINITION 5.1 (Step Correspondence). *A relation $\mathcal{R} \subseteq SC \times \mathcal{P}$ is a* step correspondence *if for all $\langle s, P \rangle \in \mathcal{R}$ and $E, A \subseteq \Pi$ the following conditions hold:*

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

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

THEOREM 5.2 (Semantic Correspondence). *Every $s \in SC$ is step-correspondent to $\llbracket s \rrbracket$.*

Proof sketch. It is sufficient to establish that $\mathcal{R} =_{\mathrm{df}} \{\langle s, \llbracket s \rrbracket \rangle \,|\, s \in \mathsf{SC}\}$ is a step correspondence, which can be done by induction on the structure of s. Intuitively, one can show that, if $T = (t_1, \dots, t_k)$ is a sequence of transitions of $s \in \mathsf{SC}$ generated by the step-construction function relative to the environment $E \subseteq \Pi$, then there exists a sequence of k internal transitions from $(\mathsf{Env}_E \,|\, \llbracket s \rrbracket) \setminus \Lambda$ to a process which can only engage in a clock transition to $(\mathbf{0} \,|\, \llbracket \mathsf{update}(s, T) \rrbracket) \setminus \Lambda$. Moreover, the lth internal transition, where $1 \leq l \leq k$, corresponds to the firing of t_l in s. Vice versa, if $(\mathsf{Env}_E \,|\, [\![s]\!]) \setminus \Lambda$ is the origin of an SPL path to a process which can only engage in a clock transition to $(\mathbf{0} \,|\, P') \setminus \Lambda$ and which mimics the triggering of a transition sequence $T = (t_1, \ldots, t_k)$, then T can be generated by the step-construction function relative to s and E. Moreover, $[\![\mathsf{update}(s, T)]\!] \doteq P'$.

The formalization of the above intuition requires the following auxiliary properties, where $s \in \mathsf{SC}$ and $E, A \subseteq \Pi$. Here, T stands for an arbitrary prefix of the above transition sequence $(t_1, \ldots, t_k)$ interpreted as set, i.e., $T = \{t_1, \ldots, t_l\}$ for some $0 \leq l \leq k$, and $\mathsf{act}(T)$ stands for $\bigcup_{t \in T} \mathsf{act}(t)$.

1. $\exists t \in \mathsf{enabled}(s, E, A, T) \setminus T$ implies $[\![s, T]\!] \xrightarrow[N']{E'} P'$ for some $E', N' \subseteq \Lambda$ and $P' \in \mathcal{P}$, such that $P' \doteq [\![s, T \cup \{t\}]\!]$, $E' = (\mathsf{ev}(t) \cap \Pi) \setminus \mathsf{act}(T)$, and $N' = \neg(\mathsf{ev}(t) \cap \neg\Pi) \cup \neg\mathsf{act}(t)$.
2. $[\![s, T]\!] \xrightarrow[N']{E'} P'$ for some $E' \subseteq E$, $N' \cap (E \cup \neg A) = \emptyset$, and $P' \in \mathcal{P}$ implies $\exists t \in \mathcal{T}.\ P' \doteq [\![s, T \cup \{t\}]\!]$, $t \in \mathsf{enabled}(s, E, A, T) \setminus T$, $E' = (\mathsf{ev}(t) \cap \Pi) \setminus \mathsf{act}(T)$, and $N' = \neg(\mathsf{ev}(t) \cap \neg\Pi) \cup \neg\mathsf{act}(t)$.
3. $\mathsf{enabled}(s, E, A, T) \setminus T = \emptyset$ implies $[\![s, T]\!] \xrightarrow{\sigma} P'$ for some $P' \in \mathcal{P}$, where $P' \doteq [\![\mathsf{update}(s, T), \emptyset]\!]$, and $\forall \langle E', N' \rangle \in \mathrm{I}([\![s, T]\!]).\ E' \setminus E \neq \emptyset$ or $N' \cap (E \cup \neg A) \neq \emptyset$.
4. $[\![s, T]\!] \xrightarrow{\sigma} P'$ for some $P' \in \mathcal{P}$ and $E' \setminus E \neq \emptyset$ or $N' \cap (E \cup \neg A) \neq \emptyset$ for all $\langle E', N' \rangle \in \mathrm{I}([\![s, T]\!])$ implies $\mathsf{enabled}(s, E, A, T) \setminus T = \emptyset$ and $P' \doteq [\![\mathsf{update}(s, T), \emptyset]\!]$.

The above properties establish a micro-step level relationship between Statecharts terms and the processes occurring in their embedding. The proof of each property can be done by induction on the structure of s and uses our extensions of the $\mathsf{enabled}$ function (cf. Section 2.3) and the embedding mapping (cf. Section 4.2). □

5.2. Preservation Results. We close the technical part by returning to the behavioral relation $\sim$ of bisimulation equivalence. First, we state a preservation result involving $\sim$ and SPL's macro-step semantics.

THEOREM 5.3. *Let $P, P', Q \in \mathcal{P}$ such that $P \sim Q$ and $P \overset{E}{\underset{A}{\Longrightarrow}} P'$. Then $\exists Q' \in \mathcal{P}.\ Q \overset{E}{\underset{A}{\Longrightarrow}} Q'$ and $P' \sim Q'$.*

The validity of this theorem relies on the congruence property of $\sim$ for SPL. When combining the insights obtained by establishing Theorems 5.2 and 5.3, one may derive the following corollary which relates bisimulation equivalence and Statecharts macro-step semantics.

COROLLARY 5.4. *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. The comparison paper [34] surveys over twenty Statecharts variants. In this section, we show how our approach can be adapted to these variants and, thereby, testify to its flexibility. We focus on the most relevant issues of Statecharts semantics, which are identified in [34].

As is immanent in this paper, we favor an *operational semantics* over a *denotational* one, since we feel that operational models are more intuitive and, therefore, easier to understand. Moreover, operational models provide an immediate interface to verification tools which implement state-exploration techniques. An important observation of this paper is that the concept of a single, global clock together with maximal progress is the key to providing a *compositional, causal* state-machine semantics for Statecharts. Although the semantics is defined on the micro-step level, it allows for an easy identification of macro steps. The clock enforces global synchronizations which mark the beginning and end of macro steps. Thus, macro steps are represented as sequences of micro steps, which encode the underlying causality of Statecharts semantics.

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*** and 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 may appear in the trigger of transitions and which are signaled by a process if it is in state n.

Another issue of Statecharts semantics concerns the question whether there exists a difference in sensing ***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. We achieve this semantics by using the signal operator which stops the signaling of events as soon as the next clock tick arrives. However, 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 basically splitting every state $t \in \mathcal{T}$ into two states that are connected via a clock transition.

The Statecharts concept of ***negated events*** forces transitions to be triggered only when certain events are absent. Negated events may be used for imposing ***priority*** between transitions and, thereby, for resolving nondeterministic choices. SPL adopts this concept by requiring input actions to be pairs of sets of events, one containing the events which must be present and the other the events which must be absent for triggering a transition. 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 former prohibits a transition, containing a negative trigger event $\neg e$, to be executed if a micro step in the same macro step produces e. This is enforced in our embedding 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 the 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, i.e., until a clock transition is executed.

In addition to the possibility of encoding priorities between transitions via negated events, one may also introduce an implicit priority mechanism along ***state hierarchy***, as is done, e.g., in Statemate [10]. More precisely, a transition leaving an or-state is given priority over any transition within this state, i.e., or-states may be viewed as ***pre-emptive interrupt*** operators. Considering this behavior in SPL requires one to modify the semantics of the disabling operator, accordingly. However, such a modification does not introduce any new semantic issues, since the necessary concept of pre-emption is the same as for the synchrony hypothesis.

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 been 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. As a consequence, we followed their suggestion to study two-leveled semantics. In our approach, the three properties hold on different levels:

compositionality holds on the micro-step level, i.e., the level of SPL action transitions, whereas responsiveness and causality is guaranteed on the macro-step level, i.e., the level on which sequences of SPL action transitions between global synchronizations, caused by clock ticks σ, are bundled together.

Uselton and Smolka [31] and Levi [17] also focused on achieving a clean, compositional semantics for Statecharts by referring to process algebras. In contrast to our approach, Uselton and Smolka's notion of transition system involves complex labels of the form $\langle E, \prec \rangle$, where E is a set of events and $\prec$ a transitive, irreflexive order on E, for encoding causality. Unfortunately, their semantics suffers from some serious problems, as pointed out in [17, 18]. Essentially, the semantics does not correspond as intended to the Statecharts semantics of Pnueli and Shalev [28]. 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. However, we believe that our semantics, where labels do not contain any order at all, profits from improved readability.

Maggiolo-Schettini et al. considered a hierarchy of equivalences for Statecharts, including isomorphism and bisimulation, and studied congruence properties with respect to Statecharts operators [18]. 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. In their semantics, labels of transitions consist of four-tuples which include information about causal orderings, global consistency, and negated events. This complexity prohibits an intuitive understanding of Statecharts semantics and an easy integration with existing analysis and verification tools. However, it should be noted that the semantic framework presented in [18] 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* [20]. 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 model-checking tools is a main objective in [16] and most recently also in a series of papers by Mikk et al. [21, 22, 23]. The first paper of this series includes a formalization of the semantics of Statemate, as defined in [8], within the specification formalism Z [30]. The second paper describes a translation from a subset of Statemate to *hierarchical state automata* which may be mapped to the specification language of the verification tool Spin [13], as shown in Mikk's third paper.

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 the process algebra SPL which is expressive enough to model the semantic principles underlying Statecharts. SPL allows one to encode a "micro-step" semantics of Statecharts in the traditional SOS-style; it is at this level that our semantics is compositional, as bisimulation may be shown to be a congruence for the language. The macro-step semantics may then be given in terms of a derived transition relation. This semantics cannot be compositional, as results of Huizing and Gerth have shown [15]. However, the algebraic basis of our semantics permits the investigation of, e.g., the largest congruence consonant within this semantics. Also, since these sequences essentially encode total closures of causal orders, *partial order methods* might be useful for avoiding unnecessary state explosion in practice [6]. Note that, although SPL is a newly developed process algebra, all of its semantic ingredients have already been studied in the process-algebra community.

We demonstrated the utility of our technique by formally embedding the Statecharts semantics of [18], which is a slight variant of Pnueli and Shalev's semantics [28], in SPL. Our embedding is sound and complete in the sense that Statecharts terms and their embeddings mutually simulate each other. The benefits of our approach include (i) a uniform semantic framework for intuitively modeling the semantics for several Statecharts variants in a compositional style, (ii) a simple method to interfacing Statecharts to existing verification tools, such as the *Concurrency Workbench of North Carolina* (CWB-NC) [5], (iii) 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. Finally, the paper gave insight in the close semantic relationship between process algebras and Statecharts and, thereby, testified to the practical importance of process algebras for design tools for reactive systems.

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 CWB-NC.

Acknowledgments. We would like to thank Peter Kelb, Ingolf Krüger, and Michael Mendler for many discussions on Statecharts, as well as Piyush Mehrotra for carefully proofreading a draft of this paper.

REFERENCES

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

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

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

[4] R. Cleaveland, G. Lüttgen, and V. Natarajan, *Priority in process algebra*, in Handbook of Process Algebra, J. Bergstra, A. Ponse, and S. Smolka, eds., Elsevier, 1999.

[5] R. Cleaveland and S. Sims, *The NCSU Concurrency Workbench*, in Computer Aided Verification (CAV '96), R. Alur and T. Henzinger, eds., Vol. 1102 of Lecture Notes in Computer Science, New Brunswick, NJ, USA, July 1996, Springer-Verlag, pp. 394–397.

[6] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, Vol. 1032 of Lecture Notes in Computer Science, Springer-Verlag, 1996.

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

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

[9] D. Harel, A. Pnueli, J. Schmidt, and R. Sherman, *On the formal semantics of Statecharts*, in Symposium on Logic in Computer Science (LICS '87), Ithaca, NY, USA, June 1987, IEEE Computer Society Press, pp. 56–64.

[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 (1995), pp. 221–239.

[12] C. Hoare, *Communicating Sequential Processes*, Prentice Hall, London, UK, 1985.

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

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

[15] C. Huizing, *Semantics of Reactive Systems: Comparison and Full Abstraction*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, March 1991.

[16] P. Kelb, *Abstraktionstechniken für automatische Verifikationsmethoden*, Ph.D. thesis, University of Oldenburg, Oldenburg, Germany, 1996.

[17] F. Levi, *Verification of Temporal and Real-Time Properties of Statecharts*, Ph.D. thesis, University of Pisa-Genova-Udine, Pisa, Italy, February 1997.

[18] A. Maggiolo-Schettini, A. Peron, and S. Tini, *Equivalences of Statecharts*, in Seventh International Conference on Concurrency Theory (CONCUR '96), U. Montanari and V. Sassone, eds., Vol. 1119 of Lecture Notes in Computer Science, Pisa, Italy, August 1996, Springer-Verlag, pp. 687–702.

[19] F. Maraninchi, *The ARGOS language: Graphical representation of automata and description of reactive systems*, in IEEE Workshop on Visual Languages, IEEE Computer Society Press, October 1991.

[20] ——, *Operational and compositional semantics of synchronous automaton compositions*, in Third International Conference on Concurrency Theory (CONCUR '92), R. Cleaveland, ed., Vol. 630 of Lecture Notes in Computer Science, Stony Brook, NY, USA, August 1992, Springer-Verlag, pp. 550–564.

[21] 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, Ilkley, UK, July 1997, Springer-Verlag.

[22] E. Mikk, Y. Lakhnech, and M. Siegel, *Hierarchical automata as model for Statecharts*, in Proceedings of Asian Computing Science Conference (ASIAN '97), Vol. 1345 of Lecture Notes in Computer Science, Springer-Verlag, December 1997.

[23] E. Mikk, Y. Lakhnech, M. Siegel, and G. Holzmann, *Verifying Statecharts with Spin*, in Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques (WIFT '98), Boca Raton, FL, USA, October 1998, IEEE Computer Society Press.

[24] R. Milner, *Communication and Concurrency*, Prentice Hall, London, UK, 1989.

[25] D. Park, *Concurrency and automata on infinite sequences*, in Proceedings of 5th G.I. Conference on Theoretical Computer Science, P. Deussen, ed., Vol. 104 of Lecture Notes in Computer Science, Springer-Verlag, 1981, pp. 167–183.

[26] G. Plotkin, *A structural approach to operational semantics*, Tech. Report DAIMI-FN-19, Computer Science Department, Aarhus University, Denmark, 1981.

[27] A. Pnueli, ed., *Linear and Branching Structures in the Semantics and Logics of Reactive Systems*, Vol. 194 of Lecture Notes in Computer Science, Springer-Verlag, 1985.

[28] A. Pnueli and M. Shalev, *What is in a step: On the semantics of Statecharts*, in Theoretical Aspects of Computer Software (TACS '91), T. Ito and A. Meyer, eds., Vol. 526 of Lecture Notes in Computer Science, Sendai, Japan, September 1991, Springer-Verlag, pp. 244–264.

[29] P. Scholz, *Design of Reactive Systems and Their Distributed Implementation with Statecharts*, Ph.D. thesis, Munich University of Technology, Munich, Germany, August 1998.

[30] J. Spivey, *Understanding Z: A Specification Language and its Formal Semantics*, Cambridge Tracts in Theoretical Computer Science 3, Cambridge University Press, Cambridge, UK, 1988.

[31] A. Uselton and S. Smolka, *A compositional semantics for Statecharts using labeled transition systems*, in Fifth International Conference on Concurrency Theory (CONCUR '94), B. Jonsson and J. Parrow, eds., Vol. 836 of Lecture Notes in Computer Science, Uppsala, Sweden, August 1994, Springer-Verlag, pp. 2–17.

[32] ——, *A process-algebraic semantics for Statecharts via state refinement*, in IFIP TC2 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94), North Holland/Elsevier, 1994.

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

[34] M. von der Beeck, *A comparison of Statecharts variants*, in Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '94), H. Langmaack, W.-P. de Roever, and J. Vytopil, eds., Vol. 863 of Lecture Notes in Computer Science, Lübeck, Germany, September 1994, Springer-Verlag, pp. 128–148.

[35] W. Yi, *CCS + time = an interleaving model for real time systems*, in Automata, Languages and Programming (ICALP '91), J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, eds., Vol. 510 of Lecture Notes in Computer Science, Madrid, Spain, July 1991, Springer-Verlag, pp. 217–228.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
October 1999

3. REPORT TYPE AND DATES COVERED
Contractor Report

4. TITLE AND SUBTITLE
Statecharts via process algebra

5. FUNDING NUMBERS
C NAS1-97046
WU 505-90-52-01

6. AUTHOR(S)
Gerald Lüttgen
Michael von der Beeck
Rance Cleaveland

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
ICASE Report No. 99-42

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-2199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/CR-1999-209713
ICASE Report No. 99-42

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Dennis M. Bushnell
Final Report
Presented at the International Conference on Concurrency Theory (CONCUR'99).

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 60, 61
Distribution: Nonstandard
Availability: NASA-CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

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, a simple process algebra, called Statecharts Process Language (SPL), is presented, which is expressive enough for encoding Statecharts in a structure-preserving and semantics-preserving manner. It is established that the behavioral relation bisimulation, when applied to SPL, preserves Statecharts semantics.

14. SUBJECT TERMS
bisimulation, compositionality, operational semantics, process algebra, statecharts

15. NUMBER OF PAGES
23

16. PRICE CODE
A03

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT