Department of Computer Science

Uniform Sampling for Timed Automata with Application to Language Inclusion

Nicolas Basset, Benoît Barbot, Marta Kwiatkowska
and Marc Beunardeau

CS-RR-16-04

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD

Uniform Sampling for Timed Automata with Application to Language Inclusion Measurement*

Benoît Barbot[1]**, Nicolas Basset[1], Marc Beunardeau[2]***, and Marta Kwiatkowska[1]

[1] Department of Computer Science, University of Oxford, United Kingdom
[2] Ingenico Labs, Paris, France, and École Normale Supérieure, Paris, France

Abstract. Monte Carlo model checking introduced by Smolka and Grosu is an approach to analyse non-probabilistic models using sampling and draw conclusions with a given confidence interval by applying statistical inference. Though not exhaustive, the method enables verification of complex models, even in cases where the underlying problem is undecidable. In this paper we develop Monte Carlo model checking techniques to evaluate quantitative properties of timed languages. Our approach is based on uniform random sampling of behaviours, as opposed to isotropic sampling that chooses the next step uniformly at random. The uniformity is defined with respect to volume measure of timed languages previously studied by Asarin, Basset and Degorre. We improve over their work by employing a zone graph abstraction instead of the region graph abstraction and incorporating uniform sampling within a zone-based Monte Carlo model checking framework. We implement our algorithms using tools PRISM, SageMath and COSMOS, and demonstrate their usefulness on statistical language inclusion measurement in terms of volume.

1 Introduction

Since the seminal work of Alur and Dill [1], timed automata (TAs) have been widely studied in the context of real-time systems verification. Several algorithms from the classical automata-theoretic verification were successfully lifted to the timed case. In spite of this, many problems become undecidable, the most important being the inclusion of timed languages. One way to circumvent undecidability is to employ statistical methods, where results are given with some confidence level. However, timed automata are non-stochastic models and it is not clear a priori with what probability to sample runs when performing statistical experiments. A natural answer is given by the maximal entropy principle: "without knowledge a priori on the distribution of probability to be taken, the

* This work is supported by ERC AdG VERIWARE.
** Now in LACL, Université Paris Est Créteil, France
*** Contributed to the work during an internship funded by ERC AdG VERIWARE

one with maximal entropy should be preferred" [16]. A maximal entropy stochastic process for timed automata was recently proposed in [7]. Essentially, this is the stochastic process that yields the most uniform sampling when the length of the timed words tends to infinity. By uniform sampling we mean that all timed words of a given length have the same density of probability to be chosen.

In this paper we propose several algorithms to achieve uniform sampling of timed words in timed languages. The methods are based on the theory of volumetry of timed languages recently developed by Asarin, Degorre and Basset [3], which provides means for quantitative measurement of languages in terms of volume. Here, we employ this theory to achieve statistical estimation of volume and demonstrate its usefulness for language inclusion measurement. The accuracy of statistical estimation depends on the ability to uniformly sample the executions. The method provided in [7], where the transitions of a TA were annotated with probability functions so that the resulting stochastic process enables random simulation in the most uniform way possible, is based on spectral attributes of a functional operator Ψ (an analogue in the TA context of the adjacency matrix of a graph) [3]. Unfortunately, it is not practical, as it relies on the region graph abstraction and the computation of eigenfunctions. In this paper, we overcome this problem by adopting a zone-based approach and approximating the probability functions of [7] with quotients of the volume functions.

Contributions. (i) We provide a zone-based computation of volume functions for TAs, which enables the first practical implementation of volumetry of timed languages. (ii) We develop three methods (Method 1-3) to sample in a (quasi) uniform manner timed words in a language recognised by a deterministic timed automaton (DTA). In particular, we propose a receding horizon framework that allows us to approximate the maximal entropy stochastic process discussed above. (iii) We apply uniform sampling for DTAs to uniform sampling and volume measurement for arbitrary timed languages, provided the membership problem for the language is decidable. (iv) We have implemented the algorithms presented here in PRISM [18] (for the splitting of the DTA into zones), SageMath [22] (for the computation of volume functions) and COSMOS [4] (for the random generation of timed words and property checking) and illustrate them on several examples, with encouraging results.

This report is an extended version of [5].

Related work. The theory of volumetry of timed languages has been studied and applied to robustness analysis [3], timed channel coding [2] and combinatorics of permutations [6], but has not yet been applied in practice.

The *recursive method* for uniform sampling is a well-known method in discrete combinatorics [13] whose generalisation to the timed case (Method 1 here) was already done for very specific timed languages in [6].

Monte Carlo model checking was proposed in [14] for discrete models to randomly explore their behaviour by means of simulating execution paths. Similarly, statistical model checking [23] uses simulation to verify temporal logic properties with statistical guarantees, and has been applied to stochastic timed/hybrid

systems [11]. This avoids state-space explosion, thus ensuring the feasibility of verification of complex models, and has also been used to check undecidable properties [11]. Here we implement Monte Carlo techniques for TAs.

Monte Carlo or statistical model checking usually employs an *isotropic* random walk to explore the executions (as explained in [21,12] for discrete models). This involves choosing uniformly at random, at each step of the simulation, the next transition from those available. It has been argued that the isotropic methods are not able to efficiently perform uniform sampling of the behaviours (see e.g. the pathological examples in [21] for sampling of lassos and [12] for sampling paths in a finite-state automaton). Here we implement uniform sampling based on the tool COSMOS, but the techniques are more generally applicable and can be implemented in other tools, for example UPPAAL-SMC [11], which supports user-defined distributions.

Statistical model checkers such as UPPAAL-SMC consider timed automata augmented with probability distribution on transitions that are either user-defined or given "by default". Thus, the model to verify is already probabilistic and specifications are written in temporal logic with probabilistic operators. Our work addresses a different and novel question: how can one use statistical experiments on a non-probabilistic timed language and draw conclusions about that language, without being given probability distributions on it?

Paper structure We first recall definitions and notions from volumetry of timed languages in Sect. 2. Then in Sect. 3, we show how to preprocess deterministic timed automata in order to compute the volumes of their languages. For the resulting split automata, we show how to perform quasi-uniform random sampling in Sect. 5. We describe applications and experiments for our methods in Sect. 4, and give conclusions and perspectives in Sect. 6. To improve readability we have relegated the proofs and further details to the Appendix.

2 Preliminaries

2.1 Timed languages and volumetry

A *timed word* $\boldsymbol{\alpha} = (t_1, a_1)\dots(t_n, a_n)$ is a word over the alphabet $\mathbb{R}_{\geq 0} \times \Sigma$, where $\mathbb{R}_{\geq 0}$ denotes the set of non-negative reals and Σ is a finite alphabet of *events*. Times t_i represent *delays* between events a_{i-1} and a_i. Throughout this paper, delays will be bounded[3] by an integer constant M. A *timed language* L is a set of timed words. Given $n \geq 0$, we denote by L_n the timed language L restricted to timed words of length n. For every timed language L and every word $w = a_1 \dots a_n \in \Sigma^n$, we define $P_w^L = \{(t_1, \dots, t_n) \mid (t_1, a_1)\dots(t_n, a_n) \in L\}$, and denote by $\texttt{Vol}(P_w^L)$ its (hyper-)volume.

Example 1 (Running example). Examples of such hyper-volumes are given in Fig. 1. Anticipating what follows, these sets correspond to the timed language restricted to timed words of length 2 of the TA depicted in Fig. 2 (Left).

[3] Our approach to timed languages is based on volume and does not apply, in its present form, to unbounded delays that result in infinite volume.

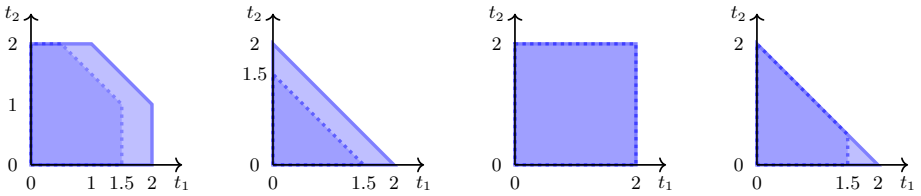

Fig. 1. From left to right, languages P^L_{ab}, P^L_{aa}, P^L_{ba} and P^L_{bb} for the running example (Example 1). The darker areas corresponds to initial clock vector $(x, y) = (0.5, 0)$.

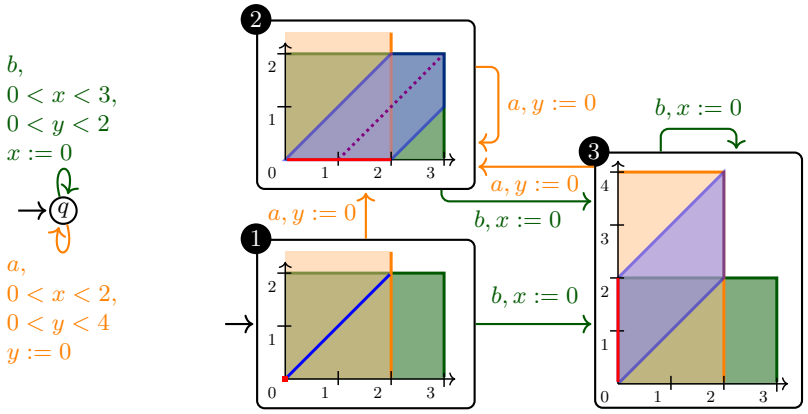

Fig. 2. Left: a DTA. Right: the same DTA obtained after applying the forward reachability algorithm. Entry zones are represented in red. Guards for a and b are the same in the two TAs. The blue part represents clock vectors reachable through entry zones by time elapsing. In location 2, the guard of transition b should be split along the dotted line to obtain the split DTA of Fig 3.

For a fixed n, we define the *n-volume* of L as follows:

$$\texttt{Vol}(L_n) = \sum_{w \in \Sigma^n} \texttt{Vol}(P^L_w) = \sum_{a_1 \in \Sigma} \int_0^M \cdots \sum_{a_n \in \Sigma} \int_0^M \mathbf{1}_{P^L_w}(\boldsymbol{t}) dt_1 \cdots dt_n.$$

Continuing the example; the hyper-volume for dimension 2 is calculated as

$$\texttt{Vol}(L_2) = \texttt{Vol}(P^L_{ab}) + \texttt{Vol}(P^L_{aa}) + \texttt{Vol}(P^L_{ba}) + \texttt{Vol}(P^L_{bb}) = 3.5 + 2 + 4 + 2 = 11.5.$$

We define the *uniform probability distribution* on a timed language L by assigning weight $1/\texttt{Vol}(L_n)$ to every timed word of length n. The main purpose of this article is to show how to sample according to that distribution when the language is recognised by a timed automaton. For instance, the probability of a uniformly sampled timed word to fall in the set $E = \{(t_1, b)(t_2, a) \mid t_1 \in (0, 1), t_2 \in (0, 2)\}$ is $\texttt{Vol}(E)/\texttt{Vol}(L_2) = 2/11.5 \approx 0.17$.

Given two timed languages L, L' over the same alphabet of events Σ, we say that L' is a *tight under-approximation* of L if, for all $w \in \Sigma^*$, $P_w^{L'} \subseteq P_w^L$ and $\mathtt{Vol}(P_w^L \setminus P_w^{L'}) = 0$; hence $\mathtt{Vol}(P_w^L) = \mathtt{Vol}(P_w^{L'})$. In particular, timed words uniformly sampled in L' are uniformly sampled in L.

2.2 Timed automata

Let X be a finite set of non-negative real-valued variables called *clocks*. Here we assume that clocks remain bounded by a constant $M \in \mathbb{N}$. A *clock constraint* has the form $x \sim c$ or $x - y \sim c$ where $\sim \in \{\leq, <, =, >, \geq\}$, $x, y \in X$, $c \in \mathbb{N}$. A *guard* is a finite conjunction of clock constraints; it is called open if its constraints involve only strict inequalities. A *zone* is a set of clock vectors $\boldsymbol{x} \in [0, M]^X$ satisfying a guard. For a clock vector $\boldsymbol{x} \in [0, M]^X$ and a non-negative real t, we denote by $\boldsymbol{x} + t$ (resp. $\boldsymbol{x} - t$) the vector $\boldsymbol{x} + (t, \ldots, t)$ (resp. $\boldsymbol{x} - (t, \ldots, t)$).

A *timed automaton* (TA) $\mathcal{A}$ is a tuple $(\Sigma, X, Q, i_0, F, \Delta)$ where Σ is a finite set of events; X is a finite set of clocks; Q is the finite set of *locations*; i_0 is the initial location; $F \subseteq Q$ is the set of final locations; and Δ is the finite set of *transitions*. Any transition $\delta \in \Delta$ has an *origin* $\delta^- \in Q$, a *destination* $\delta^+ \in Q$, a label $a_\delta \in \Sigma$, a *guard* $\mathfrak{g}_\delta$ and a *reset* function $\mathfrak{r}_\delta$ determined by a subset of clocks $B \subseteq X$: it resets to 0 all the clocks in B and does not modify the value of the other clocks.

A *timed transition* is an element (t, δ) of $\mathbb{A} \stackrel{\text{def}}{=} [0, M] \times \Delta$. The *delay* t represents the time before firing the transition δ. A *state* $s = (q, \boldsymbol{x}) \in Q \times [0, M]^X$ is a pair of a location and a clock vector. Given a state $s = (q, \boldsymbol{x})$ and a timed transition $\alpha = (t, \delta) \in \mathbb{A}$, the *successor* of s by α is denoted by s_α and defined as follows. If $\delta^- = q$ and $\boldsymbol{x} + t$ satisfies the guard $\mathfrak{g}_\delta$ then $s_\alpha = (\delta^+, \mathfrak{r}_\delta(\boldsymbol{x} + t))$ else $s_\alpha = \bot$. Here and in the rest of the paper $\bot$ represents undefined states. A sequence of timed transitions is called a *timed path*. We extend the successor action to timed paths by induction: $s_\varepsilon = s$ and $s_{(\alpha\boldsymbol{\alpha}')} = (s_\alpha)_{\boldsymbol{\alpha}'}$ for all states s, timed transitions $\alpha \in \mathbb{A}$ and timed paths $\boldsymbol{\alpha}' \in \mathbb{A}^*$. The initial state of the timed automaton is $s = (i_0, \mathbf{0})$. The *labelling* of a timed path $(t_1, \delta_1) \ldots (t_n, \delta_n)$ is the timed word $(t_1, a_{\delta_1}) \ldots (t_n, a_{\delta_n}) \in ([0, M] \times \Sigma)^*$. The *timed language* $L(\mathcal{A})$ of a timed automaton $\mathcal{A}$ is the set of timed words that are labellings of timed paths $\boldsymbol{\alpha}$ such that $s_{\boldsymbol{\alpha}} \in F \times [0, M]^X$. We also write $L_n(\mathcal{A})$ instead of $(L(\mathcal{A}))_n$.

For a guard g, we denote by $\mathtt{TE}^{-1}(g)$ the set of clock vectors from which g can be reached when time elapses; formally, $\mathtt{TE}^{-1}(g) = \{\boldsymbol{x} \mid \exists t \geq 0, \boldsymbol{x} + t \in g\}$. Given a state $s = (q, \boldsymbol{x})$ we denote by $\Delta(s)$ the set of transitions *available* from s, that is, such that $\delta^- = q$ and $\boldsymbol{x} \in \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$. Given a state $s = (q, \boldsymbol{x})$ and a transition $\delta \in \Delta(s)$, we define $\mathrm{lb}_\delta(s) \stackrel{\text{def}}{=} \inf\{t | \boldsymbol{x} + t \in \mathfrak{g}_\delta\}$ and $\mathrm{ub}_\delta(s) \stackrel{\text{def}}{=} \sup\{t | \boldsymbol{x} + t \in \mathfrak{g}_\delta\}$ so that the condition $\boldsymbol{x} + t \in \mathfrak{g}_\delta$ is equivalent to $t \in (\mathrm{lb}_\delta(s), \mathrm{ub}_\delta(s))$.

A *deterministic timed automaton* (DTA) is a TA such that no clock vector can satisfy guards of pairwise distinct transitions with the same label and origin. This implies that timed words and timed paths of a DTA are in one-to-one correspondence. We are interested in the prefixes of infinite timed words of a DTA. To be sure that $L_n(\mathcal{A})$ contains exactly the prefixes of size n, we consider

only DTAs that satisfy the two following conditions: (i) every location is final, (ii) from every reachable state, there is a timed transition that can be taken.

2.3 Equations on timed languages and volumes

Given a DTA $\mathcal{A}$, we denote by $L_n(s)$ the n-th *timed language* recognised from a state s and defined inductively as follows: $L_0(s) = \{\varepsilon\}$, and

$$L_{n+1}(s) = \bigcup_{\delta \in \Delta(s)} \bigcup_{t \in I(s,\delta)} (t, a_\delta) L_n(s_{(t,\delta)}). \tag{1}$$

For the running example and initial state $[q, (0.5, 0)]$ we have:

$$L_2([q,(0.5,0)]) = \bigcup_{t \in (0,1.5)} (t,a) L_1([q,(0.5+t,0)]) \cup \bigcup_{t \in (0,2)} (t,b) L_1([q,(0,t)]). \tag{2}$$

The language $L_2([q, (0.5, 0)])$ is depicted in Fig 1.

We also parametrise the volume by the initial state and define the n-th volume function as $v_n(s) = \texttt{Vol}(L_n(s))$. These functions can be defined recursively by replacing union over intervals by integrals and union over transitions by finite sums in (1). We obtain $v_0(s) = 1$ and

$$v_{n+1}(s) = \sum_{\delta \in \Delta(s)} \int_{\mathrm{lb}_\delta(s)}^{\mathrm{ub}_\delta(s)} v_n(s_{(t,\delta)}) \mathrm{d}t. \tag{3}$$

For the running example, passing to volumes in (2) yields

$$v_2([q,(0.5,0)]) = \int_0^{1.5} v_1([q,(0.5+t,0)])\mathrm{d}t + \int_0^2 v_1([q,(0,t)])\mathrm{d}t. \tag{4}$$

A key idea used in [3,7] is to rewrite (3) as

$$v_{n+1}(s) = \Psi(v_n)(s) \tag{5}$$

where Ψ is an integral operator defined by

$$\Psi(f)(s) = \sum_{\delta \in \Delta(s)} \Psi_\delta(f)(s) \quad \text{with} \tag{6}$$

$$\Psi_\delta(f)(s) = \int_{\mathrm{lb}_\delta(s)}^{\mathrm{ub}_\delta(s)} f(s_{(t,\delta)})\mathrm{d}t. \tag{7}$$

Thus, volume functions are defined via iteration of the operator Ψ on the constant function $\mathbf{1}$: $v_n = \Psi^n(\mathbf{1})$. In [3,7], the state space was decomposed into regions, which guaranteed algebraic properties such as polynomial volume functions at the price of an explosion of the number of locations of the TA. A TA before such a decomposition into regions has volume functions that are complicated

Table 1. First volume functions $v_n[l_i,(x,y)]$ associated to the TA of Fig. 3.

	$[l_0,(0,0)]$	$[l_1,(x,0)]$	$[l_2,(0,y)]$	$[l_3,(x,0)]$
v_0	1	1	1	1
v_1	4	$-x+4$	$-y+4$	$-2x+5$
v_2	15	$-4x+15$	$\frac{1}{2}y^2-4y+15$	$-\frac{1}{2}x^2-6x+\frac{35}{2}$
v_3	$\frac{335}{6}$	$-15x+\frac{335}{6}$	$-\frac{1}{6}y^3+2y^2-15y+\frac{335}{6}$	$-\frac{1}{6}x^3-\frac{1}{2}x^2-25x+\frac{133}{2}$

(piecewise defined), and hence difficult to handle in practice. Here we want to keep volume functions simple (polynomial) while keeping the set of locations small. For this we adopt a zone-based approach.

The idea of the zone-based decomposition described in the next section is to split the state space into several pieces in which the functions $\mathrm{lb}_\delta(s)$ and $\mathrm{ub}_\delta(s)$ have simple form, ensuring that every volume function $v_n = \Psi^n(\mathbf{1})$ restricted to any location is polynomial (see Table 1).

3 Volume function computation for DTAs

In this section we explain how to transform a DTA $\mathcal{A}$ into a DTA $\mathcal{A}'$ called *split DTA* that facilitates efficient volume computation.

Decomposition into zones. We first apply a forward reachability algorithm, implemented for instance in PRISM [18], which returns the so-called *forward-reachability graph*, that is, a finite graph with annotations, which we view as a DTA (the annotations are essentially, for each edge δ, the guard $\mathfrak{g}_\delta$ and label a_δ and, for each location l, the zone Z_l which is entered). Formally, we say that a TA is *decomposed into zones* if, for every $l \in Q$, there is a zone Z_l called the *entry zone* of l, such that the entry zone of the initial state is $\{\mathbf{0}\}$ and, for every transition δ, the successors of states in $\{\delta^-\} \times Z_{\delta^-}$ through δ with some delay are in $\{\delta^+\} \times Z_{\delta^+}$, that is, $\{\mathfrak{r}_\delta(\boldsymbol{x}+t) \mid \boldsymbol{x} \in Z_{\delta^-}, \boldsymbol{x}+t \in \mathfrak{g}_\delta\} \subseteq Z_{\delta^+}$. We denote by $\mathbb{S} = \cup_{l\in Q}\{l\} \times Z_l$ the set of states corresponding to entry zones. The forward-reachability graph for the running example is given in Fig. 2 (Right).

Guard split. Let δ be the transition from location 2 to location 3 in the automaton of Fig. 2 (Right), then $g_\delta \stackrel{\text{def}}{=} (0<x<3) \wedge (0<y<2)$. Then one can see that $\mathrm{ub}_\delta(2,(x,0)) = 2$ if $x \in (0,1)$ (due to guard $y<2$) and $\mathrm{ub}_\delta(2,(x,0)) = 3-x$ if $x \in (1,2)$ (due to guard $x<3$). The guard g_δ thus needs to be split into two (along the dotted line in the figure) to achieve a simpler form for ub_δ. It is well known how to get the tightest constraints of a guard and get rid of redundant constraints using the Floyd-Warshall algorithm (see e.g. [8]). A guard is said to be *upper-split* (*lower-split*) if there is at most one useful constraint (that is, not implied by other constraints) of the form $x_j < a$ ($x_j > a$). The guard g_δ discussed above is not upper-split as the two constraints $x<3$ and $y<2$ are both useful. Analogous definitions hold for lower-bounds and a guard is said to be *split* if it is both lower-split and upper-split.

Pre-stability. A second phenomenon we want to avoid is when the set of available transitions $\Delta(q, \boldsymbol{x})$ is not constant on the entry zone of q. A TA decomposed into zones is called *pre-stable* if, for every location q and clock vector $\boldsymbol{x} \in Z_q$, the set of transitions $\Delta(q, \boldsymbol{x})$ is exactly the set of transitions δ whose origin is q. Equivalently, a TA is pre-stable if $Z_{\delta^-} \subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$ for every δ. In case we detect a transition such that $Z_{\delta^-} \not\subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$ we will split the zone Z_{δ^-} to isolate $\mathtt{TE}^{-1}(\mathfrak{g}_\delta) \cap Z_{\delta^-}$ from its complement. Continuing the example above, after splitting g_δ the functions associated to each new guard are null for $x \in (0, 1)$ or $x \in (1, 2)$. Location 2 is split into two locations of the final TA of Fig. 3: l_1 for $(0, 1)$ and l_3 for $(1, 2)$. Every incoming transition to location 2 is split accordingly into two transitions (one orange to l_1 and one purple to l_3).

Trimming. Last but not least, we say that a TA is *trimmed* if the set of outgoing transitions of each location is non-empty. A TA is called *split* if it is pre-stable, trimmed and all the guards of its transitions are split and open. It implies, in particular, that, for every entry state $s \in \mathbb{S}$, $\Delta(s)$ is not empty and for all transition $\delta \in \Delta(s)$ it holds that $\mathrm{ub}_\delta(s) - \mathrm{lb}_\delta(s) > 0$. Note that *opening guards*, that is, transforming non-strict inequalities into strict ones is made wlog., as it only removes part of the language that has null volume measure.

Splitting algorithm. We propose an algorithm to transform a DTA into a split DTA such that the language of the latter is a tight under-approximation of the language of the former (see Theorem 1). First, we apply a forward reachability algorithm to obtain a DTA decomposed into zones and open its guards. Then we successively split zones that falsify pre-stability and guard split conditions, until the conditions are satisfied in the DTA. The splitting algorithm maintains a stack of transitions that need to be checked, which initially contains all the transitions. As the algorithm proceeds, transitions are popped from the stack and are checked against pre-stability and guard split conditions. If one test fails, the zone (or guard) is split accordingly into several zones (or open guards) and the transitions that are affected are added to the stack (incoming transitions to, and outgoing transitions from the split zone). When no more transition need to be checked (i.e. the stack is empty), the TA is split and the algorithm terminates. This occurs in a finite number of steps since transitions are added to the stack only when a zone is split into strictly smaller sub-zones, and there are finitely many zones (as the clocks are bounded by a constant M).

Theorem 1. *Given a DTA $\mathcal{A}$, one can construct (using the algorithm sketched above) a split DTA $\mathcal{A}'$ that recognises a tight under-approximation of $L(\mathcal{A})$.*

The splitting algorithm and the proof can be found in Appendix A.

Volume function of a split DTA. We have the following result.

Proposition 1. *Given a split DTA $\mathcal{A}$ and $n \in \mathbb{N}$, denote by c the maximal affine dimension of an entry zone of $\mathcal{A}$. One can compute the volume function v_k for $k \leq n$ in time and space complexity $O(n^{c+2}|Q_\mathcal{A}|)$ using dynamic programming*

based on the recursive equation (3). *Each volume function* v_k *restricted to a location* q *is a polynomial of degree at most* k *that is positive on* Z_q.

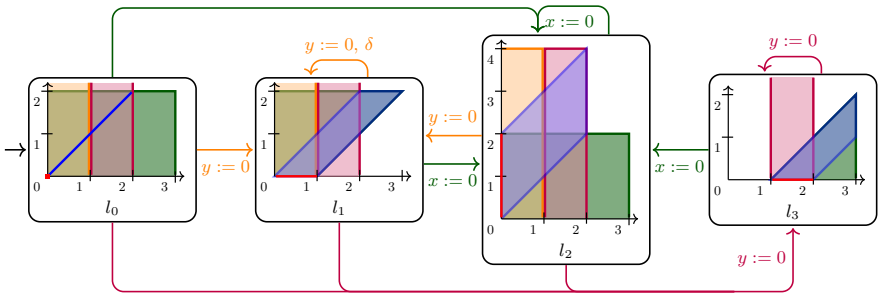

Fig. 3. The split form of the running example (Example 1).

Example 2. We have implemented the splitting algorithm sketched in Sect. 3 and applied it to the DTA of Fig. 2 (Right) to obtain the DTA of Fig. 3. Our program also returns for each transition δ of the output DTA the interval $(\mathrm{lb}_\delta, \mathrm{ub}_\delta)$, allowing us to compute with SageMath the operator Ψ as well as the volume functions. On the example, for $f : \mathbb{S} \to \mathbb{R}$, $(x, y) \in Z_l$ with $l \in \{l_0, \ldots, l_3\}$,

$$\Psi(f)[l_0,(0,0)] = \int_0^1 f(l_1,(t,0))\mathrm{d}t + \int_0^2 f(l_2,(0,t))\mathrm{d}t + \int_1^2 f(l_3,(t,0))\mathrm{d}t;$$
$$\Psi(f)[l_1,(x,0)] = \int_0^{1-x} f(l_1,(x+t,0))\mathrm{d}t + \int_0^2 f(l_2,(0,t))\mathrm{d}t + \int_{1-x}^{2-x} f(l_3,(x+t,0))\mathrm{d}t;$$
$$\Psi(f)[l_2,(0,y)] = \int_0^1 f(l_1,(t,0))\mathrm{d}t + \int_0^{2-y} f(l_2,(0,y+t))\mathrm{d}t + \int_1^2 f(l_3,(t,0))\mathrm{d}t;$$
$$\Psi(f)[l_3,(x,0)] = \int_0^{3-x} f(l_2,(0,t))\mathrm{d}t + \int_0^{2-x} f(l_3,(x+t,0))\mathrm{d}t.$$

First volume functions computed using Equation (5) are given in Table 1.

4 Sampling methods for timed languages of DTAs

In this section we consider random sampling of timed words. We first give a method that achieves exact uniform sampling when the length of timed words to be generated is finite; we speak of finite horizon. When the length is infinite or too long to be treated by the previous method, we consider a receding horizon method, where, at the k-th step of the generation, the next timed letter is chosen according to the volume of the timed words for the next m steps; these possible futures constitute a finite receding horizon. At the limit, where the receding horizon becomes infinite ($m \to \infty$), this can be interpreted as a stochastic process over runs of maximal entropy [7].

Parametric probability distributions. A *discrete probability distribution* (DPD) on a finite set A is a function $\mathtt{dpd} : A \to [0,1]$ such that $\sum_{a\in A} \mathtt{dpd}(a) = 1$. A *probability density function* (PDF) on an interval (a,b) is a Lebesgue measurable function $\mathtt{pdf} : (a,b) \to \mathbb{R}_{\geq 0}$ such that $\int_a^b \mathtt{pdf}(t)\mathrm{d}t = 1$. Values of DPD and

PDF are referred to as weights. The DPD $\texttt{isoDPD}(A)$ on a set A (resp. the PDF $\texttt{isoPDF}(a,b)$ on an interval (a,b)) that attributes the same weight to every $a \in A$ (resp. $t \in (a,b)$) is called *isotropic*. In other words, $\texttt{isoDPD}(A)(a) = 1/|A|$ for every $a \in A$ (resp. $\texttt{isoPDF}(a,b)(t) = 1/(b-a)$ for every $t \in (a,b)$). PDFs considered in the following are just polynomials on the delay variable t. Their coefficients depend on the current state (location and clock values) and on the transition to fire. Choosing a delay t according to a PDF can be done using the *inverse method*: a random number r is drawn uniformly in $(0,1)$, and the output $t \in (a,b)$ is the unique solution of $\int_a^t \texttt{pdf}(t')\mathrm{d}t' - r = 0$. In the case of the isotropic PDF on (a,b), the output t is just $a + r(b-a)$.

Random generation of timed words in $L_n(s)$ for a given state $s \in \mathbb{S}$ is achieved as follows: for $k = 1..n$, pick randomly the next transition δ according to a DPD $\texttt{dpd}_s^k$ parametrised by the current state s, then chose the delay t in $(\mathrm{lb}_\delta(s), \mathrm{ub}_\delta(s))$ according to a PDF $\texttt{pdf}_{s,\delta}^k$ parametrised by the current state s and the transition just chosen; take the successor of s by (t,δ) as the new current state s; output (t, a_δ) and repeat the loop.

This random generation outputs timed words of $L_n(s)$ with weights given by

$$Weight[(t_1,a_1)\cdots(t_n,a_n)] \stackrel{\text{def}}{=} \prod_{k=1}^{n} \texttt{dpd}^k_{s_{k-1}}(\delta_k)\texttt{pdf}^k_{s_{k-1},\delta_k}(t_k). \tag{8}$$

where, for every $k = 1..n$, s_{k-1} is the state before the kth sampling loop, (t_k, δ_k) is the kth timed transition randomly picked during the kth sampling loop and a_k is the label of δ_k.

Isotropic and uniform sampling. Isotropic sampling[4] relies on using in each step the isotropic DPD $\texttt{isoDPD}(\Delta(s))$ and the isotropic PDF $\texttt{isoPDF}(I(s,\delta))$. These distributions are particularly simple to sample, but when the length of samples grows the probability concentrates on small sections of the runs, see Fig. 4 (Left). By contrast, *uniform sampling* for $L_n(s)$ assigns the same weight $1/v_n(s)$ to every timed word. In other words, for any measurable set $B \subseteq L_n(s)$ the probability $\texttt{Vol}(B)/v_n(s)$ to fall in this set is proportional to its measure.

The recursive method for uniform sampling. The idea of the recursive method for uniform sampling of n-length timed words from a state s is to choose the first delay t and transition δ according to well chosen DPD and PDF that depend on the volume functions v_n and v_{n-1}, and then recursively apply uniform sampling to generate an $(n-1)$-length timed word from the updated state $s_{(t,\delta)}$.

Define, for every function $f : \mathbb{S} \to \mathbb{R}_{>0}$ and state s, the DPD $\omega(f,s) : \delta \mapsto \frac{\Psi_\delta(f)(s)}{\Psi(f)(s)}$. If moreover δ is given, define the PDF $\varphi(f,s,\delta) : t \mapsto \frac{f(s_{(t,\delta)})}{\Psi_\delta(f)(s)}$ from $(\mathrm{lb}_\delta(s), \mathrm{ub}_\delta(s))$ to $\mathbb{R}_{>0}$.

[4] Note that some works consider instead sampling the delay first and then the transitions available in the state updated by the delay (see [9]).

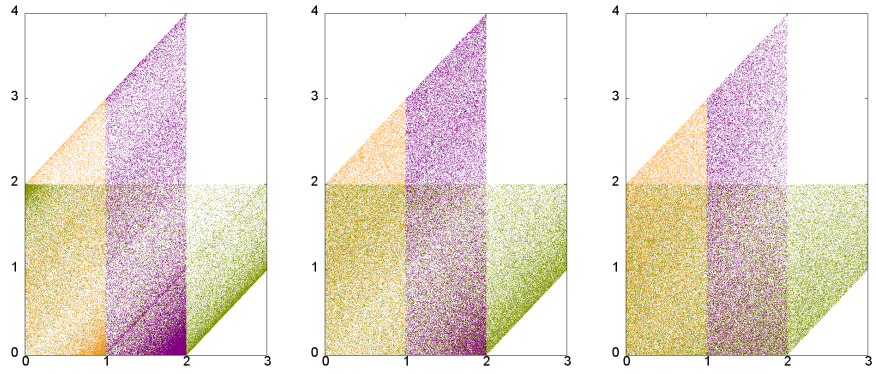

Fig. 4. Trajectories of the running example (Fig. 3) sampled using isotropic sampling (Left) and Method 2 with receding horizon $m = 0$ (Middle) and $m = 9$ (Right). Each point of a given colour corresponds to a clock vector where a transition of that colour occurs. Each plot visualises a single trajectory with $200,000$ transitions. The receding horizon $m = 9$ visibly yields the most uniform sampling. The receding horizon sampling with $m = 0$ is already more uniform than the isotropic sampling as the former assigns weights to transitions proportional to lengths of intervals $\left(\mathtt{dpd}_s = \frac{\mathrm{ub}_\delta(s) - \mathrm{lb}_\delta(s)}{v_1(s)}\right)$.

Method 1 (Exact uniform sampling) *Given a split DTA and $n \in \mathbb{N}$, precompute the volume functions $v_0 = \mathbf{1}, \ldots, v_n = \Psi^n(\mathbf{1})$ (see Proposition 1), then the uniform sampling of n-length timed words can be achieved in linear time using the following sequences of DPDs and PDFs: $(\omega(v_{n-k}, s), \varphi(v_{n-k}, s, \delta))_{k=1..n}$*

Proof. Using the same notation as in (8), it holds that

$$Weight[(t_1, a_1) \cdots (t_n, a_n)] = \prod_{k=1}^{n} \omega(v_{n-k}, s_{k-1})(\delta_k)\varphi(v_{n-k}, s_{k-1}, \delta_k)(t_k)$$

$$= \prod_{k=1}^{n} \frac{\Psi_\delta(v_{n-k})(s_{k-1})}{v_{n-k+1}(s_{k-1})} \frac{v_{n-k}(s_k)}{\Psi_\delta(v_{n-k})(s_{k-1})} = \frac{v_0(s_{n-1})}{v_n(s_0)} = \frac{1}{v_n(s_0)}.$$

Example 3. We illustrate the DPDs and PDFs used in the last but one step of the uniform random sampling for the running example, obtained from volume functions of Table 1. Consider the state $s = (l_1, (x, 0))$ with $x \in (0, 1)$ and δ the self-loop on l_1 (see Fig. 3). Then $(\mathrm{lb}_\delta(s), \mathrm{ub}_\delta(s)) = (0, 1 - x)$ and $s_{(t,\delta)} = (l_1, (x + t, 0))$. The DPD used to choose δ is

$$\mathtt{dpd}_s^{n-1}(\delta) = \frac{1}{v_2(s)} \int_{\mathrm{lb}_\delta(s)}^{\mathrm{ub}_\delta(s)} v_1(s_{(t,\delta)})dt = \int_0^{1-x} \frac{4 - x - t}{15 - 4x} dt = \frac{7 - 8x + x^2}{30 - 8x}$$

The PDF used to choose t is

$$\mathtt{pdf}_{s,\delta}^{n-1}(t) = \frac{1_{t \in (\mathrm{lb}_\delta(s), \mathrm{ub}_\delta(s))}}{\mathtt{dpd}_s^{n-1}(\delta)} \frac{v_1(s_{(t,\delta)})}{v_2(s)} = 1_{t \in (0,x)} \frac{8 - 2x - 2t}{7 - 6x + x^2}$$

Table 2. Table for Example 4.

m	$(C^+/C^-)-1$	$n_{0.01}$	m	$(C^+/C^-)-1$	$n_{0.01}$	m	$(C^+/C^-)-1$	$n_{0.01}$
0	3	1	4	3.272×10^{-4}	35	8	2.364×10^{-7}	42 098
1	0.3229	2	5	8.431×10^{-5}	124	9	2.520×10^{-8}	394 801
2	1.659×10^{-2}	3	6	9.308×10^{-6}	1 076	10	5.304×10^{-9}	1.8760×10^{6}
3	4.444×10^{-3}	6	7	1.409×10^{-6}	7 069	11	4.487×10^{-10}	2.2178×10^{7}

Random sampling with finite receding horizon. With the previous method, the k-th timed transition of a run of length n is sampled according to DPD and PDF that depend on k and n. This dependency on k and n is not suitable for large n as it requires storage of as many polynomials as the length of the run to generate n. Also, one might wish to randomly generate arbitrarily long runs without a prescribed bound on the length. To take the kth timed transition in the recursive method for uniform sampling, we use DPD and PDF that depend on v_{n-k}, that is, on the volume measure of the possible $(n-k)$ step future. The idea of the following method is to replace $(n-k)$ by a fixed $m \ll n$ at every step of the sampling. The constant m can be seen as a receding horizon used in control theory [19]. At each step we consider only the possible m step future to generate the next timed transition.

Method 2 (Random sampling with finite receding horizon m) *Given a split DTA, $n \in \mathbb{N}$ and $m \in \mathbb{N}$, precompute the volume functions $v_0 = \mathbf{1}, \ldots, v_m = \Psi^m(\mathbf{1})$ (see Proposition 1), then sample n-length timed words in linear time using the same DPD $\omega(v_m, s)$ and PDF $\varphi(v_m, s, \delta)$ for every $k = 1..n$.*

The precomputation is polynomial in m. Hence this methods is more efficient than Method 1 when $m \ll n$, but it does not yield exact uniform sampling.

Quasi-uniform random sampling. We now present a trade-off between exact uniform sampling (Method 1) and the finite receding horizon sampling (Method 2). We give bounds on the distance to uniformity for this method, which we conjecture to be small in practice for small horizon m. This conjecture is supported by theoretical results of previous works [3,7] recalled in Appendix B and by practical experiments (notably in Example 4 below).

Method 3 (Switching method for quasi-uniform sampling) *Given a split DTA, $n \in \mathbb{N}$ and $m \in \mathbb{N}$, precompute the volume functions $v_0 = \mathbf{1}, \ldots, v_m = \Psi^m(\mathbf{1})$ (see Proposition 1), then generate the $n-m$ first letters as in* Method 2 *and use* Method 1 *from the current state for the last m steps.*

This method ensures quasi-uniform sampling in the following sense.

Theorem 2. *If in* Method 3 *there exist constants $C^-, C^+ \in \mathbb{R}_{>0}$ such that $C^- v_{m+1} \leq v_m \leq C^+ v_{m+1}$, then the weight of every timed word lies in the interval $[(1-\varepsilon_{m,n})/v_n(s_0), (1+\varepsilon_{m,n})/v_n(s_0)]$, with $\varepsilon_{m,n} = (C^+/C^-)^{(n-m-1)} - 1$.*

Example 4. For the running example (Example 1) we determine the tightest constraints $C^- \stackrel{\text{def}}{=} \inf_{s\in\mathbb{S}} v_m(s)/v_{m+1}(s)$ and $C^+ \stackrel{\text{def}}{=} \sup_{s\in\mathbb{S}} v_m(s)/v_{m+1}(s)$ for $m = 0..11$. We observe empirically that C^+/C^- tends to 1 exponentially fast when m grows (see Table 2). Given a maximal tolerated error of ε, one can determine for every m the maximal n, called n_ε, such that $\varepsilon_{m,n} \leq \varepsilon$ for every $n \leq n_\varepsilon$; formally, $n_\varepsilon \stackrel{\text{def}}{=} m + 1 + \lfloor \log_{(C^+/C^-)}(1+\varepsilon) \rfloor$. First values of $n_{0.01}$ as a function of m are given in Table 2; for instance, using receding horizon for $m = 11$ one can generate timed words of length $20,000,000$ with a divergence to uniformity less than 1%.

Our sampling method requires the computation of a complete zone graph, as opposed to on-the-fly techniques used in state-of-the-art statistical model checkers; this is the price we pay for statistical evaluation of quantities of timed words in complex sub-languages as described in the next section.

5 Applications and experiments

5.1 Tackling general timed languages

It is well known that language inclusion for languages recognised by non-deterministic TAs (NTAs) is undecidable, even when a robust semantics is considered [15]. The situation is even worse for stopwatch automata, hybrid automata, etc., for which the reachability problem is undecidable. However, we can handle a statistical variant of the inclusion problem when, first, an overapproximation of the language described by a DTA is known and, second, the languages admit decision procedures for the membership problem defined as: given a language $\mathcal{L}$ and a word w, is $w \in \mathcal{L}$? Our method is based on statistical volume estimation that relies on the quasi-uniform random sampling developed in the previous section. The complexity results given below are expressed in terms of the number of membership queries one has to solve.

Application 1 (Statistical volume estimation) *Given a timed language $\mathcal{L}$, $n \in \mathbb{N}$, a confidence level θ, an error bound ε, and an over-approximation of the language recognised by a DTA $\mathcal{C}$, that is, $\mathcal{L}_n \subset L_n(\mathcal{C})$, define $N \geq (1/\varepsilon^2)\log(\theta/2)$ (Chernoff-Hoeffding bound); draw N samples uniformly at random in $L_n(\mathcal{C})$ and answer N queries for membership in $\mathcal{L}$ to return a value p such that $\mathtt{Vol}(\mathcal{L}_n)/\mathtt{Vol}(L_n(\mathcal{C}))$ lies in $[p-\varepsilon, p+\varepsilon]$ with confidence $1-\theta$.*

Application 2 (Inclusion measurement) *Given two timed languages $\mathcal{L}'$, $\mathcal{L}''$ and an over-approximation of the two languages recognised by a DTA $\mathcal{C}$ one can use the previous application with $\mathcal{L} = \mathcal{L}' \backslash \mathcal{L}''$ to evaluate the volume $\mathtt{Vol}(\mathcal{L}'_n \backslash \mathcal{L}''_n)$. If a positive value is returned, a timed word in $\mathcal{L}'_n \setminus \mathcal{L}''_n$ has been detected and one can surely claim $\mathcal{L}'_n \not\subseteq \mathcal{L}''_n$. Otherwise, a null value allows one to claim with confidence $1-\theta$ that either the inclusion holds or the difference of the two languages is smaller than $\varepsilon\mathtt{Vol}(L_n(\mathcal{C}))$.*

Application 3 (Uniform sampling) *Given a timed language $\mathcal{L}$ and $n \in \mathbb{N}$, and an over-approximation of the language recognised by a DTA $\mathcal{C}$, that is, $\mathcal{L}_n \subseteq L_n(\mathcal{C})$, draw samples uniformly at random in $L_n(\mathcal{C})$ until one falls in $\mathcal{L}_n$.*

The sampling is uniform: every timed word of $\mathcal{L}_n$ has the same density of probability to be output. The expected number of samplings in $L_n(\mathcal{C})$ to sample one timed word in $\mathcal{L}_n$ is $\texttt{Vol}(\mathcal{L}_n)/\texttt{Vol}(L_n(\mathcal{C}))$. The choice of $\mathcal{C}$ is crucial, since if $L_n(\mathcal{C})$ is a too coarse approximation of $\mathcal{L}_n$ the probability of a sample from $L_n(\mathcal{C})$ to be in $\mathcal{L}_n$ is small and the methods become inefficient. We leave as future work the design of heuristics that, given a general timed language $\mathcal{L}$, automatically generate a DTA that recognises a good over-approximation of $\mathcal{L}$.

5.2 Implementation and experiments

We implemented the techniques using three tools: PRISM [18], SageMath [22] and COSMOS [4]. The workflow is depicted in Fig. 5. We modify the tools to meet our needs. We adapted PRISM's forward reachability algorithm to implement the splitting algorithm of Sect. 3. We also export the split zone graph in a file format easy to read for SageMath. We use SageMath to compute distributions and weights of transitions as rational functions of clock valuations, which are exported and read by COSMOS in the form of a Stochastic Petri Net with general distributions. COSMOS then samples trajectories of this model, checks the membership of the language of a given NTA, and returns the probability. We have modified COSMOS to handle distributions given by arbitrary rational functions and to compute the membership of a timed word in an NTA. Our implementation can be found at `http://www.prismmodelchecker.org/files/qest16`.

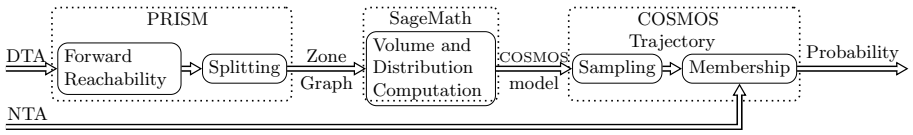

Fig. 5. Tool workflow. For the running example (Example 1), the DTA is the automaton in Fig. 2 (Left), the zone graph is the automaton in Fig. 3, the COSMOS model is the zone graph annotated with probability distributions as described in Example 3, and examples of trajectories are depicted in Fig. 4.

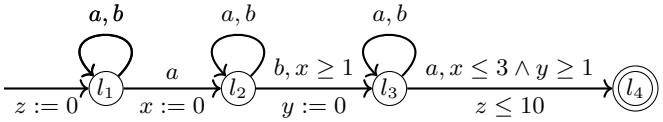

Fig. 6. Automaton $\mathcal{B}$ for Example 5. Every transition has a guard $z \leq 10$ omitted.

Example 5. Let $\mathcal{A}$ be the DTA of the running example (Example 1). The NTA $\mathcal{B}$ of Fig. 6 recognises the timed words that contain aba as a subword within the first 10 time units, where the latter a occurs at most 3 time units after

Table 3. Result of receding horizon sampling compared to isotropic sampling for the case study with two machines ($K = 2$). "Pre Time" is the pre-computation time, "Sim Time" is the simulation time. The meaning of R, $P_{50}(\mathcal{A}|\mathcal{C})$ and $P_{50}(\mathcal{B}|\mathcal{A})$ is described in the text. The receding horizon is $8 + R$. The number of samples is $100,000$.

R	Receding horizon					Isotropic		
	Pre Time	#Zones	Sim Time	$P_{50}(\mathcal{B}\|\mathcal{A})$	$P_{50}(\mathcal{A}\|\mathcal{C})$	Sim Time	$P_{50}(\mathcal{B}\|\mathcal{A})$	$P_{50}(\mathcal{A}\|\mathcal{C})$
4	45s	380	133s	0.999977	0.86539	36s	0.990439	0.03347
6	99s	581	369s	0.997717	0.58701	39s	0.975795	0.05123
8	219s	783	5005s	0.930944	0.06111	56s	0.995179	0.07052
10	417s	985	5773s	0.509091	0.00275	55s	0.999893	0.09325
12	745s	1187	7954s	0.0344828	0.00029	64s	1	0.1019

the former and there is at least 1 time unit between b and both as. We have estimated $\texttt{Vol}(L_{10}(\mathcal{A}) \cap L_{10}(\mathcal{B}))/\texttt{Vol}(L_{10}(\mathcal{A}))$ by implementing Application 1. Sampling was performed using Method 2 with $m = 5$. The result is in the interval $[0.679, 0.688]$ with confidence level 0.99; $58,000$ simulations were used in $5s$.

A case study. We additionally consider a larger case study of a failure and repair system modelled as an NTA (a detailed description of this system can be found in Appendix C). We consider a model with K machines that need to be fully repaired for the overall system to work properly. Each machine contains N levels of failure and can fail at most n_b times between two full repairs. The model is implemented by an NTA $\mathcal{A}$ with Nn_b locations and $K + 1$ clocks. The property we are interested in is encoded in another NTA $\mathcal{B}$ with 4 locations and 2 clocks. We apply our method by over-approximating the NTA $\mathcal{A}$ with a DTA $\mathcal{C}$ with $R \stackrel{\text{def}}{=} KN$ locations and 2 clocks. The results are reported in Table 3. We use our approach to sample timed words of length 50 of the DTA $\mathcal{C}$ and check their membership in $L_{50}(\mathcal{A})$ and $L_{50}(\mathcal{B})$. We compare receding horizon sampling to isotropic sampling. We observe that for isotropic sampling the probability for a timed word in $L_{50}(\mathcal{A})$ to be in $L_{50}(\mathcal{B})$ (denoted by $P_{50}(\mathcal{B}|\mathcal{A})$) tends to 1 quickly when R increases, which, for large values of R, might be interpreted as an inclusion of the languages. On the other hand, with the receding horizon sampling the same probability ($P_{50}(\mathcal{B}|\mathcal{A})$) tends to zero, which shows that the model does not satisfy the property. This result demonstrates the necessity of (quasi)-uniform sampling to explore the behaviour of the model, since the results of isotropic simulation significantly diverge from those of (quasi)-uniform simulation, and thus do not yield reliable information about the system.

We also observe that the probability for timed words in the over-approximation $L_{50}(\mathcal{C})$ to fall in $L_{50}(\mathcal{A})$ (denoted by $P_{50}(\mathcal{A}|\mathcal{C})$) tends to zero, meaning that it becomes too crude for large values of R. Thus, tight over-approximations are important to obtain efficient simulation of an NTA through a DTA.

The time required for receding horizon simulation is high compared to isotropic, since it requires sampling of complex distributions involving many polynomials.

6 Conclusion and further work

We have developed the foundations for the practical application of volumetry of timed languages to quantitative and statistical verification of complex properties for TAs. We implemented our work in a tool chain and provide first experiments.

On the theoretical side, we want to show that constants in Method 3 and Theorem 2 can be chosen to guarantee arbitrarily small divergence from exact uniform sampling and consider extending the theory to probabilistic TAs. We would also like to implement membership checking in COSMOS for general timed languages (e.g. recognised by stopwatch automata, LHA, etc.). We also plan to use our random sampling algorithms to detect forgetful cycles described in [3], which are needed to synthesise controllers robust to timing imprecision [20].

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. Eugene Asarin, Nicolas Basset, Marie-Pierre Béal, Aldric Degorre, and Dominique Perrin. Toward a timed theory of channel coding. In *FORMATS'12*, LNCS 7595, 2012.
3. Eugene Asarin, Nicolas Basset, and Aldric Degorre. Entropy of regular timed languages. *Information and Computation*, 241:142–176, 2015.
4. Paolo Ballarini, Benoît Barbot, Marie Duflot, Serge Haddad, and Nihal Pekergin. Hasl: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90(0):53 – 77, 2015.
5. Benoit Barbot, Nicolas Basset, Marc Beunardeau, and Marta Kwiatkowska. Uniform sampling for timed automata with application to language inclusion measurement. In *Proc. 13th International Conference on Quantitative Evaluation of SysTems (QEST 2016)*, LNCS. Springer, 2016. To appear.
6. Nicolas Basset. Counting and generating permutations using timed languages. In *LATIN*, LNCS 8392, pages 502–513. Springer, 2014.
7. Nicolas Basset. A maximal entropy stochastic process for a timed automaton. *Information and Computation*, 243:50–74, 2015.
8. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, 2003.
9. Dimitri Bohlender, Harold Bruintjes, Sebastian Junges, Jens Katelaan, VietYen Nguyen, and Thomas Noll. A review of statistical model checking pitfalls on real-time stochastic models. In *Leveraging Applications of Formal Methods, Verification and Validation.*, volume LNCS 8803. Springer, 2014.
10. Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.
11. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015.
12. Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, Johan Oudinet, and Sylvain Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT*, 14(1):73–93, 2012.

13. Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, 1994.
14. Radu Grosu and Scott A. Smolka. Monte carlo model checking. In *TACAS'05*, 2005.
15. Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In *HSCC 2000*, 2000.
16. E. T. Jaynes. Information Theory and Statistical Mechanics. II. *Physical Review Online Archive (Prola)*, 108(2):171–190, October 1957.
17. M.A. Krasnosel'skij, E.A. Lifshits, and A.V. Sobolev. *Positive Linear Systems: the Method of Positive Operators*. Heldermann Verlag, Berlin, 1989.
18. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV'11*, 2011.
19. Richard M Murray, John Hauser, Ali Jadbabaie, Mark B Milam, Nicolas Petit, William B Dunbar, and Ryan Franz. Online control customization via optimization-based control. *Software-Enabled Control: Information Technology for Dynamical Systems*, page 149, 2003.
20. Youssouf Oualhadj, Pierre-Alain Reynier, and Ocan Sankur. Probabilistic robust timed games. In *CONCUR'14*, 2014.
21. Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne, and Sylvain Peyronnet. Uniform monte-carlo model checking. In *FASE*, volume 6603, pages 127–140, 2011.
22. W. A. Stein et al. *Sage Mathematics Software (Version 6.9)*. The Sage Development Team, 2015. `http://www.sagemath.org`.
23. H.L.S. Younes and R.G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. and Comput.*, 204(9):1368–1409, 2006.

Appendix

This appendix includes proofs and additional details omitted from the main body of the paper, with each section corresponding to a section of the paper. We first recall definitions of regions as they are used in several sections below.

A *region* is a zone which is minimal with respect to set inclusion, e.g. the set of points (x_1, x_2, x_3, x_4) which satisfy the constraints $0 = x_2 < x_3 - 4 = x_4 - 3 < x_1 - 2 < 1$.

A timed automaton (TA) decomposed into zones is *decomposed into regions* if all the entry zones and guards are regions.

A Additional material and proofs for Section 3

We first give, in Section A.1, definitions of several additional operations on zones; then, in Section A.2, we describe the splitting algorithm sketched in the main text; in Section A.3 we prove Theorem 1; and, finally, we prove Proposition 1 in Section A.4.

A.1 Operations on zones and almost partition of zones

We describe several usual operations on zones and introduce the notion of 'almost partition' (we also apply these operations to guards, identifying a guard with the zone it describes):

(i) Time Elapse ($\mathtt{TE}$) represents the zone where we allow the time to pass as much as we want: $\boldsymbol{x} \in \mathtt{TE}(Z) \Leftrightarrow \exists \boldsymbol{x'} \in Z, t \geq 0, \boldsymbol{x} = \boldsymbol{x'} + t$.
(ii) $\mathtt{TE}^{-1}$ is the dual operation of $\mathtt{TE}$: $\boldsymbol{x} \in \mathtt{TE}^{-1}(Z) \Leftrightarrow \exists \boldsymbol{x'} \in Z, t \geq 0, \boldsymbol{x} = \boldsymbol{x'} - t \wedge \boldsymbol{x} \geq \mathbf{0}$.
(iii) $\mathfrak{r}_\delta(Z) = \{\mathfrak{r}_\delta(\boldsymbol{x}) | \boldsymbol{x} \in Z\}$: the zone obtained from Z by resetting clocks according to $\mathfrak{r}_\delta$.
(iv) $\mathfrak{r}_\delta^{-1}(Z) = \{\boldsymbol{x} \mid \mathfrak{r}_\delta(\boldsymbol{x}) \in Z\}$.
(v) $\mathtt{POST}(Z, \delta) = \mathfrak{r}_\delta(g_\delta \cap \mathtt{TE}(Z))$: the zone of clock vectors reached after having taken a timed transition (t, δ) from a state $(\delta^-, \boldsymbol{x})$ with $\boldsymbol{x} \in Z$.

Note that the condition of being decomposed into zones, namely $\{\mathfrak{r}_\delta(\boldsymbol{x}+t) \mid \boldsymbol{x} \in Z_{\delta^-}, \boldsymbol{x} + t \in \mathfrak{g}_\delta\} \subseteq Z_{\delta^+}$, can be rewritten as $\mathtt{POST}(Z_{\delta^-}, \delta) \subseteq Z_{\delta^+}$.

We write $Z' \subseteq_{\mathtt{full\text{-}dim}} Z$ if $Z' \subseteq Z$ and Z' and Z have the same affine dimension. This can be checked algorithmically as $Z' \subseteq_{\mathtt{full\text{-}dim}} Z$ iff $Z' \subseteq Z$ and every equality between clocks induced by the definition of Z' is an equality induced by the definition of Z. For a zone Z we denote by $\overline{Z}$ its topological closure, that is, obtained by making all strict inequalities non-strict in its definition. A finite family of n zones $\{Z_i\}_{i \in 1..n}$ is an *almost partition* of a zone Z if $Z_i \subseteq_{\mathtt{full\text{-}dim}} Z$ and $Z_i \cap Z_j = \emptyset$ for every $1 \leq i < j \leq n$ and $\cup_{i=1}^n \overline{Z_i} = \overline{Z}$.

A.2 Algorithms

Forward reachability algorithm The locations of a TA decomposed into zones are called loczones. The forward reachability algorithm (Algorithm 1) involves discovering step by step all the loczones reachable from the initial state, as in a graph traversal.

Proposition 2. *Algorithm 1 on a TA $\mathcal{A}$ produces a TA $\mathcal{A}'$ decomposed into zones that recognises the same language as $\mathcal{A}$.*

It is well known, thanks to the work of Bouyer [10], that forward reachability algorithm for general TAs does not terminate in general as infinitely many distinct zones can be produced. This problem cannot occur here as the clocks are bounded by a constant M, and hence the algorithm can produce only finitely many zones.

Algorithm 1 Forward reachability algorithm

```
to_explore:= {(q0, {0})}; loczones := {(q0, {0})}; Δ' := ∅
while to_explore is not empty do
  pop a loczone (q, Z) from to_explore;
  for all δ such that δ⁻ = q do
    if (δ⁺, POST(Z, δ)) ∉ loczones then
      Add (δ⁺, POST(Z, δ)) to loczones and to to_explore
      Add a new transition δ_{(q,Z),(δ⁺,POST(Z,δ))} to Δ' from loczone (q, Z) to
      (δ⁺, POST(Z, δ)) with guard 𝔤_δ and reset 𝔯_δ and label a_δ.
```

The splitting algorithm The splitting algorithm (Algorithm 2) sketched in the main text involves maintaining a set of candidate transitions to be checked and checking them all (and split zones and guards when needed) until the set is empty. At each loop, the pre-stability condition $Z_{\delta^-} \subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$ is first checked. When it fails SplitZone() is called and the loczone is split accordingly into several loczones, then the function UpdateLocZone is called on each loczone to update the guard of the incoming and outgoing transition so that the resulting TA is still decomposed into zones. When the condition $Z_{\delta^-} \subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$ is satisfied, then SplitGuard(δ) (Algorithm 3) is called. It returns a list of open guards that is an almost partition of the guard $\mathfrak{g}_\delta$ such that each guard of the list is split. If the list contains only one guard (necessarily $\mathfrak{g}_\delta$) then the guard is already split and nothing is done. Otherwise, the transition is split into several transitions, one for each sub-guard of the list, which are added to the set of candidates (these candidates, when treated, will then make the zone split into several sub-zones during SplitZone()).

We now give further details on SplitGuard().

Algorithm 2 The splitting algorithm

Require: A TA $\mathcal{A}$ decomposed into zones.
Ensure: Returns $\mathcal{A}$ in split form.
1: `cand`:= transitions($\mathcal{A}$);
2: **while** `cand` is not empty **do**
3: pop a transition δ from `cand`;
4: **if** $Z_{\delta^-} \not\subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$ **then**
5: SplitZone(δ);
6: newGuards:=SplitGuard(δ);
7: **if** Size(newGuards)> 1 **then**
8: **for all** guards $g \in$ newGuards **do**
9: Create transition δ_g with guard g, origin δ^-, destination δ^+ and label a_δ.
10: Add δ_g to `cand`;

The guard splitting algorithm The algorithm SplitGuard() (Algorithm 3) takes as input a transition δ and returns a set of split guards that is an almost-partition of the guard of the input transition. The first phase is the upper-splitting: for each useful upper-constraint $x_i < a_{0,i}$, we create a guard where $x_i < a_{0,i}$ is the unique useful upper-constraint by adding the constraints $x_i - x_j < a_{0,i} - a_{0,j}$, where $x_j < a_{j,0}$ are the other useful lower-constraints (that become useless). In the second phase we similarly perform lower-splitting for each guard created during the upper-splitting. We add the resulting guards to the output set (removing the duplicates).

Algorithm 3 SplitGuard(δ)

Require: A transition δ.
Ensure: Guards in set_result are split and form an almost partition of g_δ.
set_result:=empty set of guards;
compute useful constraints using Floyd-Warshall Algorithm;
for i = 1 to n **do**
if $x_i < a_{0,i}$ is a useful constraint of g_δ **then**
add i to lbs;
for i = 1 to n **do**
if $-x_i < a_{i,0}$ is a useful constraint of g_δ **then**
add i to ubs;
for all $l \in$ lbs **do**
for all $u \in$ ubs **do**
create a copy g' of $\mathfrak{g}_\delta$;
for all $j \in lbs \setminus \{l\}$ **do**
add the constraint $x_j - x_l < a_{0,l} - a_{0,j}$ to g';
for all $j \in ubs \setminus \{u\}$ **do**
add the constraint $x_u - x_j < a_{j,0} - a_{u,0}$ to g';
if g' is not empty **then**
add g' to set_result;
return set_result;

Algorithm 4 SplitZone(δ)

Require: A transition δ such that $Z_{\delta^-} \not\subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$
Ensure: The returned TA is a refinement of the original TA and they are different.
1: newZones := $\{Z \mid Z \subseteq Z_{\delta^-} \setminus \mathtt{TE}^{-1}(\mathfrak{g}_\delta) \wedge Z \subseteq_{\texttt{full-dim}} Z_{\delta^-}\}$
2: **for all** Z in newZones **do**
3: Create a new location l_Z with associated zone Z
4: UpdateLocZone(l_Z)
5: $Z_{\delta^-} := \mathtt{TE}^{-1}(\mathfrak{g}_\delta) \cap Z_{\delta^-}$
6: UpdateLocZone(δ^-)

Algorithm 5 UpdateLocZone(l)

Require: A location l of a TA $\mathcal{A}$ decomposed into zones.
Ensure: $\forall \delta \; s.t. \; \delta^- = l, \; \mathfrak{g}_\delta \subseteq \mathtt{TE}(Z_l) \wedge \forall \delta \; s.t. \; \delta^+ = l, \; \mathfrak{r}(\mathfrak{g}_\delta) \subseteq Z_l$
$\wedge \forall \delta \; (\mathfrak{g}_\delta \neq \emptyset \wedge \mathfrak{g}_\delta$ has been modified) $\Rightarrow \delta \in \mathtt{cand}$
1: **for all** outgoing transitions δ of l **do**
2: $\mathfrak{g}_\delta := \mathfrak{g}_\delta \cap \mathtt{TE}(Z_l)$
3: **if** $\mathfrak{g}_\delta \neq \emptyset$ **then**
4: Add δ to cand
5: **for all** incoming transitions δ of l **do**
6: $\mathfrak{g}_\delta := \mathfrak{g}_\delta \cap \mathfrak{r}_\delta^{-1}(Z_l)$
7: **if** $\mathfrak{g}_\delta \neq \emptyset$ **then**
8: Add δ to cand

A.3 Proof of Theorem 1

To prove Theorem 1 we have to prove that the algorithm terminates, that the DTA produced is split and that it recognises a tight under-approximation of the timed language of the input DTA. For this we introduce the notion of refinement of a DTA below. We show the following assertions:

A1 The refinement relation is a well-founded order (Proposition 3).
A2 Given a DTA $\mathcal{A}$, Algorithm 2 returns a DTA $\mathcal{A}'$ that is split and that is a refinement of $\mathcal{A}$ (Proposition 5).
A3 Refinement yields tight under-approximation of languages (Proposition 4).

From the two last assertions the result follows.

Refinement of a TA Given two TAs $\mathcal{A}, \mathcal{A}'$ decomposed into zones, if there exists a pair of surjective functions $\varphi_1 : Q_{\mathcal{A}'} \to Q_{\mathcal{A}}$, $\varphi_2 : \Delta_{\mathcal{A}'} \to \Delta_{\mathcal{A}}$ such that the following conditions are satisfied, then we say that $\mathcal{A}'$ is a *refinement* of $\mathcal{A}$ via (φ_1, φ_2), and denote this by $\mathcal{A} \succeq_{(\varphi_1,\varphi_2)} \mathcal{A}'$ (or just $\mathcal{A} \succeq \mathcal{A}'$).

R1 For every $\delta' \in \Delta_{\mathcal{A}'}$, $\varphi_2(\delta')^- = \varphi_1(\delta'^-)$, $\varphi_2(\delta')^+ = \varphi_1(\delta'^+)$, $r_{\delta'} = r_\delta$, $g_{\delta'} \subseteq g_{\varphi_2(\delta')}$ and $a_{\varphi_2(\delta)} = a_\delta$.
R2 For every $l \in Q_{\mathcal{A}}$ it holds that $\{Z_{l'} | l' \in \varphi_1^{-1}(l)\}$ is an almost partition of Z_l.

R3 For every $l' \in Q_{\mathcal{A}'}$, for every $\delta \in \Delta_{\mathcal{A}}$ such that $\delta^- = \varphi_1(l')$ and $g_\delta \cap \mathtt{TE}(Z_{l'}) \neq \emptyset$, it holds that $\{\mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l'}) \mid \delta'^- = l' \wedge \varphi_2(\delta') = \delta\}$ is an almost partition of $g_\delta \cap \mathtt{TE}(Z_{l'})$.

R4 $\varphi_1(i_{\mathcal{A}'}) = i_{\mathcal{A}}$.

Proposition 3. *$\succeq$ is a well-founded order and every sequence of pairwise distinct TAs $\mathcal{A}_0 \succeq \cdots \succeq \mathcal{A}_n$ satisfies $n \leq |Q_{\mathcal{A}_0}|.R_{M,X} + |\Delta_{\mathcal{A}_0}|.R^2_{M,X}$ with $R_{M,X}$ the number of regions on clocks bounded by M.*

We first prove that $\succeq$ is an order by showing transitivity (Lemma 1) and antisymmetry (Lemma 2). Reflexivity is straightforward, that is, $\mathcal{A} \succeq_{(id_{Q_{\mathcal{A}}}, id_{\Delta_{\mathcal{A}}})} \mathcal{A}$, where, for a finite set E, we write id_E, the identity function on E.

Lemma 1 (Transitivity). *If $\mathcal{A} \succeq_{(\varphi_1,\varphi_2)} \mathcal{A}'$ and $\mathcal{A}' \succeq_{(\varphi'_1,\varphi'_2)} \mathcal{A}''$ then $\mathcal{A} \succeq_{(\varphi_1\circ\varphi'_1,\varphi_2\circ\varphi'_2)} \mathcal{A}''$*

Proof. R1 is straightforward. R2 is proven by noting that an almost partition can be composed as a classical set-theoretic partition. We prove R3. Let $l'' \in Q_{\mathcal{A}''}$, $\delta \in \Delta_{\mathcal{A}}$ such that $\delta^- = \varphi_1 \circ \varphi'_1(l'')$ and $\mathfrak{g}_\delta \cap \mathtt{TE}(Z_{l''}) \neq \emptyset$. We denote by $l' = \varphi'_1(l'')$. It holds that $\delta^- = \varphi_1(l')$ and $\mathfrak{g}_\delta \cap \mathtt{TE}(Z_{l'}) \neq \emptyset$ (as $Z_{l''} \subseteq Z_{l'}$). Hence $\{\mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l'}) \mid \delta'^- = l'\}$ is an almost partition of $\mathfrak{g}_\delta \cap \mathtt{TE}(Z_{l'})$. For every δ' such that $\delta'^- = l'$ and $\mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l''}) \neq \emptyset$ it holds that $\{\mathfrak{g}_{\delta''} \cap \mathtt{TE}(Z_{l''}) \mid \delta''^- = l''\}$ is an almost partition of $\mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l''})$. Moreover $\{\mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l''}) \mid \delta'^- = l' \wedge \mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l''}) \neq \emptyset\}$ is an almost partition of $\mathfrak{g}_\delta \cap \mathtt{TE}(Z_{l''})$ (this comes from the fact that $\{\mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l'}) \mid \delta'^- = l' \wedge \mathfrak{g}_{\delta'} \cap \mathtt{TE}(Z_{l'}) \neq \emptyset\}$ is an almost partition of $\mathfrak{g}_\delta \cap \mathtt{TE}(Z_{l'})$ and $Z_{l''} \subseteq Z_{l'}$). It then holds as required that $\{\mathfrak{g}_{\delta''} \cap \mathtt{TE}(Z_{l''}) \mid \delta''^- = l''\}$ is an almost partition of $\mathfrak{g}_\delta \cap \mathtt{TE}(Z_{l''})$.

Lemma 2 (Antisymmetry). *If $\mathcal{A} \succeq \mathcal{A}'$ and $\mathcal{A}' \succeq \mathcal{A}$ then $\mathcal{A}$ and $\mathcal{A}'$ are equal up to renaming bijectively locations and transitions.*

Proof. Assume that $\mathcal{A} \succeq_{(\varphi_1,\varphi_2)} \mathcal{A}'$ and $\mathcal{A}' \succeq_{(\varphi'_1,\varphi'_2)} \mathcal{A}$. $\varphi_1 : Q_{\mathcal{A}'} \to Q_{\mathcal{A}}$ and $\varphi'_1 : Q_{\mathcal{A}} \to Q_{\mathcal{A}'}$, are surjective functions, hence $Q_{\mathcal{A}}$ and $Q_{\mathcal{A}'}$ have the same cardinalities and thus φ_1 and φ'_1 are bijective. The same reasoning holds for φ_2 and φ'_2 that are hence also bijective. Using R2 we have that, for every $l \in Q_{\mathcal{A}}$, $Z_{\varphi_1^{-1}(l)} = Z_l$. We now take $\delta' \in \Delta_{\mathcal{A}'}$ and show that $g_{\delta'} = g_{\varphi_2(\delta')}$. $\varphi'_2 \circ \varphi_2$ is a permutation of the finite set $\Delta_{\mathcal{A}'}$, so there exists some integer k such that $(\varphi'_2 \circ \varphi_2)^k(\delta') = \delta'$. Using R1 several times $g_{\delta'} \subseteq g_{\varphi_2(\delta')} \subseteq g_{(\varphi'_2\circ\varphi_2)^k(\delta')} = g_{\delta'}$; hence, all these inclusions are in fact equalities.

We now prove the statement regarding the length of decreasing subsequences. Given a TA $\mathcal{A}$ decomposed in zones, we construct a TA $\mathbf{Reg}(\mathcal{A})$ decomposed into regions as follows: let $\mathbb{S}_{\mathbf{Reg}(\mathcal{A})} = \cup_{l\in Q_{\mathcal{A}}} \cup_{\mathbf{r} \subseteq_{\mathtt{full\text{-}dim}} Z_l} l_{\mathbf{r}} \times \mathbf{r}$. For every transition δ, for every region $\mathbf{r} \subseteq_{\mathtt{full\text{-}dim}} Z_{\delta^-}$ and region $g \subseteq_{\mathtt{full\text{-}dim}} \mathfrak{g}_\delta$, if $\mathtt{TE}(\mathbf{r}) \cap g \neq \emptyset$ then add a transition $\delta_{\mathbf{r},g}$ to $\Delta_{\mathbf{Reg}(\mathcal{A})}$ with guard g origin $(\delta^-)_{\mathbf{r}}$ and destination $(\delta^+)_{\mathfrak{r}_\delta(g)}$.

The complete the proof of Lemma 3 we rely on the two following lemmas.

Lemma 3 (Minimality of $\mathbf{Reg}(\mathcal{A})$). *If $\mathcal{A} \succeq \mathcal{A}'$ then $\mathcal{A}' \succeq \mathbf{Reg}(\mathcal{A})$.*

Proof. Let (φ_1, φ_2) such that $\mathcal{A} \succeq_{(\varphi_1,\varphi_2)} \mathcal{A}'$. We define (φ_1, φ_2) such that $\mathcal{A}' \succeq_{(\varphi_1,\varphi_2)} \mathbf{Reg}(\mathcal{A})$. For $l \in Q_{\mathcal{A}}$ and $\mathbf{r} \subseteq_{\texttt{full-dim}} Z_l$, by R2, there is exactly one location in $Q_{\mathcal{A}}$, let us call it $\varphi_1'(l_{\mathbf{r}})$, such that $\mathbf{r} \subseteq_{\texttt{full-dim}} \varphi_1'(l_{\mathbf{r}})$ and $\varphi_1'(l_{\mathbf{r}}) \in \varphi_1^{-1}(l)$. For every transition $\delta \in \Delta_{\mathcal{A}}$, for every region $\mathbf{r} \subseteq_{\texttt{full-dim}} Z_{\delta^-}$ and region $g \subseteq_{\texttt{full-dim}} \mathfrak{g}_\delta$ satisfying $\texttt{TE}(\mathbf{r}) \cap g \neq \emptyset$, we will use R3, noting that $\varphi_1'((\delta^-)_{\mathbf{r}}) \in Q_{\mathcal{A}'}$ and $\varphi_1(\varphi_1'((\delta^-)_{\mathbf{r}})) = \delta^-$ and $g_\delta \cap \texttt{TE}(Z_{\varphi_1'((\delta^-)_{\mathbf{r}})}) = g_\delta \cap \texttt{TE}(\mathbf{r}) \neq \emptyset$. By R3, there is exactly one transition, let us call it $\varphi_2'(\delta_{\mathbf{r},g})$, such that $\mathfrak{g}_{\varphi_2'(\delta_{\mathbf{r},g})} \subseteq \mathfrak{g}_\delta$ and $\varphi_2'(\delta_{\mathbf{r},g})^- = \varphi_1'((\delta^-)_{\mathbf{r}})$ and $\varphi_2(\varphi_2'(\delta_{\mathbf{r},g})) = \delta$. One can show that $\mathcal{A}' \succeq_{(\varphi_1',\varphi_2')} \mathbf{Reg}(\mathcal{A})$.

Lemma 4. *If $\mathcal{A} \succeq \mathcal{A}'$ and $\mathcal{A}' \not\succeq \mathcal{A}$ then $|Q_{\mathcal{A}}| + |\Delta_{\mathcal{A}}| < |Q_{\mathcal{A}'}| + |\Delta_{\mathcal{A}'}|$*

Proof. Assume that $\mathcal{A} \succeq_{(\varphi_1,\varphi_2)} \mathcal{A}'$. As φ_1 and φ_2 are surjective then $|Q_{\mathcal{A}}| \leq |Q_{\mathcal{A}'}|$ and $|\Delta_{\mathcal{A}}| \leq |\Delta_{\mathcal{A}'}|$, and hence $|Q_{\mathcal{A}}| + |\Delta_{\mathcal{A}}| \leq |Q_{\mathcal{A}'}| + |\Delta_{\mathcal{A}'}|$ with equality if and only if these two functions are bijective. Assuming now that the two functions are bijective one can show that $\mathcal{A}' \succeq_{(\varphi_1^{-1},\varphi_2^{-1})} \mathcal{A}$ using the fact that an almost partition is reduced to only one element, and hence identifies zones and guards of the two TAs bijectively. Thus, if $\mathcal{A} \succeq \mathcal{A}'$ and $\mathcal{A}' \not\succeq \mathcal{A}$, then $|Q_{\mathcal{A}}| + |\Delta_{\mathcal{A}}| < |Q_{\mathcal{A}'}| + |\Delta_{\mathcal{A}'}|$.

End of proof of Proposition 3 From these two last lemmas, we deduce that every decreasing sequence for $\succeq$ has length bounded by $|Q_{\mathbf{Reg}(\mathcal{A}_0)}| + |\Delta_{\mathbf{Reg}(\mathcal{A}_0)}|$. $\mathbf{Reg}(\mathcal{A}_0)$ contains less than $|Q_{\mathcal{A}_0}|.R_{M,X}$ locations and less than $|\Delta_{\mathcal{A}_0}|.R_{M,X}^2$ transitions. At the end, the length of the decreasing sequence is upper bounded by $|Q_{\mathcal{A}_0}|.R_{M,X} + |\Delta_{\mathcal{A}_0}|.R_{M,X}^2$.

Proposition 4. *If $\mathcal{A}'$ is a refinement of $\mathcal{A}$ then $L(\mathcal{A}')$ is a tight under-approximation of $L(\mathcal{A})$.*

Proof. To every timed path $(t_1, \delta_1') \cdots (t_n, \delta_n')$ of $\mathcal{A}'$ we ca associate the timed path of $\mathcal{A}$ $(t_1, \varphi_2(\delta_1')) \cdots (t_n, \varphi_2(\delta_n'))$ that has exactly the same timed words $(t_1, a_{\delta_1'}) \cdots (t_n, a_{\delta_n'})$ as labels. Hence $L(\mathcal{A}') \subseteq L(\mathcal{A})$.

We now show that $\texttt{Vol}(P_w^{L(\mathcal{A})} \setminus P_w^{L(\mathcal{A}')}) = 0$ for every $w \in \Sigma^*$. Let $w = w_1 \cdots w_n$ be a word such that $\texttt{Vol}(P_w^{L(\mathcal{A}')}) \neq 0$ (otherwise the result is trivial).

We proceed in the opposite direction as before by mapping timed paths of $\mathcal{A}$ that yield timed words of $L(\mathcal{A})$ to timed paths of $\mathcal{A}'$ that yield timed words of $L(\mathcal{A}')$. From a timed path $(t_1, \delta_1) \cdots (t_n, \delta_n)$, we define by $s_0, s_1, \ldots, s_n$ the sequence of visited states where s_0 is the initial state and $s_{i+1} = (s_i)_{(t_{i+1},\delta_{i+1})}$. Using definition of a refinement (R3) one can remark that for every state $s = (q, \boldsymbol{x})$ and timed transition (t, δ) such that $s \triangleright (t, \delta) \neq \bot$, for every $s' = (q', \boldsymbol{x})$ there is at most one timed transition (t, δ') such that $\varphi_2(\delta') = \delta$ and $s'_{(t,\delta)} \neq \bot$. The only case where such a transition does not exist is when the clock vector associated to $s_{(t,\delta)}$ does not belong to any of the zones forming the almost-partition of Z_{δ^-}. This can only occur at a border of a zone and such a border has null volume. More precisely, in such a border the difference of clock values is equal

to an integer (for instance $x_1 - x_2 = 3$). We remark that both such quantities are of the form $\sum_{i=j}^{k} t_i = m$ with $j \leq k \leq n$ and $m \in \mathbb{N}$. We denote by $A_{j,k,m}$ the hyperplane of $\mathbb{R}^n$ defined by equation $\sum_{i=j}^{k} t_i = m$. The set $A \stackrel{\text{def}}{=} \cup_{j,k,m} A_{j,k,m}$ has volume null as it is a countable union of null volume sets. We have hence shown that every delay vector of $P_w^{L(\mathcal{A})}$ that avoids $\mathcal{A}$ is a delay vector of $P_w^{L(\mathcal{A}')}$. Formally, $(P_w^{L(\mathcal{A})} \setminus A) \subseteq P_w^{L(\mathcal{A}')}$. This implies that $\texttt{Vol}(P_w^{L(\mathcal{A})}) = \texttt{Vol}(P_w^{L(\mathcal{A})} \setminus A) \leq \texttt{Vol}(P_w^{L(\mathcal{A}')})$, thus $\texttt{Vol}(P_w^{L(\mathcal{A})} \setminus P_w^{L(\mathcal{A}')}) = \texttt{Vol}(P_w^{L(\mathcal{A})}) - \texttt{Vol}(P_w^{L(\mathcal{A}')}) \leq 0$ and finally $\texttt{Vol}(P_w^{L(\mathcal{A})} \setminus P_w^{L(\mathcal{A}')}) = 0$.

Proposition 5. *Given a DTA $\mathcal{A}$, Algorithm 2 returns a split DTA $\mathcal{A}'$ that is a refinement of $\mathcal{A}$.*

The algorithm terminates when the list of candidate transitions `cand` is empty. We denote by $\mathcal{A}_k$ the timed automaton just after the kth occurence of a continue statement and by $\mathcal{A}_0$ the initial timed automaton. We show the following invariants: (I1) At the evaluation of every while statement every transition not in `cand` satisfies the desired properties: its guard is split and $Z_{\delta^-} \subseteq \texttt{TE}^{-1}(\mathfrak{g}_\delta)$; let us call such a transition a good transition. (I2) $\mathcal{A}_k$ is decomposed into zones, $\mathcal{A}_k \succeq \mathcal{A}_{k-1}$ and $\mathcal{A}_k \neq \mathcal{A}_{k-1}$;

Proof of (I1). It suffices to see that

- Every transition before the first while statement and every transition created during the algorithm are added to `cand`.
- When a transition goes outside `cand` then it is either deleted either good.
- Good transitions with zones (of the origin, destination and guard) that are not changed during a loop stay good and outside `cand`.
- At any moment, if one of a zone (of the origin, destination and guard) of a transition is changed then this transition is added to `cand`.

Proof of (I2). We show (I2) by induction. At the beginning of the algorithm the TA is decomposed into zones. Let us assume that $\mathcal{A}_0 \succeq \ldots \succeq \mathcal{A}_k$ and that all these TAs are decomposed into zones. If in a loop no continue statement occurs, then the TA is not changed and no transition are added to `cand`. Thus, if no more continue statements occur the algorithm ends and (I2) is satisfied until the end. When the next continue occurs then the TA has been changed and we show that $\mathcal{A}_{k+1}$ is decomposed into zones and $\mathcal{A}_k \succeq \mathcal{A}_{k+1}$. Every transition created or modified satisfies $\texttt{POST}(Z_{\delta^-}, \delta) \subseteq Z_{\delta^+}$, and hence the TA stays decomposed into zones. We now have an explicit pair of functions (φ_1, φ_2) such that $\mathcal{A}_k \succeq_{(\varphi_1,\varphi_2)} \mathcal{A}_{k+1}$. We define φ_1 and φ_2 on non-created and non-deleted states and transitions to be the identity, in particular $\varphi_1(s_{\mathcal{A}_{k+1}}) = s_{\mathcal{A}_k}$. The only eventual creations of location are done in SplitZone; we define $\varphi_1(l_Z) = l$. For every transition created during SplitZone we define $\varphi_2(\delta'_Z) = \delta'$ and for every transition δ_g (created outside SplitZone) we define $\varphi_2(\delta_g) = \delta$. We now check that condition R1, R2 and R3 of the definition of refinement are satisfied. Condition R1 and R4 are clearly satisfied. Condition R2 is satisfied as the only zone modified is

the one that is split in an almost partition in Line 1 of SplitZone. To show R3, we note that for every incoming transition δ' of a modified zone the guard g'_δ is split in an almost-partition in Line 2 of UpdateLocZone. The other place where a guard is modified is after SplitGuard(δ), where the guard of δ is also split into an almost-partition of guards created during SplitGuard(δ).

End of the proof As every decreasing sequence for $\succeq$ is bounded (Lemma 5), the number of occurrences of continue is bounded as well, and hence the total number of additions of a transition to `cand` is bounded. As every occurrence of a while loop removes a transition from `cand`, necessarily `cand` is eventually empty and the algorithm terminates. Due to (I2) and emptiness of `cand`, the TA returned has only good transitions and hence is split.

□

A.4 Proof of Proposition 1

We remark first that, due to splitting and prestability, it holds that:

Lemma 5. *For every split DTA, for all δ (let q be its origin) the two functions $\boldsymbol{x} \mapsto \mathrm{lb}_\delta(q, \boldsymbol{x})$ and $\boldsymbol{x} \mapsto \mathrm{ub}_\delta(q, \boldsymbol{x})$ are univariate polynomial of degree less than 1 on $Z_{\delta^-} \subseteq \mathtt{TE}^{-1}(\mathfrak{g}_\delta)$.*

We now proceed with the proof of Proposition 1. We apply induction to prove that the functions v_n restricted to any location q are positive polynomial functions of degree at most n. $v_0 = 1$ is polynomial and positive. Assume that v_{n-1} is polynomial and positive and fix a location q. It suffices to show that, for every outgoing edge δ, the function $\boldsymbol{x} \mapsto \Psi_\delta(v_{n-1})(\boldsymbol{x})$ defined on $Z_q \subseteq \mathtt{TE}^{-1}(g_\delta)$ is a polynomial of degree at most n and positive everywhere on Z_q. First note that $v_{n-1}(q', \mathfrak{r}_\delta(\boldsymbol{x} + t))$ is a polynomial in $\boldsymbol{x}$ of degree at most $n-1$, where q' is the destination of δ. Let F be a primitive of this latter function. F is a polynomial of degree at most n and thus, using Lemma 5, $\Psi_\delta(v_{n-1})(q, \boldsymbol{x}) = F(\mathrm{ub}_\delta(q, \boldsymbol{x})) - F(\mathrm{lb}_\delta(q, \boldsymbol{x}))$ is a polynomial of degree at most $n-1$. Moreover, as v_{n-1} is positive and $\mathrm{ub}_\delta(q, \boldsymbol{x})) - \mathrm{lb}_\delta(q, \boldsymbol{x}) > 0$, we also have that v_n is positive on Z_q as required. □

B Additional material and proofs for Section 4

Locations and transitions of a TA define a directed graph. The period $\mathfrak{p}$ of a TA decomposed into regions is the period of this directed graph, that is, the gcd of the length of its cycles. Aperiodic TAs are TAs with period $\mathfrak{p} = 1$. For the sake of simplicity the results of this section are stated for aperiodic TAs, but they can be easily extended to TAs with an arbitrary period.

Infinite horizon and maximal entropy stochastic process.

A theoretical quasi-uniform random sampling based on the notion of maximal entropy was provided in [7]. We now show that it can be interpreted as a limit case of the receding horizon method Method 2 when the horizon m tends to infinity. The construction of [7] requires preprocessing of the TA: split the TA into regions and restrict to strongly connected components. While the former is costly, the latter is standard and already required in the discrete case of finite graphs. Moreover, the TA has to be *thick*: its volume does not decrease faster than any exponent, namely, its *entropy* $\limsup_{n\to+\infty}(1/n)\log\left(\sup_{s\in\mathbb{S}} v_n(s)\right)$ is greater than $-\infty$. As described in [3], thickness can be related in several ways to robustness, while the absence of thickness induces mathematical pathologies (weak form of Zenoness).

Theorem 3 (Perron-Frobenius-like theorem for TA, [3]). *For thick strongly connected TA $\mathcal{A}$ decomposed into regions, there exists a continuous and bounded function $v \geq 0$ such that $\Psi v = \rho v$ and v is unique up to a multiplicative constant.*

The following Theorem states that the volume functions, once renormalised, tend towards the eigenfunction v. It is a consequence of Theorem 15.4 of [17], for which the hypotheses are proved to be satisfied in [3].

Theorem 4 (Convergence in direction of the volume functions). *For thick strongly connected aperiodic TA $\mathcal{A}$ decomposed into regions, there exist constants $c \geq 1$ and $b < 1$ such that, for every $s \in \mathbb{S}_{\mathcal{A}}$, $(1/\rho^m)v_m(s) - cv(s) = O(b^m)$, where v is described in Theorem 3.*

In [7], the maximal entropy stochastic process was defined in terms of the weight of timed transitions (t,δ) from states s (denoted $p^*((t,\delta)|s)$). This weight is $p^*((t,\delta)|s) = \frac{v(s_{t,\delta})}{\rho v(s)} = \omega(v,s)(\delta)\varphi(v,s,\delta)(t)$.

Hence, sampling according to the maximal entropy stochastic process of [7] can be interpreted as sampling with *infinite receding horizon* that involves using the DPD $\omega(v,s)$ and the PDF $\varphi(v,s,\delta)$ instead of the DPD $\omega(v_m,s)$ and the PDF $\varphi(v_m,s,\delta)$ used in the finite receding horizon.

Corollary 1 (Convergence of $\omega(v_m,s)$ toward $\omega(v,s)$ (for fixed state s)). *Consider a thick strongly connected aperiodic TA $\mathcal{A}$ decomposed into regions. For every $s \in \mathbb{S}$, there exists a constant $b_s < 1$ such that*

$$\sup_{\delta} |\omega(v_m,s)(\delta) - \omega(v,s)(\delta)| = O(b_s^m)$$

and for every δ,

$$\sup_{t\in(\mathrm{lb}_\delta,\mathrm{ub}_\delta)} |\varphi(v_m,s,\delta)(t) - \varphi(v,s,\delta)(t)| = O(b_s^m)$$

where v is described in Theorem 3.

Proof of Theorem 2

Proof. Denote as usual by s_i the ith state visited during the sampling. For $i \leq n-m$, the ith timed letters have weight $\omega(v_m, s_{i-1})(\delta_i)\varphi(v_m, s_{i-1}, \delta_i)(t_i) = \frac{v_m(s_i)}{v_{m+1}(s_{i-1})}$. The m last timed letters form a timed word generated uniformly from s_{n-m}. Hence, the weight of a timed word generated by this method is:

$$\begin{aligned}
\frac{1}{v_m(s_{n-m})}\left(\prod_{i=1}^{n-m}\frac{v_m(s_i)}{v_{m+1}(s_{i-1})}\right) &= \frac{1}{v_m(s_{n-m})}\left(\frac{\prod_{i=1}^{n-m} v_m(s_i)}{\prod_{i=0}^{n-m-1} v_{m+\mathfrak{p}}(s_i)}\right) \\
&= \frac{1}{v_m(s_{n-m})}\left(\frac{v_m(s_{n-m})}{v_{m+1}(s_0)}\prod_{i=1}^{n-m-1}\frac{v_m(s_i)}{v_{m+1}(s_i)}\right) \\
&= \frac{1}{v_n(s_0)}\left(\prod_{i=1}^{n-m-1}\frac{v_{m+i+1}(s_0)}{v_{m+i}(s_0)}\right)\left(\prod_{i=1}^{n-m-1}\frac{v_m(s_i)}{v_{m+1}(s_i)}\right) \\
&= \frac{1}{v_n(s_0)}(1+\varepsilon')
\end{aligned}$$

where $\varepsilon' \stackrel{\text{def}}{=} \left(\prod_{i=1}^{n-m-1}\frac{v_{m+i+1}(s_0)}{v_{m+i}(s_0)}\right)\left(\prod_{i=1}^{n-m-1}\frac{v_m(s_i)}{v_{m+1}(s_i)}\right) - 1$.

Now we note that, for all $i = 1..n$, $\frac{v_{m+i+1}(s_0)}{v_{m+i}(s_0)} \in \left[\frac{1}{C^+}, \frac{1}{C^-}\right]$ and $\frac{v_m(s_i)}{v_{m+1}(s_i)} \in [C^+, C^-]$; hence

$$1+\varepsilon' \in \left[\left(C^-/C^+\right)^{N-1}, \left(C^+/C^-\right)^{N-1}\right]$$

We conclude that $\varepsilon' \leq \varepsilon_{n,m}$ as expected. □

C More details on the case study

In this section we describe formally the case study of Section 5.2. We consider a failure and repair system consisting of K machines running in parallel to accomplish a task. When all machines are in nominal mode they perform the task **c** together, and the delay between two consecutive such tasks is upper-bounded by t_c. When a failure occurs with **b**, all machines stop. To restart the system, each machine undergoes i repair steps (**a**) independently. During repairs, at most n_b failures may occur, which require an additional repair each. When all machines have performed their repair they can jointly perform an action **c** and enter the nominal mode. Fig. 7 depicts a TA for one machine, where dotted arrows represent edges synchronised between machines. The product automaton of the full model $\mathcal{A}$ is nondeterministic, as the choice of which machine should take a transition is unspecified. The language of this automaton is referred to as $L(\mathcal{A})$ in the following.

The failure and repair system is naturally modelled as a network of TAs that synchronise. We build the synchronised product of the whole system as a single

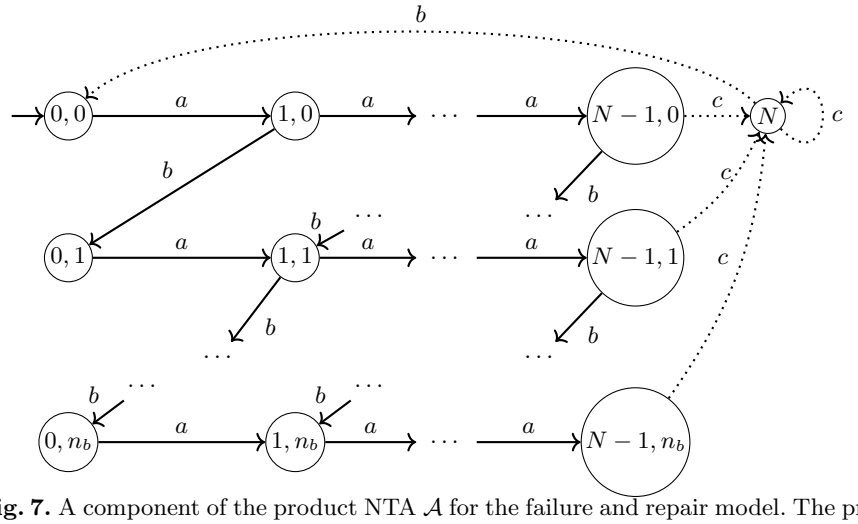

Fig. 7. A component of the product NTA $\mathcal{A}$ for the failure and repair model. The product NTA $\mathcal{A}$ is obtained by synchronising the K components along dotted transitions. Transitions labelled by a have guard $\{x > 2, y < Evt\}$ and reset $\{x := 0, y := 0\}$; transitions labelled by b have guard $\{y < Evt\}$ and reset $\{y := 0\}$; self-loops labelled by c have guard $\{y < t_c\}$ and reset $\{y := 0\}$. Other transitions labelled by c have guard $\{y < Evt\}$ and reset $\{y := 0\}$.

large NTA. The time between two repairs is bounded by 2 units of time and there is an event in each machine, failure or repair, every Evt unit of time.

We aim at checking whether the language $L(\mathcal{A})$ satisfies the property specified by the NTA $\mathcal{B}$ of Figure 8 on its prefixes of length 50. This property specifies that there exist three timed transitions labelled by $\mathbf{a}, \mathbf{c}$ and $\mathbf{a}$ such that less than c_2 time units expires between the two $\mathbf{a}$s and more than c_1 time unit between the $\mathbf{c}$ and any of the $\mathbf{a}$.

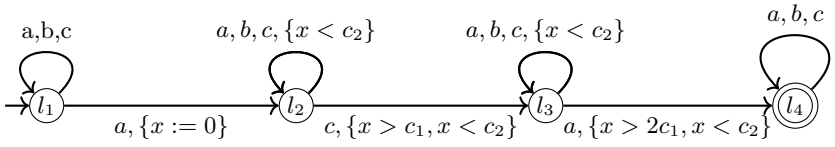

Fig. 8. Property to check for the failure and repair model.

We build a deterministic over-approximation of the language generated by this system recognised by a single DTA (referred to as automaton $\mathcal{C}$) with weaker timing constraints (Fig. 9). The quantity R is equal to the number of machines times N. The time bound $t_b \stackrel{\text{def}}{=} (N + n_b) \times Evt$ is the maximal time required for a machine to reach the nominal state.

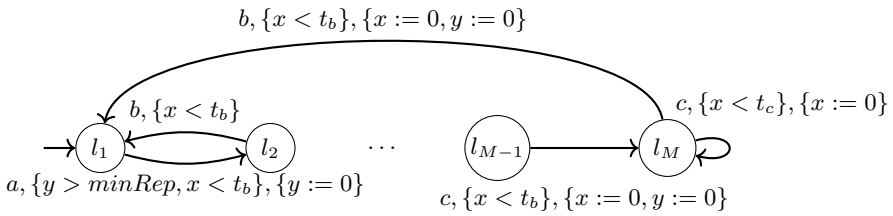

Fig. 9. Deterministic abstraction for the failure and repair model.

The experiments on this model use two machines and increasing values for N. We set the following constants: $Evt = 8$; $minRep = 1$; $c_1 = 9$; $c_2 = 20$.