29 New Unclarities in the Semantics of UML 2.0 State Machines*

Harald Fecher, Jens Schönborn, Marcel Kyas, and Willem-Paul de Roever

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

Abstract. UML 2.0, which is the standard modeling language for object-oriented systems, has only an informally given semantics. This is in particular the case for UML 2.0 state machines, which are widely used for modeling the reactive behavior of objects. In this paper, a list of 29 newly detected trouble spots consisting of ambiguities, inconsistencies, and unnecessarily strong restrictions of UML 2.0 state machines is given and illustrated using 6 state machines having a problematic meaning; suggestions for improvement are presented. In particular, we show that the concepts of history, priority, and entry/exit points have to be reconsidered.

1 Introduction

UML has become the standard modeling language for object-oriented systems. UML state machines 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 [4] and their object-oriented version [5]. The fact that the semantics of UML is only informally described leads to many ambiguities and inconsistencies in earlier versions of UML, see for example [9, 11]. Many of the detected ambiguities and inconsistencies are ruled out in UML 2.0 [8], but new ones are added.

We present a list of 29 newly detected ambiguities, inconsistencies and unnecessarily strong restrictions of UML 2.0 (behavioral) state machines [8, p. 573–639], which we found during an attempt to define their formal semantics [10]. These unclarities are illustrated on 6 state machines, which are legal according to [8] but having a problematic meaning. Some of the listed unclarities are serious, i.e., they cannot straightforwardly be eliminated. This holds, e.g., for the concepts of history, priority,

* 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)

and entry/exit points, which are discussed in Subsection 3.1 till Subsection 3.3, where also suggestions for improvement are given. Our suggestions for improving UML state machines lead to a simplified and less ambiguous semantics, in particular, all serious unclarities are eliminated.

2 UML 2.0 State Machines

The basic concepts of UML 2.0 state machines are states and transitions between them. A state may contain regions[1] (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. States that contain at least one region are called composite states, otherwise, they are called simple states. For example state 1 in Fig. 1 is a simple state, state 3 is a composite state containing one region, and state 0 is a composite state containing two regions.

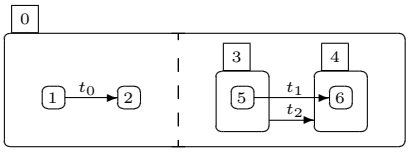

Fig. 1. State machine

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. For example, $\{0, 1, 3, 5\}$ is a configuration in Fig. 1. A state may have associated an entry behavior (evoked when the state becomes active), an exit behavior (evoked when the state becomes deactivated), and a doActivity behavior (sequence of actions, which may be (partially) executed when the state is active). In the following, to execute doActivities means the partial execution of such action sequences.

The environment may send events to the state machine. These events are collected in the event pool of the state machine. A state machine may either execute doActivities of active states or may dispatch a single event from its event pool to trigger transitions.

Beside its source and target, a transition consists of an event, a guard (a boolean expression), and an action sequence. A 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, those are fired which belong to a maximal set whose elements are pairwise conflict-free. Two transitions are in conflict if the intersection of the set

[1] Multiple regions of a state are separated by a dashed line.

of states that will be left by the firing of these transitions is non empty. A state s will be left by the firing of transitions t if it is active and either a substate of the source state of t or, roughly spoken, the transition points outside the border of s. For example, state 3 and 5 have to be left in Fig. 1 by the firing of transition t_1 or by the firing of t_2. Hence, t_1 and t_2 are in conflict.

The firing of transition t leads, in this order, (i) to the deactivation of the states that will be left by the firing of t together with the execution of the exit behavior of these states, (ii) the execution of the actions of t (which we simply call the execution of t), and (iii) the activation of its target states (the transition's target state together with the non-active states 'crossed' by the transition and some substates of the target state via a default mechanism) together with the execution of the entry behavior of these states. 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. 617].

There also exist different kinds of additional pseudostates, which are not allowed to occur in configurations, and which have special interpretations. For example, a choice pseudostate, which is depicted by a diamond-shaped symbol (see Fig. 5), leads to a new decision concerning which of its outgoing transitions will be fired without completing the run-to-completion step. In other words, the guards of transition leaving a choice pseudostate are evaluated when the choice pseudostate is left, contrary to guards of other transitions, which are evaluated when an event is dispatched. Further pseudostates are described in the next section. Transitions may also have pseudostates as their sources or targets. A compound transition is, roughly spoken, a transition obtained by subsuming transitions involving pseudostates. In particular, one (single) transition between states is also a compound transition. The semantics of firing transitions is defined on compound transitions.

3 Semantical Unclarities of UML 2.0 State Machines

The unclarities discussed here are categorized as:

Incompleteness: meaning that no clear statement is made.

Inconsistency: meaning that parts are in contradiction with other parts.

Ambiguity: meaning that interpretations in more than one way are possible.

Equivocality: meaning that it is unclear whether the implicitly given interpretation is the intended one.

3.1 History pseudostates

An initial pseudostate of composite state s indicates how the substates of s are entered if a transition t with target state s was fired: after firing t, the unique transition leaving the initial pseudostate of s, called initial transition, will be fired. A history pseudostate of a region r is used to activate those substates of r that were active when r was the last time active. The shallow history concept considers only the direct substates of r, whereas that of deep history considers 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, which is the unique transition leaving the history pseudostate, is fired instead. Final states are special simple states. Initial pseudostates are depicted by a solid filled circle, a deep history pseudostate by a circle containing an 'H*', and a final state by a circle surrounding a solid filled circle, see, e.g., Fig. 2.

Inconsistency 1 *On the one hand, history pseudostates belong to regions [8, p. 598]. On the other hand, the semantics of a history pseudostate is defined for its containing state [8, p. 591] and, therefore, no meaning of history pseudostates belonging to multiple regions of a composite state is defined.*

We proceed by applying the definition of its semantics to its containing region.

Incompleteness 2 *The firing of transitions is only defined for compound transitions, which do not include history or initial pseudostates [8, p. 625]. Hence, the semantics of firing transitions that point to history pseudostates (or initial pseudostates) is not defined.*

The solution to allow also history and initial pseudostates as targets of a compound transition does not eliminate all unclarities:

Incompleteness 3 *May a transition to a history pseudostate be fired if the guard of the history default transition evaluates to false and either the corresponding region was not visited before or a final state of the corresponding region was last active?*

UML 2.0 only mentions implicitly that initial transitions and history default transitions have to point to default states [8, p. 591], thus:

Equivocality 4 *Is it really the case that transitions from initial pseudostates or from history pseudostates may not point to pseudostates (such as a choice pseudostate)?*

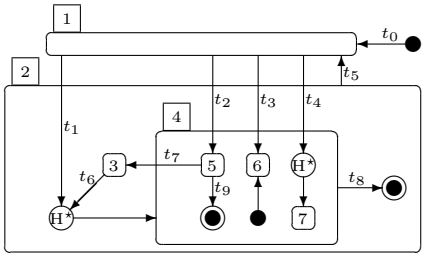

Fig. 2. History illustration

Improvement: *Interpret history and initial pseudostates as choice points with additional semantics. Then more than one transition may leave a history or a initial pseudostate, and may point to pseudostates; the model is ill-formed (i.e., any behavior is possible) when a history (or initial) pseudostate is reached and all guards of the outgoing transitions evaluate to false.*

Ambiguity 5 *The semantical behavior of transitions that point to a deep history pseudostate from inside the region containing the history pseudostate is not clear.*

Consider, e.g., the firing sequence $(t_0, t_3, t_5, t_2, t_7, t_6)$ in Fig. 2. Then which one of the states 3,5,6,7 is active? The comment given at [8, p. 591], which concerns the last active configuration before exiting, favors state 6, whereas the recursive application of the shallow history rule [8, p. 606] favors state 5, and when 'last active' does not correspond to the exiting of the state then state 3 would be active.

Furthermore, the recursive application approach mentioned leads to the following unclarities:

Equivocality 6 *Is it really the case that also the deeper nested final states will not be activated in case of a deep history activation?*

Consider, e.g., the firing sequence $(t_0, t_2, t_9, t_5, t_1)$ in Fig. 2. Then state 7 is active and not the final state that is contained in state 4.

Ambiguity 7 *How is the default activation for nested substates determined in case of deep history activation. Are they determined by the initial transitions or are they determined by the default history states of the corresponding regions?*

Consider, e.g., the firing sequence (t_0, t_1) in Fig. 2. Then, is state 6 or state 7 active?

Ambiguity 8 *How is the history information reset when a final state is reached? Are only the direct substates reset or are all substates reset?*

Consider, e.g., the firing sequence $(t_0, t_2, t_8, t_5, t_4)$ in Fig. 2. Then state 7 is active if the firing of t_6 also resets state 4. Otherwise, state 5 is active.

Improvement: *We suggest to store the history information at the point in time when the region is left. The 'last active' direct substate, instead of the 'last active' subconfiguration, is stored (deep history will use this information recursively), which reduces the complexity. The history information of region r is set to a 'not-visited' value, whenever (i) r was not visited before, or (ii) r or an outer region of r was exited after a final state was reached there. The firing of a transition pointing to a deep history pseudostate h of region r (i) fires the default history transition of h (where a default entering determined by the initial transitions takes place in its target) if the history information of r yields the 'non-visited' value, or (ii) recursively activates the states stored in the history information (also the stored final states) otherwise.*

Our suggestion has the advantage that (a) no configuration that partly consist of history information and partly consist of default information is generated; (b) a default history entering just corresponds to the firing of the corresponding history default transition; and (c) entering a region where a final state was last active has the same behavior as if the region was not visited before.

Concerning our examples this suggestions yields that (i) state 5 is active after the firing of $(t_0, t_3, t_5, t_2, t_7, t_6)$ in Fig. 2; (ii) the final state contained in state 4 is active after $(t_0, t_2, t_9, t_5, t_1)$; (iii) state 6 is active after (t_0, t_1); and (iv) state 7 is active after $(t_0, t_2, t_8, t_5, t_4)$.

3.2 Priority

Priority between transitions is used to rule out some nondeterminism in determining the set of transitions that may fire. "By definition, a transition originating from a substate has higher priority than a conflicting transition originating from any of its containing states" [8, p. 618]. Join pseudostates define a set of states, rather than a single state, as source of a compound transition. Fork pseudostates define a set of states, rather to a single state as target of a compound transition. Join and also fork pseudostates are depicted by a short heavy bar as, e.g., illustrated in Fig. 3.

Inconsistency 9 *The definition of priority of joined transition ("The priority of joined transitions is based on the priority of the transition with the most transitively nested source state" [8, p. 618]) is not well defined and in contradiction to the algorithm describing the determination of the sets of transitions that will be fired [8, p. 618].*

The priority definition for join transitions is not well defined, since the 'most transitively nested source state' (the state that has the the greatest distance to the outermost region) cannot be uniquely determined and, therefore, the priority between transitions cannot be uniquely determined. For example, it is not clear if transition t_0 in Fig. 3 has priority over t_1 or not. The contradiction between the priority definition for join transitions and the algorithm is illustrated on the following example: In Fig. 3, transition t_2 has priority over t_3 with respect to the priority definition for joined transition, but t_3 has priority over t_2 with respect to the algorithm.

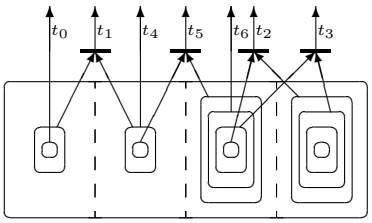

Fig. 3. Priority illustration

The algorithm mentioned contains the sentence: "For each state at a given level, all originating transitions are evaluated to determine if they are enabled" [8, p. 618].

Ambiguity 10 *The interpretation of level is not clear. Does it correspond to the maximal distance to a simple state or does it correspond to the distance to the outermost region?*

For example, if level corresponds to the maximal distance to a simple state, then in Fig. 3 transition t_4 has priority over t_5 and over t_6, and t_6 has priority over t_5. On the other hand, if level corresponds to the distance to the outermost region, then t_6 has priority over t_4 and over t_5, and no priority between t_4 and t_5 exists.

Improvement: *Use the definition given by the algorithm except that level is ignored*[2]*: t has priority over t' if every source state of t is a*

[2] Ignoring level yields less priorities and, therefore, the approaches where level is interpreted can be considered as a refinement step in the sense that less executions are allowed.

substate of a source state of t' and one is a proper one. Then in Fig. 3 transition t_6 has priority over t_4 and no further priorities exist between t_4, t_5, and t_6. The advantage of this definition is that the priority relation is completely determined by the source states (e.g., further substates are irrelevant, which is not the case if level is interpreted as distance to simple states).

3.3 Entry/exit points

Another unclarity concerns entry/exit points. Entry/exit points are pseudostates that belong to state machines or to composite states. "An entry pseudostate [and symmetrically, an exit pseudostate] is used to join an external transition terminating on that entry point to an internal transition emanating from that entry point" [8, p. 601]. Entry (exit) points are depicted by a small circle (respectively, by a small circle with a cross) on the border of the state machine or composite state.

Inconsistency 11 *Is it really the case that entry points (respectively, exit points) only exist at the topmost region of a state machine [8, p. 591] (i.e., cannot belong to composite state), since entry points (respectively, exit points) belonging to composite states are explicitly discussed at [8, pp. 592,594,603].*

In the following, we assume that entry/exit points are also allowed at composite states. Junction pseudostates describe sets of transitions obtained by combining any incoming transition with an outgoing transition.

Inconsistency 12 *On the one hand, the entry (exit) behavior of a state is executed between the transition pointing to an entry (respectively, exit) point and the transition leaving that entry (exit) point [8, pp. 601,606] (e.g., the entry behavior of state 0 in Fig. 4 is executed in between transitions t_0 and t_1). On the other hand, entry (exit) pseudostates are considered as junction pseudostates [8, pp. 607-608] and, therefore, the entry behavior of state 0 is executed after the execution of transitions t_0 and t_1 [8, pp. 625-628].*

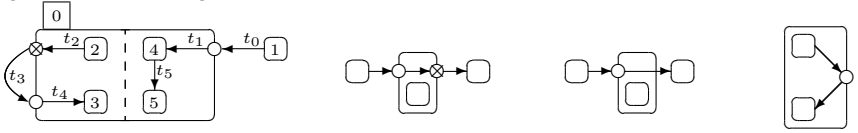

Fig. 4. Entry/exit point illustration

The approach to drop the correspondence to junction pseudostates does not eliminate all unclarities as illustrated in the following:

Incompleteness 13 *How is the behavior defined if an exit point is reached that does not have an outgoing transition? Is this an ill-formed situation?*

Incompleteness 14 *It is not explicitly mentioned that the invocation of the exit (entry) behavior of a state enforced through an exit (respectively, entry) point corresponds to the point in time when the state is exited (respectively, entered). Furthermore, it is not even clear if the state is always exited in this situation.*

For example, consider Fig. 4. Suppose the compound transition consisting of t_2, t_3, t_4 is fired. Is then only the exit (respectively, entry) behavior of state 0 executed without exiting state 0? This question is essential for execution of doActivities and conflict determination.

In the following, we assume that a state is immediately left after executing its exit behavior and that a state is immediately entered before executing its entry behavior.

Equivocality 15 *The deepest state (or region) containing the source and target state of a transition (called the least common ancestor of the transition) is not sufficient to determine the conflict relation.*

For example, in Fig. 4 the compound transition consisting of t_2, t_3, t_4 is in conflict with transition t_5, since the firing of each transition will exit state 4 (if a composite state, like 0, is exited all its substates have to be exited). But the least common ancestor of these transition yields different subregions of state 0.

Incompleteness 16 *May transitions (or transition paths on pseudostates) point from entry points to exit points as, e.g., depicted in the second picture of Fig. 4?*

More problematic, transitions from an entry point belonging to state s may point outside s (probably by using pseudostates in between), which contradicts the invariant that after a run-to-completion step a configuration will be reached [8, p. 617]. Furthermore, transitions pointing from inside state s to an entry point belonging to s would execute the entry behavior of an already active state. Therefore:

Incompleteness 17 *The following restrictions are needed: Every transition path on pseudostates starting at an entry (exit) point of a composite state s may only leave (respectively, enter) that state through an exit (respectively, entry) point of s.*

Any transition path on pseudostates starting from outside (inside) a composite state s or from an exit (respectively, entry) point of s and which (i.e., that path) does not contain an entry (respectively, exit) point of s may not end at an exit (entry) point of s.

For example, the third and the fourth state machine of Fig. 4 should be not allowed.

Inconsistency 18 *Consider the state machine of Fig. 5. Suppose t_0 will be fired. Then state 0 has to be left (independent whether transition t_1 or t_3 will be taken). But if t_1 will be fired, state 0 is left after the execution of t_1 (and, therefore, after t_0) by the semantics of exit points [8, p. 606]. On the other hand, if t_3 will be fired, then state 0 has to be left before transition t_0 is executed, since states have to be left before the compound transition is executed [8, pp. 627].*
In particular, no allowed execution order exists if the execution of the exit behavior of state 0 changes the evaluation of the guard of t_1 to true and of t_2 to false.

Fig. 5. Entry/exit point illustration (2)

Inconsistency 19 *Consider the state machine of Fig. 5. On the one hand, by the semantics of exit pseudostates state 6 has to be left after the execution of t_5. On the other hand, state 4 has to be exited before transition t_5 is executed, since states have to be left before the compound transition is executed [8, pp. 627]. Furthermore, state 4 may only be exited if all its active substates are exited, hence state 6 has to be exited before t_5 is executed.*

Equivocality 20 *Transitions from fork pseudostates may not point to entry points or to history pseudostates, since transitions from fork pseudostates may not point to pseudostates [8, p. 624].*

Equivocality 21 *Transitions from exit points may not point to join pseudostates, since transitions pointing to join pseudostates may not have pseudostates as their sources [8, p. 624].*

Improvement: *Many unclarities can be avoided by forbidding transitions crossing state borders. Instead, use always entry/exit points. This*

has the consequence that only the substates of the source (target) states of a compound transition are exited (respectively, entered) before (respectively, after) the transition is executed; all other states are exited (respectively, entered) in between the execution of the transition.
Furthermore, exit (entry) points should be considered as join (respectively, fork) pseudostates, where the source of a transition pointing to an exit point of state s has to be a direct substate of s or an exit point of a direct substate of s. The same holds, for transitions leaving entry points of s, except that they may also point to direct subpseudostates of s. Note that in this approach no explicit join or fork pseudostates are needed.

3.4 Transitions

It is unclear whether the default state (the target of an initial transition) has to be a direct substate. More problematic is that an initial transition may point outside its region, which contradicts, similarly to Incompleteness 17, the invariant that after a run-to-completion step a configuration will be reached [8, p. 617]. Therefore:

Incompleteness 22 *The following restriction is needed: Transitions from initial pseudostates or from history pseudostates may only point to substates of the region that directly contains the corresponding pseudostate.*

For example, the state machine on the left hand side of Fig. 6 should not be allowed, since if state 0 is entered by default then states 0 and 1 become active. The reason is that these states are direct substates of the same region and that only one direct substate of a region may be active [8, p. 605].

Fig. 6. Disallowed and allowed state machines

Another restriction, which is necessary to guarantee the above mentioned configuration invariant, concerns local transitions, i.e., transitions with transition kind 'local'. A local transition differs from a normal (i.e., external) transition in the sense that if it is fired, its source state is not exited (only the substates of the source state) [8, p. 634].

Incompleteness 23 *The following restriction is needed: A local transition may only point to its source state or to substates (properly reached through a fork pseudostate) of its source state.*

For example, the second state machine of Fig. 6 is not allowed, since if the local transition[3] fires, then state 3 remains active and state 5 becomes active, which is forbidden, as explained before. On the other hand, the third state machine of Fig. 6 does not yield any semantical problems.

Inconsistency 24 *Transitions crossing regions, as illustrated in the fourth picture of Fig. 6, are forbidden [8, p. 627], but their meaning is explicitly described [8, p. 627].*

If guards with side effects are used [8, p. 624,627], an ill-formed situation occurs and therefore no behavior can be guaranteed.

Incompleteness 25 *How should it be ensured that the evaluation of guards does not need time, which is a side effect?*

Improvement: *One possibility is to make the observable time steps so coarse that the execution time of the guards cannot be observed.*
Another possibility is to allow only guards that depend on single boolean attributes, which are, e.g., calculated by the entry behavior. In order to obtain a dynamic dependency, these boolean attributes can be updated by doActivities.

3.5 Nondeterminism

The determination of the set of firing transitions is not completely deterministic [8, pp. 618], since not always a priority between conflicting transitions exist. Suppose transitions t_0 and t_1 are enabled in Fig. 7, then either t_0 or t_1 fires.

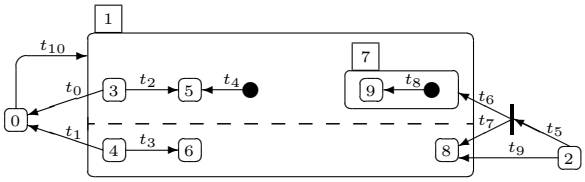

Fig. 7. Illustration of a state machine for nondeterminism

UML 2.0 seems to enforce at least determinism between the selection of transitions that have the same source state: "Each event name may appear more than once per state if the guard conditions are different" [8, p. 609].

[3] Local transitions are illustrated by the attached symbol $*$.

Ambiguity 26 *What does different guard conditions mean? Does it mean that for each pair of guards, there exists a situation where one of the guards is true and the other is false? Does different guards mean mutually exclusive guard conditions (i.e., no two guards may be true at the same time), as enforced for completion transitions*[4] *[8, p. 626]?*

The order in which transitions of a compound transition fire is not completely deterministic. More precisely, transitions to a join pseudostate (respectively, leaving a fork pseudostate) can be fired in any order. In UML 2.0, there is a contradiction in the definition of the execution order of the initial transition of a composite state that is a target of a firing transition (such as t_4 after firing t_{10}):

Inconsistency 27 *"The entry behavior of the composite state is executed before the behavior associated with the initial transition" [8, p. 605] is in contradiction to the fact that "A transition to the enclosing state represents a transition to the initial pseudostate in each region" [8, p. 600] and that actions corresponding to a compound transition are executed before entry behaviors [8, p. 627-628].*

For example, after firing transition t_{10} in Fig. 7, must the entry behavior of state 1 be executed before transition t_4 or must transition t_4 be executed before the entry behavior of state 1?

Ambiguity 28 *Suppose transitions to the enclosing state represent transitions to the initial pseudostate: In which order are actions of transitions from fork pseudostates and actions of the enabled initial transitions executed? Is there a depth-first or branching-first strategy? Is the order completely nondeterministic (except for the fact that actions of transitions to outer states have to be executed first)?*

For example, if t_5 fires in Fig. 7, then it is not clear which of the execution sequences (t_6, t_7, t_8), (t_6, t_8, t_7), (t_7, t_6, t_8) are allowed.

Equivocality 29 *Is it really the case that the actions of the initial transition do not have to be executed before the entry behavior of the target state of the initial transition, in case of default entry?*

For example, after the firing of t_9 in Fig. 7, it is not clear whether t_4 may also be executed after the entry behavior of state 5.

[4] Completion transitions are transitions that do not have an explicit trigger. They are triggered if their source states are completed. Roughly spoken, a state is completed if its doActivities are terminated and their direct subregions, if existing, have reached a final state.

4 Conclusion and Related Work

We have presented 29 inconsistencies, ambiguities, forgotten restrictions, and unnecessary strong restrictions in UML 2.0 state machines. Some of the unclarities are serious, i.e., their elimination is not straightforward. This holds for history pseudostates, priority, exit/entry points, and assuring side-effect-free guards. The serious problems are eliminated by our improvements.

Many of the detected unclarities also exist in earlier versions of UML. In particular, most unclarities concerning history pseudostates[5] (Unclarities 2-5 and 7), all unclarities concerning priority, and the assurance of side-effect-free guards also exists in UML 1.5 [7]. Entry/exit points do not exist in UML 1.5. The semantics of history and priority with respect to join pseudostates in UML 1.x are defined in the literature as follows:

In the work of van der Beeck [12], concerning history, 'last active' does not correspond to the exiting of the state. Furthermore, a transition t has priority over t' if the least common ancestor of t is below the least common ancestor of t', i.e., this yields a weaker priority concept concerning join pseudostates.

The history information in the work of Börger et al. [1] corresponds to the 'last active configuration'. Furthermore, when a state is entered via history, the history information is forgotten, i.e., in this case the semantics of transitions pointing to the history pseudostates from inside the region is unclear. Join pseudostates are encoded by completion transitions and, therefore, a priority principle similar to the one in [12] is used.

In [3], where no history pseudostates are considered, priority is handled as a variation point. Nevertheless, the authors make the suggestion that a transition t has lesser or equal priority than t' if every source state of t is below a source state of t'. This differs from our suggestion, e.g., in Fig. 3 transition t_4 has priority over t_1 in their suggestion, whereas no priority between t_4 and t_1 exists in our suggestion.

In [2], priority is also handled as variation point and in [6] join transitions are compiled away, but an exact definition of the transformation is missing and, therefore, the used priority schema is unclear. Both works do not consider history pseudostates. Most of the other works on the semantics of UML 1.x state machines, see, e.g., the references given in [10], do not consider join or history pseudostates and, therefore, do not cover the related problems. We are not aware of works different from ours [10] that define formal semantics of UML 2.0 state machines.

[5] Note that final states do not reset the history information in UML 1.5.

Future work is to define a precise formal semantics with respect to all the suggested improvements, e.g., our semantics [10] does not handle entry/exit points and choice pseudostates. The redefinition concept in UML 2.0 state machines has to be examined, e.g., to clarify to which extent redefinition corresponds to a refinement concept.

References

1. E. Börger, A. Cavarra, and E. Riccobene. Modeling the Dynamics of UML State Machines. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 223–241. Springer-Verlag, 2000.
2. R. Eshuis, D. N. Jansen, and R. Wieringa. Requirements-level semantics and model checking of object-oriented statecharts. *Requirements Engineering Journal*, 7:243–263, 2002.
3. S. Gnesi, D. Latella, and M. Massink. Modular semantics for a uml statechart diagrams kernel and its extension to multicharts and branching time model-checking. *The Journal of Logic and Algebraic Programming*, 51(1):43–75, 2002.
4. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, July 1987.
5. D. Harel and E. Gery. Executable object modeling with statecharts. *Computer*, 30(7):31–42, July 1997.
6. J. Lilius and I. P. Paltor. Formalising UML state machines for model checking. In R. France and B. Rumpe, editors, *UML*, volume 1723 of *LNCS*, pages 430–445. Springer-Verlag, 1999.
7. Object Management Group. *OMG Unified Modeling Language Specification, Version 1.5*, 2003. `http://www.omg.org/cgi-bin/doc?formal/03-03-01`.
8. Object Management Group. *UML 2.0 Superstructure Specification*, Oct. 2004. (updated version). `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`.
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. J. Schönborn. Formal semantics of UML 2.0 behavioral state machines. Master's thesis, Christian-Albrechts Universität zu Kiel, 2005. `http://www.informatik.uni-kiel.de/~jes/jsFsemUMLsm.pdf`.
11. A. J. H. Simons and I. Graham. 30 things that go wrong in object modelling with uml 1.3. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, pages 237–257. Kluwer Academic, 1999.
12. M. von der Beeck. A structured operational semantics for UML-statecharts. *Software and System Modeling*, 1(2):130–141, 2002.