A Compositional Approach to Statecharts Semantics*

Gerald Lüttgen
ICASE, NASA Langley
Research Center
Hampton, VA 23681–2199
USA
luettgen@icase.edu

Michael von der Beeck
Department of Comp. Sc.
TU Munich
D–80290 München
Germany
beeck@in.tum.de

Rance Cleaveland
Department of Comp. Sc.
SUNY at Stony Brook
Stony Brook, NY 11794–4400
USA
rance@cs.sunysb.edu

ABSTRACT

Statecharts is a visual language for specifying reactive system behavior. The formalism extends traditional finite–state machines with notions of hierarchy and concurrency, and it is used in many popular software design notations. A large part of the appeal of Statecharts derives from its basis in state machines, with their intuitive operational interpretation. The classical semantics of Statecharts, however, suffers from a serious defect: it is not compositional, meaning that the behavior of system descriptions cannot be inferred from the behavior of their subsystems. Compositionality is a prerequisite for exploiting the modular structure of Statecharts for simulation, verification, and code generation, and it also provides the necessary foundation for reusability.

This paper suggests a new compositional approach to formalizing Statecharts semantics as flattened labeled transition systems in which transitions represent system steps. The approach builds on ideas developed for timed process calculi and employs structural operational rules to define the transitions of a Statecharts expression in terms of the transitions of its subexpressions. It is first presented for a simple dialect of Statecharts, with respect to a variant of Pnueli and Shalev's semantics, and is illustrated by means of a small example. To demonstrate its flexibility, the proposed approach is then extended to deal with practically useful features available in many Statecharts variants, namely state references, history states, and priority concepts along state hierarchies.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: [languages]; D.3.1 [**Programming Languages**]: Formal Definitions and Theory; F.3.2 [**Semantics of Programming Languages**]: [operational semantics]

*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.

General Terms

Languages

Keywords

Compositionality, operational semantics, Statecharts

1. INTRODUCTION

Statecharts [9] is a visual language for specifying *reactive*, *embedded*, and *real–time systems*. The formalism extends finite–state machines with concepts of *hierarchy*, *concurrency*, and *priority*. The success of Statecharts in the Software Engineering community is founded on its graphical appeal and its capacity for modeling the complex control aspects inherent in many software systems. Different dialects of the language [36] have been employed in several software design notations — including ROOM [31], STATEMATE [12], and UML [3] — and commercial tools provide support for them. Nevertheless, precisely defining Statecharts' semantics has proved extremely challenging, with a variety of proposals being offered in the literature [7, 10, 11, 16, 17, 18, 21, 22, 25, 27, 29, 30, 33].

Existing Statecharts variants typically conform to the following interpretation of system behavior. A Statechart may respond to an event entering the system by engaging in an enabled transition, thus performing a *micro step*. This transition may generate new events which, by *causality*, may in turn trigger additional transitions while disabling others. The *synchrony hypothesis* ensures that one execution step, a so–called *macro step*, is complete as soon as this chain reaction comes to a halt. There is, however, one additional desirable ingredient that a practical Statecharts semantics should have: *compositionality*. Compositionality ensures that the semantics of a Statechart can be determined from the semantics of its components. This is of particular importance when simulating Statecharts or generating code, as one does not want to waste resources re–compiling a large Statechart if only a few of its components are changed. Compositionality is also useful when formally analyzing or verifying Statecharts. Unfortunately, all practically–relevant approaches to Statecharts semantics ignore compositionality, except for an approach presented for synchronous STATEMATE [7] whose semantics does not obey the synchrony hypothesis. Indeed, theoretical studies conducted by Huizing and Gerth [15] showed that one cannot combine the features of causality, synchrony hypothesis, and compositionality within a step semantics which labels transitions by sets of "input/output" events. In fact, the classical and well–recognized semantics of Statecharts — as defined by Pnueli and Shalev [29] — satisfies the synchrony hypothesis and causality, but is not compositional.

The aim of this paper is to present a new approach to defining Statecharts semantics which combines all three abovementioned features in a formal, yet operationally intuitive, fashion. Our semantic account borrows ideas from *timed process calculi* [13], which also employ the synchrony hypothesis [2] and which allow one to represent ordinary system behavior and clock ticks using labeled transition systems. These transition systems are defined via *structural operational rules* [28] — i.e., rules in SOS format — along the state hierarchy of the Statechart under consideration. Our semantics explicitly represents macro steps as sequences of micro steps which begin and end with the ticking of a global clock. Thereby, compositionality is achieved on the explicit micro–step level and causality and synchrony on the implicit macro–step level. The current work builds on previous research by the authors [20], which developed a compositional timed process algebra that was then used to embed a simple variant of Statecharts introduced in [21]. That work indirectly yielded a compositional operational semantics for Statecharts. In this paper, we re–develop the semantics of [20] without reference to a process algebra, thereby eliminating the rather complicated indirection. Our intention is to make the underlying semantic issues and design decisions for Statecharts more apparent and comprehensible. The paper also argues for the flexibility and elegance of our approach by extending our semantics to cope with popular Statecharts features used in practice, such as *state references*, *history states*, and *priority concepts*.

Organization of this paper. The next section gives a brief overview of Statecharts, including our notation and its classical semantics. Section 3 presents our new compositional approach to Statecharts semantics. It also establishes a coincidence result with respect to the traditional step semantics and illustrates our approach by means of an example. Section 4 shows how our framework can be extended to include various features employed in many Statecharts dialects. Finally, Section 5 discusses related work, while Section 6 contains our concluding comments as well as some directions for future research.

2. AN OVERVIEW OF STATECHARTS

Statecharts is a specification language for *reactive systems*, i.e., systems characterized by their ongoing interaction with their *environment*. The notation enriches basic finite–state machines with concepts of *hierarchy*, *concurrency*, and *priority*. In particular, one Statechart may be embedded within the state of another Statechart, and a Statechart may be composed of several simultaneously active sub–Statecharts which communicate via broadcasting events. Transitions are labeled by pairs of event sets, where the first component is referred to as *trigger* and may include *negated events*, and the second is referred to as *action*. Intuitively, if the environment offers all the positive but none of the negated events of the trigger, then the transition is enabled and can be executed, thereby generating the events in the label's action.

As an example, consider the Statechart depicted in Figure 1. It consists of an *and–state*, labeled by n_1, which denotes the parallel composition of the two Statecharts labeled by n_2 and n_3, both of which are *or–states* describing a sequential state machine. Or–state n_2 is further refined by or–state n_4 and *basic state* n_5, which are connected via transition t_1 labeled by b. The label specifies that t_1 is triggered by the occurrence of event b; its execution does not generate any new event as its action is empty. Or–state n_4 contains the basic states n_8 and n_9, connected by transition t_3 with trigger $a \wedge \neg b$ and empty action; hence, t_3 is enabled if event a occurs but event b does not. Or–state n_3 consists of two basic

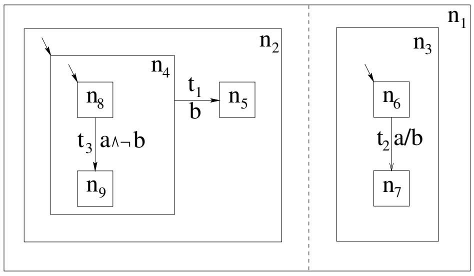

Figure 1: Example Statechart.

states n_6 and n_7 connected via transition t_2 with label a/b, so that upon occurrence of trigger event a, transition t_2 can be executed and generate event b.

In this paper, we first consider a simple dialect of Statecharts that supports a basic subset of the popular features present in many Statecharts variants. In particular, it considers hierarchy and concurrency. However, it ignores *interlevel transitions* (i.e., transitions crossing borderlines of states), *state references* (i.e., triggers of the form $\mathsf{in}(n)$, where n is the name of a state), and *history states* (remembering the last active sub–state of an or–state). In addition, it does not attach any implicit priorities to transitions at different levels in the state hierarchy. To illustrate the flexibility of our approach, we show in Section 4 how it can be extended to deal with state references, history states, and the abovementioned priority concepts. Interlevel transitions, however, cannot be brought in accordance with a *compositional* semantics, as they represent an unstructured "goto" behavior (cf. Section 5).

2.1 Term–based Syntax

For our purposes it is convenient to represent Statecharts not visually but by terms, as is done in [20, 21]. 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. For technical convenience, we assume that $\mathcal{N}$ and $\mathcal{T}$ are disjoint. With every event $e \in \Pi$ we associate a negated counterpart $\neg e$ and define $\neg\neg e =_{\mathrm{df}} e$ as well as $\neg E =_{\mathrm{df}} \{\neg e \mid e \in E\}$, for $E \subseteq \Pi \cup \{\neg e \mid e \in \Pi\}$. The set SC of Statecharts terms is then defined by the following inductive rules.

1. *Basic state:* If $n \in \mathcal{N}$, then $[n]$ is a Statecharts term.
2. *Or–state:* Suppose that $n \in \mathcal{N}$ and that $s_1, \ldots, s_k$ are Statecharts terms for $k > 0$, with $\vec{s} =_{\mathrm{df}} (s_1, \ldots, s_k)$. Also let $\rho =_{\mathrm{df}} \{1, \ldots, k\}$ and $l \in \rho$, with $T \subseteq \mathcal{T} \times \rho \times 2^{\Pi \cup \neg\Pi} \times 2^{\Pi} \times \rho$. Then $s = [n : \vec{s}; l; T]$ is a Statecharts term. Here $s_1, \ldots, s_k$ are the sub–states of s, set T contains the transitions connecting these states, s_1 is the default state of s, and s_l is the currently active sub–state of s. The transitions in T are of the form $\hat{t} =_{\mathrm{df}} \langle t, i, E, A, j \rangle$, where
   (a) t is the *name* of $\hat{t}$,
   (b) $\mathsf{source}(\hat{t}) =_{\mathrm{df}} s_i$ is the *source state* of $\hat{t}$,
   (c) $\mathsf{trg}(\hat{t}) =_{\mathrm{df}} E$ is the *trigger* of $\hat{t}$,
   (d) $\mathsf{act}(\hat{t}) =_{\mathrm{df}} A$ is the *action* of $\hat{t}$, and
   (e) $\mathsf{target}(\hat{t}) =_{\mathrm{df}} s_j$ is the *target state* of $\hat{t}$.

   In the sequel, $\mathsf{trg}^+(\hat{t})$ stands for $\mathsf{trg}(\hat{t}) \cap \Pi$ and $\mathsf{trg}^-(\hat{t})$ for $\mathsf{trg}(\hat{t}) \cap \neg\Pi$. Since we assume that all state names and transition names are mutually disjoint, we may uniquely refer to states and transitions by using their names, e.g., we may write t for $\hat{t}$. We also assume that no transition produces an event which appears negated in its trigger.

3. *And–state:* If $n \in \mathcal{N}$, if $s_1, \dots, s_k$ are Statecharts terms for $k > 0$, and if $\vec{s} =_{\text{df}} (s_1, \dots, s_k)$, then $s = [n : \vec{s}]$ is a Statecharts term, where $s_1, \dots, s_k$ are the (parallel) sub–states of s.

The Statecharts term corresponding to the Statechart depicted in Figure 1 is term s_1 which is defined as follows.

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

Note that components two and five of a transition $\langle t, i, E, A, j\rangle$ in some or–state $s = [n : \vec{s}; l; T]$ refer to the *indexes* of the source and target state in the sequence $\vec{s} = (s_1, \dots, s_k)$, respectively, and *not* to the states' names.

2.2 Classical Semantics

In this section, we sketch the semantics of Statecharts terms adopted in [21], which is a slight variant of the "classical" Statecharts semantics as proposed by Pnueli and Shalev [29]. We refer the reader to [21] for a detailed discussion of the underlying semantic issues.

As mentioned before, a Statechart s reacts to the arrival of some external events by triggering enabled micro steps in a chain–reaction manner. When this chain reaction comes to a halt, a complete macro step has been performed. More precisely, a macro step comprises a *maximal* set of micro steps, or transitions, that (i) are *relevant*, (ii) are mutually *consistent*, (iii) are triggered by events $E \subseteq \Pi$ offered by the environment or generated by other micro steps, (iv) are mutually *compatible*, and (v) obey the principle of *causality*. These notions may be defined as follows. Let $s \in \mathsf{SC}$, let t be a transition in s, let T be a set of transitions in s, and let $E \subseteq \Pi$. Transition t is *relevant* for Statecharts term s, in signs $t \in \mathsf{relevant}(s)$, if the source state of t is currently active. Transition t is *consistent* with all transitions in T, in signs $t \in \mathsf{consistent}(s, T)$, if t is not in the same parallel component as any transition in T. Transition t is *triggered* by event set E, in signs $t \in \mathsf{triggered}(s, E)$, if the positive but not the negative trigger events of t are in E. Transition t is *compatible* with all transitions in T, in signs $t \in \mathsf{compatible}(s, T)$, if no event produced by t appears negated in a trigger of a transition in T. Finally, we say that transition t is *enabled* in s with respect to event set E and transition set T, if $t \in \mathsf{enabled}(s, E, T)$, where $\mathsf{enabled}(s, E, T) =_{\text{df}} \mathsf{relevant}(s) \cap \mathsf{consistent}(s, T) \cap \mathsf{triggered}(s, E \cup \bigcup_{t \in T} \mathsf{act}(t)) \cap \mathsf{compatible}(s, T)$.

A macro step is a subset of transitions in $\mathsf{enabled}$ that is *causally well-founded*. Technically, causality holds if there exists an ordering among the transitions in a macro step such that no transition t depends on events generated by transitions occurring after t in the macro step. In [21], an operational approach for causally justifying the triggering of each transition of a macro step is given. It employs the nondeterministic *step–construction* function presented in

Table 1: 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
```

Table 1, which is adapted from Pnueli and Shalev [29]. Given a Statecharts term s and a set E of environment events, the step–construction function nondeterministically computes a set T^* of transitions. In this case, Statecharts term s may evolve in the single *macro step* $s \overset{E}{\underset{A}{\Longrightarrow}} s'$ to Statecharts term s', thereby executing the transitions in T^* and producing the events $A =_{\text{df}} \bigcup_{t \in T^*} \mathsf{act}(t)$. Term s' can be derived from s by updating the index l in every or–state $[n : (s_1, \dots, s_k); l; T]$ of s satisfying $t \in T^*$ for some $t \in T$. Observe that once one has constructed a macro step, all information about how the macro step was derived at is discarded. This is the source for the compositionality defect of this semantics for Statecharts: when two Statecharts are composed in parallel, the combination of the causality orderings may introduce newly enabled transitions (cf. [19]).

Let us illustrate a couple of macro steps of the example Statechart depicted in Figure 1. For convenience, we abbreviate a Statecharts term by its active basic states, e.g., term s_1 is abbreviated by $\langle n_8, n_6\rangle$. Moreover, we let $\Pi =_{\text{df}} \{a, b\}$ and assume that the environment only offers event a. Then, both transitions t_2 and t_3 are enabled, and the execution of t_3 results in macro step $\langle n_8, n_6\rangle \overset{\{a\}}{\underset{\emptyset}{\Longrightarrow}} \langle n_9, n_6\rangle$, i.e., a macro step in which only a single transition takes part. Although t_2 is also enabled, it cannot be executed together with t_3 in the same macro step. Otherwise, global consistency is violated since t_2 generates event b whose negated counterpart $\neg b$ is contained in the trigger of t_3. However, transitions t_2 and t_1 can take part in the same macro step, as t_1 is located in a different parallel component than t_2 and is triggered by event b which is generated by t_2. This leads to macro step $\langle n_8, n_6\rangle \overset{\{a\}}{\underset{\{b\}}{\Longrightarrow}} \langle n_5, n_7\rangle$. All macro steps of our example Statechart can be found in Figure 3.

3. A COMPOSITIONAL SEMANTICS

While Pnueli and Shalev's semantics has the advantage of simplicity, it is *not* compositional: it relies on a global analysis of an entire Statecharts expression in order to infer its macro steps. In particular, it does not compute the macro steps of a Statechart in terms of the macro steps of its subcomponents. When the Statechart in question is large, the procedure can therefore be slow, since the macro steps of subcomponents cannot be "precomputed" and reused. It also means that the semantics cannot be used as a basis for modular reasoning about Statecharts.

In the following, we present our approach to defining a *compositional* semantics for Statecharts. Our framework is based on flat labeled transition systems and is defined in the SOS style, i.e., via *structural operational rules* [28]. Each such rule is of the form

$$\textit{name} \; \frac{\textit{premise}}{\textit{conclusion}} \; \textit{side condition}$$

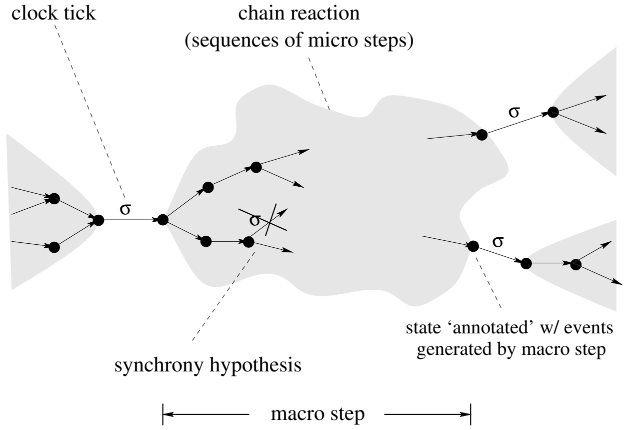

Figure 2: Illustration of our operational semantics.

and should be read as follows: Rule (*name*) is applicable if both the statements in its *premise* and its *side condition* hold; in this case, one might infer the *conclusion*. The benefits of employing SOS–style semantics are manifold. Most tools for the formal analysis and verification of systems rely on compilers which translate a textual system specification, e.g., a Statechart term, into a labeled transition system or a Kripke structure [5]. Examples of such tools are those offering equivalence checkers and/or model checkers, including the CONCURRENCY WORKBENCH [6], SPIN [14], and SMV [24]. In addition, several meta–theories regarding SOS–style semantics have been developed. Of most relevance to our work are results which infer the compositionality of a semantics from the syntactical structure of the premises, conclusions, and side conditions of the SOS rules defining it [35].

3.1 Our approach

In contrast to related work, we develop an operational semantics on the micro–step level rather than the macro–step level and represent macro steps as sequences of micro steps. Within such a setting, compositionality is easy to achieve. The challenge is to identify the states at which macro steps start and end so that Statecharts' traditional, non–compositional macro–step semantics can be recovered. Our solution is based on the observation that since Statecharts is a synchronous language, ideas from timed process calculi may be adapted. In particular, we use explicit global clock ticks to denote the boundaries of macro steps.

Our flat labeled transition systems therefore possess two kinds of transitions: those representing the execution of a Statecharts transition and those representing global clock ticks. In timed process calculi such transitions are referred to as *action transitions* and *clock transitions*, respectively. The ideas behind our semantics are illustrated in Figure 2, where clock transitions are labeled by σ. The other transitions are action transitions and actually carry pairs $\langle E', N' \rangle$ of event sets as labels. An action transition stands for a single Statechart transition which is enabled if the system environment offers all events in E' but none in N'. The states of our transition systems are annotated with (extended) Statecharts terms from which one may infer the events generated at any point of execution of the considered Statechart. Accordingly, the classical macro–step semantics of Statecharts can be recovered from our semantics as follows: assume that the global clock ticks, symbolizing the beginning of a macro step, when the system environment offers the events in E. Starting from a clock transition, follow

Table 2: Functions out (top) and default (bottom)

$\mathsf{out}([n])$	$=_{\mathrm{df}}$	$\emptyset$
$\mathsf{out}([n:\vec{s};l;T])$	$=_{\mathrm{df}}$	$\mathsf{out}(s_l)$
$\mathsf{out}([n:\vec{s};t;T])$	$=_{\mathrm{df}}$	$\mathsf{act}(t) \cup \mathsf{trg}^-(t)$
$\mathsf{out}([n::\vec{s};l;T])$	$=_{\mathrm{df}}$	$\mathsf{out}(s_l)$
$\mathsf{out}([n:(s_1,\ldots,s_k)])$	$=_{\mathrm{df}}$	$\bigcup_{l=1}^{k} \mathsf{out}(s_l)$
$\mathsf{default}([n])$	$=_{\mathrm{df}}$	$[n]$
$\mathsf{default}([n:\vec{s};l;T])$	$=_{\mathrm{df}}$	$[n:\vec{s}_{[l\mapsto \mathsf{default}(s_l)]};1;T]$
$\mathsf{default}([n:\vec{s}])$	$=_{\mathrm{df}}$	$[n:\mathsf{default}(\vec{s})]$

an arbitrary path of action transitions that are triggered by E, i.e., whose labels $\langle E', N' \rangle$ satisfy $E' \subseteq E$ and $N' \cap E = \emptyset$. When another clock transition is executed, the constructed macro step is complete. The states traversed in the path collect the events introduced by the fired Statecharts transitions along the path. Hence, from the source state of the concluding clock transition one may extract all events generated in the considered macro step. Note that, according to the synchrony hypothesis, clock transitions are prohibited unless no additional action transition can be executed relative to environment E. In a nutshell, our semantics is defined in a way that achieves compositionality on the explicit micro–step level, while causality and the synchrony hypothesis are observed on the implicit macro–step level.

3.2 Formalization

To formalize our abovementioned intuitions, we first need to extend the definition of Statecharts terms such that "Statecharts snapshots," taken after *partial* executions of macro steps, can be represented. Formally, we add the following rule to the inductive definition of Statecharts terms given in Section 2: If $[n:\vec{s};l;T]$ is a Statecharts term, then $[n:\vec{s};t;T]$, for $t \in T$, and $[n::\vec{s};l;T]$ are Statecharts terms. Intuitively, term $[n:\vec{s};t;T]$ represents an or–state after firing some 'top-level' transition $t \in T$; term $[n::\vec{s};l;T]$ represents an or–state after firing some 'inner' transition, i.e., a transition originating in the active sub–state s_l. The extended set of Statecharts terms is denoted by $\mu\mathsf{SC}$, and its elements are sometimes referred to as *micro terms*. Our formalization of Statecharts semantics also requires us to be able to extract all events $\mathsf{out}(s)$ from a micro term s, which are generated by transitions that have fired during the considered partial macro step. Additionally, $\mathsf{out}(s)$ includes all negated trigger events of the executed transitions, which is necessary to ensure the Statecharts property of global consistency, as will become clear shortly. The predicate $\mathsf{out}(s) \subseteq \Pi \cup \neg\Pi$ can be defined inductively along the structure of s, and its definition is displayed in Table 2. Finally, we need one more auxiliary function, $\mathsf{default}(s)$, which, given a Statecharts term s in SC, resets all the active states of its or–states to their respective initial states. Also this function can be defined on the structure of s as is done in Table 2. For convenience, we write $\mathsf{default}(\vec{s})$ for $\mathsf{default}((s_1,\ldots,s_k)) =_{\mathrm{df}} (\mathsf{default}(s_1),\ldots,\mathsf{default}(s_k))$ and define $\vec{s}_{[l\mapsto s']} =_{\mathrm{df}} (s_1,\ldots,s_{l-1},s',s_{l+1},\ldots,s_k)$, for $1 \leq l \leq k$ and $s' \in \mathsf{SC}$.

Now we present our semantics of a Statecharts term $s \in \mathsf{SC}$. As indicated before, the semantics of s is defined as a labeled transition system, such that (i) the states are terms in $\mu\mathsf{SC}$, (ii) the start state is s, and (iii) the two transition relations, $\longrightarrow \subseteq \mu\mathsf{SC} \times 2^{\Pi} \times 2^{\Pi\cup\neg\Pi} \times \mu\mathsf{SC}$ and $\xrightarrow{\sigma} \subseteq \mu\mathsf{SC} \times \mu\mathsf{SC}$, are defined via SOS rules.

Table 3: Operational rules: action transitions

$$\text{OR1}\quad \frac{—}{[n:\vec{s};l;T] \xrightarrow[\neg\mathsf{trg}^-(t)\cup\neg\mathsf{act}(t)]{\mathsf{trg}^+(t)}_{\bullet} [n:\vec{s};t;T]}\ \mathsf{source}(t)=s_l \qquad \text{OR2}\quad \frac{s_l \xrightarrow[N]{E}_{\alpha} s'_l}{[n:\vec{s};l;T] \xrightarrow[N]{E}_{\blacktriangledown\cdot\alpha} [n::\vec{s}_{[l\mapsto s'_l]};l;T]}$$

$$\text{AND}\quad \frac{\exists 1\le l\le|\vec{s}|.\ s_l \xrightarrow[N]{E}_{\alpha} s'_l}{[n:\vec{s}] \xrightarrow[N]{E\setminus\bigcup_{j\neq l}\mathsf{out}(s_j)}_{\|_l\cdot\alpha} [n:\vec{s}_{[l\mapsto s'_l]}]}\ N\cap\bigcup_{j\neq l}\mathsf{out}(s_j)=\emptyset \qquad \text{OR3}\quad \frac{s_l \xrightarrow[N]{E}_{\alpha} s'_l}{[n::\vec{s};l;T] \xrightarrow[N]{E}_{\blacktriangledown\cdot\alpha} [n::\vec{s}_{[l\mapsto s'_l]};l;T]}$$

Table 4: Operational rules: clock transitions

$$\text{cBAS}\quad \frac{—}{[n]\xrightarrow{\sigma}[n]} \qquad \text{cOR1}\quad \frac{—}{[n:\vec{s};t;T]\xrightarrow{\sigma}[n:\vec{s}_{[l\mapsto \mathsf{default}(s_l)]};l;T]}\ \mathsf{target}(t)=s_l \qquad \text{cOR3}\quad \frac{s_l\xrightarrow{\sigma}s'_l}{[n::\vec{s};l;T]\xrightarrow{\sigma}[n:\vec{s}_{[l\mapsto s'_l]};l;T]}$$

$$\text{cOR2}\quad \frac{—}{[n:\vec{s};l;T]\xrightarrow{\sigma}[n:\vec{s};l;T]}\ [n:\vec{s};l;T]\not\xrightarrow[\emptyset]{\emptyset} \qquad \text{cAND}\quad \frac{\forall 1\le l\le|\vec{s}|.\ s_l\xrightarrow{\sigma}s'_l}{[n:\vec{s}]\xrightarrow{\sigma}[n:\vec{s}\,']}\ [n:\vec{s}]\not\xrightarrow[\emptyset]{\emptyset}$$

The operational rules for action transitions are given in Table 3, where the subscript of the transition relation should be ignored for now; the subscript will only be needed in Section 4.3. For convenience, we write $s \xrightarrow[N]{E} s'$ instead of $\langle s, E, N, s'\rangle \in\ \longrightarrow$. Moreover, we let $\vec{s}$ stand for the sequence $(s_1, \dots, s_k)$ and write $|\vec{s}|$ for k. Rule (OR1) states that or–state $[n : \vec{s}; l; T]$ can evolve to $[n:\vec{s};t;T]$ if transition t is enabled, i.e., if (i) the source state of t is the currently active state s_l, (ii) all its positive trigger events $\mathsf{trg}^+(t)$ are offered by the environment, (iii) the positive counterparts of all its negated trigger events $\mathsf{trg}^-(t)$ are not offered by the environment, and (iv) the negated events corresponding to $\mathsf{act}(t)$ are not offered by the environment, i.e., no transition within the same macro step has already fired due to the absence of such an event. The latter is necessary for implementing global consistency in our semantics. Rules (OR2) and (OR3) deal with the case that an inner transition of the active sub–state s_l of the considered or–state is executed. Hence, sub–state s_l needs to be updated accordingly. The resulting micro term $[n :: \vec{s}_{[l\mapsto s'_l]}; l; T]$ also reflects — via the double colons — that a transition originating within the or–state has been executed, in which case the or–state may no longer engage in a transition in T during the same macro step, i.e., before executing the next clock transition. Finally, Rule (AND) deals with and–states. If sub–state s_l fires a transition $s_l \xrightarrow[N]{E} s'_l$, then the and–state can do so as well, provided that no event in N is offered by some other sub–state (cf. the rule's side condition). Moreover, for triggering the transition in the context of the and–state only those events $e\in E$ need to be offered by the environment, which are not already offered by some other 'parallel' sub–state of the and–state, i.e., for which $e \in E\setminus\bigcup_{j\neq l}\mathsf{out}(s_j)$ holds.

Clock transitions are defined by the rules in Table 4, which use the notation $s \xrightarrow{\sigma} s'$ for $\langle s, s'\rangle \in\ \xrightarrow{\sigma}$. Intuitively, a clock transition models the completion of a macro step by updating the active states in the considered micro term according to the transitions that have been executed in the macro step. Due to the synchrony hypothesis of Statecharts, this implies in particular that a clock transition can only be performed if the considered Statecharts term s cannot autonomously engage in a further action transition, i.e., if $s \not\xrightarrow[\emptyset]{\emptyset}$ holds, which stands for $\nexists s'.\ s \xrightarrow[\emptyset]{\emptyset} s'$. Note that both event sets in the label must be empty; otherwise, the action transition is not enabled with respect to all potential system environments and our semantics would not be compositional. In this vein, Rule (cBAS) states that a basic state can always accept a clock tick as it does not possess any (enabled) transitions. Rule (cOR1) reflects the update of the micro term $[n:\vec{s};t;T]$ representing an or–state after transition $t\in T$ has fired. More precisely, the sub–state of the considered or–state is updated to the target state s_l of t, where all active states of s_l are reset to their initial states. In case that no transition of the considered or–state has been executed — i.e., the or–state is represented by micro term $[n : \vec{s}; l; T]$ — and no one is enabled — i.e., $[n : \vec{s}; l; T] \not\xrightarrow[\emptyset]{\emptyset}$ holds —, a clock tick can be accepted and does not result in any change of state (cf. Rule (cOR2)). Rule (cOR3) formalizes the behavior that an or–state can engage in a clock transition if its *active* sub–state can engage in one. Finally, Rule (cAND) states that an and–state can engage in a clock transition if *all* its sub–states can, provided that there is no action transition whose execution cannot be prevented, i.e., provided that $[n:\vec{s}] \not\xrightarrow[\emptyset]{\emptyset}$ holds.

The proof that the resulting semantics is compositional can easily be inferred by studying the structure of our SOS rules [35].

3.3 Macro Step and Coincidence Result

The above rules provide a compositional semantics of Statecharts on the micro–step level. However, our consideration of a global abstract clock allows us to retrieve the classical macro–step semantics of Statecharts, as mentioned at the beginning of Section 3.

DEFINITION 1 (MACRO STEP). *For $s, s' \in \mathsf{SC}$ as well as $E, A \subseteq \Pi$ we write $s \overset{E}{\underset{A}{\Longrightarrow}} s'$ and say that s performs a* macro step

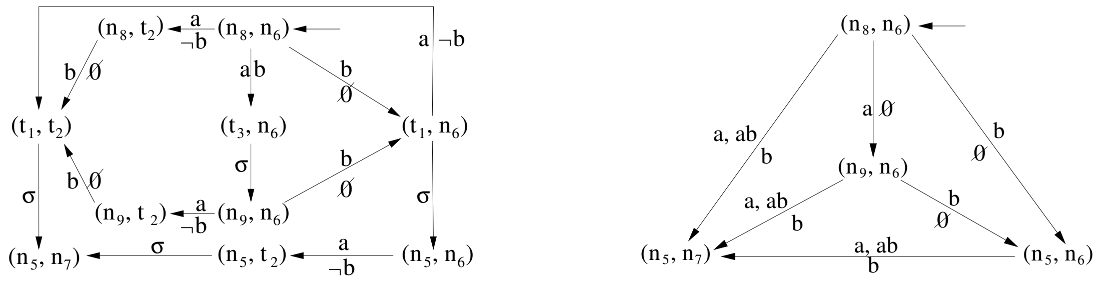

Figure 3: Our semantics (left) and the macro–step semantics (right) for the Statechart in Figure 1.

with input E and output A to s', if there exist $s_1, \dots, s_m \in \mu SC$, $E_1, \dots, E_m \subseteq \Pi$, and $N_1, \dots, N_m \subseteq \Pi \cup \neg\Pi$, for some $m \in \mathbb{N}$, such that

1. $s \xrightarrow[N_1]{E_1} s_1 \xrightarrow[N_2]{E_2} \dots \xrightarrow[N_m]{E_m} s_m \xrightarrow{\sigma} s'$,
2. $\bigcup_{i=1}^m E_i \subseteq E$,
3. $\bigcup_{i=1}^m N_i \cap E = \emptyset$,
4. $A = \mathit{out}(s_m) \cap \Pi$, *and*
5. $\nexists s_{m+1}, E_{m+1}, N_{m+1}.\ s_m \xrightarrow[N_{m+1}]{E_{m+1}} s_{m+1}$, $E_{m+1} \subseteq E$, *and* $N_{m+1} \cap E = \emptyset$.

While Conds. (2) and (3) guarantee that all considered action transitions are enabled by the environment, Cond. (5) ensures the maximality of the macro step, i.e., it implements the synchrony hypothesis. Now, we can establish the desired result, namely that our macro–step semantics coincides with the classical macro–step semantics of Statecharts.

THEOREM 3.1 (COINCIDENCE RESULT). *Let $s, s' \in SC$ as well as $E, A \subseteq \Pi$. Then, $s \overset{E}{\underset{A}{\Rrightarrow}} s'$ if and only if $s \overset{E}{\underset{A}{\Rightarrow}} s'$.*

PROOF. (*Sketch.*) Consider the following construction. If $T = (t_1, \dots, t_m)$ is a sequence of Statecharts transitions of $s \in \mathsf{SC}$ generated by the step–construction function relative to environment $E \subseteq \Pi$ and satisfying $A = \bigcup_{l=1}^m \mathsf{out}(t_l)$, then there exists a sequence of m action transitions as described in Def. 1, such that the l-th action transition corresponds to the execution of t_l in s. Vice versa, assume that the conditions of Def. 1 are satisfied for some $E \subseteq \Pi$ and that $T = (t_1, \dots, t_m)$ is the sequence of Statecharts transitions which can be identified with the considered sequence of action transitions starting in s. Then, T can be generated by the step–construction function relative to s and E, where the transitions fire in the order indicated by *sequence* T. □

3.4 Example

We now return to our example Statechart of Figure 1. Our semantics of this Statechart and its classical macro–step semantics are depicted on the left and right in Figure 3, respectively. In both diagrams, we represent a transition of the form $s \xrightarrow[N]{E} s'$ by writing E to the left of or above the arrow and N to the right of or below the arrow. We also abbreviate a set of events by listing its elements, e.g., writing ab for $\{a, b\}$, and denote alternatives for E at the same arrow by separating them by commas. Finally, we employ our notation introduced in Section 2.2 and additionally write t for the micro term $[n : \vec{s}; t; T]$.

Table 5: Revision of function out

$\mathsf{out}([n])$	$=_{\text{df}}$	$\{\mathsf{in}(n)\}$
$\mathsf{out}([n : \vec{s}; l; T])$	$=_{\text{df}}$	$\mathsf{out}(s_l) \cup \{\mathsf{in}(n)\}$
$\mathsf{out}([n :: \vec{s}; l; T])$	$=_{\text{df}}$	$\mathsf{out}(s_l) \cup \{\mathsf{in}(n)\}$
$\mathsf{out}([n : (s_1, \dots, s_k)])$	$=_{\text{df}}$	$\bigcup_{l=1}^k \mathsf{out}(s_l) \cup \{\mathsf{in}(n)\}$
$\mathsf{out}([n : \vec{s}; t; T])$	$=_{\text{df}}$	$\mathsf{act}(t) \cup \mathsf{trg}^-(t) \cup \{\mathsf{in}(n)\}$

The right diagram displayed in Figure 3 includes the macro steps $\langle n_8, n_6 \rangle \overset{\{a\}}{\underset{\emptyset}{\Rrightarrow}} \langle n_9, n_6 \rangle$ and $\langle n_8, n_6 \rangle \overset{\{a\}}{\underset{\{b\}}{\Rrightarrow}} \langle n_5, n_7 \rangle$ which we already considered in Section 2.2. According to Thm. 3.1, both macro steps can be explained in terms of sequences of micro steps displayed on the left in Figure 3, which start with state $\langle n_8, n_6 \rangle$ and end with the execution of a clock transition. The first macro step is given by the sequence $\langle n_8, n_6 \rangle \xrightarrow[\{b\}]{\{a\}} \langle t_3, n_6 \rangle \xrightarrow{\sigma} \langle n_9, n_6 \rangle$, where $\mathsf{out}(\langle t_3, n_6 \rangle) = \{\neg b\}$, and the second macro step is encoded by the sequence $\langle n_8, n_6 \rangle \xrightarrow[\{\neg b\}]{\{a\}} \langle n_8, t_2 \rangle \xrightarrow[\emptyset]{\{b\}} \langle t_1, t_2 \rangle \xrightarrow{\sigma} \langle n_5, n_7 \rangle$, where $\mathsf{out}(\langle t_1, t_2 \rangle) = \{b\}$.

4. EXTENSIONS

We now illustrate the flexibility of our approach by adapting it to incorporate features offered by many popular Statecharts variants, namely *state references*, *history mechanisms*, and *priority concepts* along the or–state hierarchy.

4.1 State References

Many Statecharts variants permit trigger events of the form $\mathsf{in}(n)$, for $n \in \mathcal{N}$, which are satisfied whenever state n is active. In our setting, we may encode this feature via the employed communication scheme. To do so, we first extend the set Π of events by the distinguished events $\mathsf{in}(n)$, for all $n \in \mathcal{N}$. Moreover, the sets $\mathsf{out}(s)$, for $s \in \mu\mathsf{SC}$, need to be re–defined — as shown in Table 5 — such that they include the events $\mathsf{in}(n)$, for any active state n in s. It is easy to see that the resulting semantics handles state references as expected.

4.2 History States

Upon entering or–states, their initial states are activated. However, in practice it is often convenient to have the option to return to the sub–state which was active when last exiting an or–state, e.g., after completing an interrupt routine. In Statecharts' visual syntax this is done by permitting distinguished *history states* in or–states to which transitions from the outside of the considered or–states may point. Such history states can have two flavors: *deep* and *shallow*. Deep means that the 'old' active state of the or–state *and* the 'old' active states of all its sub–states are restored. Shallow means that only the active state of the or–state is restored and that its sub–states are reinitialized as usual. In our term–based setting, we may model history states and transitions traversing to history states as follows. For each transition t pointing to some or–state s, we additionally record a *history flag* $\rho \in \{\mathit{none}, \mathit{deep}, \mathit{shallow}\}$. If $\rho = \mathit{none}$, then transition t is interpreted in the standard way, otherwise it is interpreted to point to the deep — if $\rho = \mathit{deep}$ — or shallow — if $\rho = \mathit{shallow}$ — history state in s.

In the light of this formalization, it is easy to integrate a history mechanism in our operational semantics. One just has to replace function $\mathsf{default}(s_j)$ in Rule (cOR1) by function $\mathsf{default}(\rho, s_j)$, where $\rho \in \{\mathit{none}, \mathit{deep}, \mathit{shallow}\}$ is the history flag of transition t. The terms $\mathsf{default}(\mathit{none}, s)$ and $\mathsf{default}(\mathit{deep}, s)$ are simply defined by $\mathsf{default}(s)$ and s, respectively. The definition of $\mathsf{default}(\mathit{shallow}, s)$ can be done along the structure of Statecharts terms as follows

1. $\mathsf{default}(\mathit{shallow}, [n]) =_{\mathrm{df}} [n]$,
2. $\mathsf{default}(\mathit{shallow}, [n : \vec{s}; l; T]) =_{\mathrm{df}} [n : \vec{s}_{[l \mapsto \mathsf{default}(s_l)]}; l; T]$,
3. $\mathsf{default}(\mathit{shallow}, [n : \vec{s}]) =_{\mathrm{df}} [n : \mathsf{default}(\mathit{shallow}, \vec{s})]$.

Here, $\mathsf{default}(\mathit{shallow}, \vec{s})$, where $\vec{s} = (s_1, \ldots, s_k)$, stands for $(\mathsf{default}(\mathit{shallow}, s_1), \ldots, \mathsf{default}(\mathit{shallow}, s_k))$. Observe that $\mathsf{default}(\rho, s)$ needs only to be defined for Statecharts terms $s \in \mathsf{SC}$ and not also for micro terms $s \in \mu\mathsf{SC} \setminus \mathsf{SC}$.

4.3 Priority Concepts

Some Statecharts dialects consider an implicit priority mechanism along the hierarchy of or–states. In UML Statecharts [3], for example, inner transitions of an or–state have priority over outer transitions, while this is the other way around in STATEMATE [10]. Let us provide a flexible scheme for encoding both priority concepts, for which we introduce the notion of *addresses* which are built according to the BNF $\alpha \ ::= \ \bullet \ | \ \blacktriangledown \cdot \alpha \ | \ \|_l \cdot \alpha$, for $l \in \mathbb{N}$. The set of all such addresses is denoted by $\mathcal{A}\mathrm{ddr}$. Each action transition is then labeled with an address pointing to *the* sub–term of the considered Statecharts term, from which the transition originates (cf. the subscripts of the transitions in Table 3). Intuitively, the symbol $\bullet$ encodes that the transition originates from the considered state, i.e., this state must be an or–state and the transition leaves the or–state's active sub–state. Address $\blacktriangledown \cdot \alpha$ also requires the state to be an or–state and the transition to originate from address α of the currently active sub–state of the or–state. Finally, address $\|_l \cdot \alpha$ indicates that the considered state is an and–state having at least l sub–states and that the transition originates from address α of the l-th sub–state.

Given an address $\alpha \in \mathcal{A}\mathrm{ddr}$, we can now define the set $\mathsf{MI}(\alpha)$ of addresses which are considered *more important* than α according to the chosen priority concept. The definitions of $\mathsf{MI}(\alpha)$ for the priority concepts of UML Statecharts and STATEMATE can be done straightforwardly along the structure of α and are given in Table 6. They do not require any extra explanation. Now, we can define a

Table 6: Priority Structure à la UML (top) and à la STATEMATE (bottom)

$$\begin{array}{lll}
\mathsf{MI}(\bullet) & =_{\mathrm{df}} & \{\blacktriangledown \cdot \beta \mid \beta \in \mathcal{A}\mathrm{ddr}\} \\
\mathsf{MI}(\blacktriangledown \cdot \alpha) & =_{\mathrm{df}} & \{\blacktriangledown \cdot \beta \mid \beta \in \mathsf{MI}(\alpha)\} \\
\mathsf{MI}(\|_l \cdot \alpha) & =_{\mathrm{df}} & \{\|_l \cdot \beta \mid \beta \in \mathsf{MI}(\alpha)\} \\
\hline
\mathsf{MI}(\bullet) & =_{\mathrm{df}} & \emptyset \\
\mathsf{MI}(\blacktriangledown \cdot \alpha) & =_{\mathrm{df}} & \{\bullet\} \cup \{\blacktriangledown \cdot \beta \mid \beta \in \mathsf{MI}(\alpha)\} \\
\mathsf{MI}(\|_l \cdot \alpha) & =_{\mathrm{df}} & \{\|_l \cdot \beta \mid \beta \in \mathsf{MI}(\alpha)\}
\end{array}$$

new transition relation $\longrightarrow\!\!\!\!\rightarrow$ for action transitions, which coincides with the original transition relation given in Section 3, except that low–priority action transitions are filtered out.

$$\text{Prio} \quad \frac{s \xrightarrow[N]{E}_\alpha s'}{s \overset{E}{\underset{N}{\twoheadrightarrow}}_\alpha s'} \quad \nexists \beta \in \mathsf{MI}(\alpha).\ s \xrightarrow[\emptyset]{\emptyset}_\beta$$

This rule states that an action transition located at address α may be executed if there exists no action transition at some more important address β, which cannot be prevented in any system environment. The justification for the fact that only action transitions with empty sets as labels have pre–emptive power over lower prioritized action transition is similar to the one regarding the pre–emption of clock transitions in Section 3. One might wonder why this "two–level" definition of Statecharts semantics is still compositional, as the above side condition concerns a global property. In order to see this, one may again employ meta–theoretic results about the compositionality of semantics defined via SOS rules [35]. Alternatively, one can distribute the side condition among the original rules for action transitions, such that compositionality becomes obvious, as is done in approaches to priority in process algebras [4]. Some details on this issue can be found in the appendix.

5. RELATED WORK

We categorize related work along the three dimensions of Statecharts semantics: causality, synchrony, and compositionality. This classification was first considered by Huizing and Gerth [15] who demonstrated that these dimensions cannot be trivially combined within a simple semantic framework.

The original Statecharts semantics, as presented by Harel et al. [11], obeys causality and synchrony. However, it ignores compositionality and the concept of global consistency. Later, Huizing et al. [16] provided a compositional denotational semantics for this variant, while Pnueli and Shalev [29] suggested the introduction of global consistency for improving the practicality of the variant. However, Pnueli and Shalev's formalization is again not compositional. Its compositionality defect has recently been analyzed by adapting ideas from intuitionistic logic, as it can be shown that the logic underlying Pnueli and Shalev's semantics does not respect the *Law of the Excluded Middle* [19].

Other researchers have developed languages whose semantics obey the synchrony hypothesis and compositionality but violate causality. Prominent representatives of such languages include Berry's ESTEREL [2], to which recently some dialect of Statecharts has been interfaced as graphical front–end [32], and Maraninchi's ARGOS [22]. Both languages are deterministic and treat causality rather conservatively in a pre–processing step, before determining the semantics of the considered program as Mealy automaton via structural operational rules [23]. Moreover, ARGOS semantics significantly differs from Statecharts semantics by allowing sequential components to fire more than once within a macro step. Another approach to formalizing Statecharts, which fits into this category, is the one of Scholz [30] who uses streams as semantic domain for defining a non–causal fixed point semantics.

The popular synchronous version of STATEMATE [10] neglects the synchrony hypothesis. Events generated in one step may not be consumed within the same step but in the next step only. The operational semantics of this dialect has been compositionally formalized by Damm et al. [7]. It was also considered by Mikk et al. [25] who translated STATEMATE specifications to input languages of model–checking tools by using hierarchical automata [26] as intermediate language. This intermediate language was employed by Latella et al. [17], too, for formalizing the semantics of UML *Statecharts* [3] in terms of Kripke structures. However, UML Statecharts discard not only the synchrony hypothesis but additionally negated events and, thereby, make the notion of global consistency obsolete. Their semantics was also investigated by Paltor and Lilius [27], who developed a semantic framework on the basis of a term–rewriting system.

Our work is, however, most closely related to approaches which aim at combining all three dimensions — causality, synchrony, and compositionality — within a single formalism. These approaches may be split into two classes. The first class adapts a process–algebraic approach, where Statecharts languages are embedded in process algebras, for which structured operational semantics based on labeled transition systems are defined. Uselton and Smolka [34] have pioneered this approach which was then refined by Levi [18]. Their notion of transition system involves complex labels of the form $\langle E, \prec \rangle$, where E is a set of events and $\prec$ is a transitive, irreflexive order on 2^E encoding causality. The second class is characterized by following essentially the same ideas but avoiding the indirection of process algebra. Research by Uselton and Smolka [33] again employs the mentioned partial order, whereas Maggiolo–Schettini et al. [21] require even more complex and intricate information about causal orderings, global consistency, and negated events. While our present work also fits into this class, although it originated in the former [20], it avoids complex labels by representing causality via micro–step sequences and by adding explicit clock transitions to retrieve macro–step information. Thereby, our semantics is not only simple and concise but also comprehensible and suited for interfacing Statecharts to existing analysis and verification tools. In addition, our approach is very flexible as we demonstrated by adding several prominent features, namely state references, history states, and priority concepts, to our initially rather primitive Statecharts dialect.

An alternative compositional semantics for Statecharts is presented by von der Beeck in [37]. That account is based on the one presented here, although its practical utility is compromised by the inclusion of unnecessary syntactic information in the semantic relation. In particular, the inclusion of transition names into the labels of execution steps allows Statecharts exhibiting the same observational behavior but differing in the naming of transitions to be distinguished.

Finally, we briefly comment on interlevel transitions which prohibit a compositional Statecharts semantics as they are based on the idea of "goto–programming." First of all, interlevel transitions jeopardize a strictly structural definition of Statecharts terms, which is a prerequisite for deriving any compositional semantics. Hence, for modeling interlevel transitions, the syntax of Statecharts must be changed in a way such that interlevel transitions may be represented by several intralevel transitions which are connected via dedicated *ports*. This can be done either explicitly, as in the *Communicating Hierarchical State Machine* language introduced by Alur et al. [1], or implicitly via a synchronization scheme along the hierarchy of or–states, as in Maraninchi's ARGOS [22].

6. CONCLUSIONS

This paper presented a new approach to formalizing Statecharts semantics, which is centered around the principle of compositionality and borrows from ideas developed for timed process algebras. In contrast to related work, our approach combines all desired features of Statecharts semantics, namely causality, synchrony, and compositionality, within a single formalism, while still being simple and comprehensible. Its foundation on structural operational rules guarantees that our semantics is easy to implement in specification and verification tools and that it can be adapted to several Statecharts dialects. The proposed semantic framework also permits the integration of many features desired in practice, as we demonstrated by extending it to dealing with state references, history states, and priority concepts. Last, but not least, we hope that this paper testifies to the utility of applying knowledge from the field of Concurrency Theory to formalizing practical specification languages rigorously yet clearly.

7. ACKNOWLEDGMENTS

We would like to thank the anonymous referees for their valuable comments and suggestions.

8. REFERENCES

[1] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *ICALP '99*, volume 1644 of *LNCS*, pages 169–178. Springer-Verlag, 1999.

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

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.

[4] R. Cleaveland and M. Hennessy. Priorities in process algebra. *Inform. & Comp.*, 87(1/2):58–77, 1990.

[5] R. Cleaveland, E. Madelaine, and S. Sims. Generating front-ends for verification tools. In *TACAS '95*, volume 1019 of *LNCS*, pages 153–173. Springer-Verlag, 1995.

[6] R. Cleaveland and S. Sims. The NCSU CONCURRENCY WORKBENCH. In *CAV '96*, volume 1102 of *LNCS*, pages 394–397. Springer-Verlag, 1996.

[7] W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In de Roever et al. [8], pages 186–238.

[8] W.-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference*, volume 1536 of *LNCS*. Springer-Verlag, 1997.

[9] D. Harel. Statecharts: A visual formalism for complex systems. *SCP*, 8:231–274, 1987.

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

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

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

[13] M. Hennessy and T. Regan. A process algebra for timed systems. *Inform. & Comp.*, 117:221–239, 1995.

[14] G. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23:279–295, 1997.

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

[16] C. Huizing, R. Gerth, and W.-P. de Roever. Modeling Statecharts behavior in a fully abstract way. In *CAAP '88*, volume 299 of *LNCS*, pages 271–294. Springer-Verlag, 1988.

[17] D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML Statechart diagrams. In *FMOODS '99*. Kluwer, 1999.

[18] F. Levi. *Verification of Temporal and Real-time Properties of Statecharts*. PhD thesis, University of Pisa, 1997.

[19] G. Lüttgen and M. Mendler. Fully-abstract Statecharts semantics via intuitionistic Kripke models. In *ICALP 2000*, volume 1853 of *LNCS*, pages 163–174. Springer-Verlag, 2000.

[20] G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via process algebra. In *CONCUR '99*, volume 1664 of *LNCS*, pages 399–414. Springer-Verlag, 1999.

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

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

[23] F. Maraninchi and N. Halbwachs. Compositional semantics of non-deterministic synchronous languages. In *ESOP '96*, volume 1058 of *LNCS*, pages 235–249. Springer-Verlag, 1996.

[24] K. McMillan. *Symbolic Model Checking: An Approach to the State-Explosion Problem*. PhD thesis, Carnegie-Mellon University, 1992.

[25] 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*. Springer-Verlag, 1997.

[26] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for Statecharts. In *ASIAN '97*, volume 1345 of *LNCS*. Springer-Verlag, 1997.

[27] I. Paltor and J. Lilius. Formalising UML state machines for model checking. In *UML '99*, volume 1723 of *LNCS*, pages 430–445. Springer-Verlag, 1999.

[28] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Aarhus University, 1981.

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

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

[31] B. Selic, G. Gullekson, and P. Ward. *Real-time Object Oriented Modeling and Design*. J. Wiley & Sons, 1994.

[32] S. Seshia, R. Shyamasundar, A. Bhattacharjee, and S. Dhodapkar. A translation of Statecharts to ESTEREL. In *FM '99*, volume 1708 of *LNCS*, pages 983–1007. Springer-Verlag, 1999.

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

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

[35] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic J. of Comp.*, 2(2):274–302, 1995.

[36] M. von der Beeck. A comparison of Statecharts variants. In *FTRTFT '94*, volume 863 of *LNCS*, pages 128–148. Springer-Verlag, 1994.

[37] M. von der Beeck. A concise compositional Statecharts semantics definition. In *FORTE/PSTV 2000*. Kluwer, 2000. To appear.

APPENDIX

A. REVISED OPERATIONAL RULES

In this appendix, we show that our semantics, when incorporating some priority concept along the hierarchy of or–states, does not need to be defined in two levels, as is done in Section 4.3. Instead one may modify the rules of action transitions presented in Section 3 to achieve a single–level semantics. However, this can only be done when having a specific priority concept — e.g., à la UML Statecharts [3] or à la STATEMATE [12] — in mind and is not as elegant as the approach presented in the main part of the paper.

If one considers the priority concept of UML Statecharts, one has to replace Rules (OR1) and (AND) by Rules (OR1') and (AND'), which are displayed in Table 7, in order to obtain a single–level semantics. In case of STATEMATE's priority concept, one must substitute Rules (OR2) and (AND) by Rules (OR2') and (AND'). It is easy to inspect that the new sets of rules lead to *compositional* semantics. The more complex side conditions in the rules presented in Table 7, when compared to the ones in the original rules, correspond to the "localizations" of the side condition of Rule (Prio) introduced in Section 4.3 and are self–explanatory. The modified

Table 7: Revised operational rules for action transitions

$$\text{OR1'}\quad \frac{-}{[n:\vec{s};l;T] \xrightarrow[\neg\mathsf{trg}^{-}(t)\cup\neg\mathsf{act}(t)]{\mathsf{trg}^{+}(t)}_{\bullet} [n:\vec{s};t;T]} \quad \mathsf{source}(t)=s_l, \forall\alpha\in\mathcal{A}\mathrm{ddr}.\ s_l \not\xrightarrow[\emptyset]{\emptyset}_{\alpha}$$

$$\text{OR2'}\quad \frac{s_l \xrightarrow[N]{E}_{\alpha} s'_l}{[n:\vec{s};l;T] \xrightarrow[N]{E}_{\blacktriangledown\cdot\alpha} [n::\vec{s}_{[l\mapsto s'_l]};l;T]} \quad \nexists t\in T.\ \mathsf{trg}(t)=\emptyset$$

$$\text{AND'}\quad \frac{\exists 1\leq l\leq|\vec{s}|.\ s_l \xrightarrow[N]{E}_{\alpha} s'_l}{[n:\vec{s}] \xrightarrow[N]{E\setminus\bigcup_{j\neq l}\mathsf{out}(s_j)}_{\|_l\cdot\alpha} [n:\vec{s}_{[l\mapsto s'_l]}]} \quad \begin{array}{l} N\cap\bigcup_{j\neq l}\mathsf{out}(s_j)=\emptyset, \\ \nexists\beta\in\mathsf{MI}(\alpha), s''_l, E', N'.\ s_l \xrightarrow[N']{E'}_{\beta} s''_l, E'\subseteq\bigcup_{j\neq l}\mathsf{out}(s_j), N'\cap\bigcup_{j\neq l}\mathsf{out}(s_j)=\emptyset \end{array}$$

transition relations for action transitions are equivalent to the transition relations $\longrightarrow\!\!\!\!\rightarrow$ introduced in Section 4.3, in both the UML Statecharts and the STATEMATE setting.

THEOREM A.1. *Let* $s, s' \in \mu\mathit{SC}$*,* $E\subseteq\Pi$*,* $N\subseteq\Pi\cup\neg\Pi$*, and* $\alpha\in\mathcal{A}\mathit{ddr}$*. Then* $s \xrightarrow[N]{E}_{\alpha} s'$ *if and only if* $s \xrightarrow[N]{E}\!\!\!\!\rightarrow_{\alpha} s'$*.*

Consequently, the operational semantics presented in Section 4.3 is compositional. The proof of this theorem bears no theoretical complexity and can be done along the structure of s. Similar proofs and constructions are standard in process–algebraic frameworks with pre–emption (see [4]).