Formalizing UML State Machines Semantics for Formal Analysis–A survey (PRELIMINARY VERSION)

Liu Shuang et al.
National University of Singapore
`lius87@comp.nus.edu.sg`

March 21, 2014

Abstract

UML state machines are widely used in modeling the dynamic behavior of object-oriented designs in industry. UML state machines specification, which is maintained by Object Management Group (OMG), is documented in natural language instead of formal language. The inherited ambiguity of natural language introduces inconsistencies to the resulting state machine model. Formalizing UML state machine specification can solve the ambiguity problem and provide a uniformed view to software designers and developers. It would also provide a foundation for automatic verification of UML state machines models, which can help to find software design vulnerabilities at an early stage and reduce the development cost. In this report, we provide a comprehensive survey of existing work related to formalizing UML state machines semantics for the purpose of conducting model checking on the design models. We have also surveyed tools which have been developed for this purpose.

Contents

1 Introduction

UML state machines, an object-oriented variation of Harel Statechart [Har87], are often used to model the dynamic behaviors of a system. The UML specification, published and managed by the Object Management Group (OMG) [OMG], introduces ambiguities and inconsistencies, which are tedious for manual detection or be verified automatically due to the lack of formal semantics. Defining formal semantics for UML state machines has been capturing more and more attention in the past decades. The benefit of a formal UML state machine semantics is threefold. Firstly, it allows more precise and efficient communication between engineers. Secondly, it yields more consistent and rigorous models. Lastly and most importantly, it enables automatic formal verification of UML state machine models through techniques like model checking, which guarantees the verification of important properties of a system in the early development stage. This results in a possible reduction in the overall cost of the software development cycle.

During the past few decades, a number of works appeared in the literature, which provide formalization for UML state machines for the purpose of model checking UML state machines. Those approaches adopt different semantic models, support different subset of UML state machines features and only a few of them provide tool supports. The existing approaches can be divided into two categories:

1. approaches that directly provide an general operational semantics for UML state machines, and
2. approaches that translate UML state machines into some specific formal languages.

The first kind of approaches provide general operational semantics for UML state machines in terms of inference rules, which is a standard format of formalizing operational semantics. The second kind of approaches provide translation rules from UML state machines to some existing specific language. These kind of approaches usually aim at adopting the formal verification power of the existing specific language, such as Promela [Proc], Petri Nets etc.

Several previous survey papers [PB04, CD05b, LRS10] summarize approaches related to UML state machine formal semantics and automatic verification. Bhaduri *et al.* [PB04] summarized approaches which translated variants of statecharts[1]. (including STATEMATE statechart and UML statechart) into input languages of SMV [CCG$^+$02] and Spin [SPI] model checkers. Crane and Dingel [CD05b], provide a categorization and comparison of

[1]There are variants of statechart diagrams, such as Harel statechart [Har87], STATEMATE statchart [HN96], etc. A comparison of the semantic differences can be found in [CD05a]

26 different approaches of formalizing UML state machine semantics (including denotational and operational semantics). Lund *et al.* [LRS10] surveyed existing works on formalizing UML sequence diagram-like and state machine diagram-like semantics. The survey focused more on sequence diagrams and it selectively discusses some representative approaches.

Since a large number of works have been published in this area of providing directly or indirectly verification support for UML state machines and there is no recent survey paper which summarizes these approaches (The work in [CD05b] is the most related in this aspect, but it is out-of-date and does not cover some related approaches.), we believe it is important to provide a comprehensive study and comparison on existing works which provide operational semantics for UML state machines, especially those contribute to formal verification of UML state machines.

In this paper, we provide a survey of existing approaches which aim at providing formalization for UML state machines (instead of the other variants of statecharts), especially for the purpose of providing verification support for UML state machines. Our survey provides comparisons of those approaches in two dimensions. The first dimension is the semantic model used by the approaches. For the second dimension, we also compare those approaches on the covered features, such as communication aspects, event pool mechanisms and tool supports to automatically verify UML state machines.

The contribution of the survey is three-fold. Firstly, it provides an overview of the status of researches in the area of formalizing UML state machines semantics. Secondly, it provides a study of tool supports for formal verification of UML state machines models. Lastly, it identify future directions of research in this area.

The rest of this report is organized as follows. Section 2 discusses the related surveys in the literature. Section 3 provides the categorization criteria of our survey. Section 4 and Section 5 discuss formalization approaches and translation approaches separately. In Section 6, we surveyed the existing tools which support formal verification of UML state machines. We conclude the paper in Section 7.

2 Related Works

There are some surveys [PB04, CD05b, LRS10] which summarize existing approaches on formal semantics and verification of UML state machines[2]. Each has a different view and coverage scope on existing works.

In 2004, Bhaduri *et al.* [PB04] summarized approaches which translated variants of statecharts (including STATEMATE statechart and UML stat-

[2]In this paper, we use UML state machines and UML statechart in interleaving, respecting the original notation in the surveyed paper.

echart) into input languages of SMV and Spin model checkers. The survey only covers a subset of approaches, with detailed descriptions and discussions about each individual work. The paper also discuss possible future research direcitons, such as adopting slicing, abstraction-based approaches or compositional verification approaches to reduce the state space in model checking, explore abstractions on environments etc. The paper only lists a subset of works which translates statecharts into SMV or Spin model checkers. It does not provide any comparisons among those works or conclusive comments on those works.

In 2005, Crane and Dingel [CD05b], provide a categorization and comparison of 26 different approaches of formalizing UML state machine semantics. They categorized those approaches based on the underlying formalism used and conducted comparisons on other dimensions such as UML state machines feature coverage, tool support etc. This paper provides a high-level comparison and discussion on different aspects of the existing 26 approaches. Though the amount of work discussed is not large, it covers different kinds of approaches and provides a good way to categorize those approaches. The categorization of the work is wide, but the coverage within each categorization is too narrow, a lot of translation based approaches are not included. Moreover, the survey is out of date and does not include recent works.

In 2010, Lund *et al.* [LRS10] surveyed existing works on formalizing UML sequence diagram-like and state machine diagram-like semantics. The survey does not focus on a thoroughly coverage of all the existing approaches; instead, it selectively discusses some representative approaches which fall in one of their categorization criteria. There are two dimensions of categorization. The main dimension is the style of the semantics, namely denotational semantics and operational semantics. The second dimension captures features such as real-time or probability. The focus is on both sequence diagram and state machine diagrams. Though the survey provides new comparison criteria such as the supported properties, refinement support, etc., it covers a limited number of approaches, especially for the UML state machine part. Moreover, no formal verification tool support information for UML state machines is discussed.

We notice the following from these surveys.

1. There are a lot of works, especially 7 recent works, that are not covered by any of these surveys.

2. There is no survey focusing specifically on formalization of UML state machines for the practical application purpose, i.e., formal verification of UML state machines.

This motivate us to provide a survey, which covers works that focus on providing formalization for UML state machines, especially for the purpose of formal verification. Tool supports for such verifications are also provided.

3 Categorization Criteria

In this paper, we survey works on providing operational formalization for UML state machines. Since all the approaches of this kind aim at providing foundations for tool supports of verification of certain kind. Some of them are more general, i.e., they directly provide operational semantics for UML state machines in the form of inference rule or SOS (structured operational semantics) so that any kind of verification tools can be developed based on it. There are also other works which conduct an indirect approach. They translate UML state machines into a specific specification language, which usually have formal verification tool supports.

Based on the above observations, we decide to categorize the surveyed approaches in two dimensions. The first and main dimension is whether the approach is a direct or indirect approach.

As the second dimension, we compare the surveyed works on the features supported, semantic models used, UML specification version, etc. Moreover, we link the approaches with tool supports which are based on those approaches, hence provides a good reference for readers who want to do use those tools.

4 The Translating Approaches

A popular approach to formalize UML state machines is to provide a translation to some existing formal languages (such as Petri nets [JK09] or Abstract State Machines [ER03]), or to the input languages of model checkers (such as Spin, SMV, UPPAAL [UPP]). Those formal languages have their own operational semantics. This kind of approaches can be regarded as an indirect way of providing formalizations for UML state machines. The purpose of this kind of approaches is more explicit, i.e., to utilize existing verification techniques and tools for UML state machines verification. So we regard these approaches as translation approaches and summarize them in this section.

We categorize these approaches based on the target formal languages they adopt, viz., abstract state machines (Section 4.1), Petri nets (Section 4.2), and translation to the input modeling language of model checkers (Section 4.3). We summarize the translation-based approaches in Section 4.4.

4.1 Translation into Abstract State Machines

Abstract State Machines (ASMs) can offer the most general notion of state (which abstract away from graphical form) in the form of structures of arbitrary data and operations which can be tailored to any desired level of abstraction. State machines' configuration changes are represented by transition rules, which consists of conditions and update functions. . On the other hand, the notion of multi-agent (distributed) ASMs can naturally reflect the interaction between objects [ER03]. Spielmann [Spi00], Castillo and Winter [GDC00] and recently Klünder *et al.* [BKKS08] provide theoretic and tool support of model checking abstract state machines.

Translating into ASM

Börger *et al.* [BCR00, BCR03, BCR04] are among the pioneers in formalizing UML state machines into ASMs. ASM contains a collection of states and a collection of rules (conditional, update, Do-forall etc) which update those states. The work [BCR00] in this direction was proposed in 2000. The syntax model is a tuple, which captures the attributes and associations of a construct (states, transitions, etc). This approach covers most UML state machines features, including deferred events, completion events and internal activities associated with states which are mostly left out by other approaches. But pseudostates such as fork, join, junction, choice, terminate are not considered. In contrast, the authors argue that these constructs can find their semantically equivalent constructs in their defined subset, where they use a transition from (resp. to) the boundary of an orthogonal composite state to replace the join (resp. fork) pseudostates. But with a join (resp. fork) pseudostate, we can decide which substates of the target orthogonal composite state are going to be entered simultaneously, while this semantic meaning is not expressible by the "equivalent" method they provide.

In 2003, another work [BCR03] extends [BCR00] to support transitions from and to orthogonal composite states[3] in the context of event deferral and RTC step.

In 2004, Börger *et al.* [BCR04] provided some further discussions about the ambiguities in the official semantics of UML state machines [UMLa] and their solutions. The works by Eörger *et al.* [BCR00, BCR03, BCR04] cover a large set of features and the formalization is easy to follow due to the abstract feature of ASM notations. But no automatic translating tool has been developed based on these work so far.

Another approach [CGHS00] which translates UML state machines into ASMs was proposed by Compton *et al.* To be precise, this work translates UML state machines into extended ASMs, in which ASMs are extended to represent inter-level transitions with multiple transitions which do not cross any boundary of states. This extension makes it easier to deal with interruptions; it also makes the formalization procedure more structured

[3]The orthogonal composite state acts as the main source/target state of the transition, i.e., the source/target of the transition can be a substate of the orthogonal composite state at any depth.

and layered (since inter-level transitions break the hierarchical structure of UML state machine and such a decomposition of inter-level transitions into multiple transitions preserve such an hierarchical structure). It shares a similar idea with [BCR00, BCR03] in the rest of translation procedure. Agents are used to process executions of UML state machines. An activity agent is used to model the execution of an activity associated with a node. The execution of agents are divided into different modes, which indicates what kind of rules (operations) the current agent should take.

Jürjens [Jür02] provided a semantics in the form of abstract state machines in 2002. The focus of this work is not on supporting various features of UML state machine. Instead, they rather focus on the communications aspects between state machines. The work explicitly models the message (with parameters) passing between state machines as well as the event queue.

Translating into variant of ASM

Jin *et al.* [JEJ04] (2004) provided an approach which syntactically defines UML statecharts as attributed graphs which are described using the Graph Type Definition Language (GTDL). They further provide some constraints in the form of predicates to specify the well-formedness rules of statecharts, which is considered as the static semantics of a UML statechart. The semantic domain is defined as an Object Mapping Automaton (OMA [JK98]), which is a variant of ASMs. Given the abstract syntax (of the attributed graph) of a well-formed statechart, they first "compile" it into OMA algebraic structures, which specifies "advanced static semantics" of a UML statechart. Based on OMA algebraic structures, two rules (viz., the initialization rule and the run-to-completion rule) are defined to describe the dynamic behavior of a UML statechart. The syntax and semantics provided by this approach, benefiting from the highly compatibility of the abstract syntax of attributed graph with UML statecharts, are more intuitive and easy to follow. But this approach supports a limited subset of UML statecharts features and does not even include concurrent composite states as well as choice vertexes.

To summarize,approaches translating UML state machines into ASMs tend to support more advanced features such as orthogonal composite states, completion/defer events, fork/join/history/choice pseudostates and inter-level transitions. The reason may be that ASMs are more flexible in terms of syntax format as well as update rules and are more suitable to express the non-structured feature of UML state machines.

4.2 Translation into Petri Nets

Petri nets is a mathematical modeling language with formal semantics. There are a variety of different Petri nets based on the application domain.

Colored Petri Nets (CPN) [JK09] are a special case of Petri nets in which the tokens identifies attributes (types). As a result, it leads to a more clear and simple representation.. Several approaches [Gom00, BP01, CKZ11, ACK12] in the literature translate UML state machines into (Colored) Petri nets. We review them in the following.

Robert *et al.* [Gom00] (2000) present an approach which uses CPN to model and validate the behavior characters of concurrent object architectures modeled by UML. The authors discuss how to map active/negative objects as well as message communications into CPN. Synchronous as well as asynchronous communications are discussed in message communications. Though not specifically dealing with UML state machines, this paper provides a general idea of transforming UML diagrams to (Colored) Petri nets.

Luciano *et al.* [BP01] (2001) propose another approach to formalize UML with high-level Petri nets, i.e., Petri nets whose places can be refined to represent composite places, and class diagrams, state diagrams and interaction diagrams are considered. Customization rules are provided for each diagram. But the authors do not provide details about those customization rules; instead, they illustrate the steps with the Hurried Philosopher Problem. The analysis and validation are also discussed, especially how to represent in UML the properties (such as absence of deadlocks, fairness etc.), as well as how to translate them into Petri nets representations.

Trowitzsch and Zimmermann [TZ05] (2005) proposed to translate a subset of timed UML state machines into Stochastic Petri nets. The work [TZ05] uses stochastic Petri nets, which contain exponential transitions, making it more suitable to model time events. The approach does not cover many UML state machines features, but time events are discussed.

Choppy *et al.* [CKZ11] (2011) propose an approach that formalizes UML state machines by translating them to hierarchical colored Petri nets. They provide a detailed pseudo algorithm for the formalization procedure. They map simple states of UML state machine into Petri nets places and composite states of UML state machine into composite Petri nets places. Transitions in UML state machines are mapped to arcs in Petri nets and corresponding triggering events are properly labeled. An extra place called *Events* is modeled with an event place in Petri nets, with each type of event assigned different color type. Entry and exit actions of UML state machines are modeled with an arc in Petri nets which is labeled with the proper event type and ends in the event place. Though the mapping from UML state machines to HCPN is clearly expressed compared to [BP01], a very limited subset of UML state machines features are supported: only the very basic features such as simple state, composite state, transitions, triggering event and entry/exit actions are discussed. How to type the events, and how to deal with concurrency invocations of a concurrent composite state are not discussed.

André *et al.* [ACK12] (2012) propose an approach different from the work by Choppy [CKZ11], and support a larger subset of UML state machine features, including state hierarchy, internal/external transitions, entry/exit/do activities, history pseudostates, etc. However, a limitation of that approach is that concurrency is left out: hence fork and join pseudostates, as well as synchronous communication between state machines is not considered.

Petri net is used in modeling work flow in industry. It is more rigorous benefiting from its mathematical supporting. But it is always hard for non-experts to understand. With automatic translators from UML state machines to Petri net, we can benefit from the rigorous verification power of existing Petri net verification tool. However, approaches translating UML state machines to Petri net usually cover a small subset of UML state machine features.

4.3 Translation into the Input Language of Model Checkers

Another kind of approaches consists in translating UML state machines into the modeling language of some model checkers (Spin, SMV, FDR or PAT). Bhaduri *et al.* [PB04] provide a good (though now somehow outdated) survey on this kind of approaches. But it focuses on many variants of Harel's statechart [Har87, HN96, HLN$^+$90], such as RSML or UML. We rather focus specifically on UML state machine, which is the object-oriented variant of Harel's statecharts.

In this section, we perform a detailed survey of this kind of approaches. We sort the different approaches based on the model checker's input language they adopt: Spin, SMV, and other model checkers.

Approaches Based on Spin

Latella *et al.* are among the first few researchers who contributed to the formal verification of UML state machines. They utilize Extended Hierarchical Automaton (EHA) as an intermediate representation of UML state machines; then they define formal semantics of EHA with Kripke structures as the semantics domain [DIM99] (1999). Based on this formalization work [DIM99], they proceed one step further in [LMM99] (1999) by providing translations from UML state machines to PROMELA, the input language of the Spin model checker. The translation function takes a hierarchical automaton as input and generates PROMELA code as output. This approach uses STEP PROMELA process to simulate a run to completion step in UML state machines, which includes dispatching of events from the environment; identify candidate transitions to fire; solve conflicts and select firable transitions; actual execution of the selected transitions (including identifying the next configuration after execution of the current transition and maybe side effects, which are events generated during the execution of actions associated with the transition). The run to completion step in UML state machine is, as indicated by the name itself, non-interruptable (but can be stopped[4]). This is guaranteed by the PROMELA atomic command. The translation process is structured since it is based on the pre-defined formal semantics of EHA [DIM99]. The authors also provide proof for the translation to guarantee the correctness of the procedure.

Timm *et al.* [SKM01] (2001) provided a method to model checking UML state machines as well as collaborations with the other UML diagrams. They compile UML state machines into a PROMELA model and collaborations into sets of Büchi automata, and then invoke the Spin model checker to verify the model against the automata. Each state in the state machine is mapped to an individual PROMELA process. Two additional PROMELA processes are generated to handle event dispatching and transitions. The event queue is modeled as buffered channels and communication among processes are modeled via unbuffered channels, i.e., they are synchronized. This approach further considers the consistencies between UML diagrams, i.e., collaboration diagram and state machine diagram. The possible communications among objects shown in a collaboration diagram should be consistent with the dynamic behavior represented in the state machine diagram. By translating collaboration diagrams into sets of Büchi automata, which is the form of property to be checked against the model, this approach cleverly checks the consistencies between the two diagrams.

Jussila *et al.* [JDJ+06] (2006) provide an approach to translate UML state machines into PROMELA. This approach considers multiple objects interacting with each other. The translation is based on a formally defined semantics of UML state machines. It supports initial and choice pseudostates as well as deferred and completion events. It further provides an action language, a subset of the Jumbala [Dub06] action language, that is used to specify guard constraints and the effects of transitions of a UML state machine. The authors implemented a tool called PROCO, that takes a UML model in the form of XMI files and outputs a PROMELA model.

Carlsson and Johansson [ML09] (2009) have designed a prototype tool to link Spin with RSARTE, a modeling tool for UML diagrams. Their work focuses on all kinds of RT-UML diagrams, i.e., UML diagrams related with real time features. As part of UML, state machines are also translated into PROMELA in their approach. Since their work is not aiming at model checking of UML state machines, it does not provide detailed discussions about each feature of UML state machines, but discusses the communications between different objects.

[4]The difference between interrupt and stop relies in the fact that interrupt means a temporary stop that needs to be resumed afterwards, whereas stop means a permanent stop without resuming.

Approaches Based on SMV and its Variants

Gihwon [Kwo00] (2000) first provides a formal semantics for UML statecharts by rule-rewriting systems, and provides a translation approach from the formalized semantics to the SMV model checker. No detailed implementation is discussed in this papers.

Compton *et al.* [CGHS00] (2000) provides another approach which uses SMV as the back end model checker to automatically verify UML state machines. The work first translates UML state machines into ASMs, which has been discussed before. Then an SMV model checker is invoked to verify the SMV specification of a UML state machine. This approach is different from the other translations approaches in the sense that it does not provide a direct translation from UML state machines to the input language of a model checker, but conducts a 2-level translation approach. That is, UML state machines are first translated into ASMs, and model checking is conducted relying on a translation tool from ASMs to SMV [GDC00].

Lam and Padget [LP04] (2004) propose a symbolic encoding of UML statecharts, and invoke NuSMV to perform the model checking. Their work adopts a three-step procedure and uses ϕ-calculus as an intermediate format for the translation. They have implemented the translator from UML statecharts to $\phi-$calculus, and claim that the implementation of a translator from $\phi-$calculus to the input language of NuSMV was ongoing (although we did not find any later updates on this).

Encarnación *et al.* [BBSCdlF05] (2005) also provide a translation from UML diagrams to the input language of SMV model checker. Instead of focusing on just UML state machines, this work focuses on the collaborations of different UML diagrams such as class diagrams, state machine diagrams and activity diagrams. This paper does not describe the detailed translation rules, but illustrates their translation procedure with an ATM machine example. Noticing that high-level model designers are unfamiliar with LTL and CTL properties which are used by model checkers, the authors also provide some aid in the form of ask and answer questions to aid the property writing.

Dubrovin and Junttila [DJ07] (2007) first provide a symbolic encoding for a UML state machine, which has been discussed in Section 5. Then they perform a translation from the defined semantics to the input language of the NuSMV [CCG$^+$02] model checker. The detailed translation steps are not discussed in the paper, but an implementation SMUML [SMU] has been provided, and some experiment results are reported in their paper.

Approaches Based on Other Model Checkers

Gnesi and Latella *et al.* [GLM99] (1999) proposed another translation approach, which is also based on the formalization of UML state machines in their early work [DIM99]. The translation is from a hierarchical automaton into a semantic automaton (LTS), which needs to be further translated into the FC2 format, which is the standard input format to Jack [BGL94].

Traoré [Tra00] (2000) and Aredo [Are00] (2000) propose to translate UML state mcahines into PVS (Prototype Verification System) [PVS], which is a specification language integrated with verification tools capable of doing theorem proving, well-formedness checking and model checking.

Alexander *et al.* [KMR02] (2002) presented an approach to translate timed UML state machines into timed automata which is used by the UPPAAL Model Checker. But the translation is not based on a formal semantics of timed UML state machines. Event queue and UML state machine are separately modeled by timed automata and the communication is modeled with a channel. This approach is implemented in a prototype tool, HUGO/RT [hug12], which can verify whether scenarios specified by ML collaborations with time constraints are consistent with the corresponding set of timed UML state machines.

Ng and Butler [NB02, NB03] (2002) proposed to translate UML state machines into CSP and utilize the FDR model checker to proceed with the model checking procedure. Due to the differences between CSP and UML state machines, some features of UML state machines, such as the priority mechanism, cannot be modeled.

Hansen *et al.* [HKL$^+$10] (2010) use another model checker, mCRL2 [mCR12], to perform model checking tasks. Their work translates xUML into mCRL2 specifications. Note that it translates both class diagram and state machines into mCRL2.

Zhang and Liu [ZL10] (2010) provide an approach which translates UML state machines into CSP#, an extension of the CSP language, which serves as the input modeling language of PAT [SLDP09]. Different from other translation approaches, the transformation is not based on a pre-defined formal semantics, but directly from the meaning of each UML state machine construct. An implementation of the translator was done and experiment results of the verification of UML state machines with PAT [SLDP09] were presented.

4.4 Summary

The translation approaches aim at utilizing the automatic verification ability of different model checkers. So the advantage of these approaches is that most of them will provide the translation rules as well as the implementation of those rules in the corresponding model checkers. But we notice that translation-based approaches suffer from the following defects:

1. Due to the semantic gaps, it may be hard to translate some syntactic features of UML state machines, introducing sometimes additional but

undesired behaviors. For example in [ZL10], extra events have to be added to each process so as to model exit behaviors of orthogonal composite states.

2. For the verification, translation approaches heavily depend on the tool support of the target formal languages. Furthermore, the additional behaviors introduced during the translation may significantly slow down the verification; and optimizations and reduction techniques (like partial order reduction) may not apply in order to preserve the semantics of the original model.

3. Lastly, when a counterexample is found by the verification tool, it is hard to map it to the original state machine execution, especially when state space reduction techniques are used. Following these remarks, we believe that a direct implementation based on an operational semantics may solve the problem.

We have provided a comparison on the supported features[5] of the surveyed approaches in Table 4.4. The symbol "$\surd$" denotes the fact that the feature is supported, "$\times$" means the feature is not supported, "$\circ$" means the featured is discussed in the paper, but is not thoroughly solved. For example, for "conflict/priority", some works considered conflict among enabled transitions, but did not discuss conflict due to deferred events. In this case, we regard the features to be partially supported.

We can conclude from the table that in the translation based approaches, time, submachine state, entry/exit pseudostate and junction pseudostate are the least supported features. Less than 5 out of all the surveyed approaches support these features. No approach using ASM as target language supports choice or time features. But they almost all support orthogonal composite state, completion event and entry/exit behaviors. All approaches use Petri net as target language do not support priority mechanism, defer and completion events. Seen from Table 4.4 , we can notice that only 5 surveyed translation approach focus on UML2.x state machine specifications and only one tool was developed based on the approach.

[5]For space consideration, we remove the features that are commonly supported by all approaches, such as simple state, transitions, initial pseudostate, etc.

Work	Multiple charts	Conflict (priority)	Time	Entry/exit behaviors	States		Pseudostates					Events		
					orthogonal	submachine	fork/join	junction	choice	history	entry/exit	defer	completion	call
[BCR00]	×	√	×	√	○	×	×	×	×	√	×	√	√	×
[BCR03]	×	√	×	√	√	×	×	×	×	√	×	√	√	×
[Jür02]	√	×	×	√	√	×	×	×	×	×	×	×	○	√
[CGHS00]	×	×	×	×	√	×	×	×	×	×	×	×	√	×
[JEJ04]	×	√	×	√	√	×	√	√	×	√	×	√	√	×
[Gom00]	√	×	×	×	×	×	×	×	×	×	×	×	×	√
[BP01]	√	×	×	×	×	×	×	×	×	×	×	×	×	√
[CKZ11]	×	×	×	×	√	×	×	×	×	×	×	×	×	×
[ACK12]	×	×	×	√	×	×	×	×	○	√	×	×	×	×
[TZ05]	×	×	√	×	○	×	√	√	√	×	×	×	×	×
[LMM99]	×	○	×	×	√	×	×	×	×	×	×	×	×	√
[JDJ+06]	√	√	×	×	√	×	×	×	√	×	×	√	√	×
[SKM01]	√	√	×	√	√[6]	×	√	×	√	×	×	×	√	√
[ML09]	√	×	√	×	×	×	×	×	×	×	×	×	×	×
[Kwo00]	×	○	×	×	√	×	×	×	×	×	×	×	×	×
[DJ07]	√	×	×	×	√	×	×	×	√	×	×	√	√	×
[LP04]	√	√	×	×	×	×	×	×	×	×	×	×	×	×
[ZL10]	×	×	×	√	√	√	√	×	×	√	√	×	√	×
[KMR02]	×	×	×	√	×	×	×	×	√	×	×	×	√	×
[Are00]	×	×	×	√	√	√	√	√	√	√	√	×	×	√
[Tra00]	×	√	√	×	√	×	√	√	√	√	×	×	√	√

Table 1: UML state machines features supported by each translation approach

Work	Target Language	Verification type	Tool developed	UML version
[BCR00]	Abstract state machines	×	×	1.3
[BCR03]	Abstract state machines	×	×	1.3
[Jür02]	Abstract state machines	–	×	1.4
[CGHS00]	Abstract state machines	Model Checking	√	<= 1.3
[JEJ04]	GDTL		√ (Moses)	1.5
[Gom00]	Colored Petri nets	deadlock and statistical analysis	√	<= 1.3
[BP01]	High-level Petri nets	model checking	×[7]	–
[CKZ11]	Hierarchical Colored Petri nets	Model checking	√(CPN-AMI)	–
[ACK12]	Colored Petri nets	–	×	2.2
[TZ05]	Stochastic Petri nets	×	×	2.0
[LMM99]	PROMELA	model checking	×	1.1
[JDJ⁺06]	PROMELA	model checking	√ (PROCO)	1.4
[SKM01]	PROMELA	model checking	√(HUGO)	1.4
[ML09]	PROMELA	model checking	√(RSARTE)	–
[Kwo00]	SMV input language	model checking	×	1.3
[DJ07]	NuSMV input language	model checking	√	–
[BBSCdlF05]	SMV input language	model checking	√	< 1.3
[LP05]	NuSMV input language	model checking	√(*SC2PiCal*)[8]	1.5
[ZL10]	CSP#	model checking	√	2.2
[KMR02]	Timed automata	model checking	√ (HUGO/RT)	1.4
[HKL⁺10]	mCRL2	model checking	×	2.2
[NB03]	CSP	model checking	√	1.4
[Are00]	PVS	Model Checking, theory proving	×	1.3
[Tra00]	PVS	model checking	√(*PrUDE*)	1.3

Table 2: Summary of translation approaches

5 Approaches Providing Operational Semantics for UML State Machines

Different from the translation-based approaches, another kind of approaches directly provides operational semantics to UML state machines, usually by defining inference rules. These approaches are of general purpose, i.e., various verification techniques can be conducted based on the operational semantics. The benefit of this kind of approaches are (1) they do not rely on the target formal languages, thus no redundancies are introduced, and (2) the semantic steps defined in the operational semantics directly coincide with UML state machines semantic step, i.e., the Run To Completion (RTC) step. A major benefit of these approaches is that state space reduction techniques can now be applied. Moreover, approaches in this category usually adopt Labeled Transition Systems (LTS) as the semantic model, which is the common semantic model for concurrent systems; furthermore, it is also well supported by many model checkers.

In this section, we categorize this kind of approaches based on the formal formats used as semantic domains. Finally, we summarize the approaches in Section 5.3.

5.1 Semantics Defined Using LTS as Semantics Model

In this section, we are going to discuss works which formalize UML state machines into the Labeled Transition Systems (LTS). We further categorize these works based on the syntax model they adopt.

Approaches using EHA as syntax model

Hierarchical Automata require a strict hierarchical structure. The existence of interlevel transitions and local transitions break the hierarchical structure. EHA extends the Hierarchical Automata to deal with interlevel transitions, i.e., an interlevel transition which crosses multiple states will be assigned to the outermost Sequence Automaton.

Latella *et al.* [DIM99] (1999) are among the pioneers who begin to focus on formalizing UML statecharts (instead of other variants of statecharts) semantics. The semantic model their formalization adopted is Kripke structure. They use a slightly modified variant of Extended Hierarchical Automata (EHA) as an intermediate model, and map the UML-statecharts into an EHA. The hierarchical structure of UML statecharts and EHA make the translation structured and intuitive. Then they define the operational semantics for EHA in the model of Kripke structures.

A following work by Gnesi Latella and Massink [GLM02] extends their previous work [DIM99] to include multicharts, i.e., multiple UML state machines communicating asynchronously. The work also discusses how to incorporate the semantics into the model checking tool–JACK.

This approach covers a quite restricted subset of UML state machine structures: no pseudostates (excepted the initial pseudostate) are considered, no actions associated with states (i.e., entry/exit/do actions, and deferred events) are considered, and the triggering events are restricted to the signal and call events without parameters. Since states in different hierarchies cannot appear in a single Sequence Automaton, it is hard [9] to express transitions which have their source and target states in different hierarchies.

Dong *et al.* [WJXZC01] (2001) further extend EHA to support more features such as entry/exit actions or parameters in actions, and provide a formal semantics for a subset of UML statecharts based on this EHA model. The authors discuss the findings on the cost of solving conflicts introduced by concurrent composite states, and the importance of modeling with multiple objects instead of modeling them with concurrent regions within one UML state machine. They consider the non-determinism caused by multiple concurrent state machines, which was not captured by [DIM99].

Although approaches using EHA as an intermediate representation do not "directly" provide the operational semantics, EHA still resemble UML state machines in the hierarchical structure, and operational semantics for EHA (which is not a formal modeling language as, e.g., CSP) are provided. For this reason we consider this kind of approaches as directly providing operational semantics.

Approaches using terms as syntax model

There are also works which uses terms or tuples as syntax model.

Von der Beek [Von02] (2002) also formalize a partial set of UML statecharts, partially based on the work proposed in [DIM99]. But it supports some more features such as history mechanisms, entry and exit actions compared to [DIM99]. The syntax used in this work is called an UML-statechart term, which is inductively defined on three kinds of terms, viz., Basic term, Or-term and And-term. All of them contain basic information about a state such as a unique ID, entry and exit actions, and sub-terms (for Or-term and And-term) which contain the hierarchical information of a UML state machine. UML-statechart terms basically represent static information about UML statechart vertices. Inter-level transitions are captured by explicitly specifying source restrictions and target determinators in an Or-term; this notation follows the idea of Latella et al [DIM99].

The dynamic behavior of UML-statechart is represented by configurations. The auxiliary semantics is defined as a mapping from a UML statechart to an LTS. Each state in the LTS is a UML-statechart term. A se-

[9]Actually transitions between states which have direct hierarchical relations not supported in [DIM99]

mantic transition is defined to proceed a single input event. Last, complete semantics is given based on auxiliary semantics and a Kripke structure. Instead of representing a UML statechart into an EHA, Von der Beek [Von02] chooses to use a UML-statechart term as the syntax domain of a UML statechart.

Kwon proposed another approach [Kwo00] which utilizes Kripke Structure as the semantic domain and aimed at model checking UML statechart. Similar to [Von02], Kwon uses terms as the syntax domain of UML statechart, which represent state hierarchy in the form of subterms as a field in a term. But Kwon [Kwo00] use the conditional rewrite rules to represent the transition relation in a UML statechart (while Breeck [Von02] explicitly defined five SOS rules). Then the semantics of UML statechart is defined as a Kripke Structure. This paper [Kwo00] also provides a translation from the defined Kripke Structure to the input language of the SMV model checker, which we have discussed in Section 4.3.

Eshuis and Wieringa [EW00] (2000) also utilize LTS as a semantic model to provide an operational semantics for UML statecharts. The syntax model used is tuple. This work focuses more on the communication and timing aspect of UML statecharts. It also considers object construction and destruction, which is not always considered by the other approaches. The approach define UML statechart syntax as sets with defined functions and the semantic model is LTS. The approach also define an action language, which includes assignment, object creation/destruction, sequence operations, signal sending operation and time expressions.

Liu *et al.* [LLA+13] (2013) is the most recent approach which provides formal semantics for UML 2.4.1 state machines. The approach covers all the features of UML state machines except for time events. Moreover, the syntax is defined totally in accordance with UML state machines specifications, which makes the approach extends easily to future changes such as refinement. The semantic domain is LTS. The approach also considered asynchronous/synchronous communications between objects.

Reggio *et al.* [RACH00] (2000) propose to use LTS as semantic model to provide a formal semantics for UML state machines. This work considers an early version (1.3) of UML specifications and discusses some inconsistencies and ambiguities in the specification. The work does not provide a clear syntax model and UML state machines are not represented formally. But it does discuss in details the event dispatching and merging operation on event pools.

5.2 Approaches using other semantic models

Lilius and Paltor [LP99c] (1999) provide an abstract syntax and semantics for a subset of UML state machines. This work uses terms as syntax model and considers most features of UML state machines. Although it does not define a clear semantic model, this work formalizes the RTC step semantics into an algorithm. The algorithm is in a high abstract level and many concepts such as history pseudostates and completion events are described in a rather informal manner. But the procedure of an RTC step is properly described.

Some features such as join, fork, junction, choice vertices unspecified and instead, claiming that these pseudostates can be replaced with extra transitions. This may be achievable using a different modeling strategy, but the expressiveness of the modeling language is weakened.

Damm *et al.* [DJPV03] (2003) provide a formal semantics for a kernel set of UML in order to model real-time applications, including static and dynamic aspects of the UML models. The formalization contains two steps. Firstly, real-time UML (rtUML) is represented in terms of the kernel subset of real-time UML (krtUML[10]). Then formal semantics of krtUML is provided. This approach provides a self-defined action language, which supports object creation/destruction, assignment and operation calls. The semantic domain is called Symbolic Transition System (STS), which conducts type-consistent first-order predicate over a set of variables. UML state machines semantics is just a component of the krtUML semantics. But the work provides a good reference for communications between different objects, such as event dispatching and handling.

Fecher and Schönborn [FS07] use core state machine, which is a subset of UML state machines, as the semantic domain for UML state machines. A core state machine is a 7-tuple including states, do actions, deferred events, transitions, initial state, set of variables and initial variable assignment. History is explicitly described by a mapping from a region to its direct substate. The work firstly formalizes both syntax and semantics of the core state machine. This paper considers more UML state machine features. 14 configuration steps are provided on the core state machine, which form the dynamic semantics of it. But there is no formal semantic model; hence, the provided semantics is not complete since just transition rules are provided. The RTC steps of a UML state machines are not defined. The transformation steps from UML state machines to core state machine is also provided. But the steps are not formally stated, only natural language descriptions with example illustrations are given. Moreover, the translation is very complex since a lot of auxiliary vertexes need to be added, such as enter/exit vertex. Both defects may make it unfeasible for automatic tool development.

Jens *et al.* [SdRFK05] (2005) provide a very comprehensive analysis about UML 2.0 behavioral state machines, including discussions about detailed semantics of each feature and the ambiguity statements. This approach covers almost all features of UML2.0 state machines, except for junction and choice vertices, which are considered as syntactic sugar and are said

10

to be easily represented by separate transitions. Termination pseudostates and completion events are also left out unconsidered. The syntax model of UML state machine is the tuple, which captures the components of each construct. Many auxiliary functions are defined to capture the execution of an RTC step, such as collecting all actions generated during transition execution and put them in the event pool. This work contributes more on the analysis of ambiguities in UML 2.0 specifications and the detailed discussion about the semantics of UML 2.0. In term of the formal semantics they have defined, though achieves a high coverage, does not have a formal semantic model. In a separate paper [FSKdR05], the same authors discuss 29 undeclared points in UML2.0 state machine specifications.

Jori and Tommi [DJ07] (2007) provide a symbolic encoding for UML state machines. The approach can be regarded as either a translation approach or an operational semantics in the domain of first-order logic. The approach discusses event dispatching mechanisms, multi-object communication (asynchronous) as well as choice pseudostate, which are often left out in other approaches. But some commonly considered constructs, such as history pseudostate, is not included in their formalization.

5.3 Summary

We provide here a detailed discussion about approaches which provide formal operational semantics for UML state machines. We are focusing on those approaches which are related to automatic verification of UML state machines. Among all the surveyed approaches, LTS-based approaches are most related to automatic verification, specifically model checking, but always support less features compared to other approaches due to the unstructured feature of UML state machines.

Table 3 provides the syntax and semantic modals used by of all the surveyed approaches in this section.

Table 4 summarizes the supported features of the surveyed approaches. $\surd$ means the feature is supported, $\times$ means the feature is not supported, $\circ$ means the featured is discussed in the paper, but is not thoroughly solved; for example, for "conflict/priority", some works consider conflict among enabled transitions, but do not discuss conflicts due to deferred events. In this case, we regard the features to be partially supported. $\oplus$ represents the situation where the corresponding feature is not directly formalized, but it is represented with the formally defined features in the semantics.

From the tables we can conclude that early works (works before 2002) tend to use LTS or variance of LTS as semantic models, but support less features (especially pseudostates, priority and event pool mechanisms) compared to later approaches.

Work	Syntax domain	Action language	Semantic domain	Event queue	UML version	Leads to tol implementation
[DIM99]	EHA	–	Kripke Structure	×	1.1	JACK
[WJXZC01]	EHA	self-defined	Kripke Structure	×	1.1	×
[Von02]	term	–	Kripke Structure	×	1.4	×
[Kwo00]	term	abstract notation	Kripke Structure	×	1.3	×
[EW00]	set with functions	self-defined	LTS	×	1.3	×
[?]	–	–	LTS	√	<= 1.3	×
[LLA+13]	tuple	–	LTS	√	2.4.1	USMMC
[LP99c]	term	–	–	×	1.3	×
[DJPV03]	krtUML	self-defined	STS	√	1.4	×
[FS07]	core state machine	–	–	×	2.0	×
[SdRFK05]	tuple	–	–	×	2.0	×
[DJ07]	tuple	abstract notation	first-order logic	√	2.0	×

Table 3: Syntax and Semantic domains of surveyed operational semantics

Work	Multiple charts	Conflict (priority)	Time	Entry/exit behaviors	States		Pseudostates					Events		
					orthogonal	submachine	fork/join	junction	choice	history	entry/exit	defer	completion	call
[DIM99]	×	○	×	×	√	×	×	×	×	×	×	×	×	√
[WJXZC01]	√	○	×	√	√	×	×	×	×	×	×	×	√	–
[Von02]	×	○	×	√	√	×	×	×	×	√	×	×	×	×
[Kwo00]	×	○	×	×	√	×	×	×	×	×	×	×	×	×
[EW00]	×	○	×	√	√	×	×	×	×	×	×	×	√	√
[?]	√	○	√	×	×	×	×	√	×	×	×	√	×	√
[LLA+13]	√	√	×	√	√	√	√	√	√	√	√	√	√	√
[LP99c]	×	√	√	√	√	×	×	×	×	√	×	√	√	√
[DJPV03]	√	×	√	×	×	×	×	×	×	×	×	×	×	×
[FS07]	×	√	√	×	⊕	√	⊕	⊕	√	⊕	⊕	√	√	×
[SdRFK05]	×	√	×	√	√	×	√	×	×	√	×	√	×	×
[DJ07]	√	×	×	×	√	×	×	×	√	×	×	√	√	×

Table 4: UML state machines features supported by the direct formalization approaches

6 Tool Support

In this section, we discuss tool support for modeling and verifying UML state machines. There are both commercial and academic tool supports for UML modeling. To the best of our knowledge, current commercial tools only support the design/graphical editing of UML models – hence, we discard them here. Some academic prototype tools were developed based on the translation approaches, which aim at automatically verifying UML state machines, as we surveyed in Section 4. We are going to survey these tools in this section and concentrate mainly on those automatic verification tools.

vUML Lilius and Porres [LP99a] (1999) report a tool vUML which aims at automatically verifying UML model behaviors specified by UML statecharts diagrams. This tool utilizes the Spin model checker as a backend to perform model checking and creates a UML sequence diagram according to the counterexample provided by Spin. The formal semantics is defined in [LP99c]. They also conduct a case study with the production cell example in [LP99b].

vUML aims at checking collaborations of UML models instead of a single UML state machine. So the PROMELA specification for a UML model is generated from class diagrams, statecharts and collaboration diagrams, where each UML class is mapped into a PROMELA process-type, each UML object is mapped into a PROMELA process and each link in the collaboration diagram is converted into a PROMELA channel for objects to exchange messages. vUML provides an event generator to emulate external events without parameters (close models) and removes external events carrying parameters in order to avoid state space explosion.

vUML can check the following properties: deadlock, livelock, reaching an invalid state, violating a constraint on an object, sending an event to a terminated object, overrunning the input queue of an object, overrunning the deferred event queue.

In order to verify those properties, two extra stereotypes are introduces, namely <<invalid>> and <<progress>>. UML model developers need to add those extra stereotypes into their models in order to use vUML to model check safety and liveness properties. Constraints also act as a way to specify properties (invariants over attributes and states). In order to verify LTL formulas with UML, the users need to understand the PROMELA model to come up with a proper LTL formula. .

JACK Gnesi *et al.* [GLM99] (1999) provide an algorithm to support direct model checking UML statecharts based on the formal semantics they have defined in [DIM99]. The implementation is based on the tool set JACK [**?**], which is an environment based on the use of process algebras, automata and temporal logic formalism, and supports many phases of the system development process by integrating different editing tools and verification tools. Different components of the JACK tool set communicate with the FC2 format. There is a model checking tool in the JACK tool set named AMC, which supports ACTL model checking. The system should be translated into the FC2 format first in order to utilize the AMC component. The users also need to specify their own ACTL property according to the model. This requires users to have a knowledge of model checking, the underlying model as well as temporal logic formulas.

HUGO Alexander *et al.* [KMR02] (2002) developed a tool called HUGO, which contains three components, each of which supports one functionality. The first is the code generation component, which is used to automatically generate Java code from UML state machines. The second is the model checking component, used to verify the consistency of UML state machines against specifications expressed as collaboration or sequence diagrams. HUGO can support LTL model checking provided knowledge of the underlying model checker(Spin) and the structure of the translation. The third component is a back end for the real-time model checker UPPAAL. In [KM02], they report an improved implementation of HUGO where they discuss code generation and state reductions in the PROMELA code. HUGO also adopts the translation approach where UML state machines are translated into PROMELA models and utilizes the Spin model checker to do the verification (the same applies to the real-time components where the UML state machines are translated into the UPPAAL modeling language).

ASM-based Verification Tools Shen *et al.* [SCH02] (2002) introduce a tool based on an ASM model checker (which is based on theSMV model checker). The semantics they adopt is defined in [CGHS00] (UML 1.3 or earlier, though not explicitly mentioned in their technical report). This tool set supports both static and dynamic checks of a UML diagrams. For static aspects, syntax as well as well-formedness rules given by OCL can be checked. Static views in UML, such as object diagram and class diagram can also be transformed into ASM and checked. For the dynamic aspects, UML state machines diagrams are transformed into ASM models, and an ASM model checker is invoked to do the model checking. This tool takes UML diagrams specified in the XMI format as input and outputs a counterexample in the form given by the SMV model checker (since the dynamic checking component of this toolset is based on the smv model checker). The counterexample trace can be fed to their analysis tool, which will analyze the error trace and produce some UML diagrams such as sequence diagrams or a collaboration diagram to the users. Details about the tool are described in [CGHS00] and details about the transformation procedures are discussed in [BCR00].

TABU Beato *et al.* [BBSCdlF05] (2005) introduce a tool called TABU (Tool for the Active Behavior of UML). TABU takes UML diagrams (activity and state diagrams) in the form of XMI as input, automatically translates it into an `.smv` representation (the input format of SMV model checker) and calls the Cadence smv model checker to verify the UML model. In addition, TABU also provides assistant for writing (LTL/CTL) properties to verify against the model. This feature makes the underlying model and the translation procedure transparent to the users, and solves the problem faced by vUML [LP99c] to some extent.

This tool is attractive in the sense that it deals with UML 2.0 specifications, which is much closer to recent UML standards. The translation covers most UML features (though not described in details in their paper) except for synchronization states, events with parameters and dynamic creation and destruction of objects. It also provides guides in writing properties. But the counterexample is given in the form of the input format of SMV model checker, which is not intuitive for model designers to map to their models.

PROCO PROCO [PROb] (2006) translates a UML state machine in the form of XMI forms into PROMELA, the input language of Spin model checker is discussed in [JDJ$^+$06]. No details are discussed in that paper, but the paper reports the bugs found by the tool, which show its effectiveness.

UML-B State Machine Animation Tool Recently, UML-B state machine Animation [UMLb] is developed as a plugin in the Rodin project [UMLb]. It is able to translate a UML-B diagram into Event-B representations and utilize ProB [Proa], a plug-in of the Rodin project, to perform the simulation and model checking tasks. The tool is available and can be installed from the Rodin platform.

USM^{2}C Liu *et al.* [LLA$^+$13] (2013) report a tool called USM^{2}C, that implements the operational semantics defined in [LLA$^+$13]. The tool supports most features of UML state machines and is capable of model checking various properties, such as LTL, safety and deadlock-checking properties.

Summary

We notice that all the available tools, except for USM^{2}C, just provide a front-end supporting translation from UML state machines to languages of model checkers. Such a translation will introduce extra cost for the verification procedure. Due to the limitation of input languages to model checkers, the complex semantics of UML state machine cannot be fully supported. There are also problems for the utility of those tool, e.g., it is hard to map

Reference	Tool name	Underlying model checker	model exchange format	Availability
[LP99a]	vUML	Spin	xmi	×
[KMR02]	HUGO/RT	Spin/UPPAAL	xmi	√
[SCH02]	ASM based	smv	xmi	×
[BBSCdlF05]	TABU	smv	xmi	×
[GLM99]	JACK	AMC	FC2	×
[JDJ+06]	PROCO	Spin	xmi	√
[UMLb]	UML-B	ProB	xmi	√
[LLA+13]	USM^{2}C	PAT	xmi	√

Table 5: Status of the tools

the found vulnerabilities to the original model. The informal translation procedure cannot guarantee the soundness of the obtained model either. Table 6 summarizes the basic information of the surveyed tools.

7 Conclusion

In this paper, we provide a thorough survey of approaches aiming at giving a formal semantics UML state machines, thus enabling their automated verification. We categorized the approaches into two major groups, viz., the translation approaches and those directly providing operational semantics. In each group, we also provide comparisons of the surveyed approaches on dimensions such as UML version, feature coverage, communication aspect, event pool mechanism, consistency with other UML diagrams and tool supports.

Bibliography

[ACK12] Étienne André, Christine Choppy, and Kais Klai. Formalizing non-concurrent UML state machines using colored Petri nets. *ACM SIGSOFT Software Engineering Notes*, 37(4):1–8, 2012. Proceedings of the 5th International workshop UML and Formal Methods (UML&FM). 8, 14, 15

[Are00] Demissie B. Aredo. Semantics of uml statecharts in pvs. In *In Proc. of the 12th Nordic Workshop on Programming Theory (NWPT00)*, 2000. 12, 14, 15

[BBSCdlF05] M. Encarnación Beato, Manuel Barrio-Solórzano, Carlos E. Cuesta, and Pablo de la Fuente. UML automatic verification tool with formal methods. *Electron. Notes Theor. Comput. Sci.*, 127(4):3–16, April 2005. 11, 15, 24, 25

[BCR00] E. Börger, A. Cavarra, and E. Riccobene. Modeling the dynamics of uml state machines. In *Abstract State Machines-Theory and Applications*, pages 167–186. Springer, 2000. 6, 7, 14, 15, 23

[BCR03] Egon Börger, Alessandra Cavarra, and Elvinia Riccobene. Modeling the meaning of transitions from and to concurrent states in UML state machines. In *Proceedings of the 2003 ACM symposium on Applied computing*, SAC '03, pages 1086–1091, New York, NY, USA, 2003. ACM. 6, 7, 14, 15

[BCR04] Egon Brger, Alessandra Cavarra, and Elvinia Riccobene. On formalizing UML state machines using asms. *Information Software Technology*, 46(5):287, 2004. 6

[BGL94] Amar Bouali, Stefania Gnesi, and Salvatore Larosa. The integration project for the jack environment. Technical report, Amsterdam, The Netherlands, The Netherlands, 1994. 12

[BKKS08] Jörg Beckers, Daniel Klünder, Stefan Kowalewski, and Bastian Schlich. Direct support for model checking abstract state machines by utilizing simulation. In Egon Brger, Michael Butler, JonathanP. Bowen, and Paul Boca, editors, *Abstract State Machines, B and Z*, volume 5238 of *Lecture Notes in Computer Science*, pages 112–124. Springer Berlin Heidelberg, 2008. 6

[BP01] L. Baresi and M. Pezze. On formalizing uml with high-level petri nets. *Concurrent Object-Oriented Programming and Petri Nets*, pages 276–304, 2001. 8, 14, 15

[CCG^{+}02] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, pages 359–364, London, UK, UK, 2002. Springer-Verlag. 2, 11

[CD05a] Michelle L. Crane and Juergen Dingel. Uml vs. classical vs. rhapsody statecharts: Not all models are created equal. In *Proc. 8th International Conf. on Model Driven Engineering Languages and Systems, Oct*, pages 2–7, 2005. 2

[CD05b] M.L. Crane and J. Dingel. On the semantics of UML state machines: Categorization and comparision. In *In Technical Report 2005-501, School of Computing, Queens*. Citeseer, 2005. 2, 3, 4

[CGHS00] Kevin Compton, Yuri Gurevich, James Huggins, and Wuwei Shen. An automatic verification tool for UML. Technical Report CSE-TR-423-00, University of Michigan, 2000. 6, 11, 14, 15, 23

[CKZ11] C. Choppy, K. Klai, and H. Zidani. Formal verification of uml state diagrams: a petri net based approach. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011. 8, 9, 14, 15

[DIM99] Latella Diego, Majzik Istvan, and Massink Mieke. Towards a formal operational semantics of UML statechart diagrams. In *Proceedings of the IFIP TC6/WG6*, volume 1, page 465, 1999. 9, 10, 12, 16, 17, 21, 22

[DJ07] Jori Dubrovin and Tommi Junttila. Symbolic model checking of hierarchical UML state machines. In *Application of Concurrency to System Design, 2008. ACSD*, number B23, pages 108–117, Espoo, Finland, December 2007. 11, 14, 15, 20, 21

[DJPV03] Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. Understanding uml: A formal semantics of concurrency and communication in real-time uml. In FrankS. Boer, MarcelloM. Bonsangue, Susanne Graf, and Willem-Paul Roever, editors, *Formal Methods for Components and Objects*, volume 2852 of *Lecture Notes in Computer Science*, pages 71–98. Springer Berlin Heidelberg, 2003. 19, 21

[Dub06] Jori Dubrovin. Jumbala — an action language for UML state machines. Technical report, Helsinki University of Technology, Laboratory for Theoretical Computer Science, 2006. 10

[ER03] Brger Egon and Strk Robert. *Abstract State Machines A Method for High-Level System Design and Analysis*. Springer-Verlag USA., 2003. 5, 6

[EW00] Rik Eshuis and Roel Wieringa. Requirements-level semantics for uml statecharts. In ScottF. Smith and CarolynL. Talcott, editors, *Formal Methods for Open Object-Based Distributed Systems IV*, volume 49 of *IFIP Advances in Information and Communication Technology*, pages 121–140. Springer US, 2000. 18, 21

[FS07] H. Fecher and J. Schönborn. UML 2.0 state machines: Complete formal semantics via core state machine. *Formal Methods: Applications and Technology*, pages 244–260, 2007. 19, 21

[FSKdR05] H. Fecher, J. Schönborn, M. Kyas, and W.P. de Roever. 29 new unclarities in the semantics of uml 2.0 state machines. *Formal Methods and Software Engineering*, pages 52–65, 2005. 20

[GDC00] Kirsten Winter Giuseppe Del Castillo. Model checking support for the asm high-level language. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346, 2000. 6, 11

[GLM99] S. Gnesi, D. Latella, and M. Massink. Model checking UML statechart diagrams using jack. In *High-Assurance Systems Engineering, 1999. Proceedings. 4th IEEE International Symposium on*, pages 46–55. IEEE, 1999. 11, 22, 25

[GLM02] Stefania Gnesi, Diego Latella, and Mieke Massink. Modular semantics for a uml statechart diagrams kernel and its extension to multicharts and branching time model-checking. *Journal of Logic and Algebraic Programming*, 51(1):43–75, 2002. 16

[Gom00] H. Gomaa. Validation of dynamic behavior in uml using colored petri nets. 2000. 8, 14, 15

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987. 2, 9

[HKL$^+$10] Helle Hvid Hansen, Jeroen Ketema, Bas Luttik, MohammadReza Mousavi, and Jaco Pol van de. Towards model checking executable uml specifications in mcrl2. *Innovations in Systems and Software Engineering*, 6(1-2):83–90, March 2010. Open Access. 12, 15

[HLN$^+$90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *Software Engineering, IEEE Transactions on*, 16(4):403–414, 1990. 9

[HN96] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996. 2, 9

[hug12] Hugo/rt website.[online] `http://www.pst.informatik.uni-muenchen.de/projekte/hugo/`, Nov 2012. 12

[JDJ$^+$06] Toni Jussila, Jori Dubrovin, Tommi Junttila, Timo Latvala, , and Ivan Porres. Model checking dynamic and hierarchical uml state machines. *Proc. MoDeV2a: Model Development, Validation and Verification*, pages 94–110, 2006. 10, 14, 15, 24, 25

[JEJ04] Y. Jin, R. Esser, and J.W. Janneck. A method for describing the syntax and semantics of UML statecharts. *Software and Systems Modeling*, 3(2):150–163, 2004. 7, 14, 15

[JK98] J.W. Janneck and P.W. Kutter. *Mapping automata: simple abstract state machines.* TIK-Report. Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology Zürich (ETH), 1998. 7

[JK09] K. Jensen and L.M. Kristensen. *Coloured Petri nets: modeling and validation of concurrent systems.* Springer-Verlag New York Inc, 2009. 5, 8

[Jür02] Jan Jürjens. A UML statecharts semantics with message-passing. In *Proceedings of the 2002 ACM symposium on Applied computing*, SAC '02, pages 1009–1013, New York, NY, USA, 2002. ACM. 7, 14, 15

[KM02] A. Knapp and S. Merz. Model checking and code generation for UML state machines and collaborations. In *Proceedings of 5th Workshop on Tools for System Design and Verification, Technical Report*, volume 11, pages 59–64, 2002. 23

[KMR02] Alexander Knapp, Stephan Merz, and Christopher Rauh. Model checking - timed uml state machines and collaborations. In *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2*, FTRTFT '02, pages 395–416, London, UK, UK, 2002. Springer-Verlag. 12, 14, 15, 23, 25

[Kwo00] Gihwon Kwon. Rewrite rules and operational semantics for model checking UML statecharts. In *Proceedings of the 3rd international conference on The unified modeling language: advancing the standard*, UML'00, pages 528–540, Berlin, Heidelberg, 2000. Springer-Verlag. 11, 14, 15, 18, 21

[LLA+13] Shuang Liu, Yang Liu, Étienne André, Christine Choppy, Jun Sun, Bimlesh Wadhwa, and Jin Song Dong. A formal semantics for the complete syntax of uml state machines with communications. In Luigia Petre and Einar Broch Johnsen, editors, *Proceedings of the 10th International Conference on Integrated Formal Methods (iFM'13)*, volume 7940 of *Lecture Notes in Computer Science*, pages 331–346. Springer, June 2013. 18, 21, 24, 25

[LMM99] Diego Latella, Istvan Majzik, and Mieke Massink. Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. *Formal Aspects of Computing*, 11(6):637–664, December 1999. 9, 14, 15

[LP99a] Johan Lilius and Ivan Porres Paltor. vUML: A tool for verifying UML models. In *Proceedings of the 14th IEEE international conference on Automated software engineering*, ASE '99, pages 255–, Washington, DC, USA, 1999. IEEE Computer Society. 22, 25

[LP99b] Johan Lilius and Iván Porres Paltor Paltor. Formalising uml state machines for model checking. In Robert France and Bernhard Rumpe, editors, *¡¡UML¿¿'99–The Unified Modeling Language*, volume 1723 of *Lecture Notes in Computer Science*, pages 430–444. Springer Berlin Heidelberg, 1999. 22

[LP99c] Johan Lilius and Ivn Porres Paltor. The semantics of UML state machines. Technical report, 1999. 18, 21, 22, 24

[LP04] Vitus S.W. Lam and Julian Padget. Symbolic model checking of uml statechart diagrams with an integrated approach. In *11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004. Proceedings.*, 2004. 11, 14

[LP05] Vitus S.W. Lam and Julian Padget. An integrated environment for communicating uml statechart diagrams. In *Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, AICCSA '05, pages 111–vii, Washington, DC, USA, 2005. IEEE Computer Society. 15

[LRS10] Mass Soldal Lund, Atle Refsdal, and Ketil Stølen. Semantics of UML models for dynamic behavior: a survey of different approaches. In *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, MBEERTS'07, pages 77–103, Berlin, Heidelberg, 2010. Springer-Verlag. 2, 3, 4

[mCR12] mCRL2, a specification language and toolset. `http://www.mcrl2.org/release/user_manual/index.html`., 2012. 12

[ML09] Carlsson Mats and Johansson Lars. Formal verification of UML-RT capsules using model checking. Master's thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Sweden, 2009. 10, 14, 15

[NB02] Muan Yong Ng and Michael Butler. Tool support for visualizing csp in uml. In Chris George and Huaikou Miao, editors, *Formal Methods and Software Engineering*, volume 2495 of *Lecture Notes in Computer Science*, pages 287–298. Springer Berlin Heidelberg, 2002. 12

[NB03] Muan Yong Ng and Michael Butler. Towards formalizing UML state diagrams in CSP. *Third IEEE International Conference on Software Engineering and Formal Methods, SEFM'03*, 0:138, 2003. 12, 15

[OMG] Object management group. `http://www.omg.org/`. 2

[PB04] S. Ramesh Purandar Bhaduri. Model checking of statechart models: Survey and research directions. *Arxiv preprint cs/0407038*, 2004. 2, 3, 9

[Proa] The prob animator and model checker, `http://www.stups.uni-duesseldorf.de/ProB/index.php5/The_ProB_Animator_and_Model_Checker`. 24

[PROb] Proco, `http://www.tcs.hut.fi/SMUML/`. 24

[Proc] The promela user menual. `http://spinroot.com/spin/Man/promela.html`. 2

[PVS] Pvs specification and verification system. `http://pvs.csl.sri.com/`. 12

[RACH00] G. Reggio, E. Astesiano, C. Choppy, and H. Hussmann. Analysing UML active classes and associated state machines - a lightweight formal approach. In *Proc. FASE 2000, number 1783 in Lecture Notes in Computer Science*, pages 127–146. Springer Verlag, 2000. 18

[SCH02] Wuwei Shen, Kevin Compton, and James Huggins. A toolset for supporting UML static and dynamic model checking. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, COMPSAC '02, pages 147–152, Washington, DC, USA, 2002. IEEE Computer Society. 23, 25

[SdRFK05] Jens Schonborn, Willem Paul de Roever, Harald Fecher, and Marcel Kyas. Formal semantics of UML 2.0 behavioral state machines. Technical report, Institute of Computer Science and Applied Mathematics, Technical Faculty, Christian-Albrechts-University of Kiel, 2005. 19, 21

[SKM01] Timm Sch"afer, Alexander Knapp, and Stephan Merz. Model checking uml state machines and collaborations. *Electronic Notes in Theoretical Computer Science*, 55(3):357–369, October 2001. 10, 14, 15

[SLDP09] J. Sun, Y. Liu, J. Dong, and J. Pang. Pat: Towards flexible verification under fairness. In *Computer Aided Verification*, pages 709–714. Springer, 2009. 12

[SMU] Symbolic methods for uml behavioural diagrams, `http://www.tcs.hut.fi/Research/Logic/SMUML.shtml`. 11

[SPI] The spin model checker `http://spinroot.com/spin/whatispin.html`. 2

[Spi00] Marc Spielmann. Model checking abstract state machines and beyond. In Yuri Gurevich, PhilippW. Kutter, Martin Odersky, and Lothar Thiele, editors, *Abstract State Machines - Theory and Applications*, volume 1912 of *Lecture Notes in Computer Science*, pages 323–340. Springer Berlin Heidelberg, 2000. 6

[Tra00] Issa Traoré. An outline of pvs semantics for uml statecharts. *Journal of Universal Computer Science*, 6:2000, 2000. 12, 14, 15

[TZ05] J. Trowitzsch and A. Zimmermann. Real-time uml state machines: An analysis approach. 2005. 8, 14, 15

[UMLa] OMG unified language superstructure specification (formal). Version 1.4, 2011-08-06. `http://www.omg.org/spec/UML/1.4/PDF/index.htm`. 6

[UMLb] UML-B statemate:achine animation, `http://wiki.event-b.org/index.php/UML-B_-_Statemachine_AnimState`. 24, 25

[UPP] The uppaal model checker. `http://www.uppaal.org/`. 5

[Von02] M. Von Der Beeck. A structured operational semantics for UML-statecharts. *Software and Systems Modeling*, 1(2):130–141, 2002. 17, 18, 21

[WJXZC01] Dong Wei, Wang Ji, Qi Xuan, and Qi Zhi-Chang. Model checking UML statecharts. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*, pages 363–370, 2001. 17, 21

[ZL10] S.J. Zhang and Y. Liu. An automatic approach to model checking uml state machines. In *Secure Software Integration and Reliability Improvement Companion (SSIRI-C), 2010 Fourth International Conference on*, pages 1–6. IEEE, 2010. 12, 13, 14, 15