A Review of Statistical Model Checking Pitfalls on Real-Time Stochastic Models*

Dimitri Bohlender[1], Harold Bruintjes[1], Sebastian Junges[1], Jens Katelaan[1], Viet Yen Nguyen[1,2], and Thomas Noll[1]

[1] Software Modeling and Verification Group, RWTH Aachen University, Germany
[2] Fraunhofer IESE, Germany

Abstract. Statistical model checking (SMC) is a technique inspired by Monte-Carlo simulation for verifying time-bounded temporal logical properties. SMC originally focused on fully stochastic models such as Markov chains, but its scope has recently been extended to cover formalisms that mix functional real-time aspects, concurrency and nondeterminism. We show by various examples using the tools UPPAAL-SMC and Modes that combining the stochastic interpretation of such models with SMC algorithms is extremely subtle. This may yield significant discrepancies in the analysis results. As these subtleties are not so obvious to the end-user, we present five semantic caveats and give a classification scheme for SMC algorithms. We argue that caution is needed and believe that the caveats and classification scheme in this paper serve as a guiding reference for thoroughly understanding them.

1 Introduction

Statistical model checking (SMC) techniques [21] have been proposed for overcoming many challenges in devising tractable numerical methods for probabilistic models. Inspired by Monte-Carlo simulation, SMC techniques simulate a random system for a particular number of times, such that statistical evidence is built up for deciding whether a property holds on the model. This method is lightweight and easily parallelizable. Drawn by their tractability, research groups have subsequently adapted them to more-expressive classes of models [17]. A particular class, which we refer to as *real-time stochastic models*, are characterized by intertwining concurrency, non-deterministic choice, real-time and probabilistic aspects into a single formalism. We consider these models as partially stochastic, as they have no common stochastic interpretation on their non-probabilistic language elements, such as concurrency. Yet, their potential applications are clear. For example, real-time stochastic models could be used for expressing safety-critical embedded systems, which are constructed for validating RAMS (reliability, availability, maintainability and safety) requirements. The probabilistic

* This work was partially supported by ESA/ESTEC (contract no. 4000107221 (HASDEL)) and the EU (project reference 318772 (D-MILS) and project reference 318490 (SENSATION)).

T. Margaria and B. Steffen (Eds.): ISoLA 2014, Part II, LNCS 8803, pp. 177–192, 2014.

features are often used for expressing uncertain faulty/erroneous behavior, like for example bit-flips, sensor glitches or simply the complete failure of a component. Real-time aspects are generally applied for expressing time-critical behavior, such as the nominal functional operation, as well as time-constrained behavior, like failure recovery mechanisms. On top of that, concurrency, both in the system and its environment, is omni-present to facilitate compositionality. Non-deterministic choice can be used to express the various potential implementations in a single model.

At this moment of writing, there are two prominent publicly available SMC tools that handle a real-time stochastic modeling formalism, namely UPPAAL-SMC [11] and Modes [7]. While using and comparing them, we noticed unintuitive discrepancies in the computed probabilities that could not be easily dismissed as bugs. After launching a full investigation, we found that there are deeply-rooted semantic caveats, as well as algorithmic design aspects that have a major impact on the computed probabilities. They emanate from mixing concurrency *and* non-deterministic choice *and* real-time *and* probabilistic aspects. This paper reports on the lessons we learned by the following contributions:

- The identification of five caveats and their interactions in Sections 3 and 4.
- A classification scheme for SMC algorithms, discussed in Section 5.
- Application and practical evaluation of the above with UPPAAL-SMC [11] and Modes [7] in Section 6.

Note that these lessons strictly apply to real-time stochastic formalisms that mix all four aspects. Model formalisms that mix two or three of those aspects tend to have a fully probabilistic semantics. Hence in those cases, such as Interactive Markov Chains [15] or Timed Automata, our learned lessons do not apply. The intent of our lessons is not to judge the use of SMC techniques on real-time stochastic models. Rather, we believe this paper provides a reference for users and developers to make an informed and deliberate decision on which SMC technique suits their purposes.

2 Preliminaries, Notations and Related Work

The results in this paper are scoped to real-time stochastic models, which we will loosely characterize in this section.

Real-Time Stochastic Models

Real-time stochastic models have a concurrent, a real-time and a probabilistic dimension. Particular examples of them are (networks of) Priced Timed Automata (PTA) as used by UPPAAL-SMC [11] and Stochastic Timed Automata (STA) as used by Modes [7]. Both models share a particular set of ingredients. The focus of this paper does not lie on the complete semantics of either model per se, but rather addresses issues related to a wider and more general class of real-time stochastic automata. Therefore, the semantics only provide a basis for

the further discussion in this paper, and do not form a single consistent model. We refer the reader to the original work on PTA and STA for their complete semantics.

Timed Processes. We first introduce a *timed process*, defined as a tuple $P = \langle L, l_0, I, T, C, A \rangle$, with L the set of locations, l_0 the initial location in L, C the set of real-valued clocks, A the set of actions including internal action τ, and

- $I : L \to Expr$ is the function that assigns to each location an invariant expression in $Expr$, which is a Boolean expression over the clocks in C. They restrict the residence time in that location based on the valuation of the clocks.
- $T \subseteq L \times A \times Expr \times 2^C \times L$ denotes the set of discrete transitions, with $Expr$ being the set of Boolean guard expressions over the clocks C.

A transition $\langle l_s, \alpha, g, X, l_t \rangle \in T$ allows the system to move from location l_s to l_t, executing action α. The transition is only enabled if the expression g evaluates to true in the current state. Upon execution of the transition, the clocks in set X are reset to zero.

The continuous state space of a timed process is defined as a set of tuples, consisting of the process' current location and valuation of the clocks. For process P at initial location p_0, with a single clock c_p assigned the value zero, the state is defined as $\langle p_0, (c_p, 0.0) \rangle$. For a process with a single clock we abbreviate this by $\langle p_0, 0.0 \rangle$. Transitions between states either delay or are discrete. *Delay* transitions progress clock values, e.g. $\langle p_0, 0.0 \rangle \xrightarrow{4.5} \langle p_0, 4.5 \rangle$. *Discrete* transitions are caused by a transition in a process' transition relation, e.g. $\langle p_0, 4.5 \rangle \xrightarrow{\alpha} \langle p_1, 4.5 \rangle$. A *path* (or sometimes also called *trace* or *run*) is a sequence of alternating delay and discrete transitions, e.g. $\langle p_1, 6.5 \rangle \xrightarrow{2} \langle p_1, 8.5 \rangle \xrightarrow{\alpha} \langle p_2, 8.5 \rangle \xrightarrow{1} \ldots$.

For expressing concurrency, we define a *network* of communicating processes (or sometimes called a network of automata). It consists of one or more timed processes composed together using a synchronization operator, denoted by $\|$. We define the state space of such a model as the cross-product of the state space of each individual process. In such a model, discrete transitions occur either concurrently, synchronizing on the shared communication *alphabet* of all the sets A of each process, or independently. When multiple independent transitions are enabled, a race condition occurs that is resolved by means of a *scheduler*. Timed transitions globally increase all clocks at the same rate.

Probabilistic Semantics. In the previous section we described the syntactical concepts for expressing a network of timed processes. We further extend upon this by introducing the possible ways of giving them a probabilistic semantics through inferring probability distributions and assuming a strategy for interpreting concurrency probabilistically.

Two types of stochastic behavior can be specified, namely for discrete transitions and delay transitions. In the case of a set of non-deterministic discrete transitions T, a discrete probability distribution $Distr(T)$ can be defined over

these transitions. This distribution specifies for each transition the probability of taking that transition, see e.g. [12,19]. Non-deterministic choice on its own, however, has no probabilistic semantics and thus typically one assumes one while simulating, like a uniform distribution. Additionally, there are formalisms that capture a discrete probabilistic distribution over successor locations. However w.l.o.g. they can be represented by a set of transitions such that we only have to restrict ourselves to models where transitions only have a single successor location. For delay transitions, a probability density function $\eta(l)$ can be specified over the possible delay times in the current location l. As we reason over a dense notion of time, this has to be a continuous distribution. This distribution can be explicitly specified by the user, or it is inferred from invariants or annotations on outgoing transitions like guards. Both the inference of a delay distribution and the choice for a discrete branching distribution come with caveats, which are discussed in Section 3.

In the face of concurrency, a statistical model checker employs a scheduler that selects the process to fire the next transition (or processes in the case of synchronization). This can be a scheduler that assigns a uniform distribution over the processes with the shortest waiting time, such as in UPPAAL-SMC. Other approaches exist as well. This also induces a caveat that is discussed in Section 3.

Recent work of [7] describes networks of Price Timed Automata, extending PTA [2,5] with exit rates for locations, as well as concurrency and probabilistic semantics for SMC. Like Timed Automata [1], they support the specification of clock variables, with location invariants and transition guards. It is the formalism implemented by the UPPAAL-SMC tool.

The work by [8] describes the MoDeST language that provides constructs for expressing the concurrency, probabilistically distributed waiting times, probabilistic branching and non-deterministic branching aspects described above. A set of MoDeST processes are mapped upon a *stochastic timed automaton (STA)*, which can be viewed as a generalization of Timed Automata and various Markov processes, e.g. CTMDPs [15]. It is the formalism implemented by the Modes tool.

Statistical Model Checking

Statistical model checking techniques [17] build upon and extend Monte-Carlo simulation techniques for verifying temporal bounded properties. These techniques rely on the following components: A path generator (sometimes also referred to as discrete event simulator), a property checker that decides whether path generation should continue or not, and an algorithm that decides if more paths need to be generated. The path generator simulates the behavior of the model by repeating discrete or delay transitions. Path generation continues until the property can be decided to hold or not, or when a boundary condition has been met. This is referred to as the *termination condition*. The Monte-Carlo method approximates the probability of a property by generating a large number of random samples. By repeatedly sampling random paths (also called traces or

simulations) from the automaton, statistical evidence is built up for disproving or proving the property of interest. Essentially, each generated path satisfies or dissatisfies the property. The outcomes of the generated paths are used to statistically decide whether additional paths are required and what the estimated probability of the property is. We refer the reader to [17] for an overview of statistical model checking techniques.

Several statistical model checkers have been built and reported. As the focus of this paper lies towards real-time stochastic models, only a few model checkers apply. The PLASMA tool [16,9] can analyze stochastic models such as Continuous Time Markov Chains (CTMCs) [4] and Markov Decision Processes (MDPs). It however does not support real-time aspects (yet), so it is omitted in the remainder of this paper. UPPAAL-SMC [11] and MODES [7] do support the aforementioned real-time stochastic formalisms and are therefore within the scope of this paper.

3 Semantic Caveats

As statistical model checking builds upon Monte-Carlo simulations, a fully probabilistic interpretation is required of the real-time stochastic model. This involves inferring or assuming probabilistic distributions on model elements that have no probabilistic semantics. The implications of this are however not always made obvious to the user. We investigated this and summarized the results as a set of *semantic caveats* which are elaborated upon in this section.

These caveats can be distinguished into two groups: The first group contains the caveats that originate from resolving underspecifications in the model, which are caveats $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$. The second group contains caveats $\mathbf{C_4}$ and $\mathbf{C_5}$, which the caveats that arise due to possible inconsistencies in the semantics of the model after inferring or assuming the probability distributions.

The diagrams in this section are denoted as follows: Nodes correspond to locations in the model, where the upper part indicates the label of the node; the lower part indicates either the invariant associated with the location, or the exit rate of the location. Edges between nodes represent discrete transitions. The enabled time interval induced by the guard of a transition is shown below the edge, the action above. Each example has an associated global clock. In order to simplify the diagrams, this clock has been hidden from the invariants and guards.

$\mathbf{C_1}$ – Underspecified Scheduling between Processes

The range of possible interleavings between concurrent processes is typically left underspecified during the design of the system and is subject to refinement in subsequent development phases. As such, we are dealing with an underspecification of the possible execution schedule(s). This ensures a conservative overapproximation of the possible behaviors in the analysis results.

Statistical model checkers employ a probabilistic interpretation over this form of underspecification. However, it is not immediately clear what this interpretation should be based on. Depending on how the one-step probabilities are calculated, this "progress" could depend on a process with the shortest waiting time (possibly uniformly chosen among multiple waiting times), but may also be distributed over any process that is capable of making progress after waiting. Finally, samples from the waiting time distribution may be discarded entirely if a process fails to find an enabled transition, requiring the sample process to start over.

$\mathbf{C_2}$ – Underspecified Choice within a Process

Within the context of communicating concurrent processes, we identify two kinds of choice, namely *internal* and *external* choice. Internal choice is when multiple internal transitions emanate from a location in the same process and they are unobservable by other processes, yet they lead to different successor locations. With external choice, we have multiple synchronizing transitions emanating from a single location.

When either choice is underspecified within a single process, i.e. $Distr(T)$ is not specified, the statistical model checker has to provide a probabilistic interpretation over it (see e.g. [13]). This is typically implicitly interpreted as equiprobabilistic, but other approaches exist, see e.g. [18] and Section 5. Issues may arise due to the introduction of a bias towards certain behavior.

Consider the example in Figure 1, with an external choice between actions α and β, and an internal choice between two α transitions. Applying a strictly uniform distribution over all transitions introduces a bias towards action α. However, applying a uniform distributions over the actions first introduces a bias towards the transition towards location p_3.

$\mathbf{C_3}$ – Underspecified Waiting Times

Transition guards and invariants are powerful modeling concepts for expressing timed behavior. They express the range of time in which certain behavior can occur, without requiring the user to specify exact time points or delay distributions. However, in the case of such underspecifications, the statistical model checker has to provide one as well, similar to caveat $\mathbf{C_2}$. As an example, Figure 2 shows a process with a transition that is enabled in the interval $[1, 5]$, and an invariant that is true for all clock values in the interval $[0, 5]$, without specifying a delay distribution.

Delay distributions may be derived from any combination of invariants, transition guards and process synchronization. Generally, a uniform distribution is derived (like in [3]), though other distributions may be used or even required, see caveat $\mathbf{C_4}$. Again, care has to be taken not to introduce any unwanted bias. Choosing whether or not to derive a distribution based on transition guards or process communication influences this, see Section 4.

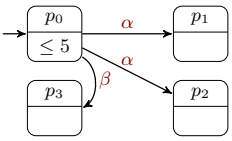

Fig. 1. Combination of internal and external choice

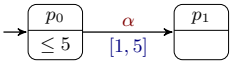

Fig. 2. Transition enabled within an interval

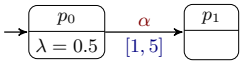

Fig. 3. Use of an exponential distribution

C_4 – Choice of Distributions

The previous caveats discussed underspecifications for which a statistical model checker has to generate probability distribution functions. However, based on the structure of the model, different distributions may be applicable. Often uniform distributions (both discrete and continuous) are employed, based on the notion of equiprobability. However, other distributions may be used or even required, for instance if the model specifies a boundary that extends to infinity, or specifies a deterministic delay by means of a point interval.

This introduces a caveat that requires attention in two directions. First, the support may not coincide with the enabledness of all transitions, which may lead for example to a deadlock (see also the accuracy dimension in Section 5). For instance, Figure 3 exemplifies this with an exponential distribution of which the support does not match the interval for which the transition is enabled. Second, and more obvious, the choice of distribution directly influences the bias on the waiting time.

C_5 – Invalid Paths

The modeling formalisms we consider come with an innate degree of abstraction. This comes with caveats by itself, namely Zeno behaviors, action-locks and deadlocks, which may generate semantically invalid paths. These caveats are well-known and widely studied [20]. Akin to analytical model checking techniques, also for statistical model checking these caveats require serious consideration.

Zeno behaviors may occur as a result of the model not allowing time to progress beyond a certain point, a timelock, or allowing paths that execute an infinite number of actions in a finite amount of time. As a result, a statistical model checker may not terminate its path generation when the termination condition is based on reaching a certain time bound.

Action-locks occur in the discrete part of the model, when the current state does not allow any further discrete transitions to be taken for any given delay.

Deadlocks are the combination of action-locks and timelocks. In a deadlocked state, neither the timed part nor the discrete part of the model can progress, and path generation will terminate prematurely. Examples of all three behaviors are shown in Figure 4. Note that for the action- and deadlocks, the behavior occurs in locations q_1 and r_1 respectively.

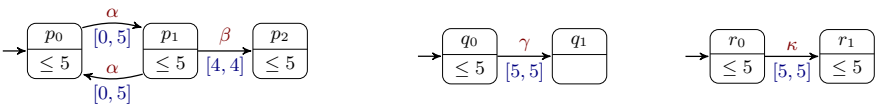

Fig. 4. Example processes showing respectively Zeno behavior, an action-lock and a deadlock

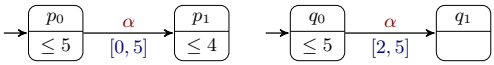

Fig. 5. Example network with synchronizing actions

4 Caveat Interactions

Thus far the caveats from the previous section have only been treated as separate instances. However, it is very well possible that they appear in the model at the same time, possibly constraining the possible distributions, or affecting the overall bias of the outcomes. Therefore, in this section we provide further examples to highlight the fact that caveats may interact and should not be considered as isolated entities.

C_1,C_3 – Event Synchronization with Partially Overlapping Enabledness Intervals

Underspecification of scheduling is generally dealt with by scheduling the process with the shortest waiting time first. Action synchronization requires two processes to take a discrete transition at the same time. Figure 5 highlights such a synchronization. Two processes P (left) and Q (right) are defined. Here, the action α is part of the communication alphabet. Thus the transitions synchronize and are therefore only enabled in the interval $[2, 4]$, due to the interaction of the guards and the invariants of the target locations.

This example highlights that action synchronization requires the path generator to consider all processes with synchronizing actions in order to prevent generating invalid paths that may for example invalidate invariants. For example, if only the local transitions in process P is considered, a waiting time of one may be sampled. However, it cannot synchronize then with process Q, as the transition is only enabled after time point two.

C_2,C_3 – Bias of Time Intervals

The bias towards a certain transition greatly depends on the time intervals. Figure 6 shows three different types of intervals that are subject to underspecification of choice as well. In all cases, if the underspecification of choice is resolved first and uniformly, selecting each outgoing transition with probability $\frac{1}{2}$, the probability of reaching either of the two target states is equal.

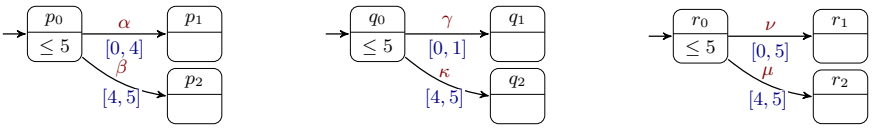

Fig. 6. Examples of underspecification of both choice and time with convex, non-convex and overlapping intervals respectively

However, resolving the underspecification of time first can cause considerable differences. In the first example (Figure 6, left), sampling in the interval in which either transition is enabled will cause a bias towards transition α (with probability $\frac{4}{5}$), as it has a larger interval. In the second case, both intervals are equally large. However, due to the union of the intervals being non-convex, a bias may be formed towards transition κ if the simulator generates a delay in the interval $[1, 4]$ as transition κ is the only transition that will become enabled after such a time point (see also Section 5.2). The third case shows the effect of the *scope* of the samples (Section 5.3). If a single sample is generated for location r_0, the waiting time is uniformly distributed. On the other hand, when generating a sample for both transitions individually, the probability distribution over the time intervals is no longer uniform, as the interval $[4, 5]$ of transition μ will increase the likelihood of picking a delay from that interval (when resolving the choice first (Section 5.1), the probability of a delay in $[4, 5]$ would become $\frac{1}{2} \cdot \frac{1}{5} + \frac{1}{2} \cdot \frac{1}{1} = \frac{3}{5}$).

5 Classifying SMC Algorithms

All statistical model checkers encounter the caveats outlined in the previous two sections and their implementations somehow deal with them. There is a range of possible solutions. We analyzed them and developed a classification scheme that has four dimensions, which we elaborate upon in the next sections.

5.1 Transition Selection Order – Early versus Delayed

The order solution dimension follows from caveats $\mathbf{C_1}$ and $\mathbf{C_2}$. It relates to the moment an enabled transition is chosen with respect to the moment that the waiting times are sampled. The choice of a particular order also impacts the accuracy dimension, which is elaborated later on.

Early ordering picks from each process a single enabled transition before sampling a waiting time. Thus, the race between transitions within a process is not determined by the execution times of the individual transitions. In our setting, this decision is usually made equiprobabilistically.

In the case of *delayed* ordering, the *waiting time(s)* are sampled first, simulating racing transitions. After sampling, the algorithm picks a winning transition which is used to extend the path. This choice is typically made equiprobabilistically between the fastest transitions, similar to racing processes.

Taking Figure 1 as an example, early ordering would pick either of the 3 transitions – usually uniformly – and sample a delay afterward. Delayed ordering does the opposite and generates a delay based on location p_0 and then pick any of the (enabled) transitions, possibly based on the delay.

A hybrid approach exists as well. In the work of [18], the preselection policy allows a set of transitions to be preselected, probabilistically or deterministically, which may then enter the race between processes. The simulation process then continues further as with delayed ordering.

5.2 Accuracy of Waiting Time Distributions – Exact versus Approximate

This dimension follows from caveat $\mathbf{C_3}$, the underspecification of time. This underspecification is resolved using a probability distribution over the possible time intervals. The simulator may generate such a distribution with a support that exactly coincides with the time intervals in which any transition is enabled, or it may approximate these intervals. Information used may stem from source and target location invariants, and transition guards. Synchronization effects may be taken into account as well.

In the *exact* case, the constructed probability distribution has a support that matches the time points that enable at least one transition. This is a strong property. Note that the constructed distribution can be non-convex. In the example of Figure 6 in the middle, this would mean the sampled waiting time is generated in the set $[0, 1] \cup [4, 5]$.

The *approximate* case allows the distribution to range over an over- or under-approximation of the set of possible waiting times. That means waiting times can be sampled for which no transition is enabled. Taking again the middle process of Figure 6 as an example, based only on the location invariant a possible interval could be $[0, 5]$. Over-approximating potentially results in an action-lock (see caveat $\mathbf{C_5}$). It is then up to the SMC algorithm what to do with the current sampled waiting time. This is the attitude policy and is discussed in Section 5.4.

5.3 Scope of Waiting Time Samples – Location Local versus Transition Local

The scope dimension follows from caveats $\mathbf{C_2}$ and $\mathbf{C_3}$ and when multiple transitions are enabled. A single delay can be sampled for the current location, or individual delays can be sampled for each transition emanating from the current location. Such a situation exists in all examples of Figure 6, where one delay may be generated entailing both transitions, or one delay per transition, totaling two.

We refer to the former solution as the *location local* scope. The distribution that is being constructed will account for all enabled time intervals of all non-deterministic choices. A waiting time sampled from that distribution does not necessarily imply which transition progresses, as the non-deterministic choices may have overlapping enabled intervals. As only one sample is generated for all transitions, the location local scope introduces a bias towards transitions that

Table 1. Summary of algorithmic policies

Policy class	Transition	Policies keeping sample	Policies rejecting sample
Memory	Enabled	Age & Enabling Memory	Resampling
	Disabled	Age Memory	Enabling Memory
Attitude	Enabled	Progressive & Conservative	-
	Disabled	Progressive	Conservative

have a larger interval, and may in fact render the simulator incapable of picking transitions with a deterministic point interval.

The alternative is referred to as the *transition local* scope. Here a sample is generated for each non-deterministic choice from a process' current location. Then, a race occurs within the process between the individual transitions. The transition local scope may introduce a bias as well. As was shown in Section 4 (Figure 6 right example), different transition intervals may induce a bias within the overall distribution of delays, in this case towards $[4, 5]$.

5.4 Race Policy

The race policy dimension addresses the lifetime of sample(s) generated for a process under a race condition. When the stochastic model is described by non-memoryless distributions, different probabilistic outcomes can be realized based on the manner in which samples are retained or discarded, which is decided by a policy. Such a policy can make different decisions based on winning or losing the race described by caveat $\mathbf{C_1}$ and whether any associated transitions are enabled or not.

Memory policies describe whether or not the generated sample will be retained for a future step if the process *lost* the race. In [18] three policies are described: age memory, which retains the sample if there is no enabled transition; enabling memory, which rejects the sample if no transition is enabled; and resampling, which always rejects the sample. Furthermore, the age and enabling policies keep the sample if the transition remains enabled.

Attitude policies dictate what samples are valid for a process that *won* the race. The conservative policy discards the sample if no transition is enabled. The progressive policy keeps the sample even if no transition is enabled (potentially causing an action-lock, see caveat $\mathbf{C_5}$). Both policies keep the sample (and use it) when a transition is enabled, and no policy rejects a sample in such a case. Both the memory and attitude policies are summarized in Table 1.

6 Applying the Systematization

To put the systematization in this paper to practical terms, we show how UPPAAL-SMC and Modes deal with the caveats and how their respective

Table 2. Model checking results from UPPAAL-SMC and MODES on the examples. The UPPAAL-SMC and MODES columns show the calculated probability intervals.

Caveats	Example	Property	UPPAAL	MODES
$\mathbf{C_2}$	Figure 1	$\mathbb{P}(\diamond^{[0,5]}(p_1 \vee p_2))$	[0.66,0.68]	[0.66,0.68]
		$\mathbb{P}(\diamond^{[0,5]}p_3)$	[0.32,0.34]	[0.32,0.34]
$\mathbf{C_3}$	Figure 2	$\mathbb{P}(\diamond^{[0,4]}p_1)$[a]	[0.74,0.76]	[0.98,1.00]
$\mathbf{C_4}$	Figure 3	$\mathbb{P}(\diamond^{[0,5]}p_1)$	[0.86,0.88]	[0.51,0.53]
$\mathbf{C_1}$,$\mathbf{C_3}$	Figure 5	$\mathbb{P}(\diamond^{[0,5]}(p_1 \wedge q_1))$	error[b]	[0.98,1.00]
$\mathbf{C_2}$,$\mathbf{C_3}$	Figure 6	$\mathbb{P}(\diamond^{[0,5]}p_1)$	[0.79,0.81]	[0.98,1.00]
		$\mathbb{P}(\diamond^{[0,5]}p_2)$	[0.19,0.21]	[0.00,0.02]
$\mathbf{C_2}$,$\mathbf{C_3}$ Non-convex	Figure 6	$\mathbb{P}(\diamond^{[0,5]}q_1)$	[0.19,0.21]	[0.98,1.00]
		$\mathbb{P}(\diamond^{[0,5]}q_2)$	[0.79,0.81]	[0.00,0.02]
$\mathbf{C_2}$,$\mathbf{C_3}$ Overlapping	Figure 6	$\mathbb{P}(\diamond^{[0,5]}r_1)$	[0.89,0.91]	[0.98,1.00]
		$\mathbb{P}(\diamond^{[0,5]}r_2)$	[0.09,0.11]	[0.00,0.02]

[a] A time bound of four was chosen to highlight the difference between UPPAAL-SMC and MODES scheduling.
[b] UPPAAL-SMC cannot execute the model as it is not input enabled.

SMC implementations can be classified. We furthermore compared these two tools using the examples presented in Sections 3 and 4. These examples have been modeled using the formalism used by the respective tool, and probabilistic time bounded reachability properties were evaluated to quantify the differences. The results are presented in Table 2.

For both tools, results were determined with a 0.99 confidence, and a 0.01 error bound. These parameters were chosen as they provide sufficient precision and confidence to compare the outcomes of the experiments. Note that due to the existence of an error bound, the results are not exact values but rather intervals. More detailed results and the sources of the models can be found online [10].

UPPAAL-SMC vs MODES

The path generation algorithm implemented by UPPAAL-SMC is described in [11]. The algorithm first determines the interval of possible waiting times by inspecting the invariant of the current active location. It then delays based on a sample from that interval, and uniformly chooses from the enabled transitions afterward. Any previously generated samples are ignored. When multiple processes are involved, the sample with shortest waiting time is selected.

The behavior of the path generation algorithm in the MODES tool [7] has been analyzed using the semantics of the MODEST language, configuration of the MODES tool and analysis of the experimental results. No information has been provided by the authors or derivative work on the path generation algorithm of

MODES. The MODES tool allows the resolution for both the underspecification of choice and time to be configured. Underspecification of choice can be resolved in four ways: The model is rejected; confluence detection is used to remove spurious non-determinism [14]; partial order reduction is used to remove spurious non-determinism [6]; or a uniform distribution is applied. Underspecification of waiting time can be resolved in two ways: Either the model is rejected, or an as-soon-as-possible (ASAP) scheduler is used (which always selects the shortest possible waiting time to enable a transition). For the experiments performed in this section, the uniform distribution is used for underspecification of choice, and the ASAP scheduler for underspecification of time.

Discussion

Most of the differences in the results can be attributed to the ASAP scheduler of the MODES tool. Despite having a delayed and approximate scheduling algorithm, the MODES tool always makes a deterministic choice to sample the shortest possible waiting time. Thus, whereas the results from UPPAAL-SMC tend towards a uniform distribution when choice is involved, MODES tends towards a Dirac-delta distribution. This can be seen for all the cases involving $\mathbf{C_3}$ (Figures 2, 5 and 6). Here, the probability of reaching a certain state within the specified time bound is approximately either 1.0 or 0.0 for MODES, indicating the chosen delay is constant.

Both the UPPAAL-SMC and MODES tools make use of a delayed order, resolving the underspecification of time before selecting a transition. For MODES, this can be seen in the results of the $\mathbf{C_3}$ case, as only the transition with the lowest time bound is chosen (due to the ASAP scheduler). In this case, MODES always chooses a delay of zero and thus, considering the examples in Figure 6, ends up in locations p_1, q_1 and r_1 respectively. Furthermore, both tools have an approximate accuracy. UPPAAL-SMC only takes the invariants into account, over-approximating the possible transition times. MODES only uses the lowest possible time value, thus under-approximating the possible transition times. For the scope dimension, both tools use the location local scope: UPPAAL-SMC uses just the invariants of a location to generate this sample whereas MODES picks the lowest possible value.

Both tools differ in the applied race policies. In case of UPPAAL-SMC, a resampling memory policy is used, as previous results are discarded when a new step is generated, and a progressive attitude policy is used, as a generated delay is always applied even if no transition is enabled. In the case of MODES, when process loses the race the age memory policy applies, as samples are explicitly assigned to process variables. However, the attitude policy can not be determined, as the scheduler ensures that there is always an enabled transition, as the difference can only be determined for delays after which no transition is enabled.

An interesting difference in results can be seen for the $\mathbf{C_4}$ case (Figure 3). This can be attributed to a difference in semantics for models containing exponential distributions to sample waiting times. In the example, location p_1 can

only be reached in the interval $[1, 5]$. Both MODES and UPPAAL-SMC interpret any sample above five as a deadlock. However, there is a difference in the interpretation of values below one. In UPPAAL-SMC, the outcome of the entire exponential distribution is shifted to the right by one unit of time, such that a waiting time below one is never generated. In MODES, a waiting time below one simply results in a deadlock, explaining the lower probability of reaching p_1.

7 Conclusions

The light-weight and scalable nature of statistical model checking techniques appeal as a practical way to handle the rich semantics of real-time stochastic models. From experimentation with two publicly available SMC tools, UPPAAL-SMC and MODES, we however encountered discrepancies in the computed probabilities on structurally equivalent models. They could not easily be dismissed as tool implementation bugs. In fact, they turned out to be semantic biases that did not align with our end-user interpretation. In our effort to study and understand these biases, we investigated how statistical model checkers deal with concurrency, non-deterministic and real-time aspects and how the discrepancies in the probabilities can be traced to key SMC tool design and implementation choices. From the lessons learned in this investigation, we systematized our observations into five caveats and a classification scheme for SMC algorithms. They can be used to understand any SMC technique on real-time stochastic models. We furthermore exemplify the caveats with concrete models, and show and discuss how two publicly available SMC tools, UPPAAL-SMC and MODES, compute significantly different probabilities on them.

We restrict our conclusion to the following: the use of SMC techniques on real-time stochastic models needs to be approached with caution. The systematization in this paper helps the end-user to identify and deal with the key points for caution. In the end, this increased understanding of SMC techniques helps the end-user to interpret the (difference in) computed probabilities by SMC tools. Their perceived biases ought not to be straightforwardly seen as an invalidation of SMC techniques. In fact, these biases may for example perfectly suit the assumptions that hold in the modeling domain (e.g. RAMS or systems biology). This paper contributes with a systematization for validating the SMC tools' behaviors against these desired assumptions, which might be for example used for tool certification.

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., La Torre, S., Pappas, G.J.: Optimal Paths in Weighted Timed Automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

3. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Probabilistic and Topological Semantics for Timed Automata. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 179–191. Springer, Heidelberg (2007)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model Checking Algorithms for Continuous-Time Markov Chains. IEEE Transactions on Software Engineering 29(6), 524–541 (2003)
5. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-Cost Reachability for Priced Timed Automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
6. Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial Order Methods for Statistical Model Checking and Simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE 2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011)
7. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and Statistical Model Checking for Modestly Nondeterministic Models. In: Schmitt, J.B. (ed.) MMB & DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012)
8. Bohnenkamp, H., D'Argenio, P.R., Hermanns, H., Katoen, J.-P.: MODEST: A Compositional Modeling Formalism for Hard and Softly Timed Systems. IEEE Transactions on Software Engineering 32(10), 812–830 (2006)
9. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013)
10. Bruintjes, H., Nguyen, V.Y.: Test results from experiments, `http://www-i2.informatik.rwth-aachen.de/~isola2014/smc/index.html` (Online; accessed May 14, 2014)
11. Bulychev, P., David, A., Larsen, K.G., Mikučionis, M., Bøgsted Poulsen, D., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. In: Wiklicky, H., Massink, M. (eds.) QAPL. Electronic Proceedings in Theoretical Computer Science, vol. 85, pp. 1–16. Open Publishing Association (2012)
12. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and Safety Verification for Stochastic Hybrid Systems. In: HSCC 2011, pp. 43–52. ACM (2011)
13. Grosu, R., Smolka, S.A.: Monte Carlo Model Checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 271–286. Springer, Heidelberg (2005)
14. Hartmanns, A., Timmer, M.: On-the-Fly Confluence Detection for Statistical Model Checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 337–351. Springer, Heidelberg (2013)
15. Hermanns, H. (ed.): Interactive Markov Chains. LNCS, vol. 2428. Springer, Heidelberg (2002)
16. Jegourel, C., Legay, A., Sedwards, S.: A Platform for High Performance Statistical Model Checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012)

17. Legay, A., Delahaye, B., Bensalem, S.: Statistical Model Checking: An Overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010)
18. Marsan, M.A., Balbo, G., Bobbio, A., Chiola, G., Conte, G., Cumani, A.: The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets. IEEE Transactions on Software Engineering 15(7), 832–846 (1989)
19. Sproston, J.: Decidable Model Checking of Probabilistic Hybrid Automata. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 31–45. Springer, Heidelberg (2000)
20. Tripakis, S.: Verifying Progress in Timed Systems. In: Katoen, J.-P. (ed.) ARTS 1999. LNCS, vol. 1601, pp. 299–314. Springer, Heidelberg (1999)
21. Younes, H.L., Simmons, R.G.: Statistical Probabilistic Model Checking With a Focus on Time-Bounded Properties. Information and Computation 204(9), 1368–1409 (2006)