Wordgen : a Timed word Generation Tool

Benoît Barbot, Nicolas Basset, Alexandre Donze

WORDGEN : a Timed word Generation Tool

Benoît Barbot
benoit.barbot@lacl.fr
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Nicolas Basset
bassetni@univ-grenoble-alpes.fr
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG Grenoble, France

Alexandre Donzé
alex@decyphir.com
Decyphir SAS
Moirans, France

ABSTRACT

Sampling timed words out of a timed language described as a timed automaton may seem a simple task: start from the initial state, choose a transition and a delay and repeat until an accepting state is reached. Unfortunately, simple approach based on local, on-the-fly rules produces timed words from distributions that are biased in some unpredictable ways. For this reason, approaches have been developed to guarantee that the sampling follows a more desirable distribution defined over the timed language and not over the automaton. One such distribution is the maximal entropy distribution, whose implementation requires several non-trivial computational steps. In this paper, we present WORDGEN which combines those different necessary steps into a lightweight standalone tool. The resulting timed words can be mapped to signals used for model-based testing and falsification of cyber-physical systems thanks to a simple interface with the Breach tool.

ACM Reference Format:
Benoît Barbot, Nicolas Basset, and Alexandre Donzé. 2023. WORDGEN : a Timed word Generation Tool. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2023), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3575870.3587116

1 INTRODUCTION

Timed automata (TA) is a well-known formalism to describe real-time systems. They are used in many verification tools such as UPPAAL [12, 16] or PRISM [15] since the algorithms behind their analysis are well understood. Verification can be conducted either exactly, using regions or zone graph abstractions, or approximately using statistical model checking (SMC) when the size of the system is too large. SMC can also deal with more complex systems, like hybrid systems and/or probabilistic systems.

Statistical model checking relies on the sampling of large numbers of trajectories of the system under test, which can be difficult. In [9], many examples are shown where the sampling of TA using current tools (e.g., UPPAAL SMC [12] or Modes[11]) is not satisfactory for various reasons. The main observation of the authors is that these tools sample TA differently leading to different and often contradicting results. Indeed when sampling a TA, discrete and continuous choices must be made in each state. The order of these choices lead to different algorithms with different results. Moreover the distribution used for both the discrete and the continuous choice should not be arbitrary (see section 2.2 or [9] for more details).

To overcome these shortfalls, probability distribution over delay and transitions must be more formally specified. Distributions that maximize the entropy of the sampling process are a good choice because they are known to be the least biased [14]. They are uniform distributions (UD) over timed languages which have the following characteristics:

(1) UD being defined over the timed language and not over the automaton, two automata recognizing the same timed language will have the same UD;
(2) UD guarantee that given two timed words of the same length they are chosen with the same probability;
(3) UD can be computed using time-language volume [1] for which tractable algorithms have been described in [5];
(4) However, UD computation relies on an analysis of the automaton which can be costly, in particular it can be significantly more costly than more "naïve" statistical model checking methods.

To our knowledge the only implementation of the computation of the uniform distribution over timed language is the tool-chain described in [5], which is not satisfactory for several reasons which we detail in Section 3.4.

The computation of UD requires the computation of the zone graph and thus is more involved than most classical computations on time automata (reachablility, emptiness, TCTL model checking). As a consequence there is no point to use UD sampling to solve these problems. UD sampling can, however, be used to construct probabilistic semi-algorithms to undecidable problems like the inclusion of time language [5].

Another application of TA sampling is the validation of Cyber Physical System (CPS). In this setting we are interested in hybrid systems containing both a discrete part (e.g., modeled by finite state automata) and a continuous part (e.g., modeled by a system of ordinary differential equations) which interact with their environment through real-valued time signals. The formal verification of such systems is known to be difficult. A commonly used alternative is, similarly to SMC, to rely on generating time signals and check that the system behaves according to its specification over these signals. It is then crucial to sample signals that cover the space of admissible input signals well. This space is not always easily described, and by extension sampled, using standard numerical constraints or even advanced specification languages such as signal temporal

This work was financed by the ANR MAVeriQ (ANR-20-CE25-0012), the join ANR-JST project CyPhAI and the Auvergne-Rhône-Alpes region project DETAI..

HSCC 2023, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0033-0/23/05…$15.00
https://doi.org/10.1145/3575870.3587116

logics (STL). In [6, 7], the authors show that TA can provide concise descriptions of the spaces of input signals for complex CPS models, and validate them using UD sampling. The cost of computing and sampling UD for a TA in these cases is negligible compared to other methods to generate admissible input signals (e.g., using STL monitoring-based rejection methods) to run and simulate the CPSs, making the method a good fit.

In this paper we presents the tool WORDGEN which takes as input a TA, compute the UD over the timed language of the TA and outputs timed words sampled from this language using UD. We also present a simple interface for timed word based signal generation with the tool Breach [13], which is a well-known tool for CPS validation. The paper is structured a follows: Section 2 describes the necessary definition, Section 3 describes the computational steps of WORDGEN, Section 4 presents some implementation details, Section 5 presents a case study, and Section 6 concludes.

2 DEFINITIONS

2.1 Timed Automata

A *timed automaton* (TA) $\mathcal{A}$ is defined as a tuple $(\Sigma, X, Q, i_0, \mathcal{F}, \Delta)$ where Σ is a finite set of events; X is a finite set of *clocks*; Q is a finite set of *locations*; i_0 is the initial location; $\mathcal{F} \subseteq Q$ is a set of final locations and Δ is a finite set of *transitions*. The finite set of clock X is a finite set of non-negative real-valued variables . A *guard* is a finite conjunction of clock constraints of the form $x \bowtie c$ with $x \in X$, $c \in \mathbb{N}^+$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. For a clock vector $\vec{x} \in [0, \infty[^X$ and a non-negative real t, we denote by $\vec{x} + t$ the vector $\vec{x} + (t, \ldots, t)$. A 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 $B \subseteq X$. Fig. 1(left) depicts an automaton with two clocks x and y, a single location l_1 and two transitions. The transition labelled by a is guarded by $x < 10 \wedge y < 3$ and reset y while transition labelled by b is guarded by $8 \leq x \leq 10 \wedge 1 \leq y \leq 2$.

Additional clock constraints can be added to each location, these are known as invariants. In our setting, an automaton without invariant is obtained by adding the constraint of the invariant to all outgoing guard of a location.

A *configuration* $s = (q, \vec{x}) \in Q \times [0, \infty)^X$ is a pair of location and a clock vector. The initial configuration of $\mathcal{A}$ is $(i_0, \vec{0})$. A *timed transition* is a pair (t, δ) of a time *delay* $t \in [0, \infty)$ followed by a discrete transition $\delta \in \Delta$. The delay t represents the time before firing the transition δ.

A run is an alternating sequence $(q_0, \vec{x}_0) \xrightarrow{t_1, \delta_1} (q_1, \vec{x}_1) \ldots \xrightarrow{t_n, \delta_n} (q_n, \vec{x}_n)$ of configurations where q_i is the successor of q_{i-1} by δ_i, and the vector $\vec{x}_{i-1} + t$ satisfies the guard $\mathfrak{g}_\delta$ and $\vec{x}_i = \mathfrak{r}_\delta(\vec{x}_{i-1} + t)$. This run is *labelled* by the *timed word* $(t_1, a_1) \cdots (t_n, a_n)$ where for every $i \leq n$, a_i is the label of δ_i. The set of timed words labelling all the runs leading from the initial configuration $(i_0, \vec{0})$ to a final configuration $(q_n \in \mathcal{F})$ is called the *timed language* of $\mathcal{A}$.

2.2 Isotropic Sampling

Isotropic sampling is the sampling of timed words implemented in tools like UPPAAL-SMC or Modes with some variation. Given an automaton $(\Sigma, X, Q, i_0, \mathcal{F}, \Delta)$ and a configuration of the automaton $(q, \vec{x})$ the next delay and next event is computed by choosing:

- The next transitions $\delta \in \Delta$ such that $\delta^- = q$ and $\exists t \geq 0$ s.t. $\vec{x} + t \models \mathfrak{g}_\delta$, i.e., transition for which there exists a time where the guard is satisfied. All such transition are given the same probability to be chosen (discrete uniform law);
- The next delay t such that $\exists \delta \in \Delta$ s.t. $\delta^- = q \wedge \vec{x} + t \models \mathfrak{g}_\delta$, i.e., delay for which there exists at least one acceptable transition at this time. Time t is chosen uniformly (continuous uniform law) among acceptable times if this set is bounded and using an exponential distribution otherwise.

The order of these two choices will substantially change the sampling algorithm and its performance. Thus isotropic sampling is not well defined and as a consequence, different tools implement different strategies. As an example, if transitions are chosen first, an analysis of reachable guards must be conducted to select a satisfiable one, but then the time has to satisfy a single guard which is usually a convex set and thus is easy to sample. The other way around requires a similar trade-off. In [9] the authors show that these choices (and other choices of implementation) lead to tools giving contradictory results on some examples.

Another problem with isotropic sampling is that it is local to each step. More precisely along a timed word, a choice of time at step i can make a guard at step $j > i$ unsatisfiable. For example, in Fig. 1 the bottom plot shows isotropic sampling of the automaton on the left. Most transitions are sampled in the bottom right corner because the local rule cannot stir the simulation toward the green transition to avoid becoming stuck in a time lock.

3 COMPUTATIONAL STEPS

Given a timed automaton, WORDGEN computes the UD over its time language and samples it. In this section, we explain the different operations performed by WORDGEN to achieve this. Fig. 2 depicts an overview of the tool architecture and Fig. 3 shows a running example illustrating each computation step. The computation starts by parsing the input file which can be either in the PRISM PTA or UPPAAL file format. The result is a data structure containing a TA $\mathcal{A}$. Fig. 3(a) shows an example of TA.

3.1 Forward Reachability and Splitting

First the automaton is transformed into a standard zone graph [8] by performing a forward reachability analysis as the traversal of a graph. Zones are sets of clock constraints implemented as Difference Bound Matrices (DBM) for which intersection time elapse (TE) and reset (*Reset*) are efficient. The vertices of the graph are pairs of automaton location and zone, edges are automaton transitions with guard specified as zone. Starting from the initial location and the zone containing the clock vector $\vec{0}$, a graph is constructed by computing for each vertex of the graph (l, z) and each transition δ s.t. $\delta^- = l$ the new vertex $(\delta^+, Reset(TE(z) \cap \mathfrak{g}_\delta, \mathfrak{r}_\delta)$. Fig. 3(b) shows the reachability graph of automaton Fig. 3(a) as computed by WORDGEN.

Then, the zone graph is split so that for any vertex of the zone graph, for each available transition δ, there exists two functions $\mathtt{lb}_\delta$ and $\mathtt{ub}_\delta$ of the clock vector $\vec{x}$ such that, for all $t \in \mathbb{R}^+$, $\vec{x} + t \models \mathfrak{g}_\delta$ is equivalent to $t \in [\mathtt{lb}(\vec{x}), \mathtt{ub}(\vec{x})]$ with lb and ub of the form $\vec{x} \mapsto c - x$ where $c \in \mathbb{N}$ and x is a clock or zero. This is always possible by splitting a zone in smaller one. Detail on the computation and proof

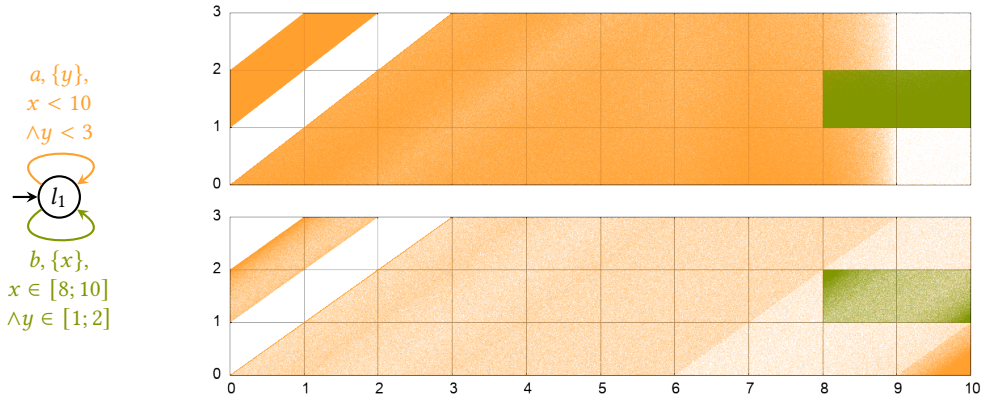


Figure 1: (Left) A TA with 2 clocks x and y, 1 location and 2 transitions. The transition labeled by a is guarded by upper-bounds on clocks, while the guard on the transition labeled by b is a small box $x \in [8;10] \wedge y \in [1;2]$. The only way to reset the clock x is to hit the box. (Top Right) discounted uniform sampling with 10 iterations (m=10), (Bottom Right) isotropic sampling, each sampling contains 50000 trajectories of length 100 where a pixel is drawn at each transitions at positions (x, y) before applying resets.

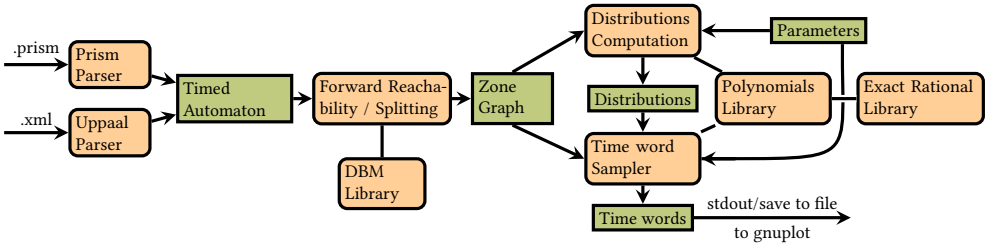


Figure 2: Architecture of WORDGEN. Round part represents computation steps while square one represents data. Arrows represent flow of data. Lines are dependencies.

of correctness can be found in [5]. In Fig. 3(b) only one transition does not satisfy the splitting constraint (in black), Fig. 3(c) shows the result of the splitting algorithm, the vertex (1) have been split. In the worst case zone are spitted until only region remains which leads to an exponential blow up in the number of clocks.

3.2 Weight and Distribution Computation

Each vertex (respectively, each edge) of the zone graph is equipped with a weight function $v_m(l, \vec{x})$ (respectively, $v_m(\delta, \vec{x})$) which maps a clock vector to the volume of the timed language reachable from this clock vector in m steps. The weight is computed with the following recursive equations following the theory of volume of timed language of [1]:

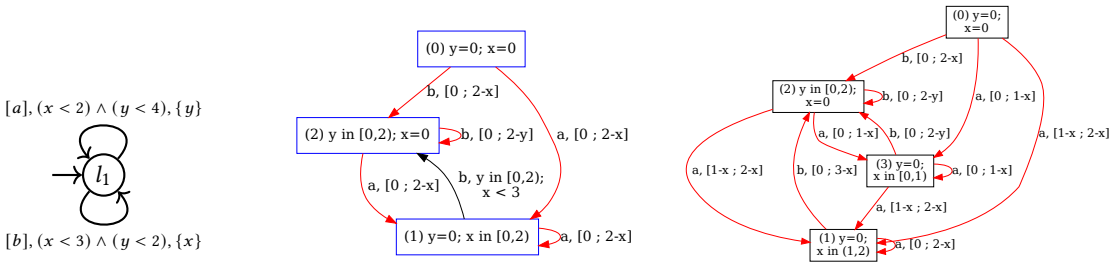
$$\begin{aligned} v_0(l, \vec{x}) &= \mathbb{1}_{l \in \mathcal{F}}; \\ v_m(\delta, \vec{x}) &= \int_{t=\mathtt{lb}_\delta}^{t=\mathtt{ub}_\delta} v_{m-1}\left(\delta^+, \mathfrak{r}_\delta(\vec{x}+t)\right) \mathrm{d}t; \\ v_m(l, \vec{x}) &= \sum_{\delta^- = l} v_m(\delta, \vec{x}). \end{aligned} \tag{1}$$

Since $\mathtt{lb}_\delta$ and $\mathtt{ub}_\delta$ are linear functions of clock vectors, $v_m(\_)$ are polynomials with the clocks set as set of variable. The degree of the polynomials are m and can be computed in polynomial time with respect to the graph size but exponential in m and the number of clocks. Figure 3(d) shows the weights computed by WORDGEN for the transition from 1 to 2 labeled by b.

The zone graph is finally augmented with a probability distribution to become a stochastic process. Discrete choices between transitions are computed using the weights of the transitions. Let us consider a vertex $(l, \vec{x})$ where k transitions are available: δ_1, $\delta_2, \ldots \delta_k$. The probability to sample δ_i is $\frac{v_m(\delta_i, \vec{x})}{\sum_{j=1}^{k} v_m(\delta_j, \vec{x})}$. Continuous time distributions are computed as the derivative of the weight normalized by the total weights, that is from vertex $(l, \vec{x})$ for transition δ the probability density function (PDF) of the time T is $T \mapsto \int_{t=\mathtt{lb}_\delta}^{t=min(T, \mathtt{ub}_\delta)} v_{m-1}\left(\delta^+, \mathfrak{r}_\delta(\vec{x}+t)\right) \mathrm{d}t \frac{1}{v_m(\delta, \vec{x})}$. Note that the variable t appears only in the numerator thus the PDF is a polynomial in t. The Cumulative Density function (CDF) is the integral of the PDF from 0 to t. It is denoted for transition δ and clock vector $\vec{x}$ as $(CDF_m(\delta, \vec{x})(t))$. It is also a polynomial in t. In Fig. 3(d) the volume and CDF of the transitions between 1 and 2 are reported.

3.3 Sampling

The last step consists of sampling the stochastic process obtained from the previous operation. To generate a timed word of length n, one starts with a sequence $(u_i)_{i=1}^{2n} \in [0, 1]$ of real values, which corresponds to a point in the unit box of dimension $2n$. These values can either be computed by WORDGEN using pseudo-random number generators or low discrepancy sequences, they can also be computed by an external optimization algorithm like in [7] where Breach [13] was used to drive the generation.



(a) (b) (c)

(d)

m	1	2	3	4
$v_m(1 \to 2, (x,y))$	$3-x$	$\frac{15-2x-x^2}{2}$	$\frac{126-24x-3x^2-x^3}{6}$	$\frac{1389-256x-48x^2-4x^3-x^4}{24}$
$CDF_m(1 \to 2, (x,y))(t)$	$\frac{t}{3-x}$	$\frac{2t-t^2}{15-2x-x^2}$	$\frac{69t-12t^2+t^3}{126-24x-3x^2-x^3}$	$\frac{760t-138t^2+16t^3-t^4}{1389-256x-48x^2-4x^3-x^4}$

(e) $1.37294[b]0.29924[a]1.14380[a]0.54786[b]0.19368[a]0.69567[b]0.13582[b]0.94777[b]$

Figure 3: Illustration of the different steps of computation. (a) a TA; (b) the forward reachability zone graph, red edges are already split; (c) the split zone graph; (d) weights and distribution computed on edge from vertices 1 to 2 with 0 to 4 iterations; (e) a timed word sampled by WORDGEN**.**

The parameter m of the distribution should be equal to $n - i$ at step i for exact uniform sampling, however, as the computation of distribution for large value of m became intractable, receding horizon can be used, a maximal value m_0 is computed then the distribution with $m = \min(m_0, n - i)$ is used at each step. The correction of this approach and rate of convergence are discussed in [5].

Starting from the initial configuration of the automaton, the timed word is computed iteratively. At step i, in configuration $(l, \vec{x})$ the next transition δ is chosen by sampling the discrete probability distribution $v_m(\delta, \vec{x})$ using real u_{2i}. Then delay t is distributed according to the PDF of δ. This is done using the so-called *inverse transform sampling* method. This consists of computing the root of the CDF minus u_{2i+1}. Such root exists as the CDF is a strictly increasing polynomial taking values in $[0, 1]$. This root can be effectively computed using Newton's method. When a timed word of desired length is produced, it is outputted and the next timed word is computed. Fig. 3(e) shown an example of a timed word. We could have sampled first the time and then the transitions; the sampling distribution would be the same (see [5]) but requires to sample from a piece-wise polynomial distribution.

3.4 Motivation for a New Implementation

In [5, 6] a tool-chain was implemented to conduct the same computation as WORDGEN. The different parts of the computation were performed by specialized tools: the TA analysis was performed by a modified version of the model checker PRISM, the probability distribution was computed by a script written in SageMath, a computer algebra system well suited for computation over polynomials, and the sampling was performed by the statistical model checker COSMOS. This work is a new standalone implementation in the OCaml programming language of all these steps. The main focus was to make this tool easier to use and to extend than the previous implementation. We describe some of the main advantages next.

Performance. While for their original purpose, each tool used in the legacy tool-chain are known to be very efficient, the specific operations they were used for in this work are not the ones they were designed and optimized for. Hence, a specialized re-implementation of these operations does not harm performance.

- PRISM forward reachability algorithm is faster than our implementation over large automata. However, automaton size is not the bottle neck here: on all examples we tried, this time was small for both tools as compared to the other computational steps.
- SageMath has an efficient library to handle polynomials formally and can handle various operations but only a few of these operations are relevant to our setting and those can be implemented with more specialized algorithms and data structures. In Table 1 we show on an example that WORDGEN is faster, even when using exact rational numbers.
- COSMOS is designed to simulate large concurrent systems specified as Petri Nets; in our setting we are only simulating single automata and thus many optimizations of COSMOS are pure overhead. In Table 1 we can see that WORDGEN and COSMOS sampling times are of the same order of magnitude, WORDGEN being faster for small automata.

Moreover, these tools have a large constant startup time (Loading the JVM for PRISM, loading Python libraries for Sage, code generation and compilation for COSMOS) making running the tool-chain a long and tedious process when designing a given model.

Usability. Making the tools of the legacy tool-chain work together is quite cumbersome. Indeed, each of them is written in a different programming languages (Java, Python/Sage , C++) which requires to serialize data to files between each operation. The coordination between the three tools requires several scripts and adding new features means implementing modifications in each individual tools as well as in the file format of the exchange file.

New features. In WORDGEN the data-structure to represent distributions can be replaced easily. On-going work uses this to implement

m	Sage	Cosmos	WORDGEN		
		Sampling	Float	Rationnal	Sampling
1	2.4	163	0.02	0.03	51
5	4.8	179	0.1	0.1	122
10	16.2	265	0.5	0.12	208
15	48	383	2.2	3.1	374
20	114	542	6.7	9.7	475

Table 1: Performance of WORDGEN compared to previous tools for distribution computation and sampling. All times are in seconds. All computations have been done on a single core of an Intel i7-6700 processor. Sampling generates 50000 trajectories of length 100, which is the number of trajectories used to plot Fig. 1

sampling algorithms over different kinds of automata: symbolic automata and extensions over the work of [5, 6]. The modularity of WORDGEN makes it easy to prototype new ideas.

4 DATA STRUCTURE AND IMPLEMENTATION

WORDGEN is written in OCaml with some borrowing from other projects. It is freely available under the GPLv3 licence here [3]. The UPPAAL file parser, time-automaton data structure and Differential Bound Matrix (DBM) library are taken from SYMROB, a tool for the symbolic robustness analysis of time automata [18]. The automata data structure compute the synchronized automaton from a network of automata in the Uppaal formalism. The PRISM file parser is taken from COSMOS [2].

4.1 Rational numbers representation

The polynomials ring is built upon a rational field, two implementations are provided for rationals, either as floating point numbers or as exact rationals using arbitrary precision integers. WORDGEN can use an exact rational number library based on GMP[1]. Floating point numbers are faster but prone to numerical errors.

4.2 Web application

The tool WORDGEN features a graphical user interface available at [4]. This webpage is a slightly modified version of the full tool compiled to JavaScript. It fully run in the browser without any installation. It featured two editors for the timed automaton either as a graph or as a PRISM file and provide visualization of the results and internal representation using graphviz and gnuplot. Some functionalities are missing notably the UPPAAL file parser and the speed of the tool is reduced compared to the native desktop version. It is very user-friendly compared to the native command-line version.

4.3 Usage of the command-line application

To run WORDGEN on a unix commandline, one is to specify the automaton, the number of iteration of the distribution computation (the m in equation 1) and the number of trajectories. Assuming the file `twoears.prism` contains the automaton describes in figure 3(a) we can run WORDGEN as follows where `--poly` specify the parameter m and `--traj` specify the number of trajectories:

```
>wordgen twoears.prism --poly 5 --traj 2
Reading Prism automaton file. [0.0s]
```

[1]https://gmplib.org

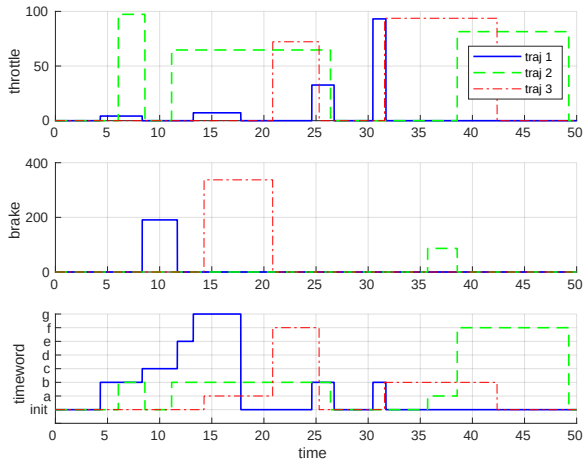


Figure 4: Three trajectories generated from the driving automata of [7].

```
Computing forward reachability graph ... 3 states found. [0.0s]
Splitting reachability graph ... 4 states and 15 DBMs found. [0.0s
    ]
Computing Distribution[...]] -> 5: [|||||] [0.0s]
Volume in initial state:240.283333333, degree of liberty:5
0.108885[b] 1.156834[b] 0.473015[a] 1.147844[b] 1.410200[a]
0.380190[a] 0.104014[a] 1.730205[b] 0.220164[a] 0.692068[a]
```

WORDGEN features many options and parameters, the web application is a great way to explore them as they are displayed as a HTML form.

4.4 Signal Generation with Breach

To convert a timed word into a $n-$dimensional signal, i.e., a function from time in $\mathbb{R}^+$ to $\mathbb{R}^n$, the idea is to map the sequence of pairs of durations and labels to a sequence of *control points* which are vectors in the time domain $\mathbb{R}^+ \times \mathbb{R}^n$ that Breach[2] can use as a base for interpolation using different interpolation schemes, e.g., constant, linear, splines, etc. In practice, we need to define parameters for control points for each transition in a timed word of a given length, which is a tedious process. To make it easier, we implemented a wrapper class that only requires the user to map each event in the alphabet Σ to values or ranges of values for signals. To sample a signal, WORDGEN sample a timed word, then Breach uses its builtin sampling methods to generate a value in the domain defined by each label of each transition.

As an example, consider the automata used in [7] to define inputs to an automatic transmission system for a car. We first indicate the name of the file defining the automaton, the timed word length and the names of the signals to generate, namely for throttle and braking.

```
TA_filename = 'driving_TA.prism';
num_evts = 10;
signals = {'throttle','brake'};
```

Then for each symbol in the alphabet of the automaton, we define either a single value or a range of values that the signals will be in when a label in a transition corresponds to this symbol.

[2]https://github.com/decyphir/breach

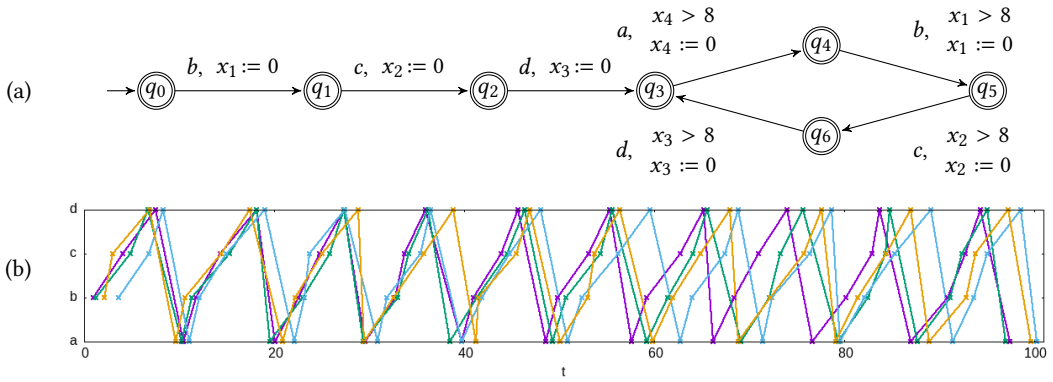


Figure 5: (a) A timed automaton for pseudo-periodic signal. To avoid overloading the figure, a global clock x (reset to 0 at each transition) and the global invariant $x_1, x_2, x_3, x_4 < 12$ and $x < 4$ hold for each guard and are not depicted. (b) Five time words of length 40 sampled from (a) with the discounted uniform sampling with $m = 5$, each action *a,b,c,d* are given a value shown in ordinates.

```
% label init: Throttle range [0, 100] but brake is 0
labels_ranges.init.brake = 0;
labels_ranges.init.throttle = [0 100];

% label a: no acceleration and no braking
labels_ranges.a.throttle =0;
labels_ranges.a.brake =0;

% label b: no acceleration, braking in range [0, 350]
labels_ranges.a.brake = [0 350];
labels_ranges.b.throttle =0;

% label c: no braking
labels_ranges.c.brake =0;
(...)
```

A `BreachTASignalGen` can then be created and used to generate signals. It is derived from Breach native class `BreachSignalGen`.

```
B_ta = BreachTASignalGen(signals, TA_filename,labels_ranges,
     num_evts);
B_ta.SampleDomain(5) % sample 5 timed words calling wordgen
B_ta.Sim(0:0.01:50); % interpolate control points for t in [0, 50]
B_ta.PlotSignals();
```

The resulting figure is shown on Fig. 4.

5 APPLICATION TO CPS FALSIFICATION

In [7] two case studies are proposed where Wordgen is used to falsify CPS by generating timed words interpreted as signals. We mentioned the driving automata in the previous section and give more details about the second one related to $\Delta\Sigma$ modulators. $\Delta\Sigma$ modulators are important components of analog-to-digital converters. Practical quantizers have limited input and output ranges, which may lead them to saturation, and we want to check whether the output ever saturates. Specifically, we consider the falsification of the absence of saturation of some quantizer signal *Out* under a certain class of nearly oscillatory inputs *In*. Formally *In* and *Out* must satisfy for some $t_s \geq 0$ and $\forall t \geq 0$,

$$|Out(t)| < 2 \tag{2}$$

$$\exists T \in [8t_s, 12t_s] \text{ such that } In(t+T) = In(T) \tag{3}$$

Standard falsification approaches translates requirements in Signal Temporal Logic (STL). Encoding (2) as an STL formula is trivial: $\varphi_{\neg\text{sat}} = \Box|Out| < p_{sat}$. However, enforcing that *In* satisfies (3) is not so simple. For instance, unbounded periodic properties are known to be beyond STL expressivity [17], and this is before considering that periods may be uncertain.

To solve this problem using Wordgen, an automata was designed as a model of a pseudo-periodic signal, shown on Fig. 5(a). The important part is the cycle of length 4, each transition of this TA has its own clock (x_1, x_2, x_3, x_4) which is reset when the transition is taken. Fig. 5(b) depicts time words sampled from this automaton by Wordgen. The idea is to extend this automaton into a signal generator interpolating the signal values between points of a periodic discrete sequence of the form:
$u_0\ \tau_0\ u_1\ \tau_1\ u_2\ \tau_2\ u_3\ \tau_3\ \ u_0\ \tau_4\ u_1\ \tau_5\ u_2\ \tau_6\ u_3\ \tau_7\ u_0\ \tau_N\ u_{\hat{N}} \ldots$
The value $In(t)$ is obtained by finding k such that $\sum_0^k \tau_i \leq t < \sum_0^{k+1} \tau_i$ and interpolating between $u_{\overline{k}}$ and $u_{\overline{k}+1}$ where $\overline{k}$ is the remainder of $k/4$. Since the discrete sequence u_i is periodic, the resulting signal satisfies (3) iff $\forall i,\ 8t_s \leq \tau_i + \tau_{i+1} + \tau_{i+2} + \tau_{i+3} \leq 12t_s$. Note that this constraint is satisfied by the delays of the timed words of our TA of Fig. 5(a). Hence by using Wordgen to generate timed words and mapping labels *a, b, c, d* to values u_0, u_1, u_2, u_3 we obtain the desired signals.

For the saturation threshold $p_{sat} = 2$ used in the model [10], the property $\varphi_{\neg\text{sat}}$ was easily falsified in our optimization setting for arbitrary long timed word. Using other falsification methods required to use rejection sampling which was only tractable for signals of length up to three periods.

6 CONCLUSION AND FUTURE WORK

We have presented Wordgen, a uniform sampler for timed language. To our knowledge it is the first standalone tool able to perform this sampling. Compared to previous tool, Wordgen is faster, easier to use, feature a graphical user interface and have more features. It has been successfully used for signal generation in the context of falsification of cyber-physical systems where other approaches fail.

Wordgen was designed as a fast prototyping tool, in the future we plan to incorporate several new features. By using Boltzmann sampling, we will be able to sample words of different lengths. Using recent work on time automaton volume computation, we will be able to control the average duration of generated timed word and to deal with unbounded clocks.

REFERENCES

[1] Eugene Asarin, Nicolas Basset, and Aldric Degorre. 2015. Entropy of regular timed languages. *Information and Computation* 241 (2015), 142–176.

[2] Paolo Ballarini, Benoît Barbot, Marie Duflot, Serge Haddad, and Nihal Pekergin. 2015. HASL: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation* 90, 0 (2015), 53 – 77.

[3] Benoît Barbot. 2023. Wordgen git repository. https://git.lacl.fr/barbot/wordgen

[4] Benoît Barbot. 2023. Wordgen Web Application. https://lacl.fr/~barbot/wordgen/

[5] Benoît Barbot, Nicolas Basset, Marc Beunardeau, and Marta Kwiatkowska. 2016. Uniform Sampling for Timed Automata with Application to Language Inclusion Measurement. In *QEST 2016, Quebec City, QC, Canada, August 23-25, 2016*. 175–190.

[6] Benoît Barbot, Nicolas Basset, and Thao Dang. 2019. Generation of Signals Under Temporal Constraints for CPS Testing. In *NASA Formal Methods - 11th International Symposium, NFM 2019*. 54–70.

[7] Benoît Barbot, Nicolas Basset, Thao Dang, Alexandre Donzé, James Kapinski, and Tomoya Yamaguchi. 2020. Falsification of Cyber-Physical Systems with Constrained Signal Spaces. In *NASA Formal Methods - 12th International Symposium, NFM, 2020*. 420–439.

[8] Johan Bengtsson and Wang Yi. 2003. Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets*. Springer, 87–124.

[9] Dimitri Bohlender, Harold Bruintjes, Sebastian Junges, Jens Katelaan, Viet Yen Nguyen, and Thomas Noll. 2014. A review of statistical model checking pitfalls on real-time stochastic models. In *ISOLA*. Springer, 177–192.

[10] S. Brigati, F. Francesconi, P. Malcovati, D. Tonietto, A. Baschirotto, and F. Maloberti. 1999. Modeling Sigma-Delta modulator non-idealities in Simulink. In *ISCAS'99. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI*, Vol. 2. 384–387 vol.2.

[11] Carlos E Budde, Pedro R D'Argenio, Arnd Hartmanns, and Sean Sedwards. 2018. A statistical model checker for nondeterminism and rare events. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 340–358.

[12] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. 2015. Uppaal SMC tutorial. *International journal on software tools for technology transfer* 17, 4 (2015), 397–415.

[13] A. Donzé. 2010. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *CAV*. 167–170.

[14] E. T. Jaynes. 1957. Information Theory and Statistical Mechanics. *Physical Review* 106, 4 (1957), 620–630. https://doi.org/10.1103/PhysRev.106.620 Publisher: American Physical Society.

[15] Marta Kwiatkowska, Gethin Norman, and David Parker. 2002. PRISM: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 200–204.

[16] Kim G Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. *International journal on software tools for technology transfer* 1, 1 (1997), 134–152.

[17] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *FORMATS/FTRTFT*. 152–166.

[18] Ocan Sankur. 2015. Symbolic quantitative robustness analysis of timed automata. In *TACAS*. Springer, 484–498.