# Rewrite-Based Statistical Model Checking of WMTL[⋆]

Peter Bulychev[1], Alexandre David[1], Kim G. Larsen[1], Axel Legay[2], Guangyuan Li[3], and Danny Bøgsted Poulsen[1]

[1] Aalborg University, Denmark
[2] INRIA Rennes – Bretagne Atlantique, France
[3] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, P.R. of China

**Abstract.** We present a new technique for verifying Weighted Metric Temporal Logic (WMTL) properties of Weighted Timed Automata. Our approach relies on Statistical Model Checking combined with a new monitoring algorithm based on rewriting rules. Contrary to existing monitoring approaches for WMTL ours is exact. The technique has been implemented in the statistical model checking engine of Uppaal and experiments indicate that the technique performs faster than existing approaches and leads to more accurate results.

## 1 Introduction

*Runtime verification* (RV) [11,1] is an emerging paradigm used to design a series of techniques whose main objective is to instrument the specification of a system (code, ...) in order to prove/disprove potentially complex properties at the execution level. Over the last years, RV has received a lot of interest and has been implemented in several toolsets. Such tools have been successfully applied on several real-life case studies.

The main problem with RV is that, contrary to classical verification techniques, it does not permit to assess the overall correctness of the entire system. *Statistical model checking* (SMC) [4,19,17] extends runtime verification capabilities by exploiting statistical algorithms to get evidence that a given system satisfies some property. The core idea of the approach is to monitor several executions of the system. The results are then used together with algorithms from statistics to decide whether the system satisfies the property with a probability greater than some threshold. Statistical model checking techniques can also be used to estimate the probability that a system satisfies a given property [12]. In contrast to classical exhaustive formal verification approaches, a simulation-based solution does of course not guarantee a result with 100% confidence. However, it is possible to bound the probability of making an error. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones,

⋆ Work partially supported by the VKR Centre of Excellence MT-LAB and the Sino-Danish Basic Research Center IDEA4CPS.

and are sometimes the only option [20]. Statistical model checking, which clearly complements RV, is widely accepted in various research areas such as software engineering, in particular for industrial applications, or even for solving problems originating from systems biology [13,10].

To get a more accurate intuition, Fig. 1 provides a schematic view of a statistical model checker and its interaction with RV procedures.

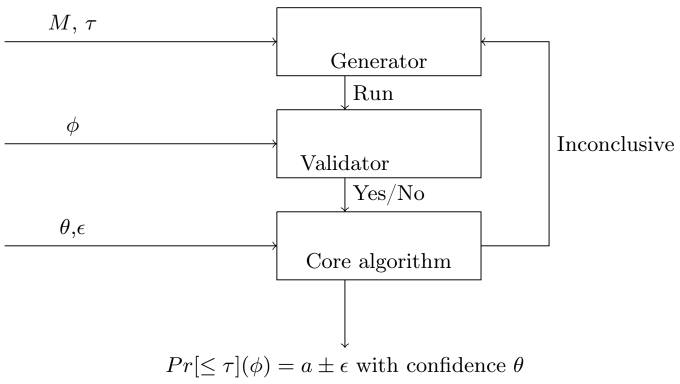


**Fig. 1.** A statistical model checker. The run generator first generates a run of $M$, which is propagated into the run validator. The run validator then validates if the run satisfies the property $\varphi$ and returns $Yes$ or $No$ to the core algorithm. Afterwards the core algorithm decides if another run is needed or if it, based on the accumulated knowledge, can draw a conclusion.

The run generator is responsible for generating runs of the model under verification and the run validator, which corresponds to the runtime verification part of the effort, validates if a run satisfy the property or not. The core algorithm collects the simulation results until sufficient samples has been obtained to provide an overall result. The core algorithm is computationally lightweight compared to the remaining two. An optimisation of SMC is therefore most easily obtained by optimising either the run generation or the run validation. In this paper, we focus on the run validation part.

In our work, we consider combining RV and SMC techniques in order to verify complex quantitative properties (performance evaluation, scheduling, ..) over rich systems. More precisely, we are interested in computing the probability that a random run of a Weighted Timed Automata(WTA)[3] satisfies a formula written in Weighted Metric Temporal Logic (WMTL)[5]. WTA is a rich formalism capable of capturing (quantitative) non-linear hybrid systems, while WMTL corresponds to the real-time extension of the linear temporal logic equipped with cost operators. In this paper, due to the use of SMC, we assume that the scope of the temporal operators is bounded, i.e., that one can decide whether a run satisfies a formula only by looking at a finite prefix. Unfortunately, it is known that, due to the expressivity of the automata-based model, the problem of verifying

WMTL with respect to WTA is undecidable [6] – hence it cannot be tackled with existing formal techniques such as model checking. Another drawback is that it is known that, even for the case where temporal operators are bounded, WMTL is more expressive than the class of deterministic timed automata [15]. This latter result implies that there is no automata-based runtime monitoring procedure for WMTL, even for the case where the scope of the temporal operators is finitely bounded. A first solution to the above problems could be to use a three-valued logic [2]. However, the absence of decision results is often unsatisfactory from an engineering point of view, especially when dealing with performance analysis.

In [8], we proposed the first SMC-based verification procedures for the eventually and always fragments of WMTL. Our work relies on a natural stochastic semantic for WTA. The work was implemented in Uppaal-smc and applied to a wide range of case studies. However, our original work does not consider nested temporal operators for which a solution was first proposed by Clarke et al. in [21]. While the approach in [21] is of clear interest, it only works for a subset of MTL where the temporal operators can only be upwards bounded, i.e., the lower bound is 0. In [7], we proposed another approach that relies on monitoring automata representing over and under approximations of solutions to the WMTL formula. This approach, which has been implemented in Casaal and Uppaal-smc, exploits confidence levels obtained on both approximations in order to estimate the probability to satisfy the formula. The first drawback with the approach in [7] is that both the under and over approximation depend on some precision that has an influence on the confidence level returned by the SMC algorithms. The second drawback is that automata-based monitors may be of large size, hence intractable.

In this paper, we propose a new monitoring approach for WMTL formulas. Contrary to existing approaches that work by first constructing a monitor for the property, ours exploit a graph-grammar procedure that rewrite the formula on-the-fly until a decision can be taken. The approach extends that of [16] to a timed logic. Contrary to existing off-line monitoring approaches [9], ours stops as soon as the formula is proved/disproved, which allows to save computation time and hence drastically improve both memory and time performances. Our approach has been implemented in Uppaal-smc and evaluated on several case studies, from random large-size formulas to concrete applications. As expected, there are many situations where we clearly outperform [7] while being more precise!

*Outline.* In section 2 we introduce our modelling formalism Networks of Weighted Timed Automata. Later in section 3 we define the WMTL logic, and section 4 describes our rewrite-based algorithm for monitoring of WMTL properties. The experiments are described in section 5.

## 2 Networks of Priced Timed Automata

In this paper, we briefly recap the formalism of networks of Weighted Timed Automata[3].

Let $\mathcal{X}$ be a set of variables called clocks. A clock valuation over $\mathcal{X}$ is a function $v : \mathcal{X} \to \mathbb{R}$ that assigns a real-valued number to each clock. We let $V(\mathcal{X})$ denote all possible valuations over $\mathcal{X}$ and let $\overline{0}$ be the valuation that assign zero to all clocks. An *upper bound* (resp. lower bound) over $\mathcal{X}$ is of the form $x \bowtie m$ where $x \in \mathcal{X}$, $m \in \mathbb{N}$, and $\bowtie \in \{<, \leq\}$ (resp. $\bowtie \in \{>, \geq\}$). We denote by $B^{\leq}(\mathcal{X})$ (resp. $B^{\geq}(\mathcal{X})$) the set of upper bounds (resp. lower bounds) over $X$. We let $B(\mathcal{X}) = B^{\leq}(\mathcal{X}) \cup B^{\geq}(\mathcal{X})$. Let $v$ be a valuation over $\mathcal{X}$ and let $g \subseteq B(\mathcal{X})$ then we write $v \models g$ if for all $(x \bowtie m) \in g$, $v(x) \bowtie m$. For a valuation $v \in V(\mathcal{X})$, a function $r : \mathcal{X} \to \mathbb{Q}$ and a $\tau \in \mathbb{R}$ we let $v + r \cdot \tau$ be the valuation over $\mathcal{X}$ such that $(v + r \cdot \tau)(x) = v(x) + r(x) \cdot \tau$ for every clock $x \in \mathcal{X}$. Let $\mathcal{X}_2 \subseteq \mathcal{X}$ then $v[\mathcal{X}_2 = 0]$ is the valuation that assigns zero to every clock in $\mathcal{X}_2$ and agress with $v$ on all other clocks. For two valuations $v_1$ and $v_2$ we let $v_2 - v_1$ be the valuation $v'$ where $v'(x) = v_2(x) - v_1(x)$ for every clock $x \in \mathcal{X}$.

**Definition 1.** *A Weighted Timed Automaton over the finite set of actions $\Sigma$ and the set of propositions $\mathcal{P}$ is a tuple $(\mathcal{L}, \ell_0, \mathcal{X}, \mathcal{X}_O^i, E, \mathcal{I}, \mathcal{R}, \mathcal{X}^{\mathcal{R}}, P)$, where*

- *$\mathcal{L}$ is a finite set of locations,*
- *$\ell_0 \in \mathcal{L}$ is the initial location,*
- *$\mathcal{X}$ is a finite set of clocks,*
- *$\mathcal{X}_O \subseteq \mathcal{X}$ is a finite set of observable clocks*
- *$E \subseteq \mathcal{L} \times 2^{B^{\geq}(\mathcal{X})} \times \Sigma \times \mathcal{X} \times \mathcal{L}$ is a set of edges,*
- *$\mathcal{I} : \mathcal{L} \to 2^{B^{\leq}(\mathcal{X})}$ assigns invariants to the locations,*
- *$\mathcal{R} : \mathcal{L} \to \mathbb{Q}$ assigns transition rates to locations,*
- *$\mathcal{X}^{\mathcal{R}} : \mathcal{L} \to \mathcal{X} \to \mathbb{Q}$ assign rates to the clocks of the WTA and*
- *$P : \mathcal{L} \to 2^{\mathcal{P}}$ assign propositions to the locations of the WTA.*

The semantics of a WTA $A = (\mathcal{L}, \ell_0, \mathcal{X}, \mathcal{X}_O^i, E, \mathcal{I}, \mathcal{R}, \mathcal{X}^{\mathcal{R}}, P)$ is given as a timed transition system with state space $\mathcal{L} \times V(\mathcal{X})$ (denoted $(SP(A))$ and initial state $(\ell_0, \overline{0})$ (denoted *init*$(A)$). For consistency, we require $\overline{0} \models \mathcal{I}(\ell_0)$. Furthermore we require that the rates of the observable clocks in any location is greater than 0. The transition rules are given below

- *delay*: $(\ell, v) \xrightarrow{d} (\ell, v')$ where $d \in \mathbb{R}_{\geq 0}$, if $v' = v + \mathcal{X}^{\mathcal{R}}(\ell) \cdot d$ and $v' \models \mathcal{I}(\ell)$
- *discrete transition*: $(\ell, v) \xrightarrow{a} (\ell', v')$ if there exists $(\ell, g, a, \mathcal{Y}, \ell') \in E$ such that $v \models g$, $v' = v[\mathcal{Y} = 0]$ and $v' \models \mathcal{I}(\ell')$.

To prepare for composition of WTAs we assume that the set of actions $\Sigma$ is partitioned into a set of input actions $\Sigma_i$ and output actions $\Sigma_o$. Also we assume the WTA is input-enabled for the input actions $\Sigma_i$, i.e. that for any $a \in \Sigma_i$ and any state $(\ell, v)$ there exists a transition $(\ell, v) \xrightarrow{a} (\ell', v')$. A WTA is deterministic for $\Sigma' \subseteq \Sigma$ if there exists at most one transition for each $a \in \Sigma$. In the paper, we let $A^i = (\mathcal{L}^i, \ell_0^i, \mathcal{X}^i, \mathcal{X}_O^i, E^i, \mathcal{I}^i, \mathcal{R}^i, \mathcal{X}^{\mathcal{R}^i}, P^i)$.

*Network of WTAs.* A network of WTAs (NWTA) is a set of WTAs executing in parallel. The automata communicate via *broadcast synchronisation.*

Let $A^1, A^2, \ldots, A^n$ be WTAs over the common set of actions $\Sigma$. Furthermore, let $\Sigma^1, \Sigma^2 \ldots \Sigma^n$ be mutually disjoint subsets of $\Sigma$ and for all $i$ let $A^i$ be deterministic and input-enabled with respect to $\Sigma \setminus \Sigma^i$ and deterministic with respect to $\Sigma^i$. Then we call $N = A^1|A^2|\ldots|A^n$ a network of WTAs over $\Sigma$ where $\Sigma^i$ is the output actions of $A^i$ and $\Sigma \setminus \Sigma^i$ is its input actions.

The semantics of the network of WTAs is a timed transition system with the state space $SP(N) = SP(A^1) \times SP(A^2) \times \cdots \times SP(A^n)$ and the initial state $(init(A^1), init(A^2), \ldots, init(A^n))$. We refer to an element $\boldsymbol{s} = (s_1, s_2, \ldots, s_n) \in SP(N)$ as a state vector of the network and let $\boldsymbol{s}_i = s_i$. The transition rules of a network is given as

- $(\boldsymbol{s}) \xrightarrow{d} (\boldsymbol{s}')$ if for all $i, 1 \leq i \leq n$, $\boldsymbol{s}_i \xrightarrow{d} \boldsymbol{s}'_i$, and $d \in \mathbb{R}_{\geq 0}$
- $(\boldsymbol{s}) \xrightarrow{a} (\boldsymbol{s}')$ if for all $i, 1 \leq i \leq n$ $\boldsymbol{s}_i \xrightarrow{a} \boldsymbol{s}'_i$, and $a \in \Sigma$.

Consider WTAs given in Fig. 2. WTAs (a) and (b) are competing to force (c) to go either to location *Left* or to location *Right*. Initially both competitors are waiting for between 3 and 5 time units whereafter one of them moves the (c) to either *Left* or *Right*. Afterwards both competitors have a period where time progresses and nothing occurs. Indeed, when one of the competitors returns it must wait for between 3 and 5 time units again and choose to either move (c) or let it be and enter a waiting period again. The primary difference between (a) and (b) is that (b) rushes to return to a position from which it can change (c) and (a) returns within 5 time units.

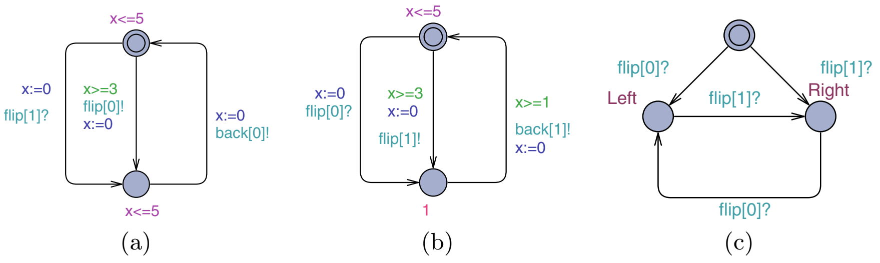


**Fig. 2.** Network of Timed Automata

Let $\boldsymbol{s} = ((\ell_1, v_1), (\ell_2, v_2), \ldots, (\ell_n, v_n))$ be a state vector for $A^1|A^2|\ldots|A^n$. Then we let $P(\boldsymbol{s}) = \bigcup_{i=1}^{n} P^i(\ell_i)$. Let $x \in \mathcal{X}_i$ for som $i$ then $V(\boldsymbol{s}, x) = v_i(x)$.

**Definition 2 (Run).** *Let $A^1|A^2|\ldots|A^n$ be a network of WTAs. A run of the network is an infinite weighted word $(\mathcal{P}_0, v_0)(\mathcal{P}_1, v_1)\ldots$ where for all $i, v_i$ is a valuation over $\mathcal{Y} = \bigcup_{i \in \{1,2,\ldots,n\}} \mathcal{X}_O^i$ and*

– $v_0 = \overline{0}$ ,
– *there exists an alternating sequence of delays and discrete transitions* $\boldsymbol{s}_0 \xrightarrow{d_0} \boldsymbol{s}'_0 \xrightarrow{a_0} \boldsymbol{s}_1 \xrightarrow{d_1} \ldots$, *where for all* $i$, $0 < i$, *and for all* $x \in \mathcal{Y}$ $v_i(x) = v_{i-1}(x) + (V(\boldsymbol{s}'_{i-1}, x) - V(\boldsymbol{s}_{i-1}, x))$.
– $\boldsymbol{s}_0 = (init(A^1), init(A^2), \ldots, init(A^n))$ *and*
– *for all* $j, j \geq 0, \mathcal{P}_j = P(s_j)$.

For a run $\omega = (\mathcal{P}_0, v_0)(\mathcal{P}_1, v_1)\ldots$, we let $\omega^i = (\mathcal{P}_i, v_i)(\mathcal{P}_{i+1}, v_{i+1})\ldots$. A run $\omega$ is called *diverging* for clock $x$ if for any $i$ there exists a $j$ such that $v_j(x) > v_i(x)+1$. A run is diverging if it is diverging for all clocks. In what follows, we assume that there always exists a clock $\tau$ in a WTA, and this clock always have a rate of 1 and is never reset, i.e. $\tau$ measures the time length of a run.

*Stochastic Semantics.* In [8] we introduced the stochastic semantics for NWTAs, i.e. proposed a probability measure on the set of all runs of a network and described an algorithm for generating *a random run.* Roughly speaking, the stochastic semantics of WTA components associates probability distributions on both the delays one can spend in a given state as well as on the transition between states. In UPPAAL-SMC uniform distributions are applied for bounded delays and exponential distributions for the case where a component has unbounded delay. In a network of WTAs the components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, with the "winner" being the component that chooses the minimum delay.

*Statistical Model Checking.* As said in the introduction, we use SMC [4,19,17] to compute the probability for a network of WTAs to satisfy a given property. Given a program $B$ and a trace-based property[1] $\phi$ , SMC refers to a series of simulation-based techniques that can be used to answer two questions: (1) *qualitative:* is the probability for $B$ to satisfy $\phi$ greater or equal to a certain threshold $\theta$ (or greater or equal to the probability to satisfy another property $\phi'$) [19]? and (2) *quantitative:* what is the probability for $B$ to satisfy $\phi$ [12]? In both cases, the answer is correct up to some confidence level, i.e., probability that the algorithm does not make mistake, whose value can be configured by the user. For the quantitative approach, which we will intensively use in this paper, the method computes a confidence interval that is an interval of probabilities that contains the true probability to satisfy the property. The confidence level is interpreted as the probability for the algorithm to compute a confidence interval that indeeds contains the probability to satisfy the property.

Our UPPAAL-SMC toolset implements a wide range of SMC algorithms for WTAs. In addition, the tool offers several features to visualize and reason on the results. Until now, the monitoring procedure for WMTL relies on a technique that computes over and under approximation monitors for the formulas. In this paper, we go one big step further and propose a more efficient and precise monitoring procedure.

[1] i.e. a property with semantics defined on traces.

## 3 Weighted Metric Temporal Logic

In this section we review the syntax and semantics of Weighted Metric Temporal Logic (WMTL) [5]. The syntax is defined as follows.

**Definition 3.** *A WMTL formula over the propositions $\mathcal{P}$ and the clocks $\mathcal{X}$ is generated by the grammar:*

$$\varphi, \varphi_1, \varphi_2 ::= \top \mid \bot \mid p \mid \neg p \mid O\,\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 U^x_{[a,b]} \varphi_2 \mid \varphi_1 R^x_{[a,b]} \varphi_2$$

*where $a, b \in \mathbb{Q}$, $a \leq b$, $p \in \mathcal{P}$ and $x \in \mathcal{X}$.*

As one can see in the syntax, we restrict to a fragment of WMTL where temporal operators are bounded. As stated in the introduction, this fragment is sufficient to break any decidability results. Observe that WMTL is an extension of Metric Temporal Logic (MTL) [14] in which $U$ and $R$ can also be bounded for arbitrary clocks. As an example, bounding $U$ and $R$ over arbitrary clock allows one to express that a communication device should recover from a state without spending more than $x$ units of energy. This can be accomplished by adding an observable clock, that measures the energy consumption, to the model and bound the $U$ and $R$ modalities over this clock.

We interpret WMTL formulas over runs of WTAs. Informally, the WMTL formula $\varphi_1 U^x_{[a,b]} \varphi_2$ is satisfied by a run if $\varphi_1$ is satisfied on the run until $\varphi_2$ is satisfied, and this should happen before the value of the clock $x$ increases with more than $b$ units starting from the beginning of the run, and after it increases for more than $a$ units. Formula $O\,\varphi$ means that $\varphi$ should be satisfied starting from the next observation of the run. The logical operators are defined as usual, and the *release* opearator $R$ is dual to $U$, and $\varphi_1 R^x_{[a,b]} \varphi_2 \equiv \neg(\neg\varphi_1 U^x_{[a,b]} \neg\varphi_2)$.

Formally, let $\omega = (\mathcal{P}_0, v_0)(\mathcal{P}_1, v_1)\ldots$ be a timed run. The satisfaction relation is inductively defined as

- $\omega \models \top$
- $\omega \models p$ if $p \in \mathcal{P}_0$
- $\omega \models \neg p$ if $p \notin \mathcal{P}_0$
- $\omega \models O\,\varphi$ if $\omega^1 \models \varphi$
- $\omega \models \varphi_1 \vee \varphi_2$ if $\omega \models \varphi_1$ or $\omega \models \varphi_2$
- $\omega \models \varphi_1 \wedge \varphi_2$ if $\omega \models \varphi_1$ and $\omega \models \varphi_2$
- $\omega \models \varphi_1 U^x_{[a,b]} \varphi_2$ if there exists $i$ such that $a \leq v_i(x) - v_0(x) \leq b, \omega^i \models \varphi_2$ and for all $j < i$ we have $\omega^j \models \varphi_1$
- $\omega \models \varphi_1 R^x_{[a,b]} \varphi_2$ if there exists $i$ such that $a \leq v_i(x) - v_0(x) \leq b, \omega^i \models \varphi_1$ and for all $j \leq i, \omega^j \models \varphi_2$, or for all $i$ such that $v_i(x) - v_0(x) \leq b$ we have $\omega^i \models \varphi_2$

In the rest of the paper, we use the following equivalences: $\Diamond^x_{[a,b]}\varphi = \top U^x_{[a,b]}\varphi$ and $\Box^x_{[a,b]}\varphi = \bot R^x_{[a,b]}\varphi$. We also use $\Box_{[a,b]}\varphi$ instead of $\Box^\tau_{[a,b]}\varphi$ for the case $\tau$ grows with rate 1.

*Example 1.* Consider again the WTAs in Fig. 2 and assume that the winner of the competition is the one who managed to have (c) located in its designated location for 8 consecutive time units. To express that (a) wins within 100 time units we need to state that (c) stays in *Left* for 8 consecutive time units at some point and that it has not stayed in *Right* for 8 consecutive time units before that point. Using WMTL this can be expressed like

$$(\neg Left \vee \Diamond_{[0,8]} Right) U_{[0,92]} (\Box_{[0,8]} Right).$$

We now focus on deciding a WMTL formula $\varphi$ on a finite prefix of an infinite diverging run $\omega = (\mathcal{P}_0, v_0)(\mathcal{P}_1, v_1)\ldots$. We first define the bound function $N(w, \varphi)$ inductively as follows:

$$\begin{aligned}
N(\omega,\top) = N(\omega,\bot) = N(\omega,p) &= 0\\
N(\omega,\neg p) &= 0\\
N(\omega,\varphi_1 \wedge \varphi_2) &= \max\{N(\omega,\varphi_1), N(\omega,\varphi_2)\}\\
N(\omega,\varphi_1 \vee \varphi_2) &= \max\{N(\omega,\varphi_1), N(\omega,\varphi_2)\}\\
N(\omega,O(\varphi)) &= 1 + N(\omega,\varphi)\\
N(\omega,\varphi_1 U^x_{[a;b]} \varphi_2) &= \max\nolimits_{i.a\le v_i(x)-v_0(x)\le b}\big(\max\{i + N(\omega^i,\varphi_2),\\
&\qquad \max\nolimits_{j<i}\{j + N(\omega^j,\varphi_1\}\}\big)\\
N(\omega,\varphi_1 R^x_{[a;b]} \varphi_2) &= \max\nolimits_{i.a\le v_i(x)-v_0(x)\le b}\big(\max\{i+1, i + N(\omega^i,\varphi_2),\\
&\qquad \max\nolimits_{j\le i}\{j + N(\omega^j,\varphi_1\}\}\big)
\end{aligned}$$

The bound function characterises the maximal prefix of $\omega$ that one needs to observe to decide $\varphi$. Observe that, contrary to [21], the bound depends not only on the formula but also on the run itself. The latter is due to the introduction of the next operator that is absent in [21].

We say that two infinite runs $\omega_1 = (\mathcal{P}^1_0, v^1_0)(\mathcal{P}^1_1, v^1_1)\ldots$ and $\omega_2 = (\mathcal{P}^2_0, v^2_0)$ $(\mathcal{P}^2_1, v^2_1)\ldots$ are $n$-equivalent, denoted $\omega_1 \equiv_n \omega_2$, if for all $i \le n$ $\mathcal{P}^1_i = \mathcal{P}^2_i$ and $v^1_i = v^2_i$. We say that $\omega$ n-boundly satisfies $\varphi$, denoted $\omega \models^n \varphi$, iff for all $\omega'$ where $\omega \equiv_n \omega'$, $\omega' \models \varphi$. We say that run n-boundly violate $\varphi$ if for all $\omega'$ where $\omega \equiv_n \omega'$, $\omega' \not\models \varphi$. It is easy to see that $\omega \models^n \varphi \implies \omega \models \varphi$ and $\omega \not\models^n \varphi \implies \omega \not\models \varphi$. We can now conclude with the following theorem that shows that any WMTL property can be decided on a finite prefix of the run.

**Theorem 1.** *Let $\omega$ be an infinite run and $\varphi$ be a WMTL formula. Then $\omega \models \varphi$ if and only if $\omega \models^{N(\omega,\varphi)} \varphi$ and $\omega \not\models \varphi$ if and only if $\omega \not\models^{N(\omega,\varphi)} \varphi$.*

## 4 Monitoring WMTL Properties

We present an efficient online monitoring algorithm for checking if a given infinite run $\omega$ of a WTA satisfies a given WMTL property $\varphi$.

**Algorithm 1.** WMTL formula satisfiability checking

```
// Input: MTL formula $\varphi$ and weighted word $\omega$
// Output: true iff $\omega \models \varphi$, false otherwise
$i$:=0
while $\varphi \neq \top \wedge \varphi \neq \bot$ do
    $\varphi$:=$\beta(\gamma(\varphi, \mathcal{P}_i, v_{i+1} - v_i))$
    i:=i+1
end
if $\varphi == \top$ then
    return true
end
if $\varphi == \bot$ then
    return false
end
```

The pseudo code of our algorithm is presented in Algorithm 1. Intuitively, the algorithm reads the elements of the input run one-by-one and rewrites the formula after reading each new element. The algorithm stops when the formula becomes $\top$ or $\bot$ meaning that any continuation of the finite prefix read so far will be accepted (or rejected) by the original formula $\varphi$. The rewriting step is performed by first applying the function $\gamma$, that *updates* the formula according to a new observation, and then applying $\beta$ function, that *simplifies* the formula and tries to reduce it to $\top$ or $\bot$.

The *rewrite* function $\gamma$ is defined by the following recursive rules where $v$ is a function that gives the change of the clock variables since the last element of the run:

- $\gamma(p, \mathcal{P}, v) = \begin{cases} \top, & \text{if } p \in \mathcal{P} \\ \bot, & \text{if } p \notin \mathcal{P} \end{cases}$
- $\gamma(\neg p, \mathcal{P}, v) = \begin{cases} \bot, & \text{if } p \in \mathcal{P} \\ \top, & \text{if } p \notin \mathcal{P} \end{cases}$
- $\gamma(\varphi_1 \wedge \varphi_2, \mathcal{P}, v) = \gamma(\varphi_1, \mathcal{P}, v) \wedge \gamma(\varphi_2, \mathcal{P}, v)$
- $\gamma(\varphi_1 \vee \varphi_2, \mathcal{P}, v) = \gamma(\varphi_1, \mathcal{P}, v) \vee \gamma(\varphi_2, \mathcal{P}, v)$
- $\gamma(O\,\varphi, \mathcal{P}, v) = \varphi$
- $\gamma(\varphi_1 U^x_{[a,\,b]} \varphi_2, \mathcal{P}, v) =$
$$\begin{cases} \gamma(\varphi_1, \mathcal{P}, v) \wedge \varphi_1 U^x_{[min(a-v(x),\,0),\,b-v(x)]} \varphi_2\,, & \text{if}\, a > 0 \wedge v(x) \leq b \\ \gamma(\varphi_2, \mathcal{P}, v) \vee (\gamma(\varphi_1, \mathcal{P}, v) \wedge \varphi_1 U^x_{[0,\,b-v(x)]} \varphi_2)\,, & \text{if}\, a = 0 \wedge v(x) \leq b \\ \gamma(\varphi_2, \mathcal{P}, v), & \text{if}\, a = 0 \wedge v(x) > b \\ \bot, & \text{if}\, a > 0 \wedge v(x) > b \end{cases}$$
- $\gamma(\varphi_1 R^x_{[a,\,b]} \varphi_2, \mathcal{P}, v) =$
$$\begin{cases} \gamma(\varphi_2, \mathcal{P}, v) \wedge \varphi_1 R^x_{[min(a-v(x),\,0),\,b-v(x)]} \varphi_2\,, & \text{if}\, a > 0 \wedge v(x) \leq b \\ \gamma(\varphi_2) \wedge (\gamma(\varphi_1, \mathcal{P}, v) \vee \varphi_1 R^x_{[0,\,b-v(x)]} \varphi_2)\,, & \text{if}\, a = 0 \wedge v(x) \leq b \\ \gamma(\varphi_2, \mathcal{P}, v), & \text{if}\, v(x) > b \end{cases}$$

The omitted cases are all rewritten into themselves. The *simplify* function $\beta$ is defined by the following recursive rules:

– $\beta(\varphi_1 \wedge \varphi_2) = \begin{cases} \bot, & \text{if } \beta(\varphi_1) = \bot \text{ or } \beta(\varphi_2) = \bot \\ \beta(\varphi_1), & \text{if } \beta(\varphi_2) = \top \\ \beta(\varphi_2), & \text{if } \beta(\varphi_1) = \top \\ \beta(\varphi_1) \wedge \beta(\varphi_2), & \text{otherwise.} \end{cases}$

– $\beta(\varphi_1 \vee \varphi_2) = \begin{cases} \top, & \text{if } \beta(\varphi_1) = \top \text{ or } \beta(\varphi_2) = \top \\ \beta(\varphi_1), & \text{if } \beta(\varphi_2) = \bot \\ \beta(\varphi_2), & \text{if } \beta(\varphi_1) = \bot \\ \beta(\varphi_1) \vee \beta(\varphi_2), & \text{otherwise.} \end{cases}$

– $\beta(\varphi) = \varphi$ in rest of the cases.

The *simplify* function $\beta$ takes into account only the logical equivalences, namely $\varphi \wedge \top \equiv \varphi$, $\varphi \wedge \bot \equiv \bot$, $\varphi \vee \top \equiv \top$, $\varphi \vee \bot \equiv \varphi$.

The correctness and termination of our algorithm is proved by the following two theorems:

**Theorem 2.** *Let $\omega = (\mathcal{P}_0, v_0), (\mathcal{P}_1, v_1), \ldots$ be an infinite weighted word, and $\varphi$ be a WMTL formula. Then $\omega^i \models \varphi$ if and only if $\omega^{i+1} \models \gamma(\varphi, \mathcal{P}_i, v_{i+1} - v_i)$ .*

**Theorem 3.** *Let $\omega = (\mathcal{P}_0, v_0), (\mathcal{P}_1, v_1), \ldots$ be an infinite weighted word that diverges for every clock used in a WMTL formula $\varphi$. Let $\varphi_0 = \varphi, \varphi_1, \ldots$ be a sequence of WMTL formulas such that for all $i > 0$ $\varphi_{i+1} = \beta(\gamma(\varphi_i, \mathcal{P}_i, v_{i+1} - v_i))$. Then there exists $k \geq 0$ such that $\varphi_k = \top$ if and only if $\omega \models \varphi_0$. Similarly, there exists $k \geq 0$ such that $\varphi_k = \bot$ if and only of $\omega \not\models \varphi_0$.*

*Example 2.* Consider the run $(\{a\}, \{\tau \mapsto 0\})(\{a\}, \{\tau \mapsto 2.5\})(\{b\}, \{\tau \mapsto 3\})$ $(\{a\}, \{\tau \mapsto 3.2\})(\{b, c\}, \{\tau \mapsto 5\})(\{a\}, \{\tau \mapsto 6\}) \ldots$ and the WMTL formula $(aU_{[0,4]}b)U_{[0,10]}c$. Our algorithms will produce the following sequence of rewriting rules. The sequence results in $\top$ thus the formula is satisfied by the run.

$$\begin{aligned}
(aU_{[0,4]}b)U_{[0,10]}c &\xrightarrow{\{a\},\{\tau\mapsto 2.5\}} (aU_{[0,1.5]}b) \wedge (aU_{[0,4]}b)U_{[0,7.5]}c \\
&\xrightarrow{\{a\},\{\tau\mapsto 0.5\}} (aU_{[0,1.0]}b) \wedge ((aU_{[0;3.5]}b) \wedge (aU_{[0,4]}b)U_{[0,7.0]}c) \\
&\xrightarrow{\{b\},\{\tau\mapsto 0.2\}} (aU_{[0,4]}b)U_{[0,6.8]}c \\
&\xrightarrow{\{a\},\{\tau\mapsto 1.8\}} (aU_{[0,3.2]}b) \wedge ((aU_{[0,4]}b)U_{[0,5.0]}c) \\
&\xrightarrow{\{b,c\},\{\tau\mapsto 1\}} \top.
\end{aligned}$$

## 5 Experiments

Our approach has been implemented in UPPAAL-SMC. We now illustrate the technique and compare it with the one in [7] that relies on automata-based monitors. If there is no deterministic automaton for the corresponding formula, [7] builds a deterministic under/over approximation that may strongly impact the confidence interval computed by SMC.

## 5.1 Size of Intermediate Formulas and Precision

Our rewriting rules are recursive in the structure of the formula, which means that the performance of the technique is highly dependent on the size of the intermediate formulas. In the following example, we show how the size of the intermediate formulas vary. We also show that our technique is often much more accurate than the one of [7].

We first study the evolution of the size of the intermediate formula generated by our technique during the monitoring of several randomly generated formulas. We also study the precision of the confidence interval returned by the SMC algorithm in case [7] uses an over or under approximation of the monitor. We also exploit an encoding in Uppaal to show how the size of the formula varies over time for a validation of a single run. In both cases runs are randomly generated by automata. This is done by choosing a delay with respect to an exponential distribution with rate parameter $r$ and after the delay with a discrete probabilistic choice set one of the propositions to true or false.

**Random Formulas.** We compute the average size of the largest intermediary formula generated in the rewriting process of different formulas. We verified each formula with a confidence level of 0.05. The results of the test are shown in Table 2(a) and Table 2(b). We also give the verifiation time and the time used in total for the monitor based approach, i.e. both the time to construct the monitor and to verify. The results show that the intermediate formula size depends on the transition rate of the model and as a result so does the validation. The monitor based approach, on the other hand, does not depend on this and the time used remain constant for all the models - due to the most significant part of the monitor based approach is constructing the monitor. However, the rewrite technique is significantly faster than the monitoring technique in all cases. For the results in Table 2(a) the monitors are tight approximations thus we gain time and not precision. However, results in Table 2(b) show that we can obtain much more accurate confidence intervals with our new technique. This is due to the monitors might be a large over approximations/a small under approximation. The variance in Table 2(a) is rather high due to the runs being random.

**Modeling Uppaal inside Uppaal.** In order to obtain a more in-depth view on how the size of formulas change over time, we have encoded the rules as Uppaal timed automata. The objective being to use the visualisation features of the tool to see how the number of automata evolve over time. Our construction is recursive in the structure of the formula in the sense that a network of observing automata for $\phi$ is obtained as one automata for $\phi$ and at least one automaton for each of the sub-formulas of $\phi$.

**Table 1.** Result of the random formula test. The $r$ column is the rate at which the run was generated. The $\#U/R$ column contains the number of until or release modalities that was in the formula. The $E_{Max}$ column is the average largest size of formula and $\sigma^2_{Max}$ is the variance thereof. $\tau_M$ and $\tau_R$ is the verification time for the monitoring technique and the rewrite technique, respectively. The verification time for the monitors are the time to construct the monitors and use them - in all the cases the monitors were not exact and both the under and over approximation was used. The $R_R$ and $R_M$ columns contain the number of runs each method required to establish the verification result. The $\%_M$ and $\%_R$ columns refer to the confidence interval obtained by the monitoring and the rewrite process respectively.

| Formula | $r$ | $\#U/R$ | Largest | $E_{Max}$ | $\sigma^2_{Max}$ | $\tau_R$ | $\tau_M$ | $R_R$ | $R_M$ |
|---|---|---|---|---|---|---|---|---|---|
| Random1 | 1 | 11 | 14 | 6.81 | 3.16 | 0.19$s$ | 5.70$s$ | 738 | 1748 |
| Random1 | 4 | 11 | 18 | 7.03 | 4.92 | 0.22$s$ | 5.83$s$ | 738 | 1748 |
| Random1 | 8 | 11 | 21 | 7.06 | 4.74 | 0.23$s$ | 5.78$s$ | 738 | 1748 |
| Random2 | 1 | 8 | 17 | 8.52 | 5.33 | 0.19$s$ | 6.13$s$ | 738 | 1748 |
| Random2 | 4 | 8 | 21 | 11.05 | 4.71 | 0.34$s$ | 6.17$s$ | 738 | 1748 |
| Random2 | 8 | 8 | 27 | 12.79 | 7.16 | 0.58$s$ | 6.26$s$ | 738 | 1748 |
| Random3 | 1 | 11 | 21 | 11.51 | 4.74 | 0.50 | 10.99$s$ | 738 | 1748 |
| Random3 | 4 | 11 | 40 | 13.58 | 16.53 | 1.08 | 11.06$s$ | 738 | 1748 |
| Random3 | 8 | 11 | 36 | 14.00 | 18.16 | 1.52 | 11.38$s$ | 738 | 1748 |

(a)

| Formula | $\#U/R$ | $r$ | $\%_R$ | $\%_M$ | $R_R$ | $R_M$ | $\tau_R$ | $\tau_M$ |
|---|---|---|---|---|---|---|---|---|
| random4 | 15 | 4 | [0.57; 0.67] | [0.57; 0.83] | 738 | 1748 | 0.34s | 7.77s |
| random5 | 15 | 4 | [0.00; 0.05] | [0.00; 0.97] | 738 | 1748 | 0.94s | 2.83s |
| random6 | 15 | 4 | [0.00; 0.05] | [0.00; 0.72] | 738 | 1748 | 0.81s | 3.18s |
| random7 | 15 | 4 | [0.00, 0.07] | [0.00; 0.43] | 738 | 1748 | 2.36s | 26.61s |

(b)

The automaton for $\phi$ starts its sub-automata through a designated *init*-channel and the sub-automata informs the $\phi$-automaton that their sub-formula has been rewritten to $\top$ or $\bot$ through designated channels. The automata for until and release rely on having multiple automata for their sub-formulas that they can start one of after each observation. If there are insufficient sub-automata an error state is reached - because of this the encoding is an under-approximation of the WMTL formula in question.

*Example 3.* Consider the run $(p, \{\tau \mapsto t_0\})(p, \{\tau \mapsto t_1\})(p, \{\tau \mapsto t_2\})(\neg p, \{\tau \mapsto t_3\})(p, \{\tau \mapsto t_4\})(p, \{\tau \mapsto t_5\})(p, \{\tau \mapsto t_6\})(p, \{\tau \mapsto t_7\})(?, \{\tau \mapsto t_8\})$ where we do not know if the proposition $p$ is true at time $t_8$ and let $t_8 - t_0 > 10$. In Fig. 3 we provide a snapshot of the set of active automata at time $t_7$. At the top we have an automaton that monitors the expression $\Diamond_{[0;10]}\Box_{[4;15]}p$ which has been active since $t_0$ thus it has $10 - (t_7 - t_0)$ time units left before its expression has been violated.

Below this automaton are automata observing the subexpression $\Box_{[4;15]}p$. These automata have been started at times $t_4, t_5$ and $t_6$ respectively and will

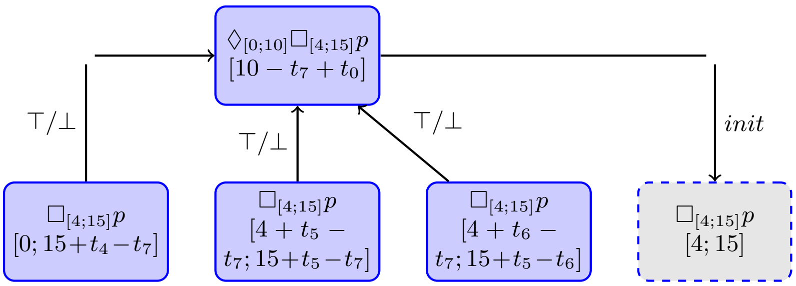
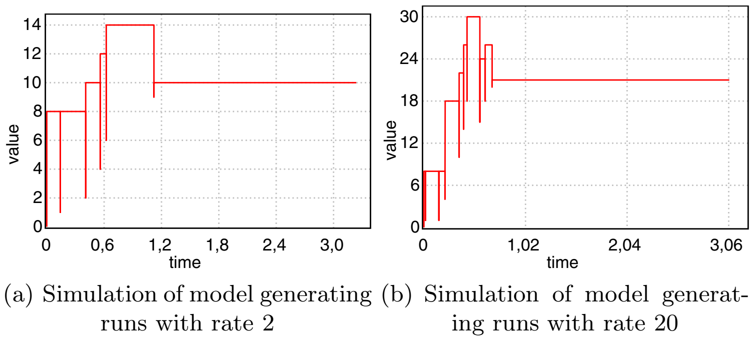


**Fig. 3.** Snapshot at time $t_7$ with 3 active automata and one being started.



(a) Simulation of model generating runs with rate 2 (b) Simulation of model generating runs with rate 20

**Fig. 4.** Plots of how the size of the formula varies over time. On the $y$-axis is plotted the number of active automata and the $x$-axis contain the time.

report $\top$ to the parent automaton at the moment they have oberved $p$ for 15 time units or $\bot$ if they observe $\neg p$. Notice that all the automata started before $t_4$ are no longer active since $\neg p$ was true at time $t_3$. Also, there is one automaton (the gray one with dashed borders) that is being started by $\Diamond_{[0;10]}\Box_{[4;15]}p$ through its *init*-channel. Since $t_8 - t_0 > 0$ the top level automaton will not start any sub-automata at time $t_8$. Instead it will merely wait for the already started automata to return either $\top$ or $\bot$. If one of them return $\top$ then the top-level automaton will return $\top$. In case all of the sub-automata return $\bot$ then the top-level automata will return $\bot$.

We encoded the formula $\Diamond_{[0;1]}(p \wedge \Box_{[0;1]}(\neg r) \wedge \Diamond_{[0;1]}(q))$, and put the resulting automata in parallel with an automata generating random runs and an automaton incrementing a counter whenever an automaton was started or decremented the counter, whenever an automaton stopped. We did this for transition rates 2 and 20 of the random run generating automaton and used the simulate query, `simulate 1 [<=3] size`.

In Fig. 4 we show the plots we obtain for runs generated with varying transition rates. One can easily see that the number of automata does not increase exponentially. We have observed the phenomena on various case studies.

### 5.2 IEEE 802.15.4 CSMA/CA Protocol

IEEE 802.15.4 standard [18] specifies the physical and media access control layers for low-cost and low-rate wireless personal area networks. Devices operating in such networks share the same wireless medium and can possibly corrupt the transmission of each other by sending data at the same time. We applied our technique to the analysis of Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) network contention protocol that is used in IEEE 802.15.4 to minimise the number of collisions.

Our objective is to estimate the probability that if a collision occurs, then all nodes participating in it will recover from the collision within a given time bound. This can be specified with $\wedge_{i=1..N}\varphi_i$, where $N$ is a number of network nodes and $\varphi_i$ specifies the behavior of a single node:

$$\varphi_i \equiv \Box_{\leq 10000}(collision_i \rightarrow \Diamond_{\leq 4000} send_i)$$

The monitor built by [7] is precise. We are thus not interested to reason on precision of the confidence interval, but rather on the evolution of computation time. In Fig. 5 we observe that both the size of intermediary formulas used to rewrite $\varphi$ and the computation time grow linearly as the number of components $N$ increases. On the other hands, both the size of monitor and the computation time with the approach in [7] grow exponentially and cannot be applied to real-life deployments of CSMA.

| Number of nodes | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Monitor-based approach (time) | <1s | 3s | 57s | 20m2s | - |
| Size of the monitor | 230 | 2049 | 16306 | 123800 | - |
| Rewrite-based approach (time) | 55s | 2m85s | 4m11s | 6m32s | 9m21.47s |
| Average formula size | 8.98 | 13.76 | 19.24 | 24 | 30.34 |

**Fig. 5.** Results for the CSMA/CA protocol

Although the monitor-based approach is faster for smaller $N$, for larger $N$ it quickly becomes intractable, while the rewrite-based approach scales well.

## 6 Conclusion

We presented a new monitoring procedure for WMTL formulas. The technique relies on a series of rewriting step for the formula and is guaranteed to terminate. Contrary to automata-based approaches, ours is precise in the sense that it does not depend on over and under approximation of the formula. We have implemented our approach in UPPAAL-SMC. Our results outperform those of the monitor-based approaches.

## References

1. Bauer, A., Leucker, M., Schallhart, C.: Comparing ltl semantics for runtime verification. J. Log. Comput. 20(3), 651–674 (2010)
2. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM TESMy 20(4), 14 (2011)
3. 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)
4. 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)
5. Bouyer, P., Larsen, K.G., Markey, N.: Model checking one-clock priced timed automata. Logical Methods in Computer Science 4(2) (2008)
6. Bouyer, P., Markey, N.: Costs Are Expensive! In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 53–68. Springer, Heidelberg (2007)
7. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Li, G., Poulsen, D.B., Stainer, A.: Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18 2012. LNCS, vol. 7180, pp. 168–182. Springer, Heidelberg (2012)
8. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical Model Checking for Networks of Priced Timed Automata. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 80–96. Springer, Heidelberg (2011)
9. Drusinsky, D.: The Temporal Rover and the ATG Rover. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 323–330. Springer, Heidelberg (2000)
10. Gong, H., Zuliani, P., Komuravelli, A., Faeder, J.R., Clarke, E.M.: Computational Modeling and Verification of Signaling Pathways in Cancer. In: Horimoto, K., Nakatsui, M., Popov, N. (eds.) ANB 2010. LNCS, vol. 6479, pp. 117–135. Springer, Heidelberg (2012)
11. Havelund, K., Roşu, G.: Synthesizing Monitors for Safety Properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002)
12. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate Probabilistic Model Checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004)
13. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A Bayesian Approach to Model Checking Biological Systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
14. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
15. Maler, O., Nickovic, D., Pnueli, A.: Real Time Temporal Logic: Past, Present, Future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005)
16. Rosu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Autom. Softw. Eng. 12(2), 151–197 (2005)

17. Sen, K., Viswanathan, M., Agha, G.: Statistical Model Checking of Black-Box Probabilistic Systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004)
18. I. C. Society. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) (2003)
19. Younes, H.L.S.: Ymer: A Statistical Model Checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005)
20. Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. STTT 8(3), 216–228 (2006)
21. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to simulink/stateflow verification. In: HSCC 2010, pp. 243–252. ACM, New York (2010)