INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Semantic Issues in UML 2.0 State Machines

Harald Fecher, Marcel Kyas, Jens Schönborn

Bericht Nr. 0507
June 2, 2005

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Semantic Issues in UML 2.0 State Machines*

Harald Fecher, Marcel Kyas, and Jens Schönborn

Christian-Albrechts-Universität zu Kiel, Germany
{hf,mky,jes}@informatik.uni-kiel.de

Abstract. A precise semantics for the modeling language UML 2.0 is necessary for code generation and for formal verification of models during the early stages of design. We present a formal semantics of the firing of transitions in UML 2.0 state machines. In particular, we handle shallow/deep history pseudostates, final states, join/fork pseudostates, entry/exit actions and the different kinds of transition. Furthermore, our semantics captures all orderings of actions in a run-to-completion step. We point out ambiguities and uncertainties in the UML 2.0 standard, especially in the meaning of history pseudostates and priority between transitions. We discuss different attempts in resolving these.

1 Introduction

UML has become the standard modeling language for object-oriented programs. But in the UML 2.0 standard [12] its semantics is only informally defined. It is, however, necessary to develop a precise formal semantics in order to rule out ambiguities and to develop tool support for UML, e.g., simulators, verifiers, or code generators.

Here, we present a formal syntax and a formal semantics of *UML 2.0 (behavioral) state machines*, which are widely used for modeling the reactive behavior of object-oriented systems. State machines have evolved from Harel's statecharts [5] and their object-oriented version [6]. The main difference between Harel's statecharts and UML state machines is that in statecharts steps are triggered by a "time tick" and the reaction is supposed to occur in zero time, whereas in state machines steps triggered by events, i.e., messages the object receives, and that the execution of actions is assumed to

* Part of this work has been financially supported by IST project Omega (IST-2001-33522) and NWO/DFG project Mobi-J (RO 1122/9-1, RO 1122/9-2)

take time[1]. Hence, the existing semantics for statecharts cannot be used for state machines.

In this paper, we focus on determining the set of transitions that are fired as a result of a given event and to which configurations the firing of the transitions will lead. The order in which transitions fire is deliberately left unspecified in the standard. Our semantics captures all orderings of actions of a run-to-completion step. The validity of our semantics is argued on the basis of explicit citations of the relevant parts of the UML standard. Our language of UML state machines includes the most important features. In particular, our language includes history pseudostates, final states, join and fork pseudostates, and transition kind.

We point out ambiguities and uncertainties in the UML 2.0 standard [12], especially in the case of history pseudostates and priority between transitions. We discuss different attempts of resolving these. We do not consider the formal definition of event selection mechanisms and a global step semantics, i.e., the semantics of a system containing many communicating state machines. In future work, a semantics based on transition system will be the foundation of the global semantics.

The paper is structured as follows: In the next section, we present the syntax, where we also justify the syntactical simplification we make. The semantics is given in Sec. 3 and related work is discussed in Sec. 4.

2 Syntax of UML State Machines

For simplicity, we do not consider:

- The concept of redefinition, which is not a behavioral issue.
- Parameters on signal events, which can be straightforwardly added.
- Events of *stereo type create* [12, p. 499,471], since we do not consider creation of new objects.

> An initial transition at the topmost level (region of a state machine) either has no trigger or it has a trigger with the stereotype create. [12, p. 499]

[1] That time is needed is a consequence of the fact that sending a synchronous message to another object blocks the sender until a response is received, but the receiver may delay this message in preference of other actions.

> [...] the transition from an initial pseudostate may be labeled with the trigger event that creates the object; otherwise, it must be unlabeled. [12, p. 471]

– *Termination pseudostates* [12, p. 471], since we could not find out when exactly the state machines vanish. Do all transitions of a firing set of transitions fire, or are the remaining transitions dropped as soon as a transition to a termination pseudostates is completed? Do all active states have to be exited before a state machine vanishes?

> Entering a terminate pseudostate implies that the execution of this state machine by means of its context object is terminated. [12, p. 471]

– *Completion event/transition* [12, p. 500], since the UML standard does not precisely say in which cases and which completion events are generated. Moreover, do completion events trigger transitions without trigger or do they trigger completion transitions corresponding to other completions (that already took place)?

> A completion transition is a transition where the source is a composite state, [...] without an explicit trigger, although it may have a guard defined. When all transitions and entry activities and state (do) activities in the currently active state are completed, a completion event is generated. This event is the implicit trigger for a completion transition. The completion event is dispatched before any other events in the pool and has no associated parameters. [12, p. 500]

In order to give our formal presentation, we use the following notations: $M \rightharpoonup M'$ denotes the set of all partial functions from M into M' and the cardinality of M is given by $|M|$. The operator $\cdot$ denotes concatenation of words (and is sometimes omitted to increase readability), ϵ denotes the empty word, $\_^*$ denotes the Kleene star operation, last the function returning the last symbol of a word, and start the word without its last symbol. We write $w \prec w'$ if w is proper prefix of w' and we write $w \preceq w'$ if $w \prec w' \vee w = w'$.

Definition 1. *Let* $\{N_{\text{comp}}, N_{\text{reg}}, N_{\text{final}}, \{\text{D}\}, \{\text{H}\}, \{\text{H}^\star\}\}$ *be a partition of a set* N *of* state names*. Set* N_{comp} *represents the names of* composite states*, set* N_{reg} *the names of* regions*, set* N_{final} *the names*

of final states*, symbol* D *an* initial state*, symbol* H *a* shallow history state*, and symbol* $\mathrm{H}^\star$ *a* deep history state[2].

A *region* collects states and transitions and is like a sub-statemachine. The transitions of a region do not necessarily originate from a state of that region or target a state of this region. In this case we speak of *inter-level* transitions.

In the following, the states and also the pseudostates of a state machine will be sequences of elements from N, i.e., the states are identified with the *path* of names to s, as is also done in OCL [1]. In particular, the region and the composite state identification will alternate in a sequence encoding a state. Before we present the formal definition, we introduce some further notation: For any set of words $W \subseteq N^*$ define its *down-set* $\downarrow W \stackrel{\text{def}}{=} \{w' \in N^* \mid \exists w \in W : w' \preceq w\}$ and its *up-set* $\uparrow W \stackrel{\text{def}}{=} \{w' \in N^* \mid \exists w \in W : w \preceq w'\}$. Furthermore, the *longest common prefix* of all words from a non-empty set of words W is denoted by $\mathrm{LCP}(W)$.

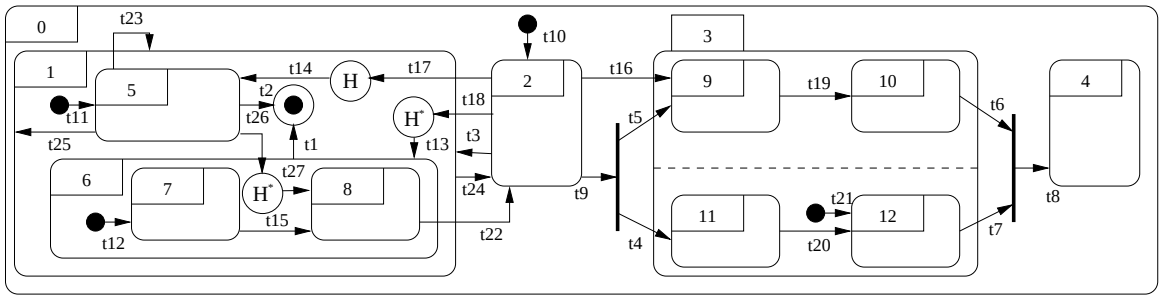
Definition 2. *A* set of states $\mathcal{S}$ *is a non-empty finite subset of* $\mathcal{N}_{\text{state}} \stackrel{\text{def}}{=} (N_{\text{comp}} \cdot N_{\text{reg}})^* \cdot (N_{\text{comp}} \cup N_{\text{final}})$ *that is prefix closed with respect to* $\mathcal{N}_{\text{state}}$*, i.e.,* $\mathcal{S} = \downarrow \mathcal{S} \cap \mathcal{N}_{\text{state}}$*. Furthermore, the set of all* regions of $\mathcal{S}$ *is* $\mathcal{S}_{\text{reg}} \stackrel{\text{def}}{=} (N_{\text{comp}} \cdot N_{\text{reg}})^* \cap \downarrow \mathcal{S}$ *and the set of all* final states of $\mathcal{S}$ *is* $\mathcal{S}_{\text{final}} \stackrel{\text{def}}{=} ((N_{\text{comp}} \cdot N_{\text{reg}})^* \cdot N_{\text{final}}) \cap \mathcal{S}$.

Example 1. Suppose $N_{\text{comp}} = \{0, 1, 2, \ldots\}$, $N_{\text{reg}} = \{a, b, c, \ldots\}$ and $N_{\text{final}} = \{\bar{0}, \bar{1}, \bar{2}, \ldots\}$. Then the states of the state machine of Fig. 1 can be encoded, e.g., as $2 \hat{=} 0a2$, $9 \hat{=} 0a3a9$, $11 \hat{=} 0a3b11$, and the final state as $0a1a\bar{0}$.

For readability, we also use the state name instead of the path when we identify the states of the state machine of Fig. 1, e.g., we use 1 instead of $0a1$.

Later, we will sometimes use *pseudostates*. Pseudostates are like states in that they may be sources or targets of transitions, but never part of a state configuration (see below). To represent them, we extend an element of N_{reg} with an additional symbol, like D, H or

[2] Please note that the star is only a notational index and has nothing to do with the Kleene star operation.



Let $t^{(j)} = (e^{(j)}, \varphi^{(j)}, \alpha^{(j)})$. The events and guards are omitted from transitions from pseudostates and also from transition t^6 and t^7.

Fig. 1. Example state machine.

H*. Note that initial pseudostates and history pseudostates are not allowed in $\mathcal{S}$, since they are only pseudostates. We call a set of states *orthogonal* if its pairwise distinct states belong to different regions of an orthogonal state [12, p. 478].

> A composite state either contains one region or is decomposed into two or more orthogonal regions. Each region has a set of mutually exclusive disjoint subvertices and a set of transitions. [...] An orthogonal composite state contains two or more regions. [12, p. 478]

Formally:

$$\mathsf{Orthogonal} \stackrel{\text{def}}{=} \{W \subseteq N^* \mid \forall w, w' \in W : \mathsf{last}(\mathrm{LCP}(\{w, w'\})) \notin N_{\mathrm{reg}} \wedge \neg(w \prec w')\}.$$

In the following definition we introduce some given sets.

Definition 3. *An* action *is a sequence of atomic actions (like changing value of attributes, sending signals, and creating new objects). In the following, we will use symbol $\mathcal{A}$ to denote the set of all actions. Let* skip $\in \mathcal{A}$ *denote the neutral element, i.e., the do-nothing action. Set $\mathcal{E}$ denotes the set of all possible* events *and set $\mathcal{G}$ denotes a set of boolean expressions, which depend on global information like the attribute values of the objects.*

For example OCL [1] can be used as a description language for $\mathcal{G}$.

Next we define the allowed transition of a state machine (see [12, p. 498,470,499]). Here the transitions are presented through compound transitions [12, pp. 500].

> A *compound transition* is a derived semantic concept, representing a "semantically complete" path made of one or more transitions, originating from a set of states [...] and targeting a set of states. [12, p. 500]

- trigger: Trigger[0..*] Specifies the triggers that may fire the transition.
- guard: Constraint[0..1] A guard is a constraint that provides a fine-grained control over the firing of the transition. The guard is evaluated when an event is dispatched by the state machine. If the guard is true at that time, the transition may be enabled, otherwise, it is disabled. Guards should be pure expressions without side effects. Guard expressions with side effects are ill formed.
- effect: Activity[0..1] Specifies an optional activity to be performed when the transition fires.
- source: Vertex[1] Designates the originating vertex (state or pseudostate) of the transition.
- target: Vertex[1] Designates the target vertex that is reached when the transition is taken.

[12, p. 498]

> [3] In a complete state machine, a join vertex must have at least two incoming transitions and exactly one outgoing transition.
> [4] All transitions incoming a join vertex must originate in different regions of an orthogonal state.
> [5] In a complete state machine, a fork [sic] vertex must have at least two outgoing transitions and exactly one incoming transition.
> [6] All transitions outgoing a fork vertex must target states in different regions of an orthogonal state. [12, p. 470]

> Transitions outgoing pseudostates may not have a trigger. [12, p. 499]

Definition 4. *A* transition *t with respect to a set of states $\mathcal{S}$ is a tuple $(\hat{S}_1, e, \varphi, \alpha, \widetilde{\alpha}_1, \widetilde{\alpha}_2, \hat{S}_2, \ell)$ such that:*

- *$\hat{S}_1$ denotes the set of* source states *where final states have no outgoing transitions, i.e., $\hat{S}_1 \neq \emptyset \wedge \hat{S}_1 \subseteq \mathcal{S} \setminus \mathcal{S}_{\text{final}}$,*
- *$\hat{S}_2$ denotes the set of* target states *where transitions from a fork pseudostate may not target pseudostates, i.e., $\hat{S}_2 \neq \emptyset \wedge (\hat{S}_2 \subseteq \mathcal{S} \vee \exists r \in \mathcal{S}_{\text{reg}} : (\hat{S}_2 = \{r \cdot \mathrm{H}\} \vee \hat{S}_2 = \{r \cdot \mathrm{H}^\star\}))$,*
- *the source states (the target states) of the transition belong to orthogonal regions, i.e., $\hat{S}_1 \in$ Orthogonal and $\hat{S}_2 \in$ Orthogonal,*
- *$e \in \mathcal{E}$ is the* event *by which the transition is triggered,*
- *$\varphi \in \mathcal{G}$ is its* guard *constraining when the transition can be triggered,*
- *$\widetilde{\alpha}_1 : \hat{S}_1 \to \mathcal{A}$ provides actions to the transitions into a join pseudostate,*
- *$\widetilde{\alpha}_2 : \hat{S}_2 \to \mathcal{A}$ provides actions to the transitions leaving a fork pseudostate,*
- *α is its action (sequence), which has to be executed between the actions from $\widetilde{\alpha}_1$ and $\widetilde{\alpha}_2$,*
- *$\ell \in \mathcal{S} \cup \mathcal{S}_{\text{reg}}$, called* scope*, determines which states are exited/entered (those states which are proper substates of ℓ) when the transition is fired. Furthermore, all substates of the source state(s) have to be exited and all substates of the target state(s) have to be entered, i.e., $\hat{S}_1 \subseteq \uparrow\{\ell\} \wedge \hat{S}_2 \subseteq \uparrow\{\ell\}$.*

Note that some transitions, e.g., those involved in a join or fork pseudostates of an UML state machines are represented by a single transition in our definition. For example, the transition corresponding to the fork of the state machine presented in Fig. 1 is represented by

$$(\{2\}, e, \varphi^{(9)}, \alpha^{(9)}, \{2, \text{skip}\}, \{(9, \alpha^{(5)}), (11, \alpha^{(4)})\}, \{9, 11\}, 0a)$$

The scope of a transition is determined by the outermost region (state) that contains the whole drawing of the transition. For example, transition $t23$ has scope $0a$, whereas transition $t25$ has scope $0a3$.

It is not clear to us whether the UML state machines are sensitive to the way they are drawn. The concept of ℓ is also used to

model transitions with *transition kind* local [12, p. 506]. A transition with transition kind local can only have a single source state, since otherwise the firing of this transition will in general contradict the invariant that the complete firing of transitions results in configurations. A transitions with transition kind local, which we also call a local transition, has as its scope its source state. This is one reason why we also allow states as the scope of transitions. Note that we do not need the restriction that the source states of transition with transition kind local or external have to be composite states.

> [1] The source state of a transition with transition kind local must be a composite state.
> [2] The source state of a transition with transition kind external must be a composite state. [12, p. 506]

Note that a transition resulting from a join pseudostate, i.e., with $|\hat{S}_1| > 1$, may not have a trigger [12, p. 499,471]

> Transitions outgoing pseudostates may not have a trigger. [12, p. 499]

> A join segment must not have guards or triggers. [12, p. 499]

> The transitions entering a join vertex cannot have guards or triggers. [12, p. 471]

Therefore, join transitions are triggered by a completion event.

In UML 2.0, sets of events are allowed as trigger. But those transitions can be equivalently represented by a set of transitions that have only a single events as their trigger.

In the following, we introduce internal transitions [12, p. 506], i.e., those transition that do not leave states. They can only has a single source state and the target state have to coincide with the source state.

> kind=internal implies that the transition, if triggered, occurs without exiting or entering the source state. Thus, it does not cause a state change. This means that the entry or exit condition of the source state will not be invoked. [12, p.506]

Definition 5. *An* internal transition t_{int} *with respect to a set of states* $\mathcal{S}$ *is a tuple* $(\hat{S}_1, e, \varphi, \alpha)$ *such that*

– $\hat{S}_1$ *is a single set of the* source state*, where final states have no internal transitions, i.e.,* $\exists s \in \mathcal{S} \setminus \mathcal{S}_{\text{final}} : \hat{S}_1 = \{s\}$*,*
– $e \in \mathcal{E}$ *is its* event *on which the transition is triggered,*
– $\varphi \in \mathcal{G}$ *is its* guard *constraining when the transition can be triggered,*
– α *is its action (sequence), which will be executed.*

Now we are ready to present our UML 2.0 state machine syntax:

Definition 6. *A* UML state machine M *is a tuple* $(\mathcal{S}, \rho_{\text{D}}, \rho_{\text{H}}, \rho_{\text{H}^\star}, \mathsf{entry}, \mathsf{exit}, \mathsf{doActivity}, \mathsf{defer}, \rightarrow, \rightarrow_{\text{int}})$*, where:*

– $\mathcal{S}$ *is a set of states,*
– $\rho_{\text{D}} : \mathcal{S}_{\text{reg}} \rightharpoonup \mathcal{S} \times \mathcal{A}$*, where* $\rho_{\text{D}}^{\mathcal{S}}$ *and* $\rho_{D}^{\mathcal{A}}$ *denote their corresponding projections, assigns to a region* s *one of its direct substates as its initial state* $\rho_{\text{D}}^{\mathcal{S}}(s)$ *initial transition* $\rho_{D}^{\mathcal{A}}(s)$*.*
– $\rho_{\text{H}} : \mathcal{S}_{\text{reg}} \rightharpoonup \mathcal{S} \times \mathcal{A}$*, where* $\rho_{\text{H}}^{\mathcal{S}}$ *and* $\rho_{\text{H}}^{\mathcal{A}}$ *denote their corresponding projections, assigns to a region one of its direct substates, which is the default state for a shallow history state and the action of its corresponding transition.*
– $\rho_{\text{H}^\star} : \mathcal{S}_{\text{reg}} \rightharpoonup \mathcal{S} \times \mathcal{A}$*, where* $\rho_{\text{H}}^{\mathcal{S}}$ *and* $\rho_{\text{H}}^{\mathcal{A}}$ *denote their corresponding projections, assigns to a region one of its direct substates, which is the default state for a deep history state and the action of its corresponding transition.*
– $\mathsf{entry} : (\mathcal{S} \setminus \mathcal{S}_{\text{final}}) \rightarrow \mathcal{A}$ *assigns to each state the action that has to be executed when the state is entered.*
– $\mathsf{exit} : (\mathcal{S} \setminus \mathcal{S}_{\text{final}}) \rightarrow \mathcal{A}$ *assigns to each state the action that has to be executed when the state is exited.*
– $\mathsf{doActivity} : (\mathcal{S} \setminus \mathcal{S}_{\text{final}}) \rightarrow \mathcal{A}$ *assigns to each state the action that can be executed when the state is active.*
– $\mathsf{defer} : \mathcal{S} \rightarrow 2^{\mathcal{E}}$ *assigns to each state those events that will be deferred.*
– $\rightarrow$ *is a set of transitions with respect to* $\mathcal{S}$*.*
– $\rightarrow_{\text{int}}$ *is a set of internal transitions with respect to* $\mathcal{S}$*.*

Note that the UML standard does not enforce an initial transition in each region or default transition on history states[3]. Therefore, we

[3] UML states that the state machine is ill defined if the semantics needs some of these non present transitions. This will also be discussed later.

use partial function on the corresponding positions in the above definitions. For example, the history default transition $t^{(13)}$ is encoded by $\rho_{\mathrm{H}^\star}(0a1) = (0a1a6, \alpha^{(13)})$.

Our syntax does not contain an explicit correspondence for submachine states (and therefore also no entry and exit pseudostates), junction pseudostates, and transition to initial pseudostates, since they can be compiled away [12, pp. 478,471].

> A submachine state is semantically equivalent to a composite state. [12, pp. 478]

> junction vertexes are semantic-free vertexes that are used to chain together multiple transitions. [12, pp. 471]

In the remainder of this paper, assume a fixed state machine

$$M = (\mathcal{S}, \rho_{\mathrm{D}}, \rho_{\mathrm{H}}, \rho_{\mathrm{H}^\star}, \mathsf{entry}, \mathsf{exit}, \mathsf{doActivity}, \mathsf{defer}, \rightarrow, \rightarrow_{\mathrm{int}}).$$

3 Semantics of UML State Machines

We define configurations of a state machine, where we also introduce the concepts of history configurations. Then a run-to-completion step of a state machine is carried out as follows: The selection mechanism of an event is out of scope of this paper. Therefore, we assume that event e is given for transition triggering, where already the deferring of events is taken into account. In Section 3.2, we determine which sets of transitions may fire. The obtained configuration after firing a set of transitions is determined in Section 3.3 and the collection of the actions that have to be executed is determined in Sect. 3.4. The execution of actions (including the do-activity actions), which can modify the global information, is again out of scope of this paper.

3.1 Configuration

In a hierarchical state machine more than one state can be active. The set of all current active states are called the *active state configuration*, or configuration for short [12, p. 481].

> Except during transition execution, the following invariants always apply to state configurations:

– If a composite state is active and not orthogonal, exactly one of its substates is active.
– If the composite state is active and orthogonal, all of its regions are active, one substate in each region.

[12, p. 481]

Definition 7. *The set of all* configurations $\mathcal{C}$ *is*

$$\mathcal{C} \stackrel{\text{def}}{=} \big\{\hat{S} \subseteq \mathcal{S} \mid (\forall s \in \hat{S} : \mathcal{S} \cap \downarrow \{s\} \subseteq \hat{S}) \wedge \forall r \in \mathcal{S}_{\text{reg}} : (r = \epsilon \vee \mathsf{start}(r) \in \hat{S}) \Rightarrow |\{n \in N \mid rn \in \hat{S}\}| = 1\big\}$$

Configurations of the state machine of Fig. 1 are, e.g., $\{0, 1, 5\}$, $\{0, 4\}$, and $\{0, 3, 9, 12\}$. On the other hand, $\{2\}$, $\{0, 3, 9\}$ and $\{0, 2, 4\}$ are no configurations.

The information which states are currently active is not enough, since we have history pseudostates. Unfortunately, the semantical behavior of transitions that point to a history pseudostate from inside the region is not clear in UML 2.0. Here, we take the approach that the last active states are activated and not, e.g., the states that were active at the moment the corresponding region was last exited (different interpretations are possible). Therefore, history configurations, which remember the last (respectively, the current) active direct substate of a region, are introduced as follows:

Definition 8. *Function:* $\text{collect} : (\mathcal{S}_{\text{reg}} \rightharpoonup N) \rightharpoonup (\mathcal{S} \to 2^{\mathcal{S}})$*, where* $\text{collect}(\chi)(s)$ *collects substates of* s*, where function* χ *indicates which direct substate of a region should be collected. Formally:*

$$\text{collect}(\chi)(s) \stackrel{\text{def}}{=} \{s\} \cup \bigcup_{\{r \in \text{dom}(\chi) \mid \mathsf{start}(r) = s \wedge r \cdot \chi(r) \in \mathcal{S}\}} \text{collect}(\chi)(r \cdot \chi(r)).$$

A history configuration χ *is a partial function* $\chi : \mathcal{S}_{\text{reg}} \rightharpoonup N$ *such that:*

– *It may assign to a region one of its direct substates (which is/was last active), i.e.,* $\forall r \in \text{dom}(\chi) : r \cdot \chi(r) \in \mathcal{S}$.
– *An outermost state was visited, i.e.,* $\chi(\epsilon)$ defined.
– *If a state was visited then all its ancestors were visited, i.e.,* $\forall r, r' \in \mathcal{S}_{\text{reg}} : (\chi(r) \text{ defined} \wedge r' \preceq r) \Rightarrow \chi(r')$ defined.

– *Every defined position corresponds to a partial configuration, i.e.,* $\forall r \in \mathcal{S}_{\text{reg}} : \chi(r)$ defined $\Rightarrow \exists \kappa \in \mathcal{C} : \text{collect}(\chi)(r \cdot \chi(r)) = \kappa \cap \uparrow \{r\}$.

The configuration κ *is compatible with the history configuration* χ *if* $\text{collect}(\chi)(\chi(\epsilon)) = \kappa$. *The configuration that is compatible with the history configuration* χ *is denoted by* κ_χ.

Example 2. $\{(\epsilon, 0), (0a, 2), (0a1a, \bar{0})\}$ is a history configuration (of the state machine of Fig. 1) that is compatible to the configuration $\{0, 2\}$, where the state encoding of Example 1 is used.

3.2 Fireable Transitions

Fireable transitions are determined through the concepts of enabled, conflict and priority of transitions [12, p. 493].

> The set of transitions that will fire is a maximal set of transitions that satisfies the following conditions:
> – All transitions in the set are enabled.
> – There are no conflicting transitions within the set.
> – There is no transition outside the set that has higher priority than a transition in the set (that is, enabled transitions with highest priorities are in the set while conflicting transitions with lower priorities are left out).
> See [12, p. 493]

Enabled transitions. They are specified in [12, p. 500].

> A transition is enabled if and only if:
> – All of its source states are in the active state configuration.
> – One of the triggers of the transition is satisfied by the current event. An event satisfies a trigger if it matches the event specified by the trigger. In case of signal events, since signals are generalized concepts, a signal event satisfies a signal event associated with the same signal or a generalization thereof.
> – If there exists at least one full path from the source state configuration to [...] the target state configuration [...] in which all guard conditions are true (transitions without guards are treated as if their guards are always true).

See [12, p. 500]

In order to determine enabled transitions, we postulate a predicate $\mathcal{G}[\![\varphi]\!] \subseteq \mathcal{C}$ that determines all configurations for which the guard $\varphi \in \mathcal{G}$ evaluates to true, where we abstract from the global information. To model generalized signals and events we assume that the set of events is partially ordered by $\leq$ with the largest element any.

Definition 9. *The set of all enabled transition with respect to configuration κ and event e is given by*

$$\mathsf{enabled}\,(\kappa, e) \stackrel{\text{def}}{=} \{(\hat{S}_1, e', \varphi, \dots) \in \rightarrow \cup \rightarrow_{\text{int}} |\ \hat{S}_1 \subseteq \kappa \wedge \kappa \in \mathcal{G}[\![\varphi]\!] \wedge e \leq e'\}.$$

Conflicting transitions. These are specified in [12, p. 492].

> Two transitions are said to conflict if [...] the intersection of the set of states they exit is non-empty. Only transitions that occur in mutually orthogonal regions may be fired simultaneously. [12, p. 492]
>
> Each orthogonal region in the active state configuration that is not decomposed into orthogonal regions (i.e., bottomlevel region) can fire at most one transition as a result of the current event. [12, p. 492]
>
> An internal transition in a state conflicts only with transitions that cause an exit from that state. [12, p. 492]

Definition 10. *The conflict relation between two transitions $t, t' \in \rightarrow \cup \rightarrow_{\text{int}}$ is given by*

$$\begin{aligned}\mathsf{conflict}\,(t, t') \iff\ & t_1 \neq t_2 \wedge \\ & \Big(\big(t = (\dots, \ell), t' = (\dots, \ell') \in \rightarrow \wedge (\uparrow \{\ell\}) \cap (\uparrow \{\ell'\}) \neq \emptyset\big) \vee \\ & \big(t = (\dots, \ell) \in \rightarrow \wedge t' = (\{s'\}, \dots) \in \rightarrow_{\text{int}} \wedge \ell \prec s'\big) \vee \\ & \big(t = (\{s\}, \dots) \in \rightarrow_{\text{int}} \wedge t' = (\dots, \ell') \in \rightarrow \wedge \ell' \prec s\big)\Big)\end{aligned}$$

The states that will be left when they are active and the local or external transition t fires is determined by $(\uparrow \{\ell\}) \setminus \{\ell\}$, compare Def. 13. In the above definition, we use $(\uparrow \{\ell\})$ instead. This is done to model that any local transition with a simple state as its source state is in conflict with any external transition that leaves this simple state.

Priority between transitions. Unfortunately, it is not clear to us how priority between transitions is defined. A description is given in the section concerning 'Firing priorities' [12, p. 493] and in the description of the Greedy algorithm that determines the set of transitions that will fire [12, p. 493].

> In situations where there are conflicting transitions, the selection of which transitions will fire is based in part on an implicit priority. These priorities resolve some transition conflicts, but not all of them. The priorities of conflicting transitions are based on their relative position in the state hierarchy. By definition, a transition originating from a substate has higher priority than a conflicting transition originating from any of its containing states.
> The priority of a transition is defined based on its source state. The priority of joined transitions is based on the priority of the transition with the most transitively nested source state. In general, if t1 is a transition whose source state is s1, and t2 has source s2, then:
> – If s1 is a direct or transitively nested substate of s2, then t1 has higher priority than t2.
> – If s1 and s2 are not in the same state configuration, then there is no priority difference between t1 and t2.
> [12, p. 493]

> This can be easily implemented by a greedy selection algorithm, with a straightforward traversal of the active state configuration. States in the active state configuration are traversed starting with the innermost nested simple states and working outwards. For each state at a given level, all originating transitions are evaluated to determine if they are enabled. This traversal guarantees that the priority principle is not violated. The only non-trivial issue is resolving transition conflicts across orthogonal states on all levels. This is resolved by terminating the search in each orthogonal state once a transition inside any one of its components is fired. [12, p. 493]

It is not even clear if priority exists between single source transitions from orthogonal regions, since a statement about this is not

made. To define priority through a "most transitively nested substate" does not make sense, since this state cannot be uniquely determined in general. Hence, the priority is nondeterministic and cycles in the priority relation are possible, which leads to contradictions. In the following we suggest how priority can be defined:

Definition 11. *That $t = (\hat{S}_1, \dots)$ has priority over $t' = (\hat{S}'_1, \dots)$, written* $\mathsf{priority}\,(t, t')$*, can be, e.g., formally defined by:*

- *t has priority over t' if there exists a source state of t that is a proper substate of a source state of t':*

$$\exists s \in \hat{S}_1, s' \in \hat{S}'_1 : s' \prec s \tag{1}$$

- *(Motivated by the given greedy algorithm)*
  - *If level is interpreted as distance to simple states:*

$$\mathsf{conflict}\,(t, t') \wedge \max_{s \in \hat{S}_1}(\max\{|w| \mid s \cdot w \in \mathcal{S}\}) < \max_{s' \in \hat{S}'_1}(\max\{|w| \mid s' \cdot w \in \mathcal{S}\}) \tag{2}$$

  - *If level is interpreted as distance to the outermost region:*

$$\mathsf{conflict}\,(t, t') \wedge \min\{|s| \mid s \in \hat{S}_1\} > \min\{|s'| \mid s' \in \hat{S}'_1\} \tag{3}$$

  - *If level is ignored: t has priority over t' when every source state of t is a substate of a source state of t' where one of which is at a proper substate:*

$$(\forall s \in \hat{S}_1 : \exists s' \in \hat{S}'_1 : s' \preceq s) \wedge (\exists s \in \hat{S}_1, s' \in \hat{S}'_1 : s' \prec s) \tag{4}$$

The consequence of the different definitions are illustrated using the partial state machines of Fig. 2:

(1): The priority order is problematic, because it contains cycles: $t3$ has priority over $t2$, $t2$ over $t1$, and $t1$ over $t3$, hence no transition can fire [12, p. 493]. It has the additional disadvantage that adding a transition generally increases interdependencies, e.g., $t5$ or $t4$ cannot fire, but if $t5$ is removed $t4$ can fire independently from $t6$.

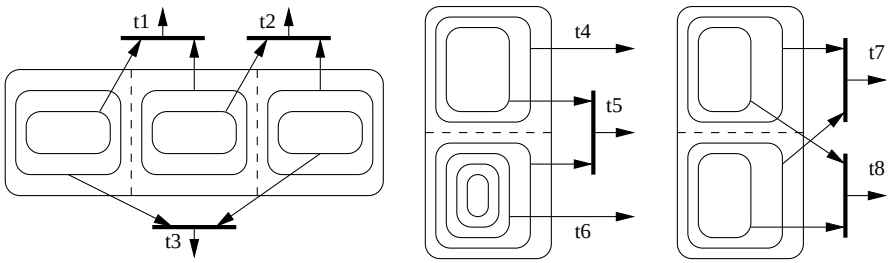


Fig. 2. Priority illustration

(2): Here $t4$ has priority over $t5$ and over $t6$, and $t6$ has priority over $t5$.
(3): Here $t6$ has priority over $t4$ and over $t5$, and no priority between $t4$ and $t5$ exists.
(4): Here $t6$ has priority over $t5$ and no further priorities exists between $t4$, $t5$ and $t6$.

In the cases (2), (3) and (4), $t8$ has always priority over $t7$ and no priorities exist between $t1$, $t2$ and $t3$.

The ambiguity obtained from priority can be avoided if no join transitions are used. Instead single source transitions can be used, where guards encode which states have to be active. Furthermore, the priority can be encoded in the guards by taking the original guard conjuncted with the negation of the guards corresponding to the transitions that have higher priority. The disadvantage of this encoding is, that, on the one hand, the guard expression language must be expressive enough and, on the other, complex guards are generated. Note that fork transitions can usually not be avoided, since fork transitions are the only way to enforce to activate certain states of orthogonal regions. It is possible to encode the activation of certain states through initial transition, but then this is fixed for all transition. Hence, it would not be possible to activate different states of orthogonal regions in different situations, which would be a strong restriction.

Fireable Transitions. As already mentioned [12, p. 493]:

> The set of transitions that will fire is a maximal set of transitions that satisfies the following conditions:

– All transitions in the set are enabled.
– There are no conflicting transitions within the set.
– There is no transition outside the set that has higher priority than a transition in the set (that is, enabled transitions with highest priorities are in the set while conflicting transitions with lower priorities are left out).

[12, p. 493]

Definition 12. *The set of transitions* $T \subseteq \rightarrow \cup \rightarrow_{\text{int}}$ *can be fired with respect to configuration* κ *and event* e *if:*

– $T \subseteq \mathsf{enabled}\,(\kappa, e)$,
– $\forall t_1, t_2 \in T : \neg\, \mathsf{conflict}\,(t_1, t_2)$,
– $\forall t' \in \mathsf{enabled}\,(\kappa, e) \setminus T : \exists t \in T : \mathsf{conflict}\,(t, t')$, *and*
– $\forall t' \in \mathsf{enabled}\,(\kappa, e) \setminus T : \forall t \in T : \neg\, \mathsf{priority}\,(t', t)$.

Note that in our case the set of transitions that can be fired is nonempty, since, otherwise, the event is deferred or dropped, which we assume is already handled by the event selecting mechanism.

Consider Fig. 1: In case transitions $t^{(1)}$ and $t^{(22)}$ have the same event, then the set $\{t^{(22)}\}$ is the only set of transitions that will fire in the configuration $\{0, 1, 6, 8\}$ with respect to all presented priority variations.

3.3 Successor Configuration

First we determine the states that are left by the firing of a transition:

Definition 13. *The states from a configuration* κ *that are left by a transition are defined by*

$$\mathsf{leave}(\kappa, t) \stackrel{\text{def}}{=} \begin{cases} \kappa \cap \big((\uparrow \{\ell\}) \setminus \{\ell\}\big) & \text{if } t = (\ldots, \ell) \in \rightarrow \\ \emptyset & \text{if } t \in \rightarrow_{\text{int}} \end{cases}$$

For example in Fig. 1, if $t^{(22)}$ fires and $t^{(22)}$ is external, then states 8, 6, and 1 are left but not state 0. In the case that $t^{(23)}$ fires in the configuration $\{0, 3, 9, 11\}$, then states 9, 11, and 3 are left. On the other hand, if $t^{(25)}$ fires in this configuration, then only states 9 and 11 are left.

In order to determine the states that are entered, we introduce function $\mathsf{activate}_\chi(s, \hat{S})$, which collects all substates of state s that have to be entered when a transition to $\hat{S}$ is taken. In particular, we have to ensure that all states from $\hat{S}$ will be entered. Depending on whether $s'' \in \hat{S}$ is a history state, we have to take the history configuration into account. In particular, the default values are used instead of the history when it is the first time the state is entered or if the last active substate of the state was a final state. Therefore, we introduce the following predicate stating this circumstance:

$$\mathsf{nonVisited}(r) \overset{\text{def}}{\iff} (\ \text{undefined}(\chi(r)) \vee \chi(r) \in N_{\text{final}})$$

and, for the opposite case, we use $\mathsf{Visited}(r) \overset{\text{def}}{\iff} \neg\mathsf{nonVisited}(r)$. In the definition of $\mathsf{activate}_\chi$, we also collect the initial pseudostates that are entered in order to easily collect later the initial transition actions.

Definition 14. *Let χ be a history configuration. Define* $\mathsf{activate}_\chi : N^* \times \mathsf{Orthogonal} \rightharpoonup 2^{N^*}$ *by:*

$$\mathsf{activate}_\chi(w, \hat{S}) \overset{\text{def}}{=} \begin{cases} \{w\} \cup \bigcup_{\{r \in \mathcal{S}_{\text{reg}} \mid \mathsf{start}(r) = w\}} \mathsf{activate}_\chi(r, \hat{S}) & \text{if } w \in \mathcal{S}, \\ \mathsf{activate}_\chi(\rho_h^{\mathcal{S}}(w), \hat{S}) & \text{if } wh \in \hat{S} \wedge h \in \{\mathrm{H}, \mathrm{H}^\star\} \wedge \mathsf{nonVisited}(w), \\ \mathsf{activate}_\chi(w\chi(w), \hat{S}) & \text{if } w\mathrm{H} \in \hat{S} \wedge \mathsf{Visited}(w), \\ (\text{collect}(\chi)(w\chi(w)) \setminus \mathcal{S}_{\text{final}}) \cup \bigcup_{s' \in \mathcal{S}_{\text{final}} \cap \text{collect}(\chi)(w\chi(w))} \mathsf{activate}_\chi(\rho_{\mathrm{D}}^{\mathcal{S}}(\mathsf{start}(s')), \hat{S}) & \\ & \text{if } w\mathrm{H}^\star \in \hat{S} \wedge \mathsf{Visited}(w), \\ \mathsf{activate}_\chi(s', \hat{S}) & \text{if } \mathsf{start}(s') = w \in \mathcal{S}_{\text{reg}} \wedge \mathsf{last}(s') \notin \{\mathrm{H}, \mathrm{H}^\star\} \wedge \\ & \quad \exists s'' \in \hat{S} : s' \preceq s'' \\ \{w\mathrm{D}\} \cup \mathsf{activate}_\chi(\rho_{\mathrm{D}}^{\mathcal{S}}(w), \hat{S}) & \text{otherwise.} \end{cases}$$

We give some comments on the definition of $\mathsf{activate}_\chi(s, \hat{S})$. If a composite state has to be activated, then all its direct subregions have to be activated. If a history state of a region has to be activated and the region was not visited before, the history default state will be activated. If a shallow history of a visited region has to be activated then the last visited state has to be activated if it was not a final

state, in which case the history default state is activated. In both cases the activation of the substates will be done by the default transitions. If a deep history of a visited region has to be activated then all substates determined by the history configuration are collected except that the substates will be activated by default in the case that a final state was last active. This is also for nested regions the case, since UML states that the rule will be applied recursively (see citation below), i.e., the possibility that the nested final states become active does not occur. If a substate of a region has to be activated ($\exists s'' \in \hat{S} : s' \preceq s''$), then the region activates its direct substate containing s''. If a region does not contain a substate that has to be activated, the substates are activated by the default transitions.

The standard does not explicitly state whether the default activation of nested states is determined by the initial pseudostates or by the deep history pseudostates when a deep history state is activated [12, pp. 481].

> *Shallow history entry:* If the transition terminates on a shallow history pseudostate, the active substate becomes the most recently active substate prior to this entry, unless the most recently active substate is the final state or if this is the first entry into this state. In the latter two cases, the default history state is entered. This is the substate that is target of the transition originating from the history pseudostate. (If no such transition is specified, the situation is ill defined and its handling is not defined.) If the active substate determined by history is a composite state, then it proceeds with its default entry.
> Deep history entry: The rule here is the same as for shallow history except that the rule is applied recursively to all levels in the active state configuration below this one. [12, pp. 481]

We decided to determine the state that is entered by the initial pseudostates, e.g., in Fig. 1 state 7 rather than state 8 becomes active after firing transition $t18$ in the history configuration of Example 2.

Note that $\mathsf{activate}_\chi(s, \hat{S})$ is not always defined. This occurs if $\rho^{\mathcal{S}}_{\mathrm{D}}(s)$, $\rho^{\mathcal{S}}_{\mathrm{H}}(s)$ or $\rho^{\mathcal{S}}_{\mathrm{H}^\star}(s)$ is not defined. The UML standard states that such a situation is ill defined [12, pp. 481].

If no such transition is specified, the situation is ill defined and its handling is not defined. [12, p. 481]

Definition 15. *The states (including initial pseudostates) that are entered through a transition t with respect to a history configuration χ are defined by*

$$\mathsf{enter}(\chi, t) \stackrel{\text{def}}{=} \begin{cases} \mathsf{activate}_\chi(\ell, \hat{S}_2) \setminus \{\ell\} & \text{if } t = (\ldots, \hat{S}_2, \ell) \in \rightarrow \\ \emptyset & \text{if } t \in \rightarrow_{\text{int}}. \end{cases}$$

For example, the states (different from pseudostates) that are entered by firing $t^{(17)}$ in Fig. 1 with respect to the history configuration given in Example 2 are 1 and 5.

Definition 16. *Let χ be a history configuration. The successor configuration and the successor history configuration with respect to a set of transition T that is fired is determined by*

$$\mathsf{succConf}(\chi, T) \stackrel{\text{def}}{=} (\kappa_\chi \setminus \bigcup_{t \in T} \mathsf{leave}(\kappa_\chi, t)) \cup (\bigcup_{t \in T} \mathsf{enter}(\chi, t) \setminus (N^* \cdot \{\mathrm{D}\}))$$

$$\mathsf{succHistConf}(\chi, T)(r) \stackrel{\text{def}}{=} \begin{cases} n & \text{if } rn \in \mathsf{succConf}(\chi, T), \\ \text{undefined} & \text{if } \exists s \in \mathsf{succConf}(\chi, T) \cap \mathcal{S}_{\text{final}} : \mathsf{start}(s) \prec r, \\ \chi(r) & \text{otherwise.} \end{cases}$$

Note that it is not clear if nested history information is reset to default, as is done in our definition, as soon as a final state of an upper region becomes active [12, pp. 481]. That this makes a difference is illustrated as follows: consider in Fig. 1 the firing sequence $(t^{(18)}, t^{(1)}, t^{(24)}, t^{(17)}, t^{(26)})$. In the case that it is not reset then state 7 will be entered. In the case that it is reset, as is done in our definition, state 8 instead of 7 will be entered.

Proposition 1. *The successor configuration (the successor history configuration) yields, if existent, a configuration (respectively, a history configuration). Furthermore, the successor configuration is compatible with the successor history configuration.*

3.4 Collecting Actions

The action execution corresponding to a transition is performed by exiting the source state, executing the actions of the transition, and entering the target states [12, p. 501].

> Once a transition is enabled and is selected to fire, the following steps are carried out in order:
> - The main source state is properly exited.
> - Activities are executed in sequence following their linear order along the segments of the transition: The closer the activity to the source state, the earlier it is executed.
> - [...]
> - The main target state is properly entered.
>
> [12, p. 501]

Before we collect the corresponding actions, we investigate a mechanism that allows to specify more flexible action sequence execution and we introduce special actions corresponding to entering/exiting a state.

Non-deterministic action sequence. In order to order the collected actions of a firing set of transitions, we introduce *non-deterministic action sequences*, denoted by β, which are either an action sequence, a sequence of non-deterministic action sequences or a finite collection of ordered non-deterministic action sequences, where the later is denoted by $\bigoplus^{(\mathcal{I},\leq)}(\beta_i)^{i\in I}$, with $(\mathcal{I},\leq)$ a partially ordered set and I a finite subset of $\mathcal{I}$. The interpretation of a non-deterministic action sequence yields a set of possible action sequences such that the order $\leq$ is respected. For example,

$$\Big(\bigoplus^{(\mathcal{S},\preceq^{-1})}(\alpha_i)^{i\in\{r,rn,r\hat{n}\}}\Big);\Big(\bigoplus^{(\mathcal{S},\preceq)}(\tilde{\alpha}_i)^{i\in\{r,rn,r\hat{n}\}}\Big) = \big\{\alpha_{rn};\alpha_{r\hat{n}};\alpha_r;\tilde{\alpha}_r;\tilde{\alpha}_{rn};\tilde{\alpha}_{r\hat{n}}\ ,$$
$$\alpha_{rn};\alpha_{r\hat{n}};\alpha_r;\tilde{\alpha}_r;\tilde{\alpha}_{r\hat{n}};\tilde{\alpha}_{rn}\ ,\ \alpha_{r\hat{n}};\alpha_{rn};\alpha_r;\tilde{\alpha}_r;\tilde{\alpha}_{rn};\tilde{\alpha}_{r\hat{n}}\ ,\ \alpha_{r\hat{n}};\alpha_{rn};\alpha_r;\tilde{\alpha}_r;\tilde{\alpha}_{r\hat{n}};\tilde{\alpha}_{rn}\big\}.$$

In the special case that I is the empty set, $\bigoplus^{(\mathcal{I},\leq)}(\beta_i)^{i\in I}$ yields the do-nothing action.

Actions for entering and exiting states. We introduce special actions indicating the entering and exiting of a state. Such actions update the state configurations, because conditions and actions may test whether an intermediate state is active, if OCL is used as an expression language [1].

Definition 17. *Let* α_s^{enter} *be the action indicating that state* s *is entered and let* α_s^{exit} *be the action indicating that state* s *is exited.*

Collecting exit actions. States are exited from the innermost to the outermost [12, p. 479,482,480,506] and the state is immediately left after the execution of the exit actions [12, p. 480]:

> exit: Activity[0..1] An optional activity that is executed whenever this state is exited regardless of which transition was taken out of the state. If defined, exit actions are always executed to completion only after all internal activities and transition actions have completed execution. [12, p. 479]
>
> *Exiting non-orthogonal states:* When exiting from a composite state, the active substate is exited recursively. This means that the exit activities are executed in sequence starting with the innermost active state in the current state configuration.
> *Exiting an orthogonal states* When exiting from an orthogonal state, each of its regions is exited. After that, the exit activities of the state are executed. [12, p. 482]
>
> [. . .] whenever a state is exited, it executes its exit activity as the final step prior to leaving the state. [12, p. 480]
>
> kind=internal implies that the transition, if triggered, occur without exiting or entering the source state. Thus, it does not cause a state change. This means that the entry or exit condition of the source state will not be invoked. An internal transition can be taken even if the state machine is in one or more regions nested within this state. [12, p.506]

Definition 18. *The non-deterministic action sequence corresponding to the exiting of the firing transition* t *in configuration* κ *is*

$$\mathsf{actExit}(\kappa, t) = \bigoplus^{(N^*, \preceq^{-1})} (\mathsf{exit}(s); \alpha_s^{\text{exit}})^{s \in \mathsf{leave}(\kappa, t)},$$

where $\mathsf{exit}(s)$ *denotes the do-nothing action if* $s \in \mathcal{S}_{\text{final}}$.

Note that in case t is an internal transition, $\mathsf{actExit}(\kappa, t)$ yields the do-nothing action.

Collecting transition actions. Actions of transitions are executed along the transition [12, p. 501].

> Activities are executed in sequence following their linear order along the segments of the transition: The closer the activity to the source state, the earlier it is executed. [12, p. 501]

Furthermore, "A transition to the enclosing state represents a transition to the initial pseudostate in each region." [12, p. 478][4]

Definition 19. *The non-deterministic action sequence corresponding to transition* t *with respect to a history configuration* χ *is given by*

$$\mathsf{actTrans}(\chi, t) \stackrel{\text{def}}{=} \begin{cases} \left(\bigoplus^{(N^*,\mathrm{id})} (\widetilde{\alpha}_1(s_1))^{s_1 \in \hat{S}_1}\right); \alpha; \hat{\alpha} & \text{if } t = (\hat{S}_1, \ldots, \alpha, \widetilde{\alpha}_1, \widetilde{\alpha}_2, \hat{S}_2, \ell) \in \rightarrow \\ \alpha & \text{if } t = (\ldots, \alpha) \in \rightarrow_{\text{int}} \end{cases}$$

where $\hat{\alpha}$ *is given by*

$$\hat{\alpha} \stackrel{\text{def}}{=} \begin{cases} \widetilde{\alpha}_2(rh); \rho_h^{\mathcal{A}}(r); \widehat{\beta}_{\rho_h^{\mathcal{S}}(r)} & \text{if } \{rh\} = \hat{S}_2 \wedge h \in \{\mathrm{H}, \mathrm{H}^\star\} \wedge \mathsf{nonVisited}(r), \\ \widetilde{\alpha}_2(r\mathrm{H}); \widehat{\beta}_{\chi(r)} & \text{if } \{r\mathrm{H}\} = \hat{S}_2 \wedge \mathsf{Visited}(r), \\ \widetilde{\alpha}_2(r\mathrm{H}^\star) & \text{if } \{r\mathrm{H}^\star\} = \hat{S}_2 \wedge \mathsf{Visited}(r), \\ \bigoplus^{(N^*,\preceq)} (\widehat{\alpha}_s)^{s \in \hat{S}_2 \cup \left((\mathcal{S}_{\text{reg}} \cdot \{\mathrm{D}\}) \cap \bigcup_{s_2 \in \hat{S}_2} \mathsf{activate}_\chi(s_2, \emptyset)\right)} & \text{if } \hat{S}_2 \subseteq \mathcal{S} \end{cases}$$

with $\widehat{\beta}_s \stackrel{\text{def}}{=} \bigoplus^{(N^*,\preceq)} \left(\rho_D^{\mathcal{A}}(\mathsf{start}(s'))\right)^{s' \in ((\mathcal{S}_{\text{reg}} \cdot \{\mathrm{D}\}) \cap \mathsf{activate}_\chi(s, \emptyset))}$ *and*

$$\widehat{\alpha}_s \stackrel{\text{def}}{=} \begin{cases} \widetilde{\alpha}_2(s) & \text{if } s \in \hat{S}_2, \\ \rho_D^{\mathcal{A}}(\hat{r}) & \text{if } s = \hat{r} \cdot \mathrm{D}. \end{cases}$$

[4] This actually contradicts "The entry activity of the composite state is executed before the activity associated with the initial transition [12, p. 481]". In this paper, we chose to have the transitions target the initial pseudostate.

In the following, we give some comments on this definition.The execution of an internal transition is clear. In the case of a compound transition: First the actions corresponding to the transition into the join pseudostates are executed in no specific order. Then the actions of the transition followed by the actions corresponding to the transition 'after the fork pseudostate' are executed. In case the transition does not point to a history pseudostate (it points to a set of real states) the action of the transition leaving the fork states executed together with the implicit triggered initial transition is executed, where an initial transition can only be executed when its outermost initial transitions (respectively, the corresponding fork transition) have been executed. It is not clear to us what the UML standard mean with closest in the case of the firing of the initial pseudostates ("The closer the activity to the source state, the earlier it is executed."[12, p. 501]). More precisely, does the transition of a fork execute, e.g., each branch (with respect to the initial transition of orthogonal regions) completely before another branch is considered (depth-first strategy), or do, e.g., all outermost triggered initial transition execute before the initial transitions that are one step deeper are executed (branching-first strategy)? In order to be more general, we do not enforce any further ordering on these executions.

In case the target is a history pseudostate and its corresponding region was not visited before (or a final state was last active) then the actions of the default transition is executed; and thereafter the implicit triggered initial transition is executed as described before. If the target is a deep history state and its region was visited before, then no further action is executed during that phase. On the other hand, if the target is a shallow history state and its region was visited before, then the implicit triggered initial transition of the last active direct substate has to be executed, since "A transition coming into the shallow history vertex is equivalent to a transition coming into the most recent active substate of a state" [12, p. 470].

The collected actions corresponding to the fork transition in Fig. 1 is $\alpha^{(9)};\bigoplus^{(N^*,id)}(\beta_i)^{i\in\{0a3a9,0a3b11\}}$ with $\beta_{0a3a9}=\alpha^{(5)}$ and $\beta_{0a3b11}=\alpha^{(4)}$.

Collecting entry actions. States are entered from the outermost to the innermost [12, p. 479,481] and the state is immediately entered before the execution of the entry actions [12, p. 480].

> entry: Activity[0..1] An optional activity that is executed whenever this state is entered regardless of the transition taken to reach the state. If defined, entry actions are always executed to completion prior to any internal activity or transitions performed within the state. [12, p. 479]

> *Entering a non-orthogonal composite state*
> [...]
> *Explicit entry:* If the transition goes to a substate of the composite state, then that substate becomes active and its entry code is executed after the execution of the entry code of the composite state. This rule applies recursively if the transition terminates on a transitively nested substate.
> [...]
> *Entering an orthogonal composite state*
> Whenever an orthogonal composite state is entered, each one of its orthogonal regions is also entered, either by default or explicitly. If the transition terminates on the edge of the composite state, then all the regions are entered using default entry. If the transition explicitly enters one or more regions (in case of a fork), these regions are entered explicitly and the others by default. [12, p. 481]

> Whenever a state is entered, it executes its entry activity before any other action is executed. [12, p. 480]

Furthermore, all initial transition executed in the collecting transition part have to be ruled out.

Definition 20. *The non-deterministic action sequences corresponding to the entering of the firing transition* $t = (\ldots, \hat{S}_2, \ell)$ *in the his-*

tory configuration χ *are*

$$\mathsf{actEnter}(\chi,t) \stackrel{\text{def}}{=} \begin{cases} \bigoplus^{(N^*,\preceq)} (\alpha_s^{\text{enter}}; \mathsf{entry}(s))^{s\in\mathsf{enter}(\chi,t)\setminus\left((\mathcal{S}_{\text{reg}}\cdot\{\text{D}\})\cap\mathsf{activate}_\chi(\rho_h^{\mathcal{S}}(r),\emptyset)\right)} \\ \qquad \textit{if } t=(\ldots,\hat{S}_2,\ell)\in\!\!\rightarrow \wedge\{rh\}=\hat{S}_2\wedge h\in\{\text{H},\text{H}^\star\}\wedge\mathsf{nonVisited}(r), \\ \bigoplus^{(N^*,\preceq)} (\alpha_s^{\text{enter}}; \mathsf{entry}(s))^{s\in\mathsf{enter}(\chi,t)\setminus((\mathcal{S}_{\text{reg}}\cdot\{\text{D}\})\cap\mathsf{activate}_\chi(\chi(r),\emptyset))} \\ \qquad \textit{if } t=(\ldots,\hat{S}_2,\ell)\in\!\!\rightarrow \wedge\{r\text{H}\}=\hat{S}_2\wedge\mathsf{Visited}(r), \\ \bigoplus^{(N^*,\preceq)} (\alpha_s^{\text{enter}}; \mathsf{entry}(s))^{s\in\mathsf{enter}(\chi,t)} \\ \qquad \textit{if } t=(\ldots,\hat{S}_2,\ell)\in\!\!\rightarrow \wedge\{r\text{H}^\star\}=\hat{S}_2\wedge\mathsf{Visited}(r), \\ \bigoplus^{(N^*,\preceq)} (\alpha_s^{\text{enter}}; \mathsf{entry}(s))^{s\in\mathsf{enter}(\chi,t)\setminus\left((\mathcal{S}_{\text{reg}}\cdot\{\text{D}\})\cap\bigcup_{s_2\in\hat{S}_2}\mathsf{activate}_\chi(s_2,\emptyset)\right)} \\ \qquad \textit{if } t=(\ldots,\hat{S}_2,\ell)\in\!\!\rightarrow \wedge\hat{S}_2\subseteq\mathcal{S} \\ \emptyset \qquad \textit{if } t\in\!\!\rightarrow_{\text{int}} \end{cases}$$

where $\mathsf{entry}(s\text{D})$ *is considered to be* $\rho_D^{\mathcal{A}}(s)$, $\alpha_{s'\text{D}}^{\text{enter}}$ *denotes the do-nothing action, and if* $s\in\mathcal{S}_{\text{final}}$ *then* $\mathsf{entry}(s)$ *denotes the do-nothing action.*

Note that when, e.g., $t^{(16)}$ fires in Fig. 1 the action of the initial transition $t^{(21)}$ can be independently executed from the entry activity execution of state 12. This seems strange, but we could not find a comment that enforces an order on these executions.

Collecting all actions.

Definition 21. *The possible action sequences obtained by firing an allowed set of transitions* $T\subseteq\!\!\rightarrow$ *in the history configuration* χ *is*

$$\mathsf{actCollect}(\chi,T)=\bigoplus^{(2^{\rightarrow},\text{id})}\Big(\mathsf{actExit}(\kappa_\chi,t);\mathsf{actTrans}(\chi,t);\mathsf{actEnter}(\chi,t)\Big)^{t\in T}.$$

Note that the set of states obtained by modifying the configuration via the actions α_s^{exit} and α_s^{enter} in $\mathsf{actCollect}(\chi,T)$ leads to the configuration $\mathsf{succConf}(\chi,T)$, as required.

4 Related Work

Operational semantics of state machines for the purpose of model checking have been defined in, e.g., [9], based on the work presented

in [10]. As most of the following work it does not cover fork and join states. Von der Beeck presents a semantics for a subset of UML state machines in [13] using an approach similar to ours. While a semantics for history states is defined in this paper, join and fork transitions are not considered. Its semantics focuses on communication and control and does not discuss history states, action execution, and activities.

Most language features have been formalized by Börger et al. [2, 3], who define a semantics for UML 1.x state machines using abstract state machines. UML 2.0 state machines are different from UML 1.x state machines; therefore, a semantics for UML 1.x state machines cannot be easily adapted to our case.

The semantics defined in Damm et. al. [4] presents a semantics for UML state machines which differs in many ways from the informally defined semantics in UML 2.0. There, a run-to-completion step is defined by selecting one enabled transition instead of a set of transitions. In [4] a transition which does not specify a trigger is taken immediately, whereas the standard requires such a transition requires a *completion event* to trigger. The notion of an activity group introduced by Damm has no counterpart in UML 2.0 and introduces additional synchronization and scheduling problems. This semantics was used in [7, 8, 11], where state machines were formalized for theorem proving or model checking.

All mentioned work is based on UML 1.x. We are not aware of a paper that covers UML 2.0 state machines to the extent we do. In particular, we cover the new semantics of history states and the semantics implied by transition kinds. Except for the work of Börger, none of the mentioned work covers fork and join transitions.

References

1. Boldsoft and Rational Software Corporation and IONA and Adaptive Ltd. *UML 2.0 OCL Specification*, Oct. 2003. Available for download at `http://www.omg.org/cgi-bin/doc?ptc/2003-10-14`.
2. E. Börger, A. Cavarra, and E. Riccobene. Solving conflicts in UML state machines concurrent states. In *Workshop on Concurrency Issues in UML (CIUML), Toronto, October 2, 2001*, 2001.
3. E. Börger, A. Cavarra, and E. Riccobene. Modeling the meaning of transitions from and to concurrent states in UML state machines. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), March 9–12, 2003, Melbourne, FL, USA*, pages 1086–1091. ACM Press, 2003.

4. W. Damm, B. Josko, A. Pnueli, and A. Votintseva. Understanding UML: A formal semantics of concurrency and communication in real-time uml. In F. S. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Formal Methods for Components and Objects*, number 2852 in Lecture Notes in Computer Science. Springer-Verlag, 2003.
5. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, July 1987.
6. D. Harel and E. Gery. Executable object modeling with statecharts. *Computer*, 30(7):31–42, July 1997.
7. J. Hooman and M. van der Zwaag. A semantics of communicating reactive objects with timing. In S. Graf, O. Haugen, I. Ober, and B. Selic, editors, *1st Workshop on Specification and Validation of UML Models for Real Time and Embedded Systems, SVERTS 2003*, Verimag technical report 2003/10/22. Verimag, 2003. Available online at `http://www-verimag.imag.fr/EVENTS/2003/SVERTS/`.
8. M. Kyas, H. Fecher, F. S. de Boer, M. van der Zwaag, J. Hooman, T. Arons, and H. Kugler. Formalizing UML models and OCL constraints in PVS. In G. Lüttgen, N. M. Madrid, and M. Mendler, editors, *Proceedings of the Second Workshop on Semantic Foundations of Engineering Design Languages (SFEDL 2004)*, volume 115 of *Electronic Notes in Theoretical Computer Science*, pages 39–47. Elsevier, 2005.
9. D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11(6):637–664, 1999.
10. E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In R. K. Shyamasundar and K. Ueda, editors, *Advances in Computing Science - ASIAN '97, Third Asian Computing Science Conference, Kathmandu, Nepal, December 9-11, 1997, Proceedings*, number 1345 in Lecture Notes in Computer Science, pages 181–196. Springer-Verlag, 1997.
11. I. Ober, S. Graf, and I. Ober. Validation of UML models via a mapping to communicating extended timed automata. In S. Graf and L. Mounier, editors, *Model Checking Software: 11th International SPIN Workshop*, number 2989 in Lecture Notes in Computer Science, pages 127–145. Springer-Verlag, 2004.
12. Object Management Group. *UML 2.0 Superstructure Specification*, Oct. 2004. `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`.
13. M. von der Beeck. A structured operational semantics for UML-statecharts. *Software and Systems Modeling*, 1(2):130–141, Dec. 2002.