Softw Syst Model (2002) 1: 130–141 / Digital Object Identifier (DOI) 10.1007/s10270-002-0012-8

A structured operational semantics for UML-statecharts

Michael von der Beeck

BMW Group, Funktionsentwicklungsprozess, Max-Diamand-Str. 5, 80937 München, Germany;
E-mail: Michael.Beeck@bmw.de

Initial submission: 19 February 2002/Revised submission: 28 October 2002
Published online: 2 December 2002 –

Abstract. The Unified Modeling Language (UML) has gained wide acceptance in very short time because of its variety of well-known and intuitive graphical notations. However, this comes at the price of an unprecise and incomplete semantics definition. This insufficiency concerns single UML diagram notations on their own as well as their integration. In this paper, we focus on the notation of UML-statecharts. Starting with a precise textual syntax definition, we develop a precise structured operational semantics (SOS) for UML-statecharts. Besides the support of interlevel transitions and in contrast to related work, our semantics definition supports characteristic UML-statechart features like the history mechanism as well as entry and exit actions.

Keywords: Statecharts – UML – UML-statecharts – Formal semantics – Structured operational semantics (SOS) – Labeled transition systems

1 Introduction

The Unified Modeling Language (UML) [23] has become a very successful analysis and design language in very short time. It already constitutes the de-facto-standard for industrial applications in many areas – especially in the object-oriented domain, but it also gains in importance for the modeling of embedded real-time systems. Its very advantages are given by a great variety of intuitive and mostly well-known notations for different kind of information to be specified: requirements, static structure, interactive and dynamic behaviour as well as physical implementation structures. However, this intuitive appeal suffers from an insufficient definition. Whereas the UML syntax is defined in quite a precise and complete manner, its semantics is not. The official UML documentation [17] contains a chapter titled "semantics", but on the one hand a considerable part of it only considers what usually is called "static semantics" in compiler theory, i.e. the parts of the syntax which are not context-free, on the other hand the parts which in fact consider semantics contain English prose and therefore lack preciseness and completeness. This serious insufficiency concerns single diagram notations of the UML like statecharts, sequence diagrams, and class diagrams as well as their integration. In this paper we focus on the statecharts version contained in UML – in the following denoted as UML-statecharts.

The "classical" statecharts language, developed by Harel [6], constitutes a very successful intuitive graphical state-based specification language which enhances finite automata by hierarchy, concurrency, and a sort of broadcast communication. However, its semantics definition required tremendous efforts. A lot of work deals with precisely defining the statecharts semantics in such a way that certain semantic properties – statecharts specific ones like causality as well as general ones like compositionality – are fulfilled [7, 13–16, 21, 25, 26]. Some of the semantic difficulties therein also exist for UML-statecharts e.g. with respect to interlevel transitions, whereas some intricacies – e.g. those caused by set-valued events or negated events – do not exist. However, in comparison with the huge amount of published work on classical statecharts only a relatively small amount of work has been published on UML-statecharts up to now.

The main contribution of this paper consists of a formal, structured operational semantics (SOS) definition for a subset of UML-statecharts. The SOS-approach of Plotkin [20] combines an intuitive operational understanding with precisenenss and supports the definition of a compositional semantics.

Our work is partly based on earlier work of Latella, Majzik, and Massink [10] for UML-statecharts, but to some extent also on previous work for classical state-

charts, namely on the work of Mikk, Lakhnech, and Siegel [16], on the work of Lüttgen, von der Beeck, and Cleaveland [13, 14], as well as on our previous work [26]. Furthermore, this article represents an elaborated version of [27].

In contrast to the work of Latella et al. [10][1] our approach

- includes the history mechanism (shallow and deep cases),[2]
- supports entry and exit actions,
- uses statechart terms for the syntax definition which are "more similar" to the original UML-statecharts syntax than Enhanced Hierarchical Automata (EHA) which are used in [10] (and [16]),
- enables a more liberal modeling of interlevel transitions, because we use complete as well as incomplete configurations for specifiying the source states and the target states of interlevel transitions, and
- reduces the complexity of the semantics definition by reducing redundant information.

As opposed to our above-mentioned earlier work [27] this contribution presents the following improvements:

- consideration of sequences of UML-statechart transition executions instead of only single UML-statechart transition executions by use of Kripke structures as the semantic domain,
- introduction of nondeterminism in the semantics definition concerning the execution of
  - entry and exit actions in And-terms and
  - action parts in And-terms,
- redefinition of subconfigurations as well as of (complete and incomplete) potential configurations,
- comprehensive explanation of taken design decisions concerning our syntax and semantics definition as e.g.
  - only local (as opposed to global) separation of structural and behavioural information,
  - use of stuttering rules in the formal semantics definition, and
  - avoidance of redundancy.

The rest of the paper is organized as follows: We briefly discuss differences between classical statecharts and UML-statecharts in Sect. 2. In Sect. 3 we define our textual syntax of UML-statecharts, whereas in Sect. 4 we present our formal operational semantics for UML-statecharts. Related work is discussed in Sect. 5. We conclude and discuss future work in Sect. 6.

2 UML-statecharts versus classical statecharts

Some of the problems inherent in classical statecharts also exist in UML-statecharts. However, there are some important differences which simplify or complicate the semantics definition, respectively. In contrast to classical statecharts, the UML-statechart language shows the following features:

- Only one input event is processed at each point of time. Especially, a generated event is not sensed within the same "step", i.e. it can not trigger a transition within the same step.
- The trigger part of a transition label neither contains conjunctions of events nor negated events.
- If an event is taken from the input event queue which does not enable any transitions in the currently active state, then this event is simply ignored.[3]
- If several conflicting transitions, i.e. transitions which must not be taken simultaneously, are simultaneously enabled, then a transition on a lower level has priority over transitions on a higher level (for short: lower-first priority). In the case of classical statecharts either no priority or upper-first priority (e.g. in the STATEMATE tool) is given.
- Entry/exit actions associated to states exist which are executed whenever the corresponding state is entered or exited, respectively.

The first two restrictions considerably simplify the semantics definition of UML-statecharts in comparison with classical statecharts. For example the problem of achieving global consistency does not occur. In general, causality conflicts – caused by cyclic transition executions – do not exist.

We take all of these differences into account when defining the UML-statecharts semantics. Furthermore, we also consider the history mechanism which has been presented by Harel [6], but which has been neglected in most of the semantics definitions of classical statecharts.

3 Syntax

UML-statechart terms. UML-statecharts is a visual language. However, for our aim to define a formal semantics, it is convenient to represent UML-statecharts not visually but by textual terms. This is also done in related work for classical statecharts [15, 25] as well as for UML-statecharts [10]. Essentially, our term syntax enhances the syntax presented in [15] by entry and exit actions, by a possibility to model interlevel transitions, and by a possibilty to specify different history types.

Let $\mathcal{N}, \mathcal{T}, \Pi, \mathcal{A}$ be countable sets of state names, transition names, events, and actions, respectively, with $\Pi \subseteq \mathcal{A}$.[4] We denote events and actions by $a, b, c, \ldots$ and sequences of events as well as sequences of actions by $\alpha, \beta, \gamma, \ldots$. For a set M let M^* denote the set of finite sequences over M. Then, the set UML-SC of *UML-statechart terms* is inductively defined to be the least

[1] The work of Latella et al. [10] for UML-statecharts is based on the work of Mikk et al. [16] for classical statecharts.

[2] Note that the history mechanism has been neglected in most of the semantics definitions of classical statecharts.

[3] We do not consider deferring of events.

[4] At a later stage we have to identify the sets Π and $\mathcal{A}$.

set satisfying the following conditions, where $n \in \mathcal{N}$ and $en, ex \in \mathcal{A}^*$. (The interpretations of n, en and ex will be given later.)

1. **Basic term:** $s = [n, (en, ex)]$ is a UML-statechart term with $\mathsf{type}(s) = \mathsf{basic}$. Therefore s is also called a *basic term*.
2. **Or-term:** If $s_1, \ldots, s_k$ are UML-statechart terms for $k > 0$, $\rho = \{1, \ldots, k\}$, $l \in \rho$, $\mathsf{HT} = \{\mathsf{none}, \mathsf{deep}, \mathsf{shallow}\}$, and $T \subseteq \mathsf{TR} =_{\mathrm{df}} \mathcal{T} \times \rho \times 2^{\mathcal{N}} \times \Pi \times \mathcal{A}^* \times 2^{\mathcal{N}} \times \rho \times \mathsf{HT}$, then $s = [n, (s_1, \ldots, s_k), l, T, (en, ex)]$ is a UML-Statechart term with $\mathsf{type}(s) = \mathsf{or}$. Therefore, s is also called an *Or-term*. Here, $s_1, \ldots, s_k$ are the *subterms* of s, T is the set of *transitions* between the subterms of s, s_1 is the *default subterm* of s, l is called the *active state index* of s (or for short: the *index* of s), and s_l is the *currently active* subterm of s (or for short: s_l is *active*). Note that active state index $l \in \{1, \ldots, k\}$ denotes the l-th term within the k-tuple $(s_1, \ldots, s_k)$ of the subterms of s.
   For each transition $t = (\hat{t}, i, sr, e, \alpha, td, j, ht) \in T$ we require that additional constraints are fulfilled, namely $sr \in \mathsf{confAll}(s_i)$ and $td \in \mathsf{confAll}(s_j)$[5]. Furthermore, we define $\mathsf{name}(t) =_{\mathrm{df}} \hat{t}$, $\mathsf{sou}(t) =_{\mathrm{df}} s_i$, $\mathsf{souRes}(t) =_{\mathrm{df}} sr$, $\mathsf{ev}(t) =_{\mathrm{df}} e$, $\mathsf{act}(t) =_{\mathrm{df}} \alpha$, $\mathsf{tarDet}(t) =_{\mathrm{df}} td$, $\mathsf{tar}(t) =_{\mathrm{df}} s_j$, and $\mathsf{historyType}(t) =_{\mathrm{df}} ht$. $\mathsf{name}(t)$ is called the *transition name* of t, $\mathsf{ev}(t)$ and $\mathsf{act}(t)$ are called the *trigger part* and *action part* of t, respectively,[6] $\mathsf{sou}(t)$ and $\mathsf{tar}(t)$ are called the *source* and *target* of t, respectively, whereas $\mathsf{souRes}(t)$ and $\mathsf{tarDet}(t)$ are called the *source restriction* and *target determinator*. Source restriction and target determinator provide a means for modeling an interlevel transition by a simple transition on the level of the uppermost states the interlevel transition exits and enters. The source and target of the interlevel transition are represented as additional label information by the source restriction and the target determinator.[7] Transition t is called an interlevel transition, if its source restriction sr or its target determinator td differ from the empty set. Finally, $\mathsf{historyType}(t)$ is called the *history type* of t.
3. **And-term:** If $s_1, \ldots, s_k$ are UML-statechart terms for $k > 0$, then $s = [n, (s_1, \ldots, s_k), (en, ex)]$ is a UML-statechart term with $\mathsf{type}(s) = \mathsf{and}$. Thus, s is also called an *And-term*. Here, $s_1, \ldots, s_k$ are the (parallel) *subterms* of s.

In all three cases we refer to n as the *root name* of s and write $\mathsf{root}(s) =_{\mathrm{df}} n$. Furthermore, en and ex are the *sequence of entry* and the *sequence of exit actions* of s, respectively. If $a_1, \ldots, a_k$ are actions, then the sequence of actions $a_1, \ldots, a_k$ is denoted by $\langle a_1, \ldots, a_k \rangle$. In particular, the empty sequence is denoted by $\langle\rangle$. We assume that all root names and transition names are mutually disjoint, so that terms and transitions within UML-statechart terms are uniquely referred to by their names. For convenience, we sometimes write "state" instead of "term" and abbreviate $(s_1, \ldots, s_k)$ by $(s_{1..k})$.

In the following we will discuss some of our design decisions taken within the definition of UML-SC :

- We have decided to only include the indexes of states, but not the states themselves in the definition of transitions. The reason is that we do not want to overwhelm the definition of UML-statechart terms. To be more precise, in the definition of an Or-term $s = [n, (s_1, \ldots, s_k), l, T, (en, ex)]$ the definition of transition set T reduces redundancy (and therefore also complexity) within s: the substate information is already contained in tupel $(s_1, \ldots, s_k)$, but not repeated within T.
- Note that according to our definition of Or-terms a UML-statechart term does not only contain the static term structure (e.g. the information which subterms exist), but also dynamic information, i.e. the information which of the subterms of an Or-term is the currently active one. One could discuss whether it would be better to separate these two kinds of information (structure and behaviour) completely. We decided to define statecharts terms in a way which closely resembles the intuitive notion of statecharts, such that each substate of an Or-term on each level contains structural as well as behavioural information.
- In contrast to the work of Mikk et al. [16] we decided to define our (formal) syntax in a way that the formulation of Or- and And-states is not mingled in one common structure as it is the case with EHAs (Enhanced Hierarchical Automata), but to clearly separate And- and Or-states, because this is also the case in the graphical syntax of UML-statecharts. Therefore the comprehension of the translation between graphical and formal (textual) syntax is simplified.

We explain our term syntax with Fig. 1. The term syntax of the UML-statechart of Fig. 1 is as follows:

$$
\begin{aligned}
s_1 &= [n_1, (s_2, s_3), (\langle\rangle, \langle\rangle)] \\
s_2 &= [n_2, (s_4, s_5), 1, \{t_1, t_4, t_5, t_6\}, (\langle\rangle, \langle\rangle)] \\
s_3 &= [n_3, (s_6, s_7), 1, \{t_2\}, (\langle\rangle, \langle\rangle)] \\
s_4 &= [n_4, (s_8, s_9), 1, \{t_3\}, (\langle\rangle, \langle d \rangle)] \\
s_5 &= [n_5, (\langle e \rangle, \langle\rangle)] \\
s_i &= [n_i, (\langle\rangle, \langle\rangle)] \; (6 \leq i \leq 9)
\end{aligned}
$$

where s_1 is an And-state, s_2, s_3, and s_4 are Or-states, and $s_5, \ldots, s_9$ are basic states. Only s_4 has an exit action and only s_5 has an entry action. For the Or-states, we have selected the default substate as the currently active substate. Therefore, the active state index equals 1 for all the Or-states considered before.

A UML-statechart transition $(\hat{t}, i, sr, e, \alpha, td, j, ht)$ with $sr = \emptyset = td$ and $ht = \mathsf{none}$ is represented by an arrow from state s_i to s_j with label $t : e/\alpha$. Therefore, the

[5] The definition of $\mathsf{confAll}(s_j)$ is postponed to the end of this section.

[6] Note that the trigger part e is a single element of Π, whereas the action part α is a sequence of elements of $\mathcal{A}$ (with $\Pi \subseteq \mathcal{A}$).

[7] This idea stems from Mikk et al. [16].

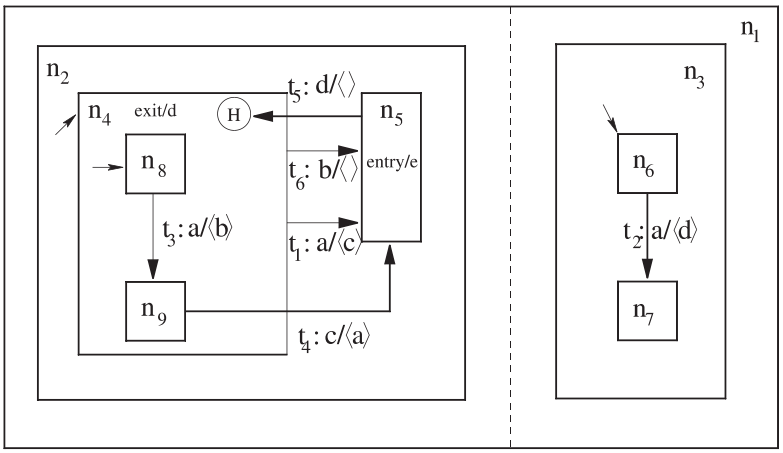

Fig. 1. UML-statechart example

transitions $t_1, t_2, \ldots, t_6$ are given as follows:

$t_1 = (\hat{t}_1, 1, \emptyset, a, \langle c\rangle, \emptyset, 2, \mathsf{none})$
$t_2 = (\hat{t}_2, 1, \emptyset, a, \langle d\rangle, \emptyset, 2, \mathsf{none})$
$t_3 = (\hat{t}_3, 1, \emptyset, a, \langle b\rangle, \emptyset, 2, \mathsf{none})$
$t_4 = (\hat{t}_4, 1, \{n_9\}, c, \langle a\rangle, \emptyset, 2, \mathsf{none})$
$t_5 = (\hat{t}_5, 2, \emptyset, d, \langle\rangle, \emptyset, 1, \mathsf{shallow})$
$t_6 = (\hat{t}_6, 1, \emptyset, b, \langle\rangle, \emptyset, 2, \mathsf{none})$

Note that components two and seven of a transition $t = (\hat{t}, i, sr, e, \alpha, td, j, ht) \in T$ – namely i and j – of an Or-term $s = [n, (s_1, \ldots, s_k), l, T, (en, ex)]$ refer to the i-th and j-th term of the k-tuple $(s_1, \ldots, s_k)$, respectively, but not to the indexes of the states' names in the k-tuple. Otherwise, e.g. transition t_1 would be given by $t_1 = (\hat{t}_1, 4, \emptyset, a, \langle c\rangle, \emptyset, 5, \mathsf{none})$.

Transition t_4 is the only interlevel transition, because its source restriction $\{n_9\}$ differs from the empty set. Furthermore, t_5 is the only transition which uses the history mechanism. We say that a transition t uses the history mechanism, if its history type is shallow or deep (i.e. $\mathsf{historyType}(t) \in \{\mathsf{deep}, \mathsf{shallow}\}$).

Later on, interlevel transitions and history-mechanism will be dealt with in more detail.

Configurations. In the following we consider different kinds of configurations. Function $\mathsf{conf} : \mathsf{UML\text{-}SC} \longrightarrow 2^{\mathcal{N}}$ which is inductively defined along the structure of UML-statechart terms computes the *(complete) current configuration* of a given UML-statechart term s, i.e. the set of the root names of all currently active substates within s also including the root name of s.[8]

$$
\begin{array}{ll}
\mathsf{conf}([n, \_]) & =_{\mathrm{df}} \{n\} \\
\mathsf{conf}([n, (s_{1..k}), l, T, \_]) & =_{\mathrm{df}} \{n\} \cup \mathsf{conf}(s_l) \\
\mathsf{conf}([n, (s_{1..k}), \_]) & =_{\mathrm{df}} \{n\} \cup \bigcup_{i=1}^{k} \mathsf{conf}(s_i)
\end{array}
$$

[8] The underscore "_" is used as a placeholder for arguments which can be neglected for the present consideration.

For example in the UML-statechart of Fig. 1 we have:[9]

$$\mathsf{conf}(s_1) = \{n_1, n_2, n_4, n_8, n_3, n_6\} \quad \mathsf{conf}(s_2) = \{n_2, n_4, n_8\}$$

For the sake of brevity, in the following we sometimes use subconfigurations instead of (complete) configurations. Intuitively, a subconfiguration of a UML-statechart term s is the set of all root names in the configuration of s which denote basic states. Formally, we define a function $\mathsf{subconf} : \mathsf{UML\text{-}SC} \longrightarrow 2^{\mathcal{N}}$ by

$$
\begin{array}{ll}
\mathsf{subconf}([n, \_]) & =_{\mathrm{df}} \{n\} \\
\mathsf{subconf}([n, (s_{1..k}), l, T, \_]) & =_{\mathrm{df}} \mathsf{subconf}(s_l) \\
\mathsf{subconf}([n, (s_{1..k}), \_]) & =_{\mathrm{df}} \bigcup_{i=1}^{k} \mathsf{subconf}(s_i)
\end{array}
$$

For example, in the UML-statechart of Fig. 1 we have:

$$\mathsf{subconf}(s_1) = \{n_8, n_6\} \qquad \mathsf{subconf}(s_2) = \{n_8\}$$

Function $\mathsf{confAll} : \mathsf{UML\text{-}SC} \longrightarrow 2^{2^{\mathcal{N}}}$ applied to UML-statechart s computes the set of all *potential* configurations of s, which can be complete or *incomplete*.[10]

- The term "potential" denotes that not only the currently active substate of each Or-state s' within s is considered, but all possibilities for choosing a substate of s'. This difference between conf and subconf on the one side and confAll on the other side implies that $\mathsf{conf}([n, (s_{1..k}), l, T, \_])$ as well as $\mathsf{subconf}([n, (s_{1..k}), l, T, \_])$ depend on active state index l, whereas $\mathsf{confAll}([n, (s_{1..k}), l, T, \_])$ does not.
- The term "incomplete" denotes a configuration which results from an application of confAll to state s, where the recursion within confAll terminates before the basic states of s are reached. Therefore, an incomplete

[9] Remember that for the present consideration we assume that the currently active substate of a (currently active) Or-state is given by the default substate of the Or-state.

[10] Remember that we have used function confAll to define constraints on the transition syntax.

configuration is upward-closed with respect to the state hierarchy, but not downward-closed, whereas a complete configuration is both.

Function confAll is also defined along the structure of UML-SC:[11]

$$\mathsf{confAll}([n,\_]) =_{\mathrm{df}} \{\{n\}\}$$
$$\mathsf{confAll}([n,(s_{1..k}),l,T,\_]) =_{\mathrm{df}} \{\{n\} \cup c \mid \exists j \in \{1..k\} \,.\, c \in \mathsf{confAll}(s_j)\} \cup \{\{n\}\}$$
$$\mathsf{confAll}([n,(s_{1..k}),\_]) =_{\mathrm{df}} \{\{n\} \cup \textstyle\bigcup_{i=1}^{k} c_i \mid c_i \in \mathsf{confAll}(s_i)\} \cup \{\{n\}\}$$

Incomplete configurations are realized in the second and third case of the definition of confAll by the union with term $\{\{n\}\}$. Note that $\mathsf{conf}(s)$ is an element of $\mathsf{confAll}(s)$ for each UML-statechart term s, formally $\forall s \in \mathsf{UML\text{-}SC} : \mathsf{conf}(s) \in \mathsf{confAll}(s)$.

We explain the notion of confAll by the UML-statechart of Fig. 1:

$$\mathsf{confAll}(s_1) \supseteq \{\{n_1,n_2,n_4,n_8,n_3,n_6\}, \{n_1,n_2,n_4,n_9,n_3,n_6\}, \{n_1,n_2,n_5,n_3,n_6\}, \{n_1,n_2,n_4,n_8,n_3,n_7\}, \{n_1,n_2,n_4,n_9,n_3,n_7\}, \{n_1,n_2,n_5,n_3,n_7\}\}$$
$$\mathsf{confAll}(s_2) \supseteq \{\{n_2,n_4,n_8\}, \{n_2,n_4,n_9\}, \{n_2,n_5\}\}$$

In this example, we have only listed the set of all complete potential configurations, but not the incomplete potential ones.

Having defined confAll and assuming that transition $t = (\_, i, sr, \_, \_, td, j, \_)$ with source s_i, target s_j, source restriction $sr \neq \emptyset$, and target determinator $td \neq \emptyset$ is given, now the reason for the constraints $sr \in \mathsf{confAll}(s_i)$ and $td \in \mathsf{confAll}(s_j)$ stated in the definition of transitions in Or-terms becomes clear. The source restriction and the target determinator are (possibly incomplete) configurations of s_i and s_j and specify that transition t models an interlevel transition as follows:

- Source restriction sr of t specifies the source state of the interlevel transition.
- Target determinator td of t specifies the target state of the interlevel transition.

Let us consider an example: The only interlevel transition of UML-statechart term s_1 of Fig. 1 is $t_4 = (\hat{t}_4, 1, \{n_9\}, c, \langle a\rangle, \emptyset, 2, \mathsf{none})$, because in all other transitions both the source restriction as well as the target determinator equal the empty set. Due to the source restriction $\{n_9\}$ of t_4 this transition is represented in Fig. 1 by an arrow from state s_9 (instead of s_4) to s_5 with label $t_4 : c/\langle a\rangle$.

Note that our definition of confAll which allows *incomplete* configurations enables – together with the conditions $sr, td \in \mathsf{confAll}(s_i)$ – a more liberal modeling of interlevel transitions than the definition of transitions in the work of Mikk et al. [16], where source restriction and target determinator must be *complete* configurations.

As can be seen from our UML-statecharts term syntax and as in the work of [10] we do not consider the following features of UML-statecharts:

- initial, final, and junction pseudostates,
- deferred, time, and change events,
- branch segments and completion transitions,
- guards, variables, and data dependencies in transition labels,
- termination, creation, destruction of objects and send clauses within actions as well as do actions, and
- dynamic choicepoints.

4 Semantics

In the following subsections, we proceed as follows: At first we recall the intuitive semantics definition of UML-statecharts. Afterwards we formalize the treatment of entry and exit actions. Then we define how the state resulting from transition execution is computed. Finally, we use the achieved results to formally define the semantics of UML-statecharts.

4.1 Intuition

We recall the intuitive semantics definition of UML-statecharts by considering again our example of Fig. 1. Initially, UML-statechart term s_1 is in subconfiguration $\{n_8, n_6\}$. If the input event queue offers event

- b, then transition t_6 can be taken, so that the next subconfiguration of s_1 will be $\{n_5, n_6\}$ and the sequence $\langle d, e\rangle$ of actions is executed, because $\langle d\rangle$ is the sequence of exit actions of state s_4, t_6 has the empty sequence $\langle\rangle$ as action part, and $\langle e\rangle$ is the sequence of entry actions of state s_5,
- a, then transitions t_3 and t_2 can simultaneously be taken, so that $\{n_9, n_7\}$ will be the next subconfiguration and the sequence $\langle b, d\rangle$ of actions will be executed. In this case it is not allowed that only one of both transitions is taken. Note that due to the lower-first priority in UML-statecharts the transition t_1 can not be taken if state s_8 is active, because the source s_8 of transition t_3 is on a lower level than the source s_4 of transition t_1,[12]
- $e \in \Pi \setminus \{a, b\}$, then the configuration does not change, because no transition can be taken.

History mechanism. Harel already presented the history mechanism in his early classical statecharts paper [6].

[11] We will abbreviate tuple $\{1, \ldots, k\}$ by $\{1..k\}$.

[12] The level of a transition t is defined by the level of the source of t, not by the level of its source restriction.

However, most proposals for a precise semantics definition of classical statecharts neglected it.

The history mechanism allows to reenter an Or-state s, such that the same substate of s becomes the currently active substate as it has been the case, when s has been active the last time. If the Or-state has never been active before, the default substate of the Or-state becomes the currently active substate.

The history mechanism allows two kinds of scoping: the "memory" effect can be restricted to the direct substates of the considered Or-state ("shallow", graphical symbol: "H") or can be unrestricted ("deep", graphical symbol: "H*"), such that it recursively remembers the active substates along the state hierarchy down to the basic states. We will consider both cases in our semantics.

In the UML-statechart of Fig. 1 only transition t_5 uses the history mechanism – with $\mathsf{historyType}(t_5) = \mathsf{shallow}$. If t_5 is performed, then state s_4 becomes active. Furthermore, substate s_8 (s_9) becomes active, if s_8 (s_9) has been active, when s_4 has been active last time before. In the case that s_4 has never been active before, then the default state s_8 of s_4 becomes active.

Semantic problem. The main difficulty in our enhancements with respect to the work of Latella et al. [10] (and Mikk et al. [16]) arises from the fact that our semantics shall additionally support the history mechanism which exhibits quite intricate dependencies with interlevel transitions.[13] To clarify this point let us assume that transition t_5 in Fig. 1 which uses the history mechanism (cf. $\mathsf{historyType}(t_5) = \mathsf{shallow}$) is modified to an interlevel transition t_5' by changing its target determinator from empty set $\emptyset$ to $\{n_8\}$. If t_5' is taken one could argue that due to the history mechanism the next currently active subterm of s_4 should be s_9 if s_9 had been the last active subterm of s_4. Alternatively, one could argue that due to the target determinator $\{n_8\}$ of t_5' the next currently active subterm of s_4 should be s_8. However, we will define the semantics such that the target determinator information of a transition has priority over the history mechanism. For the above-mentioned example this means that the second possibility is chosen.

4.2 Entry and exit actions

When a UML-statechart transition t is taken, a (possibly empty) set of actions is executed: at first the sequence of exit actions of the source of t is executed (according to an inner-first approach), then the action part $\mathsf{act}(t)$ of t, and finally the sequence of entry actions of t (according to an outer-first approach).

We explain the execution of entry and exit actions with the UML-statechart example of Fig. 1. If transition t_1 is taken then the action sequence $\langle d, c, e\rangle$ is generated, because at first the sequence $\langle d\rangle$ of exit actions of source s_4 of transition t_1 is executed, then the action part $\langle c\rangle$ of t_1, and finally the sequence $\langle e\rangle$ of entry actions of target s_5 of t_1.

In general, if a transition $(\_, l, \_, \_, \alpha, \_, i, \_)$ from state s_l to state s_i with action part α is taken, then a sequence $ex :: \alpha :: en$ of actions is executed, with $ex \in \mathsf{exit}(s_l), en \in \mathsf{entry}(s_i)$. Intuitively, $\mathsf{exit}(s_l)$ is the set of all possible sequences of exit actions of s_l and $\mathsf{entry}(s_i)$ is the set of all possible sequences of entry actions of s_i. Here, we assume that :: is an operator in infix-notation which concatenates action sequences.[14] The mentioned intuitive meanings of functions exit and entry already indicate that we introduce a sort of nondeterminism when defining the UML-statecharts semantics. The UML definition in [17] does not define in which order the entry actions of the substates of an And-state $[n, (s_{1..k}), (en, ex)]$ have to be executed with respect to each other. The same is valid for the exit actions of the substates of an And-state. Therefore we allow each permutation of the entry actions and each permutation of the exit actions as possible execution sequences. This nondeterminism is achieved by defining entry and exit as two set-valued functions $\mathsf{entry} : \mathsf{UML\text{-}SC} \longrightarrow 2^{\mathcal{A}^*}$ and $\mathsf{exit} : \mathsf{UML\text{-}SC} \longrightarrow 2^{\mathcal{A}^*}$ which are inductively defined along the structure of UML-SC as follows:

$$
\begin{array}{l}
\mathsf{entry}([n, (en, ex)]) =_{\mathrm{df}} \{en\} \\
\mathsf{entry}([n, (s_{1..k}), l, T, (en, ex)]) \\
\quad =_{\mathrm{df}} \{en :: en' \mid en' \in \mathsf{entry}(s_l)\} \\
\mathsf{entry}([n, (s_{1..k}), (en, ex)]) \\
\quad =_{\mathrm{df}} \{en :: m_1 :: \ldots :: m_k \mid \\
\qquad \exists \text{ bijection } b : \{1..k\} \longrightarrow \{1..k\} \,. \\
\qquad m_i \in \mathsf{entry}(s_{b(i)}) \forall i \in \{1..k\}\}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{exit}([n, (en, ex)]) =_{\mathrm{df}} \{ex\} \\
\mathsf{exit}([n, (s_{1..k}), l, T, (en, ex)]) \\
\quad =_{\mathrm{df}} \{ex' :: ex \mid ex' \in \mathsf{exit}(s_l)\} \\
\mathsf{exit}([n, (s_{1..k}), (en, ex)]) \\
\quad =_{\mathrm{df}} \{m_1 :: \ldots :: m_k :: ex \mid \\
\qquad \exists \text{ bijection } b : \{1..k\} \longrightarrow \{1..k\} \,. \\
\qquad m_i \in \mathsf{exit}(s_{b(i)}) \forall i \in \{1..k\}\}
\end{array}
$$

In the definition of both functions entry and exit the nondeterminism is realized by existential quantification over bijections in order to consider all possible permutations of the corresponding sequences of entry actions and exit actions, respectively.

4.3 Computing the next state

If a UML-statechart transition t is executed, particularly its history type and – if t is an interlevel transition – t's

[13] Remember that our UML-statecharts term syntax represents interlevel transitions by using additional information in the transition labels (i.e. source restriction and target determinator) based on the approach of [16].

[14] The above-mentioned term $ex :: \alpha :: en$ will be used in SOS rule OR-1 of the (auxiliary) semantics of UML-statecharts to be presented in Sect. 4.4.

target determinator have to be considered. Therefore we define function next which computes the state which results from a transition execution. Later on this function is used in the SOS rule which handles transition execution (in an OR-state).

Given a UML-statechart transition t with target s, history type $ht = \mathsf{historyType}(t)$, and target determinator $N = \mathsf{tarDet}(t)$ of t, the function $\mathsf{next} : \mathsf{HT} \times \mathcal{N} \times \mathsf{UML\text{-}SC} \longrightarrow \mathsf{UML\text{-}SC}$ computes the UML-statechart term $s' = \mathsf{next}(ht, \mathsf{tarDet}(t), s)$ which results after execution of transition t. Note that the terms s and s' have identical static structure, only their dynamic information – specifying the currently active substates – may differ. In order to simplify the presentation of next as well as the presentation of several subsequent definitions (functions next_stop, default, and the SOS rules), in the following we sometimes abstract from entry and exit actions within UML-statechart terms: for example, we write $[n]$ instead of $[n, (en, ex)]$, we write $[n, (s_{1..k}), l, T]$ instead of $[n, (s_{1..k}), l, T, (en, ex)]$, and we write $[n, (s_{1..k})]$ instead of $[n, (s_{1..k}), (en, ex)]$. Furthermore, we use the substitution notation $._{[./.]}$ as follows: If t is a term, then $t_{[a/b]}$ is the term which results from replacing all occurrences of a in t by b. Finally, for $l \in \{1, \ldots, k\}$ we abbreviate $(s_1, \ldots, s_{l-1}, s'_l, s_{l+1}, \ldots, s_k)$ by $(s_{1..k})_{[s_l/s'_l]}$.

$$
\begin{aligned}
&\mathsf{next}(ht, N, [n]) =_{\mathrm{df}} [n] \\
&\mathsf{next}(ht, N, [n, (s_{1..k}), l, T]) \\
&\quad =_{\mathrm{df}} \begin{cases} [n, (s_{1..k})_{[s_j/\mathsf{next}(ht, N, s_j)]}, j, T] & \text{if } \exists n' \in N, \\ & j \in \{1, \ldots, k\}\,. \\ & n' = \mathsf{name}(s_j) \\ \mathsf{next\_stop}(ht, [n, (s_{1..k}), l, T]) & \text{otherwise} \end{cases} \\
&\mathsf{next}(ht, N, [n, (s_{1..k})]) \\
&\quad =_{\mathrm{df}} [n, (\mathsf{next}(ht, N, s_1), \ldots, \mathsf{next}(ht, N, s_k))]
\end{aligned}
$$

The second case of the definition – application of function next to an Or-state $[n, (s_{1..k}), l, T]$ – requires some explanation:

- If N contains a name n' of one of the state names of the substates $s_1, \ldots, s_k$, i.e. $\exists n' \in N, j \in \{1, \ldots, k\}$, such that $n' = \mathsf{name}(s_j)$ (denoted as condition $*$), then active state index l will be replaced by active state index j and function next is recursively applied to s_j. Therefore, if $N = \mathsf{tarDet}(t)$, then the target determinator information of t is exploited in function next when zooming into the state hierarchy as long as condition $*$ is fulfilled, i.e. as long as adequate target determinator information exists.
- Otherwise, i.e. if N does not contain a name n' of one of the state names of the substates $s_1, \ldots, s_k$, function $\mathsf{next\_stop} : \mathsf{HT} \times \mathsf{UML\text{-}SC\text{-}OR} \longrightarrow \mathsf{UML\text{-}SC\text{-}OR}$ is called which uses the history type information to determine currently active substates of a state, with $\mathsf{UML\text{-}SC\text{-}OR} =_{\mathrm{df}} \{s \mid s \in \mathsf{UML\text{-}SC}, \mathsf{type}(s) = \mathsf{or}\}$, i.e. UML-SC-OR is the set of all UML-statechart terms which are Or-states. In the definition of function next_stop the following case distinction occurs:
  - If history type $ht = \mathsf{deep}$, then the state $[n, (s_{1..k}), l, T]$ does not change at all.
  - If history type $ht = \mathsf{none}$, then active state index l is replaced by active state index 1 and function default is used to initialize substate s_1.
  - If history type $ht = \mathsf{shallow}$, then active state index l does not change, but function default initializes all lower levels of s_l.

$$
\begin{aligned}
&\mathsf{next\_stop}(ht, [n, (s_{1..k}), l, T]) \\
&\quad =_{\mathrm{df}} \begin{cases} [n, (s_{1..k}), l, T] & \text{if } ht = \mathsf{deep} \\ [n, (s_{1..k})_{[s_1/\mathsf{default}(s_1)]}, 1, T] & \text{if } ht = \mathsf{none} \\ [n, (s_{1..k})_{[s_l/\mathsf{default}(s_l)]}, l, T] & \text{if } ht = \mathsf{shallow} \end{cases}
\end{aligned}
$$

Note the difference between the second and third case of the definition of next_stop:

- The second case ($ht = \mathsf{none}$) is independent from l, i.e. from the active state index of the currently active substate of the Or-state – $\mathsf{default}(s_1)$ becomes the new currently active substate of s_1.
- In contrast the third case ($ht = \mathsf{shallow}$) depends on l, because $\mathsf{default}(s_l)$ becomes the new currently active substate of the Or-state.

The definition of next_stop uses function $\mathsf{default} : \mathsf{UML\text{-}SC} \longrightarrow \mathsf{UML\text{-}SC}$ which defines for an Or-state that its currently active substate is given by its default substate.

$$
\begin{aligned}
&\mathsf{default}([n]) =_{\mathrm{df}} [n] \\
&\mathsf{default}([n, (s_{1..k}), l, T]) =_{\mathrm{df}} [n, (s_{1..k})_{[s_1/\mathsf{default}(s_1)]}, 1, T] \\
&\mathsf{default}([n, (s_{1..k})]) \\
&\quad =_{\mathrm{df}} [n, (\mathsf{default}(s_1), \ldots, \mathsf{default}(s_k))]
\end{aligned}
$$

4.4 Semantics definition

In this section we will develop a formal UML-statecharts semantics definition. The semantics will be defined for the textual UML-statecharts syntax as given by the set UML-SC of UML-statechart terms.

To develop a comprehensible – though precise – UML-statecharts semantics definition we will modularize it as follows: First of all we will define the semantics in two phases: In the first phase we will define an auxiliary UML-statecharts semantics which only deals with processing single input events, but not with sequences of input events. In a second phase we use this auxiliary semantics to define the (complete) UML-statecharts semantics dealing with processing sequences of input events. This separation already supports modularity. Furthermore, for the first phase we follow the SOS-approach of Plotkin: We

take labeled transition systems as semantic domain and use SOS-rules to define the (auxiliary) semantics of UML-statecharts – restricted to the processing of single input events – in a modular way. More precisely, we will define the auxiliary semantics by a function $[\![.]\!]_{aux} : \mathsf{UML\text{-}SC} \longrightarrow \mathsf{LTS}$, where LTS is the set of *labeled transition systems* and where the (semantic) transitions[15] work on single input events $e \in \Pi$. For the second phase we use Kripke structures as semantic domain, because this selection simplifies the processing of event sequences considerably, since Kripke structures are very appropriate for modeling that the output of one step serves as (part of) the input of the next step.

Both phases constitute an operational and modular approach, such that comprehension as well as flexibility (e.g. with respect to subsequent enhancements) are supported – without restricting preciseness.

Priority mechanism. As opposed to the work of Latella et al. [10] we will not parameterize our semantics definition with a priority mechanism for transition execution, since lower-first priority is stipulated in the official UML specification of the OMG [17]. Therefore, we directly encode lower-first priority in our semantics.

Auxiliary semantics. The auxiliary semantics $[\![s]\!]_{aux}$ of a UML-statechart term $s \in \mathsf{UML\text{-}SC}$ is given by the labeled transition system $(\mathsf{UML\text{-}SC}, L, \longrightarrow, s) \in \mathsf{LTS}$, where

- $\mathsf{UML\text{-}SC}$ is the set of states,[16]
- $L = \Pi \times \mathcal{A}^* \times \{0,1\}$ is the set of labels,
- $\longrightarrow \subseteq \mathsf{UML\text{-}SC} \times L \times \mathsf{UML\text{-}SC}$ is the transition relation, and
- s is the start state.

[15] We use the term "semantic transition" in order to distinguish transitions of the semantics of UML-statecharts from "UML-statechart transitions", which occur in the syntax, more precisely in UML-statechart terms of type Or.

[16] This implies that each state of the transition system is given by a UML-statechart term.

For the sake of simplicity, we write $s \xrightarrow[\alpha]{e}_f s'$ instead of $(s, (e, \alpha, f), s') \in \longrightarrow$ and $s \not\xrightarrow{e}_f$ instead of $\nexists s', \alpha.\ s \xrightarrow[\alpha]{e}_f s'$, where s and s' are called the *source* and the *target* of these (semantic) transitions, respectively, e and α are called the *input* and *output*, respectively, and f is called the *stuttering flag* (or for short *flag*). We say that term s may perform a (semantic) transition with input e, output α, and flag f to term s'. If appropriate, we do no mention the input, output, and/or target of the transition. Intuitively, stuttering flag f states whether a semantic transition is performed,

- either because at least one UML-statechart transition is taken (in this case $f = 1$, denoted as *positive flag*)
- or without taking any UML-statechart transition (in this case $f = 0$, denoted as *negative flag*). In this case only the input event is "consumed", whereas source and target are identical. This is usually denoted as a *stuttering step*.

The flag is needed to assure that stuttering steps can only occur, if no non-stuttering step is possible.

In contrast to the work of Latella et al. [10] we do not need to annotate a semantic transition with the explicit set of UML-statechart transitions which are taken when the semantic transition is performed. Instead, in our case it suffices to annotate the boolean information whether at least one UML-statechart transition is taken. This simplification reduces the complexity of the semantics and therefore could ease the implementation of the semantics. Furthermore, a better performance of the implemented semantics could result.

Transition relation $\longrightarrow$ is defined as presented in Table 1 by five SOS rules using the following rule format:

$$name\ \frac{premise}{conclusion}\ (condition)$$

Explanation of the SOS rules:

- BAS (stuttering)
  A basic state may perform a semantic transition with

Table 1. SOS rules of the auxiliary semantics

$$\textit{BAS}\quad \frac{\mathsf{true}}{[n] \xrightarrow[\langle\rangle]{e}_0 [n]}$$

$$\textit{OR-1}\quad \frac{(\_, l, sr, e, \alpha, td, i, ht) \in T,\ sr \subseteq \mathsf{conf}(s_l),\ s_l \not\xrightarrow{e}_1}{[n, (s_{1..k}), l, T] \xrightarrow[ex::\alpha::en]{e}_1 [n, (s_{1..k})_{[s_i/\mathsf{next}(ht,td,s_i)]}, i, T]}\ \begin{pmatrix} ex \in \mathsf{exit}(s_l), \\ en \in \mathsf{entry}(\mathsf{next}(ht, td, s_i)) \end{pmatrix}$$

$$\textit{OR-2}\quad \frac{s_l \xrightarrow[\alpha]{e}_1 s_l',}{[n, (s_{1..k}), l, T] \xrightarrow[\alpha]{e}_1 [n, (s_{1..k})_{[s_l/s_l']}, l, T]}$$

$$\textit{OR-3}\quad \frac{s_l \xrightarrow[\langle\rangle]{e}_0 s_l,\ [n, (s_{1..k}), l, T] \not\xrightarrow{e}_1}{[n, (s_{1..k}), l, T] \xrightarrow[\langle\rangle]{e}_0 [n, (s_{1..k}), l, T]}$$

$$\textit{AND}\quad \frac{\forall j \in \{1, \ldots, k\}\ .\ s_j \xrightarrow[\alpha_j]{e}_{f_j} s_j'}{[n, (s_{1..k})] \xrightarrow[\alpha]{e}_{\bigvee_{j=1}^k f_j} [n, (s'_{1..k})]}\ \begin{pmatrix} \alpha \in \{\alpha_{b(1)} :: \ldots :: \alpha_{b(k)} \mid \\ \exists \text{ bijection } b : \{1..k\} \longrightarrow \{1..k\}\} \end{pmatrix}$$

arbitrary input event e, empty output, and negative flag such that the state does not change, i.e. that the input event is just consumed.

– OR-1 (progress)
If t is a UML-statechart transition of an Or-state s with trigger part e, then s can perform a semantic transition with input e and positive flag
  • if the source restriction sr of t is a subset of the complete current configuration of the currently active substate s_l of s ($sr \subseteq \mathsf{conf}(s_l)$) and
  • if s_l cannot perform a semantic transition with input e and positive flag ($s_l \overset{e}{\not\rightarrow}_1$).
The former condition assures the enabledness of transition t, whereas the last condition assures the lower-first priority of UML-statecharts. Rule OR-1 treats the execution of entry and exit actions: it is the only rule in which additional entry and exit actions, i.e. entry and exit actions not already occuring in the output part of the transition in the rule's premise, can occur in the output part of the transition in the conclusion. Note that the source restriction sr and the target determinator td of a UML-statechart transition only appear in this rule: $sr \subseteq \mathsf{conf}(s_l)$ assures the enabledness of the considered transition, whereas td – used within $\mathsf{next}(ht, td, s_i)$ – precisely defines the target state and therefore also the entry actions to be executed. The target of the semantic transition differs from its source by changing the currently active substate from s_l to s_i, because s_l and s_i are the source and target of the UML-statechart transition t, respectively. Furthermore, the dynamic information of s_i is updated according to the history type ht and the target determinator td of t using function next. This update is performed by the substitution $(s_{1..k})_{[s_i/\mathsf{next}(ht,td,s_i)]}$. Finally, the output of the semantic transition is given by concatenating a sequence ex of the exit actions of the old currently active substate s_l with the output part α of the UML-statechart transition t and with a sequence en of the entry actions of the new currently active substate s_i', where s_i' is the result of updating s_i using function next as explained before.

– OR-2 (propagation of progress)
If a substate of an Or-state may perform a semantic transition with a positive flag, then the Or-state may perform a semantic transition with the same label.

– OR-3 (propagation of stuttering)
If a substate of an Or-state may perform a semantic transition with a negative flag (i.e. no UML-statechart transition can be taken within the Or-state) and if the Or-state cannot perform a semantic transition with positive flag, then the Or-state may also perform a semantic transition with the same label (in particular with negative flag). The condition, that the Or-state cannot perform a semantic transition with positive flag supports the maximality condition which will be dealt with later on in more detail.

– AND (composition)
If every substate s_j of an And-state s can perform a semantic transition with input e, output α_j, and flag b_j, then And-state s can also perform a semantic transition with the same input e, but with output α resulting from concatenating the substate outputs α_j in an arbitrary order, and with flag $\bigvee_{j=1}^{k} f_j$ given by the logical disjunction of all flags f_j. Here, we identify "0" and "1" with the boolean values "false" and "true", respectively, to evaluate term $\bigvee_{j=1}^{k} f_j$.

Summing up, the SOS rules define that for every input event $e \in \Pi$ and for every state $s \in \mathsf{UML\text{-}SC}$

– either a semantic transition $s \overset{e}{\underset{\alpha}{\rightarrow}}_1 s'$ with output $\alpha \in \mathcal{A}^*$ and state $s' \in \mathsf{UML\text{-}SC}$ exists or
– a semantic transition $s \overset{e}{\underset{\langle\rangle}{\rightarrow}}_0 s$ exists – with empty output and without a state change.

In particular our semantics definition fulfills the maximality condition of UML-statecharts: a maximal number of non-conflicting UML-statechart transitions is taken, when a semantic transition is performed. This condition is assured by the following facts:

– The AND-rule assures that an And-term can only perform a semantic transition, if all of its (parallel) substates perform a (semantic) transition.
– The set of Or-rules make sure that performing a semantic transition with positive flag (denoting the execution of at least one UML-statechart transition) has priority over performing a semantic transition with negative flag (denoting the execution of no UML-statechart transition): the premise of rule OR-3 (propagation of stuttering) can only become true, if the premise of rule OR-1 (progress) evaluates to false.

In the SOS rules we have used stuttering steps in order to simplify the formulation of (parallel) composition in rule AND: by allowing stuttering steps we can (and in fact must) assume that all its parallel substates perform a semantic transition. If we would not use stuttering steps, it would be difficult to define the semantics such that the aforementioned maximality condition will be satisfied.

As an example Fig. 2 presents the auxiliary semantics, i.e. a labeled transition system lts, for the UML-statechart of Fig. 1 in a graphical way as state transition diagram std, where each std-state represents an lts-state s by the current subconfiguration of s. For the sake of simplicity, Fig. 2 only presents those semantic transitions which have positive flags. Semantic transitions with negative flags would have to be presented by cyclic transitions at every state. More precisely for every state s of the transition diagram of Fig. 2 for which there does not exist an outgoing transition with input $e \in \Pi$ a cyclic transition with label $e/\langle\rangle$ at state s would have to be drawn. Furthermore, Fig. 2 only shows those states which are reachable from the start state (n_8, n_6).

Complete semantics. In order to define the UML-statechart semantics in a more complete way, we have to

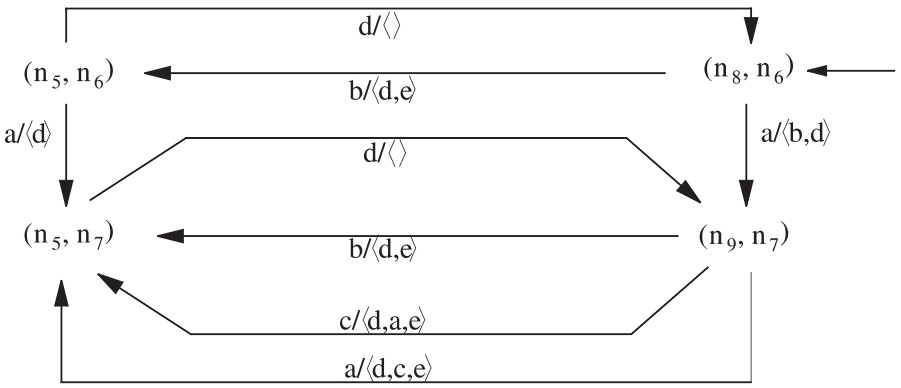

Fig. 2. Auxiliary semantics of UML-statechart from Fig. 1

consider that – given a sequence of input events – a UML-statechart performs a sequence of steps, such that during each of these steps

- one event of the current sequence of input events is consumed and therefore deleted from this sequence and
- a sequence of actions is generated which is added to the shortened sequence of input events resulting in a new sequence of input events to be used in the following step.

Kripke structures can be used to model this processing. In particular, they are very appropriate for modeling that the output of one step serves as the input for the following step. Therefore, after having defined the auxiliary semantics $[\![.]\!]_{aux}$ in a first phase, in a second phase we use Kripke structures and the (auxiliary) semantics $[\![.]\!]_{aux}$ to define the (complete) semantics $[\![.]\!] : \mathsf{UML\text{-}SC} \longrightarrow \mathcal{K}$ for UML-statechart terms, where $\mathcal{K}$ is the set of *Kripke structures*.

The (complete) semantics $[\![s]\!]$ of a UML-statechart term $s \in \mathsf{UML\text{-}SC}$ is given by a Kripke structure $K = (S, st, \longrightarrow\!\!\blacktriangleright) \in \mathcal{K}$, where

- $S = \mathsf{UML\text{-}SC} \times \Pi^*$ is the set of Kripke states of K,
- $st = (s, \epsilon_0) \in S$ is the start state of K with $\epsilon_0 \in \Pi^*$
- $\longrightarrow\!\!\blacktriangleright \subseteq S \times S$ is the transition relation of K.

Due to the choice of Kripke structures as semantic domain we have to require $\Pi = \mathcal{A}$, because the "output" of a step of a Kripke structure serves as the "input" of the next step.[17]

For the sake of simplicity, we write $(s, \epsilon) \longrightarrow\!\!\blacktriangleright (s', \alpha)$ instead of $(s, \epsilon, s', \alpha) \in \longrightarrow\!\!\blacktriangleright$.

The following SOS rule (called get-inp) defines transition relation $\longrightarrow\!\!\blacktriangleright$ of the complete semantics by use of transition relation $\longrightarrow$ of the auxiliary semantics.

$$\textit{get-inp} \qquad \frac{s \xrightarrow[\alpha]{e/f} s'}{(s,\epsilon) \longrightarrow\!\!\blacktriangleright (s',\epsilon'')} (\exists(\epsilon, e, \epsilon') \in \mathsf{sel}, \exists(\alpha, \epsilon', \epsilon'') \in \mathsf{join})$$

Explanation of SOS rule get-inp:

If a sequence ϵ of events is given, such that relation sel can separate it in a single event e and the rest sequence ϵ' and if state s may perform a transition according to transition relation $\longrightarrow$ with input e (being a single event), output α, and flag f to state s', then Kripke state (s, ϵ) may also perform a transition according to transition relation $\longrightarrow\!\!\blacktriangleright$ to state (s', ϵ''), if relation join can compose output α and rest sequence ϵ' to sequence ϵ''.

According to the UML definition which does not define the scheduling strategy of the input event queue of UML-statecharts (and thereby following [10]) we use two relations $\mathsf{sel} \subseteq \Pi^* \times (\Pi \times \Pi^*)$ and $\mathsf{join} \subseteq (\Pi^* \times \Pi^*) \times \Pi^*$ which still have to be defined accordingly for a concrete scheduling strategy of the input event queue.

After having defined the semantics we will discuss the advantages and disadvantages of interlevel transitions.

- On the one hand interlevel transitions are included in Harel's classical statecharts [6] as well as in UML-statecharts of all UML versions 1.x [17] and also in the UML 2.0 Proposal of the U2-partners [18].
- On the other hand interlevel transitions imply that (UML-)statecharts are not defined in a modular way, so that the definition of a compositional (UML-) statecharts semantics is impeded. Therefore interlevel transition are not allowed in most work related to the formalization of statecharts semantics [11, 12, 15, 21, 24, 25] (also in our previous work [13, 14, 26, 27]).
- However, in the work presented above we have incorporated interlevel transitions within our UML-statechart term syntax, although their prohibition would have significantly simplified our syntax and semantics definition of UML-statecharts:
  - Syntax:
    Transitions of Or-states would neither contain source restriction nor target determinator information. Therefore also function confAll would not be necessary any more.
  - Semantics:
    We could skip function conf which is used in SOS-rule OR-1. Furthermore, the definition of function next would be significantly less complex.

[17] Up to now, we only had to require $\Pi \subseteq \mathcal{A}$.

5 Related work

In the following we discuss related work dealing with a precise semantics definition of UML-statecharts.

The work of Latella et al. [10] has been one starting point of our work. The enhancements of our work with respect to their work have already been described before.

Paltor and Lilius [19] as well as Kwon [9] define an operational semantics for UML-statecharts in terms of rewrite rules. Since Paltor et al. do not use a structured approach like SOS their semantics does not offer the same level of clarity as ours.

Compton et al. [2] outline a UML-statecharts semantics based on Abstract State Machines. They consider entry and exit actions, but – in contrast to our work – not the history mechanism.

Gogolla et al. [5] present a formal semantics of UML-statecharts by mapping UML-statecharts into a more simplified machine using graph rewriting techniques.

Reggio et al. [22] define the semantics of a UML-statechart associated with an active UML-class by a labeled transition system which is formally specified by the algebraic specification language CASL. In contrast to our approach the authors neither consider entry and exit actions nor the history mechanism.

Börger et al. [1] present a precise and quite modular semantics for UML-statecharts based on Abstract State Machines in particular covering the history mechanism as well as entry and exit actions.

Engels, Hausmann, Heckel, and Sauer [3] propose a meta modeling approach to define the operational semantics of behavioural UML diagrams – especially a fragment of UML-statecharts – based on collaboration diagrams. The way how collaboration diagrams are used resembles Plotkin's Structured Operational Semantics.

Eshuis and Wieringa [4] present a formal semantics of UML-statecharts in terms of labelled transition systems. In contrast to our work they do not include the history mechanism.

Kuske [8] proposes a formal operational semantics for a subset of UML-statecharts – not including the history mechanism – based on structured graph transformations.

6 Conclusions and further work

We presented a formal semantics of UML-statecharts. In contrast to related work (like the approach of Latella et al. [10] which has been one starting point of our work) we additionally include UML-statechart features like the history mechanism (in both kinds) as well as entry and exit actions. Furthermore, the use of our syntax, namely UML-statechart terms, as well as our approach of defining an SOS-style semantics results in quite a succinct and well adaptable semantics which could be used as the basis for formal analysis techniques like model checking, equivalence checking, refinement checking, consistency checking or for defining transformations between tools which support different UML-statechart dialects.

In future, we will consider additional features of UML-statecharts – e.g. guards and deferred events – within the semantics definition. Furthermore, we will examine how our approach can be adapted to other behavioural UML notations like sequence diagrams.

Another major point of our future work is given by the development of adequate formal notions for equivalence, refinement, and consistency based on the semantics definition of UML behavioural notations providing the basis of a development method for UML behavioural notations.

Acknowledgements. We would like to thank the anonymous reviewers for very constructive and detailed suggestions for improvements.

References

1. Börger, E., Cavarra, A., Riccobene, E.: Modeling the dynamics of UML state machines. In: Abstract State Machines. Theory and Applications. LNCS, vol. 1912. Springer, 2000
2. Compton, K., Huggins, J., Shen, W.: A Semantic Model for the State Machine in the Unified Modeling Language. In: Proceedings Dynamic Behaviour in UML Models: Semantic Questions. Ludwig-Maximilians-Universität München, Institut für Informatik, Bericht 0006, 2000, pp. 25–31
3. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000 – The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings. LNCS, vol. 1939. Springer, 2000, pp. 323–337
4. Eshuis, R., Wieringa, R.: Requirements-level semantics for UML-statecharts. In: Proceedings of FMOODS 2000. Kluwer, 2000
5. Gogolla, M., Parisi-Presicce, F.: State diagrams in UML: A formal semantics using graph transformations. In: Broy, M., Coleman, D., Maibaum, T.S.E., Rumpe, B. (eds.) Proceedings PSMT'98 Workshop on Precise Semantics for Modeling Techniques. Technische Universität München, TUM-I9803, 1998
6. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8: 231–274, 1987
7. Harel, D., Naamad, A.: The STATEMATE semantics of Statecharts. ACM Transactions on Software Engineering, 5(4): 293–333, October 1996
8. Kuske, S.: A Formal Semantics of UML State Machines Based on Structured Graph Transformation. In: Gogolla, M., Kobryn, C. (eds.) UML 2001 – The Unified Modeling Language: Modeling Languages, Concepts, Tools. Lecture Notes in Computer Science, vol. 2185. Springer-Verlag, 2001, pp. 241–256
9. Kwon, G.: Rewrite rules and operational semantics for model checking UML statecharts. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000 – The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings. LNCS, vol. 1939. Springer, 2000, pp. 528–540
10. Latella, D., Majzik, I., Massink, M.: Towards a formal operational semantics of UML statechart diagrams. In: Formal Methods for Open Object-based Distributed Systems. Chapman & Hall, 1999
11. Levi, F.: Verification of Temporal and Real-Time Properties of Statecharts. PhD thesis, University of Pisa-Genova-Udine, Pisa, Italy, February 1997
12. Lüttgen, G., Mendler, M.: The intuitionism behind statecharts steps. In: ICALP 2000. Lecture Notes in Computer Science, vol. 1853. Springer-Verlag, 2000, pp. 163–174

13. Lüttgen, G., von der Beeck, M., Cleaveland, R.: Statecharts via process algebra. In: Baeten, J.C.M., Mauw, S. (eds.) Concurrency Theory (CONCUR '99). Lecture Notes in Computer Science, vol. 1664. Springer-Verlag, Eindhoven, The Netherlands, August 1999, pp. 399–414
14. Lüttgen, G., von der Beeck, M., Cleaveland, R.: A Compositional Approach to Statecharts Semantics. In: Proc. of ACM SIGSOFT Eighth Int. Symp. on the Foundations of Software Engineering (FSE-8). ACM, 2000, pp. 120–129
15. Maggiolo-Schettini, A., Peron, A., Tini, S.: Equivalences of Statecharts. In: Montanari, U., Sassone, V. (eds.) CONCUR '96 (Concurrency Theory). Lecture Notes in Computer Science, vol. 1119. Springer-Verlag, Pisa, Italy, August 1996, pp. 687–702
16. Mikk, E., Lakhnech, Y., Siegel, M.: Hierarchical automata as model for Statecharts. In: Proceedings of Asian Computing Science Conference (ASIAN '97). Lecture Notes in Computer Science, vol. 1345. Springer-Verlag, December 1997
17. OMG. OMG Unified Modeling Language Specification. Version 1.4, 2001
18. U2 Partners. Unified Modeling Language 2.0 Proposal, version 0.671 (draft). `http://www.u2-partners.org`, 2002
19. Paltor, I., Lilius, J.: Formalising UML state machines for model checking. In: France, R., Rumpe, B. (eds.) UML'99 – The Unified Modeling Language. Beyond the Standard. LNCS, vol. 1723. Springer, 1999
20. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Department, Aarhus University, Denmark, 1981
21. Pnueli, A., Shalev, M.: What is in a step: On the semantics of Statecharts. In: Ito, T., Meyer, A. (eds.) Theoretical Aspects of Computer Software (TACS '91) Lecture Notes in Computer Science, vol. 526. Springer-Verlag, Sendai, Japan, September 1991, pp. 244–264
22. Reggio, G., Astesiano, E., Choppy, C., Hussmann, H.: Analysing UML Active Classes and Associated State Machines – A Lightwight Formal Approach. In: Fundamental Approaches to Software Engineering. LNCS, vol. 1783. Springer, 2000, pp. 127–146
23. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1998
24. Scholz, P.: Design of Reactive Systems and their Distributed Implementation with Statecharts. PhD thesis, Munich University of Technology, Munich, Germany, August 1998
25. Uselton, A., Smolka, S.: A compositional semantics for Statecharts using labeled transition systems. In: Jonsson, B., Parrow, J. (eds.) CONCUR '94 (Concurrency Theory). Lecture Notes in Computer Science, vol. 836. Springer-Verlag, Uppsala, Sweden, August 1994, pp. 2–17
26. von der Beeck, M.: A Concise Compositional Statecharts Semantics Definition. In: Proc. of FORTE/PSTV 2000. Kluwer, 2000, pp. 335–350
27. von der Beeck, M.: Formalization of UML-Statecharts. In: Gogolla, M., Kobryn, C. (eds.) UML 2001 – The Unified Modeling Language: Modeling Languages, Concepts, and Tools. Lecture Notes in Computer Science, vol. 2185. Springer-Verlag, 2001, pp. 406–421

Michael von der Beeck works as research and development engineer in the department Function Development Process of BMW Group, Munich, Germany. He is the author or co-author of more than twenty refereed conference and workshop papers on software engineering and formal description techniques, in particular on the UML and statecharts. He has been working for program committees of several international workshops.