

# UML 2.0 State Machines: Complete Formal Semantics via Core State Machines \*

Harald Fecher and Jens Schönborn

Christian-Albrechts-University at Kiel, Germany

hf@informatik.uni-kiel.de and jes@informatik.uni-kiel.de

**Abstract.** UML has become the standard modeling language for object-oriented systems. The informal description of UML and its continuous extension cause many ambiguities. Therefore, a formal semantics for UML is necessary, especially for formal reasoning and tool development. We present a formal semantics of UML 2.0 state machines, which are used for modeling the reactive behavior of objects, by (i) deriving core state machines with fewer design features and a precise syntax, (ii) developing a formal semantics for core state machines, and (iii) presenting a complete transformation from UML 2.0 state machines to core state machines. Such a transformational approach provides the opportunity of easy adaption to future changes of the semantics of UML state machines.

## 1 Introduction

UML [8] has become the standard modeling language for object-oriented systems. It constitutes the de-facto-standard for industrial applications in many areas - especially in the object-oriented domain, but it also gains in importance for modeling embedded real-time systems. Its advantages are given by a great variety of intuitive and mostly well-known notations for different kinds of information to be specified: requirements, static structure, interactive and dynamic behavior, as well as physical implementation structures. However, it suffers from an insufficiently precise definition. This and the continuous extension of UML cause many ambiguities and inconsistencies, see, e.g., [9, 3, 10]. Therefore, a formal semantics for UML is necessary, especially for formal reasoning and tool development.

A standard technique to handle such high complexities is to derive a sublanguage, sometimes called core language, for which a precise syntax and a formal semantics are defined. Then the semantics of the original language is given via transformation into the core language. Having such a transformational approach for UML state machines semantics provides, e.g., the opportunity of easy adaption to future changes and independent examination and tool development can be build on the core languages.

In this paper we focus on defining a complete formal semantics of single UML 2.0 state machines, which are one of the most important constituents of UML, since they are widely used for modeling the reactive behavior of objects. UML state machines have evolved from Harel's statecharts [6] and their object-oriented version [5]. In particular, our contributions are:

---

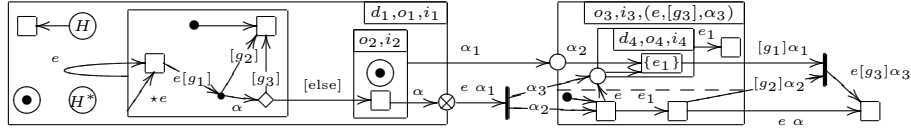
\* Part of this work is financially supported by DFG project Refism (FE 942/1-1)

- In Section 3, we present core state machines consisting of composite and final states, regions, choice and entry/exit pseudostates, internal/external transitions, do actions, and event deferral.
- In Section 4, we present a formal semantics of core state machines in terms of transition systems, which are chosen as semantical model in order to enhance readability. A translation of our semantics to abstract state machines [4], which are more suitable for verification, can be straightforwardly defined.
- In Section 5, we give a transformation from UML 2.0 state machines into core state machines such that a precise semantics coincident with the UML standard is obtained. In particular, all aspects of single UML 2.0 state machines, i.e., join, fork, junction, choice, entry, exit, history, initial pseudostates, do, entry, exit actions, event deferral, completion, local, internal, external, conflicting transitions, and priority relations between transitions are handled. Parameters on events, which are omitted for readability, can straightforwardly be added. By this transformation, we rule out the ambiguities and inconsistencies of UML 2.0 state machines observed earlier by us in [3], where possible solutions are illustrated.
- Especially, we handle the following non-trivial aspects that are in our opinion not adequately handled in the literature: choice pseudostates (including the decision which state is left at which point in time); the order of execution of actions of compound transitions (including entry/exit pseudostates); transitions pointing to a history pseudostate from inside its containing region; the effect that event deferral can have priority over transition firing.

Note that core state machines can also be reused as semantical basis for other variations of statecharts.

*Related work.* For an overview of existing formal semantics of UML state machines see [1], where 26 different approaches to the semantics are well compared. We only discuss the most relevant works in detail. All of the cited papers there, except of [11] and a technical report of us [2] consider previous versions of UML state machines, i.e., they do not deal with UML 2.0 state machines. This is significant, since there have been major changes and additions in the syntax and semantics, i.e., entry/exit pseudostates are allowed at any composite state, the concept of local transitions has been added, and event deferral can have priority over transition firing. None of these challenging aspects are considered in [11], where only a small subset of UML 2.0 state machines is considered. In contrast to our technical report [2], we now handle entry/exit pseudostates at any composite state and the fact that event deferral can have priority over transition firing, as well as choice pseudostates and completion transitions. It is not obvious how the existing semantics can be extended to handle exit/entry pseudostates at any composite state, i.e., how to define a formalization for the order of execution of the actions of compound transitions.

The paper that handles most state machines aspects is [7]. Besides the new aspects of UML 2.0 state machines, the authors do not address in detail the order of execution of actions (which already occurs in earlier UML versions by, e.g., the usage of join/fork pseudostates). All other works mentioned in [1] handle fewer aspects than [7].



**Fig. 1.** A UML 2.0 state machine. Here we depict states as rectangular, final states as circles surrounding a solid filled smaller circle, shallow (deep) history pseudostates as circles containing ‘H’ (resp. ‘H\*’), initial/junction pseudostates as solid filled circles, join/fork as fat lines, choice pseudostates as diamond-shaped symbols, and entry (exit) pseudostates as small circles (resp., as small circles with a cross) on the border of the state. Regions are separated by dashed lines. The set of events deferred at a state is written inside the state. Transition labels divide or are written closely to the transition, where guard true and action skip are omitted. Events are written as  $e$ , guards as  $g$ , and transition actions as  $\alpha$ , possibly attached with indices. One of the transitions outgoing a choice or a junction pseudostate may be labeled with guard  $[else]$ . Transitions with an additional label  $*$  have type local. Do, exit, entry actions and internal transitions are written, if present, in a small box in the upper right corner of the corresponding state via  $d, o, i, (e[g]\alpha)$ , respectively.

## 2 Informal description of UML 2.0 state machines

Before we give the informal description, we refer to Figure 1, where a state machine is illustrated. Note that this picture is mainly used to illustrate our transformation in Section 5 and therefore is not always meaningful, e.g., the final states are meaningless, since no transitions pointing to them.

The basic concepts of UML 2.0 state machines are states and transitions between states. A state may contain regions (called direct subregions of that state) and a region must contain states (called direct substates of that region) such that this hierarchy yields a tree structure. A configuration describes the currently active states, where exactly one direct substate of an active region must be active and all regions of an active state must be active. Entering a state (via firing a transition) corresponds to its activation whereas exiting a state corresponds to its deactivation. A state may have *entry* actions (executed when the state is entered), *exit* actions (executed when the state is left), and *do* actions ((partly) executed as long as the state is active). In the following, executing do actions means (partial) execution of such actions. The environment may send events to the state machine. These are collected in the event pool of the state machine. A state machine may either execute do actions of active states or may dispatch a single event from its event pool to trigger transitions. UML state machines follow the *run-to-completion* assumption, i.e., “an event occurrence can only be taken from the pool and dispatched if the processing of the previous current occurrence is fully completed” [8, p. 546]. Furthermore, a dispatched event that does not trigger transitions is either discarded or deferred, if there is an active state that is specified to defer the corresponding event. When a state completes its do actions and only final states (that are special states without outgoing transitions) are active in its subregions, then a completion transition<sup>1</sup> of that state, selected depending on the guard, becomes enabled. The completion transition has to be fired before another event can be dispatched.

<sup>1</sup> encoded via compound transitions having no event

Besides its source and target, a transition consists of an event (optional), a guard (a boolean expression), and actions. Join and fork pseudostates, which may also be source and target of transitions, are used to collect different transitions into a compound transition having a set of sources and a set of targets. Note that further pseudostates, which will be discussed later, are used to obtain more complex compound transitions. A compound transition is enabled if its source state is currently active, its event is dispatched from the event pool, and its guard evaluates to true. Among the enabled transitions a maximal set of pairwise conflict-free elements is fired. Two compound transitions are in conflict if the intersection of the sets of states that will be left by firing these transitions is non empty. An active state  $s$  will be exited by firing transition  $t$  if, roughly speaking, the target of  $t$  is not a substate of  $s$ . A non-active state  $s$  will be entered by firing transition  $t$  if, roughly speaking, (i)  $s$  is the target of  $t$ , (ii) the target of  $t$  is a substate of  $s$  or (iii)  $s$  is an element of a recursively defined chain of substates (by using initial pseudostates) from the target of  $t$ . Furthermore, a priority relation determining which one of conflicting and enabled compound transitions will fire exists. But this relation does not resolve nondeterminism completely. Roughly speaking compound transitions having deeper nested source states have priority. The firing of a compound transition  $t$  leads, in this order, to (i) exiting the corresponding states together with the execution of their exit actions, (ii) the execution of the actions of  $t$ , and (iii) entering the corresponding states together with the execution of their entry actions. The order of the action execution can be changed (to a certain degree) by using exit and entry pseudostates: the actions of the states and transitions that are above the exit (below the entry) pseudostates are executed after the actions of the states and transitions that are below the exit (resp., above the entry) pseudostates w.r.t. state hierarchy.

Further pseudostates are junction pseudostates, that are a shorthand notation for the set of transitions obtained by combining any incoming with any outgoing transition; choice pseudostates lead to a new decision, where the effects of the previously executed transitions are taken into account when determining the outgoing transition to be fired; history pseudostates activate those substates of the region, to which the history pseudostate belongs, that were active when the region was the last time active. The shallow-history pseudostates consider only the direct substates of a region  $r$ , whereas deep history pseudostates consider also deeper nested substates. In case  $r$  was not active before or the last active direct substate of  $r$  is a final state, the history default transition, that is the (unique) transition leaving the history pseudostate, is fired instead. Furthermore, a transition can be external, local, or internal. An internal transition does not exit any state and a local compound transition does not exit its source state, but its substates.

### 3 Syntax of core state machines

Core state machines, derived from UML 2.0 state machines, use only composite and final states, regions, do actions, event deferral, and choice, entry, exit pseudostates. No interlevel transitions, i.e., transitions crossing a state border, are allowed, instead additional exit, entry pseudostates are used. Default exit and entry must be explicitly modeled. Exit and entry pseudostates are used as junction as well as join (resp., fork) pseudostates in some sense and have, therefore, a slightly augmented meaning as in

UML 2.0. We use three different kinds of exit pseudostates: a normal one, a priority relevant one, and a completion relevant one, which is only ‘enabled’ if the do actions of the corresponding state have terminated (but not necessarily its corresponding regions). States are only allowed as sources and targets of internal transitions. Two additional predicates, occurring in guards of transitions, are used for the modeling of history entry. This construct classification is summarized below.

core constructs		derived constructs
unmodified	modified	
composite and final states, regions, do actions, event deferral, choice, external/local transitions, completion	exit (pr, npr, cp) and entry pseudostates	interlevel transitions, join/fork, junction, history, exit and entry pseudostates, internal transitions, initial pseudostates

**Definition 1 (States & regions).**  $\mathcal{S}$  denotes a set of states, partitioned into composite, final, exit, entry, and choice states, denoted by  $\mathcal{S}_{\text{com}}$ ,  $\mathcal{S}_{\text{fin}}$ ,  $\mathcal{S}_{\text{exit}}$ ,  $\mathcal{S}_{\text{entry}}$ , and  $\mathcal{S}_{\text{choice}}$ , respectively. Furthermore,  $\mathcal{S}_{\text{exit}}$  is partitioned into priority, non-priority, and completion (relevant) exit states, denoted by  $\mathcal{S}_{\text{exit}}^{\text{pr}}$ ,  $\mathcal{S}_{\text{exit}}^{\text{npr}}$ , and  $\mathcal{S}_{\text{exit}}^{\text{cp}}$ , respectively. Set  $\mathcal{S}$  is consistent with a set of regions  $\mathcal{R}$  and parent function  $\text{parent}$  if the outermost region, denoted by  $\epsilon$ , is contained in  $\mathcal{R}$ ,  $\text{parent}$  maps composite, final, and choice states to regions, maps regions (different from  $\epsilon$ ) and entry states to composite states, and maps exit states to composite or final states such that the derived containing relation  $\succ$  defined as the transitive closure of  $\{(z, \text{parent}(z)) \mid z \in \mathcal{S} \cup \mathcal{R} \setminus \{\epsilon\}\}$  is irreflexive.

Note that for technical reasons we also allow that exit states belong to final states and not only to composite states. In the following, we assume fixed  $\mathcal{S}$ ,  $\mathcal{R}$ , and  $\text{parent}$  such that  $\mathcal{S}$  is consistent w.r.t.  $\mathcal{R}$  and  $\text{parent}$ . Furthermore,  $\succ$  denotes the derived containing relation from  $\text{parent}$  and  $\succeq$  denotes the reflexive closure of  $\succ$ . Moreover, functions  $\text{stateOf} : \mathcal{S} \rightarrow (\mathcal{S}_{\text{com}} \cup \mathcal{S}_{\text{fin}} \cup \mathcal{S}_{\text{choice}})$  and  $\text{regOf} : \mathcal{S} \rightarrow \mathcal{R}$  yield the deepest composite, final, or choice state (resp. region) that contains the argument. Formally:

$$\text{stateOf}(s) = \begin{cases} s & \text{if } s \in \mathcal{S}_{\text{com}} \cup \mathcal{S}_{\text{fin}} \cup \mathcal{S}_{\text{choice}} \\ \text{parent}(s) & \text{otherwise} \end{cases}$$

$$\text{regOf}(s) = \text{parent}(\text{stateOf}(s))$$

In the following definition, we introduce the sets of actions, events, and guards, whose elements are, e.g., used as transition labels.

**Definition 2 (Actions).** 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  $\text{Actions}$  to denote the set of all actions. Let  $\text{skip} \in \text{Actions}$  denote the termination of the sequence execution. Furthermore, let  $(\mathcal{B}, \text{Actions}, \rightsquigarrow, \surd)$  be a labeled transition system having a labeled termination predicate, i.e.,  $\mathcal{B}$  is the set of transition system states,  $\rightsquigarrow \subseteq \mathcal{B} \times \text{Actions} \times \mathcal{B}$ , and  $\surd \subseteq \mathcal{B} \times \text{Actions}$ , where  $(B, \alpha) \in \surd$  indicates that an execution of  $\alpha$  which leads to termination is possible. To simplify definitions, we assume  $\text{skip} \in \mathcal{B}$  with  $(\text{skip}, \text{skip}) \in \surd$  and  $\forall (B, \alpha, B') \in \rightsquigarrow: B \neq \text{skip}$ .

Note that  $\mathcal{B}$  is later used to model interleaving points. More precisely,  $B \in \mathcal{B}$  encodes, roughly speaking, a set of sequences of actions; between every action of such

a sequence  $\vartheta$  an interleaving point exists, i.e., another action (of a transition firing in parallel) can be executed before  $\vartheta$  is continued. Note that do actions can interleave at any point, not necessarily at the modeled interleaving points. In particular, transitions of the state machines will be labeled with elements of  $\mathcal{B}$  instead of Actions, compare with Section 5.

**Definition 3 (Events/guards).** Set  $\mathcal{E}$  denotes the set of all events and set  $\mathcal{G}$  denotes a set of boolean expressions, which depend on global information such as the attribute values of the objects. Furthermore, the atomic predicates  $wla$  and  $nab$  are contained in  $\mathcal{G}$ , where  $wla$  indicates that the target state of the transition having  $wla$  as guard was last active and  $nab$  indicates that the region of the target state of the transition having  $nab$  as guard was not active before. These two atomic predicates are later used to model UML 2.0 history pseudostates.

Next we define what are allowed transitions between states:

**Definition 4 (Transitions).** A transition  $t$  w.r.t. a set of states  $\mathcal{S}$  is a tuple  $(s_1, e, g, B, s_2)$  such that:

- $s_1 \in \mathcal{S} \setminus \mathcal{S}_{\text{fin}}$  (i.e.,  $s_1 \in \mathcal{S}_{\text{exit}} \cup \mathcal{S}_{\text{entry}} \cup \mathcal{S}_{\text{choice}} \cup \mathcal{S}_{\text{com}}$ ) is called its source state,
- $s_2 \in \mathcal{S}$  its target state where
  - exactly the internal transitions have composite states as sources or as target, in which case the source and target must also be equal, i.e.,  
 $(s_1 \in \mathcal{S}_{\text{com}} \vee s_2 \in \mathcal{S}_{\text{com}}) \Rightarrow s_1 = s_2$
  - transitions targeting exit pseudostates have exit pseudostates as source, i.e.,  
 $s_2 \in \mathcal{S}_{\text{exit}} \Rightarrow s_1 \in \mathcal{S}_{\text{exit}}$ ,
  - exit pseudostates may only be the source of transitions targeting exit pseudostates at exactly one level higher in the state hierarchy or targeting non-exit pseudostates at the same level of hierarchy, i.e.,  
 $(s_1 \in \mathcal{S}_{\text{exit}} \wedge s_2 \in \mathcal{S}_{\text{exit}}) \Rightarrow \text{stateOf}(\text{stateOf}(s_1)) = \text{stateOf}(s_2)$  and  
 $(s_1 \in \mathcal{S}_{\text{exit}} \wedge s_2 \notin \mathcal{S}_{\text{exit}}) \Rightarrow \text{regOf}(s_2) = \text{regOf}(s_1)$ ,
  - transitions outgoing from exit pseudostates of final states may only target exit pseudostates, i.e.,  $\text{stateOf}(s_1) \in \mathcal{S}_{\text{fin}} \Rightarrow s_2 \in \mathcal{S}_{\text{exit}}$ ,
  - transitions outgoing from entry pseudostates may only have targets at one level of hierarchy downwards, i.e.,  
 $s_1 \in \mathcal{S}_{\text{entry}} \Rightarrow \text{stateOf}(\text{stateOf}(s_2)) = \text{stateOf}(s_1)$ , and
  - choice pseudostates must have targets at the same level of hierarchy, i.e.,  
 $s_1 \in \mathcal{S}_{\text{choice}} \Rightarrow \text{regOf}(s_1) = \text{regOf}(s_2)$ .
- $e \in \mathcal{E} \cup \{\emptyset\}$  is called its necessary event (which has to be provided to enable the transition), where events may not occur at transitions leaving entry or choice pseudostates, i.e.,  $e \neq \emptyset \Rightarrow s_1 \in \mathcal{S}_{\text{exit}} \cup \mathcal{S}_{\text{int}}$ ,
- $g \in \mathcal{G}$  is called its guard, constraining the necessary condition for the enabling of the transition, and
- $B \in \mathcal{B}$  is called its action encoding.

The projections of transitions to the corresponding components are denoted by  $\pi_{\text{sor}}$ ,  $\pi_{\text{ev}}$ ,  $\pi_{\text{gua}}$ ,  $\pi_{\text{act}}$ , and  $\pi_{\text{tar}}$ , respectively. Furthermore, the direct subregions of a composite state  $s \in \mathcal{S}_{\text{com}}$  are given by

$$\text{dsr}(s) = \{r \in \mathcal{R} \mid \text{parent}(r) = s\}$$

Now we are ready to present the syntax of core state machines, where  $\mathbb{P}(U)$  denotes the power set of set  $U$ :

**Definition 5 (Core state machines).** A core state machine  $M$  is a tuple

$$((\mathcal{S}, \mathcal{R}, \text{parent}), \text{doAct}, \text{defer}, \mathbb{T}, s_{\text{start}}, \text{Var}, \sigma_{\text{init}}), \text{ where}$$

- $\mathcal{S}$  is a set of states that is consistent w.r.t. the set of regions  $\mathcal{R}$  and with function  $\text{parent}$ ,
- $\text{doAct} : \mathcal{S}_{\text{com}} \rightarrow \text{Act}$  assigns to each state the do action that can be executed when the state is active,
- $\text{defer} : \mathcal{E} \rightarrow \mathbb{P}(\mathcal{S}_{\text{com}})$  assigns to each event those states in which it will be deferred,
- $\mathbb{T}$  is a set of transitions w.r.t.  $\mathcal{S}$ , where  $\mathbb{T}_{\text{int}}$  denotes the set of the singleton sets of internal transitions, i.e.,  $\mathbb{T}_{\text{int}} = \{\{t\} \mid t \in \mathbb{T} \wedge \pi_{\text{sor}}(t) \in \mathcal{S}_{\text{com}}\}$ ,
- $s_{\text{start}} \in \mathcal{S}_{\text{com}}$  its initial state belonging to the uppermost region and having no subregions, i.e.,  $\text{regOf}(s_{\text{start}}) = \epsilon \wedge \text{dSr}(s) = \emptyset$ ,
- $\text{Var}$  a set of variables ranging over some fixed domain, and
- $\sigma_{\text{init}} \in \text{VarAss}$  its initial variable assignment, where  $\text{VarAss}$  denotes the set of all possible variable assignments over  $\text{Var}$ .

Note that if an entry pseudostate  $s$  is ‘active’ and there is a direct subregion ‘of’  $s$  for which no enabled transition outgoing  $s$  points to (which is, in particular, the case if the state contains a region that does not contain a state), a deadlock occurs (this situation corresponds to the absence of initial pseudostates in UML state machines). A similar case arises if there is no enabled transition leaving an ‘active’ choice pseudostate. Note that this situation is ill-formed in UML 2.0 [8, p. 523].

In the following,  $((\mathcal{S}, \mathcal{R}, \text{parent}), \text{doAct}, \text{defer}, \mathbb{T}, s_{\text{start}}, \text{Var}, \sigma_{\text{init}})$  denotes a fixed core state machine  $M$ . Next we define the compound transitions  $\text{CoTr}$  of a core state machine. A compound transition is either a set consisting of one internal transition ( $\in \mathbb{T}_{\text{int}}$ ) or a collection of (non-internal) transitions such that a single outermost state is exited. Note that here, contrary to UML state machines, only transitions outgoing exit pseudostates are collected in a compound transition. This is formalized by using function  $\mathcal{Y} : \mathcal{S}_{\text{exit}} \rightarrow \mathbb{P}(\mathbb{P}(\mathbb{T}))$ , where  $\mathcal{Y}(s)$  collects all sets of transitions ‘below’  $s$  that can belong to a compound transition that ‘includes’  $s$ . Formally:

$$\mathcal{Y}(s) = \left\{ \bigcup_{r \in \text{dSr}(s)} (\{f(r)\} \cup F(r)) \mid f : \text{dSr}(s) \rightarrow \mathbb{T} \wedge F : \text{dSr}(s) \rightarrow \mathbb{P}(\mathbb{T}) \wedge \forall r \in \text{dSr}(s) : \pi_{\text{tar}}(f(r)) = s \wedge \text{regOf}(\pi_{\text{sor}}(f(r))) = r \wedge F(r) \in \mathcal{Y}(\pi_{\text{sor}}(f(r))) \right\}$$

$$\text{CoTr} = \{\{t\} \cup T \mid t \in \mathbb{T} \wedge \pi_{\text{sor}}(t) \in \mathcal{S}_{\text{exit}} \wedge \pi_{\text{tar}}(t) \notin \mathcal{S}_{\text{exit}} \wedge T \in \mathcal{Y}(\pi_{\text{sor}}(t))\} \cup \mathbb{T}_{\text{int}}$$

Note that exit states not targeted by transitions from all direct subregions may not occur in a compound transition. In order to simplify definitions, we also allow that different events may occur in a compound transition. This is not problematic, since compound transitions where more than one event occurs cannot be enabled.

## 4 Formal semantics of core state machines

A history maps a region  $r$  to its direct substate that was active the last time the region was left. If the region was not active before (or a final state was last active),  $r$  is mapped to the default value  $\perp$ .<sup>2</sup> Therefore, the set of histories is

$$\mathcal{H} = \{H : \mathcal{R} \rightarrow \mathcal{S}_{\text{com}} \cup \mathcal{S}_{\text{fin}} \cup \{\perp\} \mid \forall r \in \mathcal{R} : H(r) \neq \perp \Rightarrow r = \text{regOf}(H(r))\},$$

Configurations describe a snapshot of a state machine execution. They are introduced below, where we, in order to simplify the definition, allow configurations that cannot occur during execution.

**Definition 6.** A csm-configuration  $\mathcal{C}$  w.r.t.  $M$  is a tuple  $(\sigma, \mathcal{A}, \text{do}, H, \alpha, \tilde{s}, \beta, T, \tilde{T})$  where

- $\sigma \in \text{VarAss}$ , denoting the current variable assignment,
- $\mathcal{A} \subseteq \mathcal{S}_{\text{com}} \cup \mathcal{S}_{\text{fin}}$ , denoting which states are currently active,
- $\text{do} : \mathcal{S}_{\text{com}} \rightarrow \text{Act}$  denoting the corresponding do actions which remain to be executed,
- $H \in \mathcal{H}$ , denoting its current history information,
- $\alpha \in \text{Actions}$ , denoting the action that has to be executed next w.r.t. transition execution,
- $\tilde{s} \in \{\emptyset\} \cup \mathcal{S}_{\text{fin}} \cup \mathcal{S}_{\text{exit}} \cup \mathcal{S}_{\text{entry}} \cup \mathcal{S}_{\text{choice}}$ , denoting the (pseudo)state that has to be activated after  $\alpha$  is completed,
- $\beta \subseteq \mathcal{B} \times (\{\emptyset\} \cup \mathcal{S}_{\text{fin}} \cup \mathcal{S}_{\text{exit}} \cup \mathcal{S}_{\text{entry}} \cup \mathcal{S}_{\text{choice}})$ , denoting the currently executing transitions (i.e., remaining actions and target states),
- $T \in \text{CoTr} \cup \{\emptyset\}$ , denoting the currently executing compound transition, and
- $\tilde{T} \subseteq \text{CoTr}$ , denoting the compound transitions that are left to be executed in order to complete the run-to-completion step.

The initial csm-configuration is

$$(\sigma_{\text{init}}, \{s_{\text{start}}\}, \text{doAct}, \{(\epsilon, s_{\text{start}})\} \cup \{(r, \perp) \mid r \in \mathcal{R} \setminus \{\epsilon\}\}, \text{skip}, \emptyset, \emptyset, \emptyset, \emptyset).$$

In the following we give some preliminaries for the definition of the relation between csm-configurations.

We assume that a function  $\text{eval} : (\text{VarAss} \times \mathcal{H} \times \mathcal{S}) \rightarrow \mathbb{P}(\mathcal{G})$ , that evaluates the guards, is given such that  $\text{wla}$  holds in  $\text{eval}(\sigma, H, s)$  if ‘ $s$ ’ was the last active state of the corresponding region, i.e.,  $\text{wla} \in \text{eval}(\sigma, H, s) \Leftrightarrow H(\text{regOf}(s)) = \text{stateOf}(s)$ , and  $\text{nab}$  holds in  $\text{eval}(\sigma, H, s)$  if the region of ‘ $s$ ’ was not visited before or a final state was last active there, i.e.,  $\text{nab} \in \text{eval}(\sigma, H, s) \Leftrightarrow H(\text{regOf}(s)) \in \{\perp\} \cup \mathcal{S}_{\text{fin}}$ . A compound transition is enabled for trigger  $e \in \mathcal{E} \cup \{\emptyset\}$ , variable assignment  $\sigma$ , active states  $\mathcal{A}$ , history  $H$ , and current do actions of the active states: if the sources of its transitions are active, if the triggers of all of its transitions are in  $e$ , if the guards of its transitions

<sup>2</sup> Note that the history information is ambiguous in UML 2.0 [8], e.g., it is not clear if the last active subconfiguration instead of the subconfiguration generated by the corresponding last active states is chosen, see [3] for more detail.

evaluate to true, and if the do actions are terminated for its transitions having elements from  $\mathcal{S}_{\text{exit}}^{\text{CP}}$  as its sources. Formally:

$$\text{Enable}_{e,\sigma,\mathcal{A},H,\text{do}} = \left\{ T \in \text{CoTr} \mid \forall t \in T : \text{stateOf}(\pi_{\text{sor}}(t)) \in \mathcal{A} \wedge \pi_{\text{ev}}(t) \subseteq \{e\} \wedge \right. \\ \left. \pi_{\text{gua}}(t) \in \text{eval}(\sigma, H, \pi_{\text{tar}}(t)) \wedge (\pi_{\text{sor}}(t) \in \mathcal{S}_{\text{exit}}^{\text{CP}} \Rightarrow \text{do}(\text{stateOf}(\pi_{\text{sor}}(t)))) = \text{skip} \right\}$$

Two compound transitions are in conflict if their states that have to be left (for an internal transition: the set consisting of its source state) are not disjoint. Formally:

$$\text{Conflict} = \{(T_1, T_2) \in \text{CoTr} \times \text{CoTr} \mid \text{stateOf}(\pi_{\text{sor}}(T_1)) \cap \text{stateOf}(\pi_{\text{sor}}(T_2)) \neq \emptyset\}$$

Note that two internal transition of the same state are in conflict with each other. Note that in UML no two internal transitions of the same state can be enabled at the same time [8, p. 539]. A compound transition  $T_1$  has priority over another one  $T_2$  if every priority relevant source state of  $T_1$  is a substate of a priority relevant source state of  $T_2$ , i.e.,  $\text{PrBelow}(T_1, T_2) \iff \forall t_1 \in T_1 : \pi_{\text{sor}}(t_1) \in \mathcal{S}_{\text{exit}}^{\text{Pr}} \cup \mathcal{S}_{\text{com}} \Rightarrow \exists t_2 \in T_2 : \pi_{\text{sor}}(t_2) \in \mathcal{S}_{\text{exit}}^{\text{Pr}} \cup \mathcal{S}_{\text{com}} \wedge \text{stateOf}(\pi_{\text{sor}}(t_2)) \succeq \text{stateOf}(\pi_{\text{sor}}(t_1))$ , and one of the highest priority relevant source states of  $T_1$  is a proper substate. The later fact is here equivalent to the existence of a priority relevant source state of  $T_2$ , for which no upper or at the same level priority relevant source state of  $T_1$  exists, i.e.,  $\text{PrStrBelow}(T_1, T_2) \iff \exists t_2 \in T_2 : \pi_{\text{sor}}(t_2) \in \mathcal{S}_{\text{exit}}^{\text{Pr}} \cup \mathcal{S}_{\text{com}} \wedge \forall t_1 \in T_1 : \pi_{\text{sor}}(t_1) \in \mathcal{S}_{\text{exit}}^{\text{Pr}} \cup \mathcal{S}_{\text{com}} \Rightarrow \neg(\text{stateOf}(\pi_{\text{sor}}(t_1)) \succeq \text{stateOf}(\pi_{\text{sor}}(t_2)))$ :

$$\text{Priority} = \{(T_1, T_2) \in \text{CoTr} \times \text{CoTr} \mid \text{PrBelow}(T_1, T_2) \wedge \text{PrStrBelow}(T_1, T_2)\}$$

Note that the outermost prioritized exit pseudostates decide about priority. A set of compound transitions is fire-able if it is a non-empty maximal set of enabled and conflict-free compound transitions such that no enabled compound transition with higher priority exists. The non-emptiness is used to model deferral of an event, which is only possible if no corresponding fire-able compound transition exists. Formally:

$$\text{Fireable}_{e,\sigma,\mathcal{A},H,\text{do}} = \{\ddot{T} \subseteq \text{Enable}_{e,\sigma,\mathcal{A},H,\text{do}} \mid \ddot{T} \neq \emptyset \wedge \forall T' \in \text{Enable}_{e,\sigma,\mathcal{A},H,\text{do}} \setminus \ddot{T} : \\ (\forall T \in \ddot{T} : (T', T) \notin \text{Priority}) \wedge (\exists T \in \ddot{T} : (T, T') \in \text{Conflict}) \wedge \\ \forall T_1, T_2 \in \ddot{T} : (T_1, T_2) \in \text{Conflict} \Rightarrow T_1 = T_2\}$$

In the following, we assume that a function  $\text{calc} : (\text{Actions} \times \text{VarAss}) \rightarrow (\mathcal{L} \times \text{Actions} \times \text{VarAss})$ , calculating the effect of action execution (observable communication [encoded via elements of a given set  $\mathcal{L}$ ], remaining actions, and obtained variable assignment), is given. Furthermore,  $f[x \mapsto v]$ , that is straightforwardly extended to sequences  $(x_i \mapsto v_i)^{i \in I}$ , denotes the function that is everywhere equal to  $f$  except on  $x$  (if it is in its range) where it is equal to  $v$ .

The rules defining the relation between csm-configurations is given in Table 1, where a transition step without a label corresponds to an internal step. The rules are explained below: RULE do-act describes an atomic action execution of a do action of an active composite state, where condition  $\alpha = \text{skip} \Rightarrow \ddot{s} = \emptyset$  ensures that the entry of a state finishing the execution of a corresponding transition cannot be interleaved by

do-act	$\frac{s \in \mathcal{A} \cap \mathcal{S}_{\text{com}} \quad \text{do}(s) \neq \text{skip} \quad \text{calc}(\text{do}(s), \sigma) = (\ell, \alpha', \sigma') \quad \alpha = \text{skip} \Rightarrow \ddot{s} = \emptyset}{(\sigma, \mathcal{A}, \text{do}, H, \alpha, \ddot{s}, \beta, T, \ddot{T}) \xrightarrow{\ell} (\sigma', \mathcal{A}, \text{do}[s \mapsto \alpha'], H, \alpha, \ddot{s}, \beta, T, \ddot{T})}$
cur-act	$\frac{\alpha \neq \text{skip} \quad \text{calc}(\alpha, \sigma) = (\ell, \alpha', \sigma')}{(\sigma, \mathcal{A}, \text{do}, H, \alpha, \ddot{s}, \beta, T, \ddot{T}) \xrightarrow{\ell} (\sigma', \mathcal{A}, \text{do}, H, \alpha', \ddot{s}, \beta, T, \ddot{T})}$
next-tr-1	$\frac{(B, \ddot{s}) \in \beta \quad B \overset{\alpha}{\rightsquigarrow} B' \quad \beta' = \{(B', \ddot{s})\} \cup \beta \setminus \{(B, \ddot{s})\}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \alpha, \emptyset, \beta', T, \ddot{T})}$
next-tr-2	$\frac{(B, \ddot{s}) \in \beta \quad (B, \alpha) \in \checkmark \quad \beta' = \beta \setminus \{(B, \ddot{s})\}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \alpha, \ddot{s}, \beta', T, \ddot{T})}$
next-com	$\frac{T' \in \ddot{T} \setminus \mathbb{T}_{\text{int}} \quad \beta = \{(\text{skip}, \pi_{\text{sor}}(t)) \mid t \in T' \wedge \forall t' \in T' : \neg(\text{stateOf}(\pi_{\text{sor}}(t)) \succ \text{stateOf}(\pi_{\text{sor}}(t')))\}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \ddot{T}) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \beta, T', \ddot{T} \setminus \{T'\})}$
next-int	$\frac{\{t\} \in \ddot{T} \cap \mathbb{T}_{\text{int}} \quad \beta = \{(\pi_{\text{act}}(t), \emptyset)\}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \ddot{T}) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \beta, \{t\}, \ddot{T} \setminus \{t\})}$
next-completion	$\frac{\ddot{T} \in \text{Fireable}_{\emptyset, \sigma, \mathcal{A}, H, \text{do}} \quad T' \in \ddot{T}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \emptyset) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \{T'\})}$
next-trigger	$\frac{e \in \mathcal{E} \quad \text{Fireable}_{\emptyset, \sigma, \mathcal{A}, H, \text{do}} = \emptyset \quad \ddot{T} \in \text{Fireable}_{e, \sigma, \mathcal{A}, H, \text{do}}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \emptyset) \xrightarrow{e} (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \ddot{T})}$
defer	$\frac{\text{defer}(e) \cap \mathcal{A} \neq \emptyset \quad \text{Fireable}_{e, \sigma, \mathcal{A}, H, \text{do}} = \emptyset}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \emptyset) \xrightarrow{\text{defer}(e)} (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \emptyset, T, \emptyset)}$
a-fin	$\frac{\ddot{s} \in \mathcal{S}_{\text{fin}} \quad H' = (H[(r \mapsto \perp)^{r \in \mathcal{R} \cap \uparrow \{\text{regOf}(\ddot{s})\}]})}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \ddot{s}, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A} \cup \{\ddot{s}\}, \text{do}, H', \text{skip}, \emptyset, \beta, T, \ddot{T})}$
a-ch	$\frac{\ddot{s} \in \mathcal{S}_{\text{choice}} \quad t \in \mathbb{T} \quad \pi_{\text{sor}}(t) = \ddot{s} \quad \pi_{\text{gua}}(t) \in \text{eval}(\sigma, H, \pi_{\text{tar}}(t))}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \ddot{s}, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \beta \cup \{(\pi_{\text{act}}(t), \pi_{\text{tar}}(t))\}, T, \ddot{T})}$
a-en	$\frac{\begin{array}{l} \ddot{s} \in \mathcal{S}_{\text{entry}} \quad \text{do}' = \text{do}[\text{stateOf}(\ddot{s}) \mapsto \text{doAct}(\text{stateOf}(\ddot{s}))] \\ f : \text{dsr}(\text{stateOf}(\ddot{s})) \rightarrow \mathbb{T} \quad \beta' = \beta \cup \bigcup_{r \in \text{dsr}(\text{stateOf}(\ddot{s}))} \{(\pi_{\text{act}}(f(r)), \pi_{\text{tar}}(f(r)))\} \\ \forall r \in \text{dsr}(\text{stateOf}(\ddot{s})) : \pi_{\text{sor}}(f(r)) = \ddot{s} \wedge \pi_{\text{gua}}(f(r)) \in \text{eval}(\sigma, H, \pi_{\text{tar}}(t)) \end{array}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \ddot{s}, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A} \cup \{\text{stateOf}(\ddot{s})\}, \text{do}', H, \text{skip}, \emptyset, \beta', T, \ddot{T})}$
a-ex-1	$\frac{\begin{array}{l} \ddot{s} \in \mathcal{S}_{\text{exit}} \quad \forall (B, \ddot{s}') \in \beta : \ddot{s}' \neq \ddot{s} \quad \forall s \in \mathcal{A} : \neg(\text{stateOf}(\ddot{s}) \succ s) \quad t \in T \\ \pi_{\text{sor}}(t) = \ddot{s} \quad \beta' = \beta \cup \{(\pi_{\text{act}}(t), \pi_{\text{tar}}(t))\} \quad H' = H[\text{regOf}(\ddot{s}) \mapsto \text{stateOf}(\ddot{s})] \end{array}}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \ddot{s}, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A} \setminus \{\text{stateOf}(\ddot{s})\}, \text{do}, H', \text{skip}, \emptyset, \beta', T, \ddot{T})}$
a-ex-2	$\frac{\ddot{s} \in \mathcal{S}_{\text{exit}} \quad \exists B : (B, \ddot{s}) \in \beta \vee \exists s \in \mathcal{A} : \text{stateOf}(\ddot{s}) \succ s}{(\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \ddot{s}, \beta, T, \ddot{T}) \mapsto (\sigma, \mathcal{A}, \text{do}, H, \text{skip}, \emptyset, \beta, T, \ddot{T})}$

**Table 1.** Configuration steps.

a do action. Note that a do action can also execute at any point during the execution of actions of transitions. RULE cur-act describes the next atomic action execution of the atomic action sequence currently being executed. RULES next-tr-1 and next-tr-2 select from the transitions currently being fired ( $(B, \dot{s}) \in \beta$ ) a new atomic action sequence  $\alpha$  (determined from  $B \xrightarrow{\alpha} B'$ , resp.  $(B, \alpha) \in \checkmark$ ) that will be executed next. This is only possible if no action has to be executed and no pseudostate has to be activated ( $\dot{s} = \emptyset$ ). In next-tr-2, contrary to next-tr-1, the target of the transition has to be activated after the execution of  $\alpha$ , since this completes the firing of the transition.

RULE next-com determines the next non-internal compound transition  $T'$  from  $\ddot{T}$  which will be fired, that is only possible if the previous fired compound transition is completed: no state has to be activated, no action and no transition remain to be executed. Furthermore, the deepest source exit pseudostates of  $T'$  are remembered to be activated. This is done by using special transitions formed by deepest source exit pseudostates of  $T'$  as targets and skip as actions. These transitions determine  $\beta$ . RULE next-int determines the next internal transition  $\{t\}$  of  $\ddot{T}$  which will be fired, that is only possible if the previously fired compound transition is finished. RULE next-completion determines the next trigger-free compound transition (which corresponds in UML to transitions triggered by completion) that will be fired. This is only possible if the previous set of compound transitions is completely executed (which corresponds in UML to the termination of the previous run to completion step). RULE next-trigger is similar to Rule next-completion except that transitions triggered by an event are considered. Note that completion transitions have priority over triggered transitions, which is ensured by  $\text{Fireable}_{\emptyset, \sigma, \mathcal{A}, H, \text{do}} = \emptyset$  stating that no completion transition is enabled. RULE defer describes the deferral of events, which is possible if a state that defers the event is active and no completion transition and no compound transition having this trigger is enabled.

RULE a-fin activates a final state  $\dot{s}$  directly contained in region  $\ddot{r} = \text{regOf}(\dot{s})$  and resets the history information of all subregions of  $\ddot{r}$  to  $\perp$ , since they are now considered as not visited before [3]. RULE a-ch activates a choice pseudostate, where it is immediately determined which of its currently enabled outgoing transitions is fired. RULE a-en activates an entry pseudostate  $s$ , thereby the composite state  $s'$  to which  $s$  belongs becomes active and the current do action of  $s'$  is reset to the do action specified in the state machine, i.e., it starts from the beginning also in the case of history entry. Furthermore, as for choice pseudostates, it is immediately determined for every direct subregion which one of the currently enabled outgoing transitions of state  $s$  is fired. RULES acti-ex-1 and acti-ex-2 deal with the activation of exit pseudostates, that will not happen if there is a transition with a source below the exit pseudostate, w.r.t. state hierarchy, which has not yet been completely executed (Rule acti-ex-2). In the case of activation (Rule acti-ex-1) the unique transition from  $T$  having the exit pseudostate as source is added to  $\beta$ . Furthermore, the state to which the exit pseudostate belongs is deactivated and the history information of the exited region is adapted.

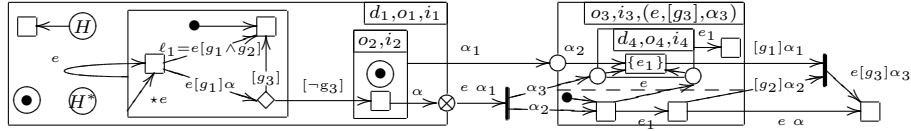
## 5 Embedding of UML 2.0 State Machines

Many ambiguities in UML 2.0 state machines are resolved here along our suggestions given in [3]. Another ambiguity, which is not mentioned in [3] concerns the influence

of event deferral on enabling transitions: “The conflict resolution [between deferring or consuming] follows the triggering priorities, where nested states override enclosing states. In case of a conflict between states in different orthogonal regions, a consumer state overrides a deferring state.”[8, p. 536]. Consider, e.g., the two transition having event  $e_1$  in Figure 1. Will the above transition fire, since the enabling the lower transition avoids the deferral of the event? We decide that a compound transition with trigger  $e$  is disabled if it has a source state that has a deeper state which defers  $e$ . Thus in our example the upper transition will not fire in contrast to the other one.

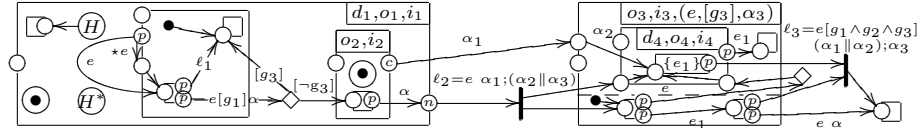
The transformation of UML 2.0 state machines is divided into 5 succeeding steps. In the following, prototype transitions are transitions with no event, with guard true, and with action skip, except that the label is explicitly specified differently. Furthermore, the transition system encoding the transitions’ actions in core state machines is built upon a simple process algebra consisting of actions, sequential composition, and parallel composition without synchronization mechanism. Weak-compound transitions combine transitions connected via join, fork, and junction pseudostates. Note that we decide to use a not optimal transformation w.r.t. space, in order to increase understandability. In order to increase comprehension, the UML 2.0 state machine of Figure 1 is transformed along the different steps. Besides the resolving of junction pseudostates, the complexity of the transformation is in  $\mathcal{O}(n \cdot t)$ , where  $n$  is the number of states and  $t$  the number of transitions.

**Step 1:** Here some simplifications are made: Replace every guard ‘else (resp., otherwise)’ at a transition  $t$ , by the negation of the disjunction of the guards of the other transitions outgoing from the source of  $t$ . Replace a transition with a set of events as trigger by copies of the transition, one for each event in the set, which yields the event label of that copy. Resolve junction pseudostates, by introducing for every pair of incoming and outgoing transitions a new transition labeled with the union of the events (must be empty or singleton), with the conjunction of the guards, and the actions combined via the sequential composition of the process algebra. Next, we ensure that no choice, entry, exit pseudostate can occur in two different compound transitions: Copy choice, entry, exit pseudostates, and adapt (including copying) the transitions such that every choice and entry pseudostate has exactly one incoming transition, every exit pseudostate has exactly one outgoing transition, and every direct subregion of the state corresponding to an exit (entry) pseudostate has at most one transition pointing to (resp., leaving) this exit (resp., entry) pseudostate. After applying Step 1 to the UML 2.0 state machine from Figure 1 we obtain:



**Step 2:** In this step sources and targets become exit, resp., entry pseudostates and further modifications making clear which states are left, resp., entered are made: For every local transition  $t$  having its target below its source  $s$  add a new entry pseudostate at  $s$ , which becomes the new target of  $t$  (note that  $t$  remains local), and add a prototype transition from the new entry pseudostate to the original target of  $t$ . Every weak-compound transition of the current state machine is labeled by its event, by the conjunction of

its transitions' guards, and by a process algebra term encoding its transitions' actions. Every exit pseudostate that does not have an incoming transition becomes a priority relevant exit pseudostate, the other exit pseudostates become non-priority relevant exit pseudostates. To every composite state add a so called default entry pseudostate. For every transition  $t$  outgoing a composite state  $s$ , add a new completion (priority) exit pseudostate to  $s$  if  $t$  belongs to a completion (resp., if  $t$  belongs to a non-completion) compound transition and this exit pseudostate becomes the new source state of the transition. Every transition targeting a composite state is redirected such that its target becomes the composite state's default entry pseudostate. Next, we make explicit which states of a compound transition are left (resp., entered): Divide by a new choice pseudostate every weak-compound transition whose sources are different from choice pseudostates and whose sources and targets are in two orthogonal regions (i.e., none contains the other). This new choice pseudostate belongs to the deepest region that contains all sources (and join pseudostates). The label of the old weak-compound transition is written at the weak-compound transition pointing to the new choice pseudostate, whereas the weak-compound transition leaving the new choice pseudostate with no event, with guard true, and with action skip as label. Thereafter, relocate every choice pseudostate to the outermost region that can be reached via a chain of transitions starting from the choice pseudostate, where transitions crossing regions reach also the lowest region containing both those regions.<sup>3</sup> After applying Step 2 on our running example we obtain, where circled  $p(n, c)$  indicates priority (resp., non-priority, completion) exit pseudostates:

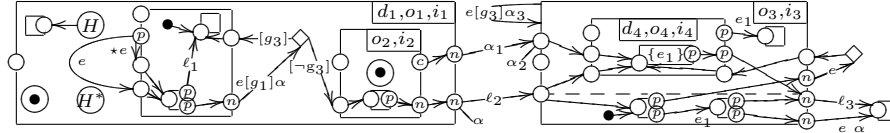


**Step 3:** Intuitively, in this step a transition is divided by an exit (resp., entry) pseudostate, where it crosses the border of a state. In order to handle also join and fork pseudostates the transformation is precisely given: For every weak-compound transition  $T$  having interlevel transitions, add a non-priority exit pseudostate at every composite state that will be left by  $T$  and that strictly contains a source of  $T$ , and add an entry pseudostate at every composite state that will be entered by  $T$  and that strictly contains a target of  $T$ . Then connect those new pseudostates for  $T$  as well as the sources and targets of  $T$  with non-interlevel prototype transitions along the transition path of  $T$ . The event, guard, and action of  $T$  are written (temporarily till the end of Step 5) at  $T$ 's target if it is an exit pseudostate<sup>4</sup>, at  $T$ 's source if it is an entry pseudostate, and at the outermost transition added for  $T$  otherwise. Different transitions labels at an exit (resp., entry) pseudostate, which all correspond to a single compound transition (ensured by Step 1), are transformed into a single label, by taking the union of the events (must be empty or singleton), the conjunction of the guards, and the actions combined via the

<sup>3</sup> In our interpretation, transitions between orthogonal regions lead to leaving (and entry) the lowest containing composite state. Note that the other interpretation that only the regions of the lowest containing composite state and not the state itself are left and entered can also be handled via transformation to core state machines.

<sup>4</sup> Note that if a compound transition  $T$  has an exit (entry) pseudostate as target (resp., source) then  $T$  has exactly one target (resp., source).

parallel composition of the process algebra. Remove all interlevel transitions and all join/fork pseudostates. Finally, internal transitions are replaced via transitions having the corresponding label and having the corresponding state as source and target. After applying Step 3 on our running example we obtain:

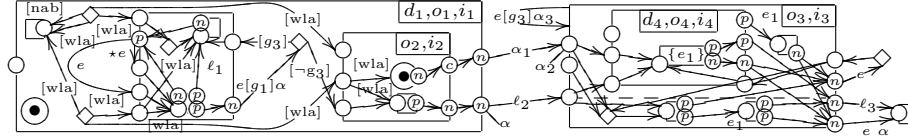


**Step 4:** Default exit and entry are obtained as follows: To every composite or final state add a non-priority exit pseudostate, called its default exit pseudostate. To every composite state add for every event  $e$  a non-priority exit pseudostate, called its default exit pseudostate w.r.t.  $e$ . Note that default exit pseudostates w.r.t. events are used for modeling the fact that deferral of events can disable transitions. Furthermore, every priority exit pseudostate  $s$  is considered as an exit pseudostate w.r.t. event  $e$  if  $s$  belongs to a compound transition that has trigger  $e$ . For every exit pseudostate  $s$  that is different from a completion exit pseudostate and from an exit pseudostate w.r.t. an event and for every direct subregion  $r$  of  $\text{stateOf}(s)$  such that no transition pointing to  $s$  exists in  $r$ , add prototype transitions from the default exit pseudostate of every composite/final state of  $r$  to  $s$ . For completion exit pseudostates add transitions in the same way except that only the default exit pseudostates of final states are used as sources. For an exit pseudostate w.r.t.  $e$  add transitions in the same way except that default exit pseudostates w.r.t.  $e$  are used at composite states (final states use their default exit pseudostate) and only composite states that do not defer event  $e$  are allowed.<sup>5</sup> For every entry pseudostate  $s$  and direct subregion  $r$  of  $\text{stateOf}(s)$  such that no transition outgoing  $s$  exists in  $r$ , add a default transition from  $s$  to the initial pseudostate of  $r$  (if present).<sup>6</sup> History entry is obtained as follows: The guards of transitions leaving history pseudostates are conjunctively extended by predicate nab. Add an entry pseudostate to every composite state for which an upper region has a deep history pseudostate. These newly introduced entry pseudostates are called the deeper-history entry pseudostates of the corresponding states. Add non-interlevel prototype transitions having guard wla (i) from shallow history pseudostates to default entry pseudostates, (ii) from deep history pseudostates to deeper-history entry pseudostate, and (iii) from deeper-history entry pseudostates to deeper-history entry pseudostates or final states. Finally, transform history and initial pseudostates into choice pseudostates, except of the initial pseudostate belonging to the outermost region which becomes an exit pseudostate of the initial state  $s_{\text{start}}$ , which also is added. After applying Step 4 on our running example, we obtain the following

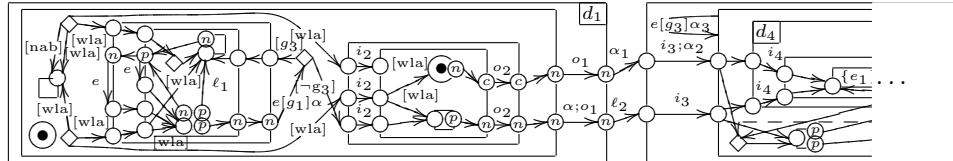
<sup>5</sup> The transformation follows our decision that a transition  $t_1$  has priority over  $t_2$  when every source of  $t_1$  is below or equal a source of  $t_2$  and (one is strictly below or there is a subregion for which  $t_2$  has a source but not  $t_1$ ).

<sup>6</sup> Note that UML 2.0 provides a variation point for default entry if no initial pseudostate exists [8, p. 532]. The first alternative is an ill-defined behavior, as we interpret it. The second one is not to activate any state of such regions (using partial configurations). The second interpretation can be handled by transformation as follows: an initial pseudostate pointing to a new state having no entry, exit, do actions, and no regions, is added to every region that has no initial pseudostate.

structure: Here, only the relevant exit pseudostates are depicted (e.g., the default exit points on the outermost states and those pointing to them are omitted) and the default entry pseudostates on states not containing regions are used there for the deeper-history entry pseudostates.



**Step 5:** In the final step exit and entry actions are transformed: For every composite state  $s$  (different from  $s_{start}$ ), we introduce a new composite state  $\tilde{s}$  that (i) belongs to the region to which  $s$  belonged, (ii) contains exactly one region that exactly contains  $s$ , (iii) has the do actions of  $s$  (the do action of  $s$  becomes skip), (iv) has a non-priority exit pseudostate for every priority or non-priority exit pseudostate of  $s$ , and (v) has a completion exit pseudostate for every completion exit pseudostate of  $s$ . Transitions between the exit (entry) pseudostates of  $s$  and their corresponding exit (resp., entry) pseudostates of  $\tilde{s}$  are added. These transitions are labeled with the labels of the corresponding exit (entry) pseudostates of  $s$ , if present, and with no event, with guard true, and with action skip, otherwise. The exit (resp., entry) actions of  $s$  are sequentially attached after (resp., before) the action of every outgoing (resp., incoming) transition of exit (resp., entry) pseudostates of  $s$ , which corresponds to the new introduced transitions. This sequentially attaching is done via the sequential process algebra operator. External transitions targeting entry (outgoing from exit) pseudostates of  $s$  are redirected such that they are targeting (resp., outgoing from) the corresponding entry (resp., exit) pseudostates of  $\tilde{s}$ . Finally, all entry/exit actions at a state and all labels written at entry (exit) pseudostates are removed. After applying Step 5 on our running example we obtain the following structure: Here, only the relevant composite states are copied and the non relevant entry points are removed.



## 6 Conclusion

To structure the cumbersome process of resolving the ambiguities and removing the inconsistencies in the semantics of UML state machines (as published by the OMG) we propose a new strategy: Introduce core state machines with their precise semantics. These core state machine contain all essential features of UML state machines in a compact way. Then transform UML state machines to these core state machine. Thereby we have presented a complete, formal semantics of single UML 2.0 state machines.

Because of the low space complexity of the transformation, core state machines yield an appropriate basis model for verification of UML 2.0 state machines and probably also for other statecharts variations: tools can be based on core state machines and,

e.g., UML 2.0 state machines are handled via transformation. Thus more independent tool support can be developed. In our opinion, core state machines are optimal in the sense that the syntax of core state machines cannot be further reduced by maintaining the hierarchical structure and by maintaining the expressive power of UML 2.0 state machines. Furthermore, core state machines have even more expressive power than UML state machines, e.g., a more specific history entry can be defined.

Note that do actions can (partly) be executed at any point in time provided that the corresponding state is active. This big amount of interleaving points leads to a large state explosion. Here, the core state machines' explicitly modeled action-interleaving points (modeled by encoding actions via a state of a labeled transition systems) are also appropriate for handling the interleaving with the do actions: Modify the semantics such that do actions may only execute at the interleaving points of the transitions. Furthermore, action-interleaving can also be used to model more atomic executions inside do actions by slightly modifying the syntax and semantics. Another possible variation in the semantics of core state machines is to allow in case of history entry that do actions continue their execution at those positions when the corresponding state was exited. Note that the behavior of UML 2.0 state machines is not totally clear in this situation.

Future work will be the modeling of communication between state machines, the development of formal semantics taking further UML diagrams into account, as well as the development of tool support for core state machines.

## References

- [1] M. L. Crane and J. Dingel. On the semantics of uml state machines: Categorization and comparison. Technical Report 501, Queen's University, 2005.
- [2] H. Fecher, M. Kyas, and J. Schönborn. Semantic issues in UML 2.0 state machines. Technical Report 0507, Christian-Albrechts-Universität zu Kiel, 2005.
- [3] H. Fecher, J. Schönborn, M. Kyas, and W. P. de Roever. 29 new unclarities in the semantics of uml 2.0 state machines. In *ICFEM*, volume 3785 of *LNCS*, pages 52–65. Springer, 2005.
- [4] Y. Gurevich. Evolving algebras 1993: Lipari guide. In *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
- [5] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.
- [6] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of statecharts (extended abstract). In *LICS*, pages 54–64. IEEE Computer Society Press, 1987.
- [7] Y. Jin, R. Esser, and J. W. Janneck. A method for describing the syntax and semantics of uml statecharts. *Software and System Modeling*, 3(2):150–163, 2004.
- [8] Object Management Group. *UML Superstructure Specification, v2.0 formal/05-07-04*, 2005.
- [9] G. Reggio and R. Wieringa. Thirty one problems in the semantics of uml 1.3 dynamics. In *OOPSLA'99 workshop, Rigorous Modelling and Analysis of the UML: Challenges and Limitations*, 1999.
- [10] A. J. H. Simons and I. Graham. 30 things that go wrong in object modelling with uml 1.3. In *Behavioral Specifications of Businesses and Systems*, pages 237–257. Kluwer Academic, 1999.
- [11] X. Zhan and H. Miao. An approach to formalizing the semantics of uml statecharts. In *ER*, volume 3288 of *LNCS*, pages 753–765. Springer, 2004.