Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 250 (2009) 71–86
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Refinement Sensitive Formal Semantics of State Machines With Persistent Choice[1]

Harald Fecher[2] and Michael Huth[3]

Imperial College London, United Kingdom

Heiko Schmidt[4] and Jens Schönborn[5]

Christian-Albrechts-Universität zu Kiel, Germany

Abstract

Modeling languages usually support two kinds of nondeterminism, an external one for interactions of a system with its environment, and one that stems from under-specification as familiar in models of behavioral requirements. Both forms of nondeterminism are resolvable by composing a system with an environment model and by refining under-specified behavior (respectively). Modeling languages usually don't support nondeterminism that is persistent in that neither the composition with an environment nor refinements of under-specification will resolve it. Persistent nondeterminism is used, e.g., for modeling faulty systems. We present a formal semantics for UML state machines enriched with an operator "persistent choice" that models persistent nondeterminism. This semantics is based on abstract models – μ-automata with a novel refinement relation – and a sound three-valued satisfaction relation for properties expressed in the μ-calculus.

Keywords: modeling language, nondeterminism, μ-calculus, 3-valued satisfaction, formal semantics

1 Introduction

Nondeterminism is a central concept in modeling and analysis of systems. For example, the abstraction of a program's control and data introduces nondeterminism that analyzers have to control. In program analysis and software verification, such control is achieved by a judicious over- or under-approximation of the deterministic program, which yields sound but potentially imprecise results.

[1] This work is in part financially supported by the DFG projects *FE 942/1-1*, *FE 942/2-1*, and by the UK EPSRC project *Complete and Efficient Checks for Branching-Time Abstractions* EP/E028985/1.

[2] Email: hfecher@doc.ic.ac.uk

[3] Email: mrh@doc.ic.ac.uk

[4] Email: hsc@informatik.uni-kiel.de

[5] Email: jes@informatik.uni-kiel.de

doi:10.1016/j.entcs.2009.08.006

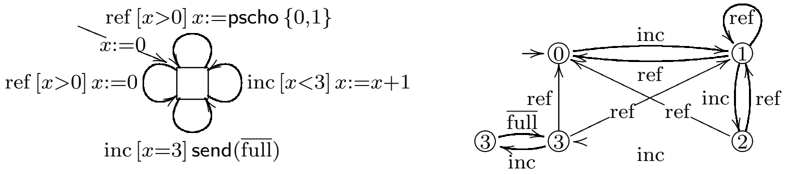

Fig. 1. Left: state machine with initial value $x = 0$. Transitions are labeled with triples of synchronization event, guard, and execution description; e.g. on event *inc* variable x is incremented by 1 if $x < 3$, and an output event $\overline{\text{full}}$ is sent if $x = 3$. Right: Transition system as semantics of state machine on the left. The presence of pscho makes this semantics incorrect for refinements based on bisimulation or ready simulation.

Open systems have well defined interfaces that specify how a system can interact with any environment. For example, the header of a method specifies input values and types for method invocation and such an invocation resolves the nondeterminism of that interface at runtime. Parallel composition of process algebra terms and modular model checking also express and analyze open systems and the external nondeterminism stemming from their interaction with environments.

Descriptions of requirements may exhibit an additional form of nondeterminism, that of under-specification. A local access control policy may be silent on accesses pertaining to a non-local resource. An architecture may specify that at most four servers may be used for data processing of an e-commerce application. A state machine may say that the implementation of an audit logging facility is optional. Etc. This nondeterminism is resolvable, ideally in a stepwise fashion, by a controlled choice of implementation alternatives, e.g. the decision not to implement audit logs or the decision to deny all accesses to non-local resources.

A further form of nondeterminism may be needed in the modeling of behavior. Bisimulation and its probabilistic variants treat nondeterminism as persistent. For a labeled transition system its bisimulation quotient is a behavioral model that has not lost any nondeterminism present in the original labeled transition system. Another example of persistent nondeterminism can be found in modeling and reasoning about faulty systems (as seen below), or in the modeling of random behavior without knowledge of any distributions [26]. A method header may have an input x of static type `String` but a complex precondition may enforce semantic constraints on legal inputs, e.g. "at most two coding errors occur in string x". One then wants to reason about the correctness of an implementation of that method for all inputs with at most two coding errors. The implementor or verifier is then not in control over the choice of input (zero, one or two coding errors) that meets these constraints, and seeks assurance against all such choices. In particular, this need to consider all scenarios persists even when the method body is fully implemented – including its communication details with external services.

Modeling languages usually handle well the resolvable forms of nondeterminism, whereas they may not adequately support persistent nondeterminism. In this paper we focus on UML state machines [25] – a variant of statecharts [12,13,14] – which model behavioral aspects of an object in an object-oriented application. The example below already exhibits resolvable and persistent nondeterminism.

Example 1.1 The state machine on the left in Fig. 1 models the part of a system that counts the number of currently occupied sectors of memory, stored in variable x. Setting x to 0 models reformatting, all sectors are freed up again. This is the initial state and up to three sectors may be occupied. External nondeterminism is present, e.g., in the choice of events *ref* (reformatting) and *inc* (increment sector count) when $0 < x \leq 3$. Persistent nondeterminism is expressed with the "persistent choice" operator $\mathsf{pscho}\,\{0, 1, 2, 3\}$. On event *ref* and positive value of x, that variable can be set to any value n between 0 and 3 where n is understood to be the leftmost *non-faulty* sector. We cannot control the choice of that value since $x := \mathsf{pscho}\,\{0, 1, 2, 3\}$ models all relevant fault scenarios. (However, to keep this running example small, we will use only a persistent choice between no fault and a fault in the first sector, expressible as $x := \mathsf{pscho}\,\{0, 1\}$.) This state machine has resolvable nondeterminism between this persistent choice and that of $x := 0$. The latter transition is risky as it ignores any fault in any modelled sector of memory.

In the above example, program and faulty memory are combined into a single model. This is advantageous in model checking, when the verification property won't have to specify any fault scenarios, since property specifications become more transparent and reusable and one can compensate for the increased complexity of the model through further model abstraction.

Persistent nondeterminism is not supported in UML state machines explicitly. But even without such explicit support, it has an implicit presence as soon as the underlying programming language has a "persistent choice" operator as seen in Example 1.1 – which could also be implemented by a random choice. To our knowledge, no formal semantics of UML state machines in the extant literature can express such a construct adequately.

Contribution and outline of paper.

Our aim is to support verification on an abstract level for systems that also have persistently nondeterministic behavior, where we mainly focus on state machine, respectively statechart, formalisms. Specifically:

- Section 2: We develop a new class of models, which employ hypertransitions, together with a refinement notion that can handle all external and persistent nondeterminism and under-specification.
- Section 3: We define a 3-valued satisfaction relation between our models and the modal μ-calculus [19,33], and prove its soundness under our refinement notion.
- Section 4: We discuss state machine semantics in terms of our new class of semantic models.
  - Subsection 4.1: We present a formal semantics for a simple state-machine variant with a persistent choice operator for arithmetic expressions.
  - Subsection 4.3: We discuss how existing formal semantics of more complex state machine variants can be accommodated to handle persistent choice operators adequately with respect to the common refinement notion in state machines.
- Section 5: We sketch how our technique can be generalized to state machine variants with, possibly underspecified, random choice operators.

2 ν-automata

Before we introduce our new class of semantic models, we present labeled transition systems. We do not consider predicates on states, as they appear, e.g., in Kripke structures, but our theory can be easily extended to allow for this.

We use the following notation: $\mathcal{L}$ denotes a set of transition labels. $|A|$ denotes the cardinality of a set A. The set of total functions from A to B is denoted by B^A. Therefore we may write 2^A for the power set of A. For a binary relation $R \subseteq M_1 \times M_2$ and for $X \subseteq M_1$, we define $X.R = \{m_2 \in M_2 \mid \exists m_1 \in X : (m_1, m_2) \in R\}$. For a ternary relation $\rightsquigarrow\ \subseteq M_1 \times \mathcal{L} \times M_2$ we write $m_1 \overset{e}{\rightsquigarrow} m_2$ for $(m_1, e, m_2) \in\ \rightsquigarrow$ and $\overset{e}{\rightsquigarrow}$ for the binary relation $\{(m_1, m_2) \mid m_1 \overset{e}{\rightsquigarrow} m_2\}$, thus $\{m_1\}.\overset{e}{\rightsquigarrow}\ = \{m_2 \in M_2 \mid m_1 \overset{e}{\rightsquigarrow} m_2\}$.

2.1 Transition systems

Definition 2.1 A *labeled transition system* (TS) with respect to the label set $\mathcal{L}$ is a tuple $(C, c^{\mathrm{i}}, \rightsquigarrow)$, where C is its set of states, c^{i} its initial state, and $\rightsquigarrow\ \subseteq C \times \mathcal{L} \times C$ its transition relation.

Nondeterminism in transition systems appears as multiple transitions going out from the same state and having the same label. A resolvable approach to this nondeterminism is obtained by using, e.g., the refinement notion of ready-simulation [2], whereas a persistent approach is, e.g., obtained by using bisimulation [24] as underlying equivalence notion. Both notions are made formal as follows:

Definition 2.2 Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are two TSs. Then $\mathcal{T}_1$ *is ready-simulated by* $\mathcal{T}_2$ if there exists a relation $R \subseteq C_1 \times C_2$ such that the initial states are related, i.e.,

- $(c_1^{\mathrm{i}}, c_2^{\mathrm{i}}) \in R$, and
- for $(c_1, c_2) \in R$ we have

(i) $\forall e \in \mathcal{L}, c_1' \in \{c_1\}.\overset{e}{\rightsquigarrow}_1 : \exists c_2' \in \{c_2\}.\overset{e}{\rightsquigarrow}_2 : (c_1', c_2') \in R$ and

(ii) $\{e \in \mathcal{L} \mid \exists c_1' : c_1 \overset{e}{\rightsquigarrow}_1 c_1'\} = \{e \in \mathcal{L} \mid \exists c_2' : c_2 \overset{e}{\rightsquigarrow}_2 c_2'\}$.

Constraint (i) states that every concrete transition has an abstract counterpart and (ii) states that every label present at the abstract level has to be present at the concrete level (and vice versa by constraint (i)). Similarly, in the definition of bisimulation every transition in $\mathcal{T}_1$ has a counterpart in $\mathcal{T}_2$ and vice versa.

Definition 2.3 Two TSs $\mathcal{T}_1$ and $\mathcal{T}_2$ are *bisimilar* if there exists a relation $R \subseteq C_1 \times C_2$ such that the initial states are related, i.e.,

- $(c_1^{\mathrm{i}}, c_2^{\mathrm{i}}) \in R$, and
- for $(c_1, c_2) \in R$ we have

(a) $\forall e \in \mathcal{L}, c_1' \in \{c_1\}.\overset{e}{\rightsquigarrow}_1 : \exists c_2' \in \{c_2\}.\overset{e}{\rightsquigarrow}_2 : (c_1', c_2') \in R$ and

(b) $\forall e \in \mathcal{L}, c_2' \in \{c_2\}.\overset{e}{\rightsquigarrow}_2 : \exists c_1' \in \{c_1\}.\overset{e}{\rightsquigarrow}_1 : (c_1', c_2') \in R$.

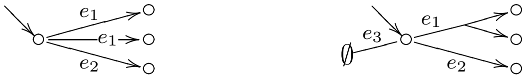

Fig. 2. Transformation of a transition system, interpreted with persistent nondeterminism and depicted on the left, to the ν-automata syntax, depicted on the right. The set of labels is $\mathcal{L} = \{e_1, e_2, e_3\}$.

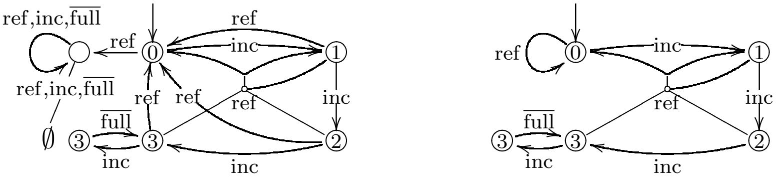

Fig. 3. Two ν-automata. Hypertransitions are illustrated by an arrow with a single starting point, a single label and several targets. A set of hypertransitions sharing the same targets and label are combined using a small circle. All states that do not have for $e \in \mathcal{L}$ a drawn outgoing transition have an implicit transition with label e to $\emptyset$. For later references, some states are named by numbers, drawn inside the circles. Here, the set of labels is $\mathcal{L} = \{\text{ref}, \text{inc}, \text{full}\}$.

2.2 ν-automata syntax

Note that in transition systems, it is only possible to opt for one of the interpretations of nondeterminism if such an interpretation is to be applied uniformly within a transition system. This suggests to use another class of models for expressing resolvable and persistent kinds of nondeterminism in the same model. The idea behind the approach proposed in this paper is to use again a kind of ready-simulation. Therefore, we treat the persistent nondeterminism present in transition systems not in the conventional manner of using multiple transitions per label and source state; rather we use exactly one transition per label and source state that points to a set of successor states. In particular, a state with no outgoing transitions with label e has, in this approach, one e-labeled transition pointing to the empty set.

Example 2.4 Consider Fig. 2 where a transition system, interpreted with persistent nondeterminism, and its transformed system are presented. The transformation will be given formally in Proposition 2.7.

If we now allow more than one outgoing hypertransition per state and label, we can model resolvable nondeterminism in the sense of ready-simulation. We first define the models of interest.

Definition 2.5 A *ν-automaton* with respect to $\mathcal{L}$ is a tuple $(C, C^{\mathrm{i}}, \longmapsto)$, with C its set of states, $C^{\mathrm{i}} \subseteq C$ its initial states, and $\longmapsto \ \subseteq C \times \mathcal{L} \times 2^C$ its hypertransition relation. A ν-automaton is *concrete* if it has exactly one initial state ($|C^{\mathrm{i}}| = 1$) and exactly one transition per label and per source state: $\forall c \in C, e \in \mathcal{L} : |\{c\}. \overset{e}{\longmapsto} | = 1$.

A state that does not have outgoing transitions for a label from $\mathcal{L}$ cannot be related to a concrete state, i.e., such a state is unsatisfiable (shown below). Examples of ν-automata are depicted in Fig. 3. We now define the desired refinement notion.

2.3 ν-refinement

The adaption of ready-simulation to our class of models leads to the following refinement notion, where every refining initial state has a related abstract initial state; and any refining hypertransition can be matched by an abstract hypertransition with the same label and target sets that are pairwise related. Formally:

Definition 2.6 We say the ν-automaton $(C_1, C_1^{\mathsf{i}}, \longmapsto_1)$ ν-refines the ν-automaton $(C_2, C_2^{\mathsf{i}}, \longmapsto_2)$ if there is a ν-refinement R, i.e., a relation $R \subseteq C_1 \times C_2$ such that

- $\forall c_1^{\mathsf{i}} \in C_1^{\mathsf{i}} : \exists c_2^{\mathsf{i}} \in C_2^{\mathsf{i}} : (c_1^{\mathsf{i}}, c_2^{\mathsf{i}}) \in R$, and
- $\forall (c_1, c_2) \in R, e \in \mathcal{L}, \Theta_1 \in \{c_1\}.\overset{e}{\longmapsto}_1 : \exists \Theta_2 \in \{c_2\}.\overset{e}{\longmapsto}_2$:
  $(\forall c_1' \in \Theta_1 : \exists c_2' \in \Theta_2 : c_1' R c_2') \wedge (\forall c_2' \in \Theta_2 : \exists c_1' \in \Theta_1 : c_1' R c_2')$.

For example, the ν-automaton on the right hand side in Fig. 3 is a ν-refinement of the ν-automaton on the left hand side in the same figure. Transition systems with resolvable nondeterminism, as well as transition systems with persistent nondeterminism, can be embedded into ν-automata:

Proposition 2.7 (i) *The embedding δ^{ps} from transition systems into ν-automata, defined by $\delta^{\mathsf{ps}}(C, c^{\mathsf{i}}, \rightsquigarrow) = (C, \{c^{\mathsf{i}}\}, \{(c, e, \Theta) \mid c \in C \wedge e \in \mathcal{L} \wedge \Theta = \{c' \mid c \overset{e}{\rightsquigarrow} c'\}\})$ ensures that two transition systems are related via bisimulation iff their transformations are related via ν-refinement. Moreover, the image of δ^{ps} captures exactly the concrete ν-automata.*

(ii) *The embedding $\delta^{r}(C, c^{\mathsf{i}}, \rightsquigarrow)$ of the transition system $(C, c^{\mathsf{i}}, \rightsquigarrow)$s into a ν-automata is defined as $\delta^{r}(C, c^{\mathsf{i}}, \rightsquigarrow) = (C, \{c^{\mathsf{i}}\}, \{(c, e, \{c'\}) \mid c \overset{e}{\rightsquigarrow} c'\} \cup \{(c, e, \emptyset) \mid \{c\}.\overset{e}{\rightsquigarrow} = \emptyset\})$, and ensures that two transition systems are ready bisimilar iff their transformation are related via ν-refinement.*

2.4 Discussion

μ-automata [16] are extensions of transition systems, which are also able to handle resolvable and persistent nondeterminism in a single formalism. The syntax of μ-automata can be defined to be that for ν-automata, but they differ in their refinement notion and so also in their concrete elements. In μ-automata, every transition at the abstract level has to be matched by the concrete system, but with possibly less target states. In other words, persistent nondeterminism is encoded via the outgoing transitions and resolvable nondeterminism is encoded by using a set of states as transition target. This is dual to ν-automata, where resolvable nondeterminism is encoded via outgoing transitions having the same label and persistent nondeterminism is encoded by using a set of states as transition target.

In our approach we can freely alternate persistent and resolvable non-determinism, since at each state of a ν-automata the modeler can decide whether to express persistent or (non-exclusively) resolvable non-determinism.

We consider every label of a transition system to be *reactive*, i.e., to denote an incoming communication. But *generative* labels correspond to asynchronous output communication of the system (e.g., label full in Fig. 3). If different generative labels are used, the set of concrete systems of ν-automata has a different constraint for generative labels: From any state there may be at most one outgoing transition with a generative label. Our refinement notion then has to be amended such that transitions having generative labels can be removed, as long as one remains present.

3 Satisfaction

We give a 3-valued satisfaction relation between ν-automata and the μ-calculus [19]. We chose this logic since persistent choice leads us into the world of branching time and the μ-calculus is a canonical branching-time logic in which many popular branching-time logics, such as CTL, can be translated. The set of all μ-calculus formulas $\mathcal{F}$ is generated by the following BNF-grammar:

$$\phi ::= \mathtt{tt} \mid \mathtt{ff} \mid Z \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle e \rangle\, \phi \mid [e]\, \phi \mid \mu Z.\phi \mid \nu Z.\phi$$

where $e \in \mathcal{L}$ and Z is from a μ-calculus variable set $\mathcal{V}ar$. The dual formula $\mathsf{dual}(\phi)$ of a μ-calculus formula ϕ is obtained by replacing $\mathtt{tt}$ by $\mathtt{ff}$, $\wedge$ by $\vee$, $\langle e \rangle$ by $[e]$, μ by ν, and vice versa.

Definition 3.1 Suppose $\mathcal{A} = (C, C^{\mathrm{i}}, \longmapsto)$ is a ν-automaton. Then the semantic function $[\![\_]\!]_{\_} : \mathcal{F} \times (\mathcal{V}ar \to 2^C) \to 2^C$ with respect to $\mathcal{A}$ is:

$$[\![\mathtt{tt}]\!]_\rho = C \qquad [\![\mathtt{ff}]\!]_\rho = \emptyset \qquad [\![Z]\!]_\rho = \rho(Z)$$

$$[\![\phi_1 \wedge \phi_2]\!]_\rho = [\![\phi_1]\!]_\rho \cap [\![\phi_2]\!]_\rho \qquad [\![\langle e \rangle\, \phi]\!]_\rho = \{c \in C \mid \forall \Theta \in \{c\}.\xmapsto{e}: \Theta \cap [\![\phi]\!]_\rho \neq \emptyset\}$$

$$[\![\phi_1 \vee \phi_2]\!]_\rho = [\![\phi_1]\!]_\rho \cup [\![\phi_2]\!]_\rho \qquad [\![[e]\, \phi]\!]_\rho = \{c \in C \mid \forall \Theta \in \{c\}.\xmapsto{e}: \Theta \subseteq [\![\phi]\!]_\rho\}$$

$$[\![\mu Z.\phi]\!]_\rho = \bigcap\{\ddot{C} \mid [\![\phi]\!]_{\rho[Z \to \ddot{C}]} \subseteq \ddot{C}\} \qquad [\![\nu Z.\phi]\!]_\rho = \bigcup\{\ddot{C} \mid \ddot{C} \subseteq [\![\phi]\!]_{\rho[Z \to \ddot{C}]}\}$$

Let ϕ be a closed formula, i.e., every variable Z only occurs in ϕ under the scope of μZ or νZ. Then $\mathcal{A}$ *satisfies* ϕ, written $\mathcal{A} \models \phi$, if $C^{\mathrm{i}} \subseteq [\![\phi]\!]_{\rho_0}$, where ρ_0 maps every variable to the empty set. Furthermore, $\mathcal{A}$ *falsifies* ϕ if $\mathcal{A} \models \mathsf{dual}(\phi)$.

The above satisfaction definition is standard except for the diamond and box operators: To guarantee that satisfaction of $[\![\langle e \rangle\, \phi]\!]_\rho$ under ρ is being preserved at every refinement, all transitions labeled e in the ν-automaton (refinement can remove all but one) must have a suitable candidate in its target set Θ, i.e., $\Theta \cap [\![\phi]\!]_\rho \neq \emptyset$ is required. To guarantee that satisfaction of $[\![[e]\, \phi]\!]_\rho$ under ρ is being preserved at every refinement, all targets Θ of any transition labeled e in the ν-automata must be suitable candidates: $\Theta \subseteq [\![\phi]\!]_\rho$.

An example illustrates that this satisfaction definition is indeed 3-valued: The ν-automaton $\widetilde{\mathcal{A}} = (\{c\}, \{c\}, \{(c, e, \{c\}), (c, e, \emptyset)\})$ neither satisfies nor falsifies the formula $\langle e \rangle\, \mathtt{tt}$. However, for concrete ν-automata the satisfaction definition is two-valued, and corresponds to the normal μ-calculus satisfaction for transition systems (where the dual formula corresponds to negation) in their image under δ^{ps}. Note that $\mathcal{A}$ might not satisfy ϕ, although all concrete refinements of $\mathcal{A}$ satisfy ϕ. This is, e.g., the case for the above ν-automaton $\widetilde{\mathcal{A}}$ and the formula $\phi = \langle e \rangle\, \mathtt{tt} \vee [e]\, \mathtt{ff}$. However, our satisfaction is sound, as stated by the following theorem:

Theorem 3.2 *Satisfaction is sound for refinement: If a ν-automaton $\mathcal{A}_1$ refines a ν-automaton $\mathcal{A}_2$, then for any closed formula $\phi \in \mathcal{F}$ we have $\mathcal{A}_2 \models \phi \Rightarrow \mathcal{A}_1 \models \phi$.*

4 State machine semantics

4.1 Simple state machine with a persistent choice operator

The basic modules of UML state machines[6] are variables, states and annotated transitions between them. The structure of, and notation for, our simple state machines is given as follows: A simple state machine SM is a tuple $(V, \sigma_{init}, \mathcal{S}, s_{init}, T)$ comprising a set of integer-variables V, an initial variable assignment σ_{init}, a set of states $\mathcal{S}$, an initial state $s_{init} \in \mathcal{S}$, and a set of transitions T. A transition t comprises a source $\pi_{\mathrm{src}}(t) \in \mathcal{S}$, a target $\pi_{\mathrm{tgt}}(t) \in \mathcal{S}$, an event $\pi_{\mathrm{ev}}(t) \in E$ denoting the external trigger, a guard $\pi_{\mathrm{gd}}(t)$ denoting a necessary condition for enabledness, and an SM-action $\pi_{\mathrm{act}}(t)$, which has to be executed when the transition is taken. For sake of simplicity we restrict the set of guards $(g \in)G$ to boolean expressions over V. An SM-action consists of a 'do nothing'-action skip, the sending of an event, or a variable assignment – where exp may contain instances of the **p**er**s**istent **cho**ice operator (pscho) that models persistent nondeterminism for variable assignments. Formally, an SM-action α is given by the following BNF-grammar:

$$\alpha ::= \mathsf{skip} \mid \mathsf{send}(\overline{e}) \mid v := exp \qquad\qquad exp ::= v \mid n \mid exp \otimes exp \mid \mathsf{pscho}\, N$$

where v is a variable from V, n an integer-constant, $\overline{e}$ an element from the set of output events E of the state machine, $\otimes$ any standard binary operator on integers such as addition or subtraction, and N is a (finite) set of integer-constants – the possible return values of the persistent choice operator. The set of all SM-actions is denoted by Act.

Remark 4.1 So far we modeled only persistent nondeterminism with respect to variable assignments. Choice pseudostates[7], which exist in UML state machines, can be used to model persistent nondeterminism with respect to state machine states $\{s_i \mid i \in I\}$. To that end we take a choice pseudostate, let it point to state machine targets s_i with guards $[x = i]$ for $i \in I$, and then put a SM-action $x := \mathsf{pscho}\, I$ on the incoming transition for said choice pseudo-state.

We first discuss the semantics of state machines informally. At each step a single event e is dispatched from the event pool where the order of dispatching is left open in [25]. This event enables a transition if (i) its source is currently active, (ii) e is its trigger, and (iii) its guard evaluates to true with respect to the current variable assignment. Among all enabled transitions, one is chosen, its corresponding SM-

[6] Note that it is possible to introduce persistent nondeterminism to statechart semantics in a similar way. But since the semantics of triggering events are different (sets of events in statecharts in contrast to single events in state machines) this simple but cumbersome adaption is omitted.

[7] Choice pseudo-states are intermediate states with respect to the *run to completion* steps – where decisions via guarded transitions are made and no external communication is excepted.

action is executed, and the active state becomes its target instead of its source. If no transition is currently enabled for e, different interpretations are possible:

U ('no enabled transition', corresponds to under-specification): Arbitrary behavior is possible after consuming e. A transition in state machines can be added as long as its guard conjoined with the guard of any other transition from the same source and with the same event is unsatisfiable. This harmonizes with the refinement pattern of [28].

D ('no enabled transition', corresponds to discarding): e is consumed without executing any SM-action. This corresponds to the standard interpretation of UML state machines[8].

We first provide formal semantics to the **U**-approach, the formal semantics to the **D**-approach is given in Example 4.2.

$\mathcal{V}$ denotes the set of all possible variable assignments of V. The guards are evaluated with respect to a variable assignment in the usual way by a function eval : $G \times \mathcal{V} \to \{\top, \bot\}$, where $\top$ corresponds to the boolean value true and $\bot$ to false. Function $\mathsf{calc} : \mathrm{Act} \times \mathcal{V} \to 2^{\mathcal{V}}$ yields the set of updated variable assignments, which is in general a singleton set except in the case of the usage of the persistent choice operator. Formally:

$$\mathsf{calc}(\alpha, \sigma) = \begin{cases} \{\sigma[v \mapsto n] \mid n \in \widetilde{\mathsf{calc}}(exp, \sigma)\} & \text{if } \alpha \text{ is } v := exp \\ \{\sigma\} & \text{otherwise} \end{cases}$$

$$\widetilde{\mathsf{calc}}(exp, \sigma) = \begin{cases} \{\sigma(v)\} & \text{if } exp \text{ is } v \in V \\ \{n\} & \text{if } exp \text{ is constant } n \\ \{n_1 \otimes n_2 \mid \forall j \in \{1,2\} : n_j \in \widetilde{\mathsf{calc}}(exp_j, \sigma)\} & \text{if } exp \text{ is } exp_1 \otimes exp_2 \\ N & \text{if } exp \text{ is } \mathsf{pscho}\, N \end{cases}$$

The formal semantics is given as a ν-automaton: Its state space consists of (i) configurations – SM-states combined with variable assignments – (ii) intermediate states for modeling the sending of output events, and (iii) an additional state c_r, which corresponds to a ν-automaton state abstracting anything. Formally, its state space is $(S \times \mathcal{V}) \cup (E \times S \times \mathcal{V}) \cup \{c_r\}$. Its only initial state is the initial SM-state combined with the initial variable assignment $(s_{init}, \sigma_{init})$. Its hypertransitions are given by the derivation rules in Fig. 4, where its underlying set of labels is $E \cup E$.

Example 4.2 The formal semantics of the state machine on the left in Fig. 1 and of the one in Fig. 5 is the ν-automaton on the left (respectively right) in Fig. 3. Here, the state machine constraint "all other events discarded" means that the **D**-approach is taken, in which case rule (f) is replaced by

$$(\mathrm{f}')\frac{\forall t \in T : (\pi_{\mathrm{src}}(t) = s \wedge \pi_{\mathrm{ev}}(t) = e) \Rightarrow \mathrm{eval}(\pi_{\mathrm{gd}}(t), \sigma) = \bot}{(s, \sigma) \overset{e}{\rightarrowtail} \{(s, \sigma)\}}$$

[8] In UML state machines an event can also be declared to be deferred at a state, in which case the triggering or discarding of this event is moved to a subsequent state where it is no longer deferred and becomes active.

$$\text{(a)}\ \frac{t \in T \quad \text{eval}(\pi_{\text{gd}}(t), \sigma) = \top \quad \pi_{\text{act}}(t) = \mathsf{send}(\overline{e})}{(\pi_{\text{src}}(t), \sigma) \overset{\pi_{\text{ev}}(t)}{\rightarrowtail} \{(\overline{e}, \pi_{\text{tgt}}(t), \sigma)\}} \qquad \text{(b)}\ \frac{\overline{e} \in \overline{E}}{(s, \sigma) \overset{\overline{e}}{\rightarrowtail} \emptyset} \qquad \text{(c)}\ \frac{e \in E \cup \overline{E}}{c_r \overset{e}{\rightarrowtail} \{c_r\} \quad c_r \overset{e}{\rightarrowtail} \emptyset}$$

$$\text{(d)}\ \frac{t \in T \quad \text{eval}(\pi_{\text{gd}}(t), \sigma) = \top \quad \forall \overline{e} \in \overline{E} : \pi_{\text{act}}(t) \neq \mathsf{send}(\overline{e})}{(\pi_{\text{src}}(t), \sigma) \overset{\pi_{\text{ev}}(t)}{\rightarrowtail} \{(\pi_{\text{tgt}}(t), \sigma') \mid \sigma' \in \mathsf{calc}(\pi_{\text{act}}(t), \sigma)\}} \qquad \text{(e)}\ \frac{}{(\overline{e}, s, \sigma) \overset{\overline{e}}{\rightarrowtail} \{(s, \sigma)\}}$$

$$\text{(f)}\ \frac{\forall t \in T : (\pi_{\text{src}}(t) = s \wedge \pi_{\text{ev}}(t) = e) \Rightarrow \text{eval}(\pi_{\text{gd}}(t), \sigma) = \bot}{(s, \sigma) \overset{e}{\rightarrowtail} \{c_r\}} \qquad \text{(g)}\ \frac{e \in E \cup \overline{E} \setminus \{\overline{e}\}}{(\overline{e}, s, \sigma) \overset{e}{\rightarrowtail} \emptyset}$$

Fig. 4. Transition derivation rules. The firing of a transition that sends an output event leads (by adapting the active SM-state) to an intermediate state (a), which only sends exactly the corresponding output event (e) but nothing else (g). Output events can only be sent in such intermediate states (b). If the SM-action is different from a send-action, a hypertransition subsuming all the possible outcomes with respect to calc is derived (d). In case no transition is enabled for an event (f), a transition to the state c_r, which abstracts everything ensured by (c), is derived. Note that (c) encodes two rules: one where $c_r \overset{e}{\rightarrowtail} \{c_r\}$ is the conclusion, another where $c_r \overset{e}{\rightarrowtail} \emptyset$ is the conclusion – both having the same premise.

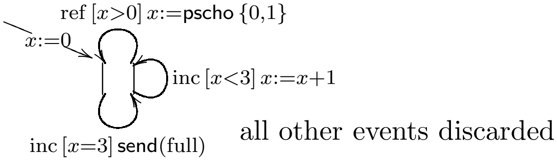

Fig. 5. State machine from Fig. 1 after its leftmost transition, labeled with ref, is removed and non-present events are interpreted as "discarded".

where self loops (discarding) – rather than the reaching of the state that abstracts everything – happen. Note that adding the constraint "all other events discarded" is a refinement step. This, and that the removal of transitions (as long as for every variable assignment the set of enabled events remains the same) is a refinement step, are illustrated by the example where the ν-automata on the right in Fig. 3 is a refinement of the one on the left in Fig. 3.

4.2 Discussion

The order of persistent and resolvable nondeterminism defined in μ-automata, introduced in Subsection 2.4, does not fit well the requirements for a semantics of state machines with persistent choice operator: ν-automata with their ready simulation approach of refinement allow direct use of variable assignments as ν-automaton states. This is possible, because under-specification is modeled similarly in state machines and ν-automata: Removing transitions in the state machine directly corresponds to removing transitions in the semantic model. This correspondence does not exist between state machines and μ-automata. Consequently, it is necessary to use more complex μ-automaton states: In addition to variable assignments, additional information is needed for expressing every possible choice of taken transitions

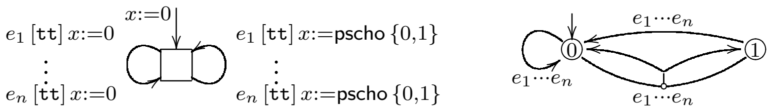

Fig. 6. A state machine together with its semantics in terms of ν-automata.

for each SM-action. This exponentially blows up the state space:

Example 4.3 A state machine and its semantics in terms of ν-automata are given in Fig. 6. The semantics in terms of μ-automata is significantly more complex, and for this reason not depicted. Its set of states is $\{0,1\} \times 2^n$, its set of initial states is $\{0\} \times 2^n$, and its transition relation is

$$\bigcup_{j\in\{1,2\},\ddot{C}\in 2^n} (\{((j,\ddot{C}),\{0\}\times 2^n)\} \cup \{((j,\ddot{C}),\{1\}\times 2^n) \mid j \in \ddot{C}\}$$

In different settings, μ-automata may well be more suitable than ν-automata; e.g. in giving an operational semantics for a process algebra with parallel composition and a persistent choice operator [9].

4.3 Adaption of existing semantics

We sketch how existing formal semantics of state machines can be easily adapted such that they can handle also persistent choice operators. Since our extension of state machines with persistent choice is orthogonal to their run to completion steps, concurrency, and hierarchies, those notions are still adequately handled in the presence of persistent choice operators.

We assume that the formal semantics is given in terms of transition systems that are obtained by derivation rules (small step semantics). Also, we assume that the derivation rules use an evaluation function calc for the SM-actions, i.e., calc maps a SM-action together with a variable assignment to a variable assignment (and possibly additional information). This kind of formal semantics occurs, e.g., in [10,32].

As the first transformation, function calc is adapted to SM-actions containing also persistent choice operators. This is achieved by changing the range of calc to sets (rather than elements) of variable assignments.

Next, the derivation rules are adapted such that hypertransitions are obtained. This is done by replacing the target of the derived transition by the set consisting of the original target. But when calc is used for the derivation, the target set contains all possible variations with respect to the result of calc. For example rule

$$\frac{\alpha \neq \mathsf{skip} \quad \mathsf{calc}(\alpha,\sigma) = (\ell,\alpha',\sigma')}{(\sigma,\mathcal{A},\mathsf{do},H,\alpha,\ddot{s},\beta,T,\ddot{T}) \overset{\ell}{\rightarrowtail} (\sigma',\mathcal{A},\mathsf{do},H,\alpha',\ddot{s},\beta,T,\ddot{T})}$$

taken from [10] – describing the execution of a SM-action – is transformed into

$$\frac{\alpha \neq \mathsf{skip}}{(\sigma, \mathcal{A}, \mathsf{do}, H, \alpha, \ddot{s}, \beta, T, \ddot{T}) \overset{\ell}{\rightarrowtail} \{(\sigma', \mathcal{A}, \mathsf{do}, H, \alpha', \ddot{s}, \beta, T, \ddot{T}) \mid (\ell, \alpha', \sigma') \in \mathsf{calc}(\alpha, \sigma)\}}$$

Note that in the semantics of [10] the evaluation function also yields (i) a label (e.g. the sending of an output event), which will be the same for all elements of $\mathsf{calc}(\alpha, \sigma)$, (ii) a SM-action, which remains to be executed, and (iii) a variable assignment.

Finally, absence of outgoing transitions for a given label is handled such that it harmonizes with ν-refinement. If at a state there is no outgoing transition with a generative event (see Subsection 2.4), then a corresponding transition pointing to the empty set is added. The same is done for events whenever no events can be accepted in the current state, e.g., if some computation must first be finished. The absence of outgoing transitions – for an event e in a semantic state c where events can be accepted – is handled by a case analysis on the different interpretations introduced in Section 4.1:

U: Add a new state c_r to the ν-automaton and add the additional two derivation rules (c) from Fig. 4. Thus c_r represents a state that is refined by any ν-automaton state. A transition labeled with e from c to $\{c_r\}$ is also added.[9]

D: A self-loop from c labeled with event e is added.

5 Random choice operators

A random choice operator can be used to express a probability distribution. Probability distributions are a form of persistent choice. For example, the probabilistic choice of a successor state of state s in a Markov chain expresses a nondeterministic choice that remains to be nondeterministic each time state s is visited. Of course, this persistence is controllable to some degree through the use of familiar tools from stochastic processes.

5.1 ν-automata and random choice

We sketch how ν-automata can be adapted to support random choice operators with an underlying distribution: First, they are extended by moving from power sets to distributions as transition targets, i.e., $\longmapsto \;\subseteq C \times \mathcal{L} \times \mathsf{Dist}(C)$ where $\mathsf{Dist}(C)$ denotes the possible discrete distributions over the set C (total functions f from C to $[0, 1]$ such that $\sum_{c \in C} f(c) = 1$). Furthermore, the refinement notion is adapted to handle probabilities by lifting the refinement relation to distributions, i.e., *probabilistic automata* with *strong simulation* [29] as refinement notion. Then calc has to be adapted such that distributions rather than sets of states are obtained. The adaption of the existing semantics is then made similar to that in Section 4.3. Note that in such a probabilistic setting, satisfaction is defined over probabilistic logics, like PCTL [11], rather than over the μ-calculus.

[9] An additional transition to the empty set is not needed here, since *input enabledness* is usually guaranteed. Otherwise, e.g. in the context of event deferral, such a transition has to be added.

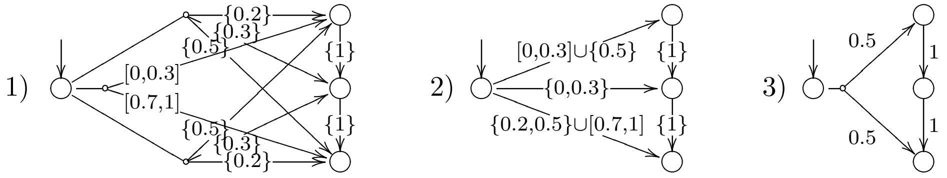

Fig. 7. Illustration of 1) a ν-automaton extended with sets of probabilities, 2) a probabilistic specification system, which has all concrete refinements of 1) and is minimal with respect to refinement among those models having all concrete refinements of 1); and 3) a deterministic probabilistic automaton being a concrete refinement of 2) but not of 1). All three models have the same single event-label, which is therefore omitted.

5.2 *Under-specified random choice*

This approach can also deal with randomized choice that is under-specified. Exact knowledge of distributions is often not a realistic assumption and does not facilitate top-down-development, because abstract systems already need to specify exact probabilities. Therefore, approaches emerged [18] where only sets or intervals of allowed probabilities are given. We can accommodate this as follows:

Probabilistic automata (PA) are extended by moving from distributions to sets of possible distributions as transition targets, i.e., $\longmapsto\ \subseteq C \times \mathcal{L} \times (2^{[0,1]})^C$ which yields a generalization of *probabilistic specification systems* (PSS) [18], where the transition relation is a subset of $C \times (2^{[0,1]}) \times C$. Furthermore the refinement notion has to be adequately adapted, similar to the treatment in [18]. This new class of models, which we denote by PSSA, has more expressive power than PSS in the sense that more sets of concrete refinements can be described via refinement. This is informally argued in Fig. 7. Note that PSSA has also more expressive power than PA, first with respect to strong simulation – since probabilistic intervals cannot be finitely described by nondeterminism – and, second with respect to *strong probabilistic simulation* [29], since this has already less expressive power than PSS.

In order to give the formal semantics, calc has to be adapted such that sets of distributions rather than distributions are obtained. The adaption of the existing semantics is then made similar to that in Section 4.3. The exact development of the theoretical foundation of this class of models, including precise refinement and satisfaction definitions, is a topic of future work.

6 Related work

The process algebra CSP [15,27] has internal and external choice operators for expressing nondeterminism. Since all CSP process terms appear to have deterministic refinements, CSP and FDR cannot explicitly support models with persistent choice but one can represent persistent non-determinism in an environment and instrument refinement checks such that all choices of the environment are non-resolvable for all implementations. Process algebras having persistent as well as resolvable choice operators occur in [1,9], differing in their interpretation of parallel composition. Their semantics is given in terms of μ-automata and "persistent choice" is called "inherent choice" in loc. cit.

In the world of predicate transformers and Hoare logics as their cousins, there is a rich body of work on combining probabilistic and (resolvable) non-deterministic choice, e.g. [23]. Such work is in the tradition of deductive verification and may exploit algebraic identities for the simplification of proofs. Such deductive reasoning, e.g. in the form of assume-guarantee proof rules for model checking [7], is outside the scope of this paper.

In Subsection 4.2, we argued why μ-automata [16] are not suitable as semantic models for state machines having persistent choice operators. Further abstract models that can express persistent as well as resolvable nondeterminism are, e.g., (disjunctive) modal transition systems [21,22], mixed transition systems [4], generalized Kripke modal transition systems [30], hypermixed Kripke structures [8], and modal automata [5]. These models have additional may-transitions meaning that refinements may have those transitions but not necessarily so. Hyper Kripke modal transition systems [31] interpret must-hypertransitions analogously to the μ-refinement approach, whereas may-hypertransitions are interpreted analogously to the ν-refinement approach. None of these models are suitable as semantic models for UML state machines by the same arguments as put forward for μ-automata. Note that the additional may-transitions do not help to define the semantics, since there is no concept in state machines that supports such kind of modeling directly (in state machine refinements, a transition per label has to remain, i.e., not all transitions can be removed [28]). A comparison to probabilistic models is already made in Section 5. In [20], the authors use probabilistic games to differentiate between probabilistic (persistent) and resolvable nondeterminism. Their model is close to PSSA based on *strong probabilistic simulation*.

An overview of existing formal semantics of UML state machines is given in [3], where 26 different approaches are compared – and none of them features persistent choice. [10], which is not referenced in [3], gives a complete formal semantics of UML 2.0 state machines without persistent choice. As already mentioned, refinement patterns of statecharts are presented in [28], where the underlying formal semantics are streams and therefore no persistent nondeterminism is handled. In [6] state machines are mapped to terms of CSP and so a stronger semantics is given by considering trace- and failure-based refinement.

The work most closely related to ours is [17], where formal semantics in terms of probabilistic automata based on strong simulation is given to state machines having an additional randomized pseudo-state. Their semantic mapping corresponds more to a *run to completion* step rather than to the firing of transitions. This leads to the problems that (i) their refinement notion – implicitly given via probabilistic automata – is only sound but imprecise, i.e., there are unexpected semantic refinements, illustrated in Fig. 8, (ii) a well formed condition – that every variable may only be changed by one of the firing transitions – is needed, and (iii) their state space is exponential in the number of enabled probabilistic transitions. None of these problems occur in our definition of probabilistic pseudo-states via choice pseudo-states (see the discussion in Remark 4.1).

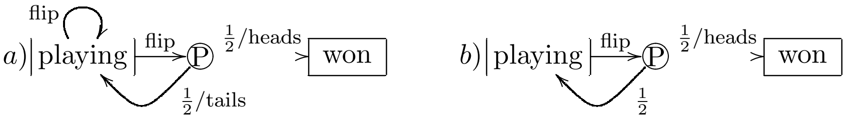

Fig. 8. a) A state machine having probabilistic pseudo-states and b) one of its possible (unwanted) refinement with respect to the formal semantics of [17], where heads, but no tails can occur.

7 Conclusion

We developed a new class of models, ν-automata, which describes sets of transition systems by a suitable refinement notion. ν-automata employ hypertransitions, their refinement notion generalizes ready simulation, and they are suitable as semantic models for systems that feature or benefit from persistent nondeterminism as well as resolvable nondeterminism. In particular, a formal semantics for a simple state machine language was given in ν-automata and it was sketched how existing semantics of state machines can be adapted if persistent choice operators are added to the underlying programming language. A compositional 3-valued satisfaction definition over the μ-calculus was given for ν-automata and proved to be sound for property verification.

It is future work to establish tool support based on the ν-automata semantics for satisfaction and refinement. In particular, it should be examined how ν-automata can be used in the context of abstraction in order to speed up verification.

References

[1] Baeten, J. C. M. and J. A. Bergstra, *Process algebra with partial choice.*, in: B. Jonsson and J. Parrow, editors, *CONCUR*, LNCS **836** (1994), pp. 465–480.

[2] Bloom, B., S. Istrail and A. Meyer, *Bisimulation can't be traced*, J. ACM **42** (1995), pp. 232–268.

[3] Crane, M. L. and J. Dingel, *On the semantics of UML state machines: Categorization and comparison*, Technical Report 2005-501, Queen's University (2005).
URL http://www.cs.queensu.ca/TechReports/Reports/2005-501.pdf

[4] Dams, D., R. Gerth and O. Grumberg, *Abstract interpretation of reactive systems*, ACM Trans. Program. Lang. Syst. **19** (1997), pp. 253–291.

[5] Dams, D. and K. S. Namjoshi, *Automata as abstractions*, in: R. Cousot, editor, *VMCAI*, LNCS **3385** (2005), pp. 216–232.

[6] Davies, J. and C. Crichton, *Concurrency and refinement in the unified modeling language*, Formal Aspects of Computing **15** (2003), pp. 118–145.

[7] de Roever, W.-P., H. Langmaack and A. Pnueli, editors, "Compositionality: The Significant Difference," LNCS **1536**, Springer, 1998.

[8] Fecher, H. and M. Huth, *Ranked predicate abstraction for branching time: Complete, incremental, and precise.*, in: S. Graf and W. Zhang, editors, *ATVA*, LNCS **4218** (2006), pp. 322–336.

[9] Fecher, H. and H. Schmidt, *Process algebra having inherent choice: Revised semantics for concurrent systems.*, in: *SOS*, 2007, will appear in ENiTCS.
URL http://www.informatik.uni-kiel.de/~hf/papers/Fecher07sos.pdf

[10] Fecher, H. and J. Schönborn, *UML 2.0 state machines: Complete formal semantics via core state machines.*, in: L. Brim, M. Leucker, B. R. Haverkort and J. van de Pol, editors, *FMICS and PDMC 2006*, LNCS **4346** (2007), pp. 244–260.

[11] Hansson, H. and B. Jonsson, *A logic for reasoning about time and reliability.*, Formal Asp. Comput. **6** (1994), pp. 512–535.

[12] Harel, D., *Statecharts: A visual formulation for complex systems.*, Sci. Comput. Program. **8** (1987), pp. 231–274.

[13] Harel, D. and E. Gery, *Executable object modeling with statecharts.*, IEEE Computer **30** (1997), pp. 31–42.

[14] Harel, D. and H. Kugler, *The Rhapsody Semantics of Statecharts (or, on the executable core of the UML)*, in: H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder and E. Westkämper, editors, *SoftSpez Final Report*, LNCS **3147** (2004), pp. 325–354.

[15] Hoare, C. A. R., "Communications Sequential Processes," International Series in Computer Science, Prentice Hall, 1985.

[16] Janin, D. and I. Walukiewicz, *Automata for the modal mu-calculus and related results*, in: J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science*, LNCS **969** (1995), pp. 552–562.

[17] Jansen, D. N., H. Hermanns and J.-P. Katoen, *A probabilistic extension of UML statecharts.*, in: W. Damm and E.-R. Olderog, editors, *FTRTFT*, LNCS **2469** (2002), pp. 355–374.

[18] Jonsson, B. and K. G. Larsen, *Specification and refinement of probabilistic processes*, in: *LICS* (1991), pp. 266–277.

[19] Kozen, D., *Results on the propositional μ-calculus*, Theor. Comput. Sci. **27** (1983), pp. 333–354.

[20] Kwiatkowska, M. Z., G. Norman and D. Parker, *Game-based abstraction for Markov decision processes.*, in: *QEST* (2006), pp. 157–166.

[21] Larsen, K. G. and B. Thomsen, *A modal process logic*, in: *LICS* (1988), pp. 203–210.

[22] Larsen, K. G. and L. Xinxin, *Equation solving using modal transition systems*, in: *LICS* (1990), pp. 108–117.

[23] McIver, A. and C. Morgan, "Abstraction, Refinement and Proof for Probabilistic Systems," Springer, 2004.

[24] Milner, R., "Communication and Concurrency," International Series in Computer Science, Prentice Hall, 1989.

[25] Object Management Group, "UML Superstructure Specification, v2.0 formal/05-07-04," (2005). URL http://www.omg.org/cgi-bin/doc?formal/05-07-04

[26] Refsdal, A., R. K. Runde and K. Stølen, *Underspecification, inherent nondeterminism and probability in sequence diagrams.*, in: R. Gorrieri and H. Wehrheim, editors, *FMOODS*, LNCS **4037** (2006), pp. 138–155.

[27] A. W. Roscoe. *Theory and Practice of Concurrency.* Prentice Hall, 1997.

[28] Rumpe, B., "Formale Methodik des Entwurfs verteilter objektorientierter Systeme," TUM Doktorarbeit, Technische Universität München (1996).

[29] Segala, R., *Probability and nondeterminism in operational models of concurrency.*, in: C. Baier and H. Hermanns, editors, *CONCUR*, LNCS **4137** (2006), pp. 64–78.

[30] Shoham, S. and O. Grumberg, *Monotonic abstraction-refinement for CTL*, in: K. Jensen and A. Podelski, editors, *TACAS*, LNCS **2988** (2004), pp. 546–560.

[31] Shoham, S. and O. Grumberg, *3-valued abstraction: More precision at less cost.*, in: *LICS* (2006), pp. 399–410.

[32] von der Beeck, M., *A structured operational semantics for UML-statecharts*, Software and System Modeling **1** (2002), pp. 130–141.

[33] Wilke, Th., *Alternating tree automata, parity games, and modal μ-calculus*, Bull. Soc. Math. Belg. **8** (2001), pp. 359–391.