Timescales: A Benchmark Generator for MTL Monitoring Tools

Dogan Ulus*

Boston University
Boston, MA, USA

Abstract. This article presents a benchmark generator, Timescales, which can be used to evaluate the performance and scalability of runtime verification tools using Metric Temporal Logic (MTL) formulas as their specifications. We mainly target runtime verification of cyber-physical systems and generate traces similar to the qualitative behavior of sensor readings and state variables of such systems that are observed/sampled continuously. Since such systems are composed of many heterogeneous components that work over very different time scales, it is crucial to measure the performance of the MTL monitoring tool for a wide range of timing parameters in specifications. Hence, Timescales supports the generation of benchmarks for 10 typical timed properties for any given trace length and timing parameters with several other useful features. Finally, we include some default benchmark suites generated by Timescales.

1 Introduction

Cyber physical systems (CPS) refer to large-scale interconnected control and communication systems that incorporate physical and computing components at various levels. Real-time decision making is the defining characteristic of cyber physical systems such that it does not only mean responding to soft and hard deadlines but also making decisions in a timely manner based on an ongoing interaction between the system and environment. The design of CPS often involves heteregeneous components that operate on different temporal scales and the correct operation requires a high level of coordination and cooperation among such components. Overall the design results in very complex artifacts to verify the correctness and evaluate the performance. Therefore, we usually resort to conventional simulation and model based testing methods for the analysis of CPS. Current research efforts include developing fast, scalable, and versatile runtime verification techniques and tools that handle complex timing requirements of CPS and provide an additional level of rigor and effectiveness over conventional testing methods.

Metric Temporal Logic (MTL) [6] is a popular formalism to specify temporal properties with timing constraints over the behavior of cyber physical systems.

* Affiliated with Samsung Semiconductor, Inc. at the time of acceptance

Several existing runtime verification (RV) tools support MTL as their specification and employ different techniques and algorithms to monitor MTL formulas over temporal behaviors (traces) [2, 1, 7, 8, 9]. From any MTL monitoring tool, we typically expect a linear-time performance in both the length of the trace and the size of the formula. Furthermore, the insensitivity to the base time unit and numeric time values in formulas is a very much desired feature for MTL monitoring tools. This becomes especially crucial for CPS applications since we usually have to use small and large numerical constants in the same formula when specifying timing requirements of components that operate on different timescales. Hence, in this paper, we present Timescales benchmark generator, which can be used to evaluate the performance and scalability of MTL monitoring tools. In particular, Timescales generates temporal behaviors for a predefined set of MTL formulas with parameterized timing constraints. We consider the past and future fragments of MTL seperately and our benchmarks consist of one trace file in the comma-seperated values (CSV[1]) format and two (past and future) specification files in the YAML[2] format. Besides we provide ANTLR[3] grammar files to parse our MTL formulas. These file formats are widely supported and their implementations are already available for many programming languages.

Timescales is essentially developed to help measure the typical performance of MTL monitoring tools. Therefore, we, first and foremost, consider the most common types of timed properties encountered in real system designs. Such typical properties have been studied in the papers [5, 4], which employs the template system developed by Dwyer et. al for untimed specifications [3]. In this template system, a property consists of (1) a pattern, which describes what must be observed, and (2) a scope, which describes the temporal extent of the pattern. Each property further contains one or two timing parameters, which can be controlled by the user. In Timescales, we support a total of 10 typical timed properties over 4 pattern (absence, universality, recurrence, response) and 4 temporal scopes (before, after, between, globally) for the benchmark generation. Then, the tool generates a discrete time behavior that satisfy the property for a given property, duration, and values of timing parameters. We also provide an option to generate dense time behaviors where the maximum density is controlled by the user. Since we usually want to generate a collection of benchmarks by varying all these parameters over a grid, we include some example scripts to generate such benchmark suites with some predefined values.

The structure of this paper is as follows. Section 2 overviews the syntax and semantics of MTL as used in this paper. In Section 3, we describe supported timed properties and corresponding MTL formulas for the benchmark generation. Section 4 explains the benchmark generation generally and gives further details about the implementation including trace generation, output formats, and default benchmark suites generated by Timescales.

[1] CSV is a common text-based data exchange format to store a sequence of data fields. (`https://en.wikipedia.org/wiki/Comma-separated_values`)
[2] YAML is a human-readable configuration file format. (`https://yaml.org`)
[3] ANTLR is a powerful parser generator tool. (`https://www.antlr.org`)

2 Metric Temporal Logic

Metric Temporal Logic (MTL) [6] is an extension of linear temporal logic (LTL) in which temporal operators are endowed with timing constraints. In this paper, we interpret MTL formulas over a bounded discrete time domain $\mathbb{T} = [1, N]$ of total duration N and use so-called non-strict (reflexive) semantics of temporal operators, *timed since* ($\mathcal{S}_I$) and *timed until* ($\mathcal{U}_I$). Given a finite set P of atomic propositions, the formulas of MTL are defined by the following grammar:

$$\varphi \;=\; p \mid \neg\,\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\,\mathcal{S}_I\,\varphi_2 \mid \varphi_1\,\mathcal{U}_I\,\varphi_2$$

where $p \in P$ and $I \subseteq [0, \infty)$. Then the satisfaction relation $(w, t) \vDash \varphi$ indicates that the Boolean temporal behavior $w : \mathbb{T} \to \mathbb{B}^P$ satisfies the formula φ at the time point $t \in \mathbb{T}$ as follows.

$$\begin{array}{lll}
(w,t) \vDash p & \leftrightarrow w_p(t) = \top \\
(w,t) \vDash \neg\,\varphi & \leftrightarrow (w,t) \nvDash \varphi \\
(w,t) \vDash \varphi_1 \vee \varphi_2 & \leftrightarrow (w,t) \vDash \varphi_1 \text{ or } (w,t) \vDash \varphi_2 \\
(w,t) \vDash \varphi_1\ \mathcal{S}_I\ \varphi_2 & \leftrightarrow \exists t' \leq t.\ (w,t') \vDash \varphi_2 \text{ and} \\
 & \qquad \forall t'' \in (t', t].\ (w,t'') \vDash \varphi_1 \text{ and} \\
 & \qquad\quad t - t' \in I \\
(w,t) \vDash \varphi_1\ \mathcal{U}_I\ \varphi_2 & \leftrightarrow \exists t' \geq t.\ (w,t') \vDash \varphi_2 \text{ and} \\
 & \qquad \forall t'' \in [t, t').\ (w,t'') \vDash \varphi_1 \text{ and} \\
 & \qquad\quad t' - t \in I
\end{array}$$

where we use $w_p : \mathbb{T} \to \mathbb{B}$ to denote the projection of w onto its component p. Other timed modalities include time constrained variants of *sometime in the past* ($\blacklozenge_I$) and *always in the past* ($\blacksquare_I$) as well as *sometime in the future* ($\lozenge_I$) and *always in the future* ($\square_I$). For any temporal operator, we usually omit the time bound if there is no constraint and call such an operator untimed. The past fragment of MTL, or past MTL in short, is defined as a restriction of MTL without the until operator. Similarly we do not use the since operator in the future fragment. In this paper, we write MTL formulas either in the past or future fragment of MTL.

3 Supported Properties

This section overviews supported timed properties in Timescales. By default a single benchmark consists of a logical formula that capture the property either in the past or future fragment of MTL and a temporal behavior that satisfies the formula. We select 10 typical timed properties for the benchmark generation and we cover in these properties all types of temporal operators and different types of intervals for timing constraints. Each timed property is parameterized with one or two timing parameters in order to generate benchmarks with different timing characteristics. Note that we do not claim or seek the logical equivalence between the past and future MTL formulas in the following but capture the intention in both fragments.

Bounded Absence After Q. This property has one timing parameter u. Intuitively it means that it is always the case that the event P does not occur at least for `U` time units after the event Q occurs. We capture this property by the formula

$$\blacksquare\big(\ \blacklozenge_{[0,u]}\ Q \longrightarrow (\neg P\ \mathcal{S}\ Q)\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ Q \longrightarrow \square_{[0,u]}\neg P\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Absence Before R. This property has one timing parameter u. Intuitively it means that it is always the case that the event P does not occur at least for u time units before the event R occurs. We capture this property by the formula

$$\blacksquare\big(\ R \longrightarrow \blacksquare_{[0,u]}\neg P\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ \lozenge_{[0,u]}\ R \longrightarrow (\neg P\ \mathcal{U}\ R)\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Absence Between Q and R. This property has two timing parameters l and u. Intuitively it means that it is always the case that the event P does not occur between events Q and R and the duration between Q and R is in l and u time units. We capture this property by the formula

$$\blacksquare\big(\ (R \wedge \neg Q \wedge \blacklozenge Q) \longrightarrow (\neg P\ \mathcal{S}_{[l,u]}\ Q)\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ (Q \wedge \neg R \wedge \lozenge R) \longrightarrow (\neg P\ \mathcal{U}_{[l,u]}\ R)\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Universality After Q. This property has one timing parameter u. Intuitively it means that it is always the case that the event P always occurs at least for u time units after the event Q occurs. We capture this property by the formula

$$\blacksquare\big(\ \blacklozenge_{[0,u]}\ Q \longrightarrow (P\ \mathcal{S}\ Q)\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ Q \longrightarrow \square_{[0,u]}P\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Universality Before R This property has one timing parameter u. Intuitively it means that it is always the case that the event P always occurs at least for u time units before the event R occurs. We capture this property by the formula

$$\blacksquare\big(\ R \longrightarrow \blacksquare_{[0,u]} P\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ \Diamond_{[0,u]}\ R \longrightarrow (P\ \mathcal{U}\ R)\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Universality Between Q and R. This property has two timing parameters l and u. Intuitively it means that it is always the case that the event P always occurs between events Q and R and the duration between Q and R is in l and u time units. We capture this property by the formula

$$\blacksquare\big(\ (R \wedge \neg Q \wedge \blacklozenge Q) \longrightarrow (P\ \mathcal{S}_{[l,u]}\ Q)\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ (Q \wedge \neg R \wedge \Diamond R) \longrightarrow (P\ \mathcal{U}_{[l,u]}\ R)\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Recurrence Globally. This property has one timing parameter u. Intuitively it means that it is always the case that the event P occurs at least for every u time units. We capture this property by the formula

$$\blacksquare\blacklozenge_{[0,u]} P \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\Diamond_{[0,u]} P \tag{Future}$$

in the future fragment of MTL.

Bounded Recurrence Between Q and R. This property has one timing parameter u. Intuitively it means that it is always the case that the event P occurs at least for every u time units between events Q and R. We capture this property by the formula

$$\blacksquare\big(\ (R \wedge \neg Q \wedge \blacklozenge Q) \longrightarrow (\blacklozenge_{[0,u]}(P \vee Q)\ \mathcal{S}\ Q)\big) \tag{Past}$$

in the past fragment of MTL and by the formula

$$\square\big(\ (Q \wedge \neg R \wedge \Diamond R) \longrightarrow (\Diamond_{[0,u]}(P \vee R)\ \mathcal{U}\ R)\big) \tag{Future}$$

in the future fragment of MTL.

Bounded Response Globally. This property has two timing parameters l and u. Intuitively it means that it is always the case that the event S responds to the event P in l and u time units. We capture this property by the formula

$$\blacksquare\big((S \longrightarrow \blacklozenge_{[l,u]} P) \wedge \neg(\neg S \; \mathcal{S}_{[u,\infty)} \; P)\big) \qquad \text{(Past)}$$

in the past fragment of MTL and by the formula

$$\square\big(P \longrightarrow \lozenge_{[l,u]} S\big) \qquad \text{(Future)}$$

in the future fragment of MTL.

Bounded Response Between Q and R. This property has two timing parameters l and u. Intuitively it means that it is always the case that the event S responds to the event P in l and u time units between events Q and R. We capture this property by the formula

$$\blacksquare\Big((R \wedge \neg Q \wedge \blacklozenge Q) \longrightarrow \big((S \longrightarrow \blacklozenge_{[l,u]} P) \wedge \neg(\neg S \; \mathcal{S}_{[u,\infty)} P)\big)\Big) \qquad \text{(Past)}$$

in the past fragment of MTL and by the formula

$$\square\Big((Q \wedge \neg R \wedge \lozenge R) \longrightarrow \big(P \longrightarrow (\neg R \; \mathcal{U}_{[l,u]}(\neg R \wedge S)) \; \mathcal{U} \; R\big)\Big) \qquad \text{(Future)}$$

in the future fragment of MTL.

4 Implementation

Timescales is an open source command line program[4] written in Python. For example, using Timescales, we generate a benchmark for the property bounded universality property between Q and R by executing the command

```
timescales always_bqr --lbound 300 --ubound 600 --duration 1000
```

where the argument specifies the duration of trace and arguments `lbound` and `ubound` specify the value for lower and upper timing parameters of the property, respectively. Then Timescales produces a concrete specification file, which contains an MTL formula, as a standard YAML file and a trace as a standard CSV file. These output formats are simple human-readable text files and very well supported in virtually all major programming languages. In the following, we give further details on the trace generation, output formats, and default benchmark suites.

[4] `https://github.com/doganulus/timescales`

4.1 Trace Generation

Given a timed property and parameter values, Timescales generate a temporal behavior in a periodic fashion where each period contains a sequence of propositional values that satisfies the property. For each period, we randomly determine actual timings of the sequence according to timing constraints specified by `--lbound` and `--ubound` arguments. Consequently, the duration of each period varies accordingly and we terminate the generation process once we exceed the total duration specified by the argument `--duration`. For example, suppose we want to generate a trace for the property `always_bqr` for $l = 300$ and $u = 600$. We start each period with a time point where Q holds and end with a time point where R holds. Then the actual number of time points where P holds between Q and R is randomly selected between 300 and 600. We repeat this procedure as many times as needed.

For benchmark generation, the command line interface of Timescales provides a few more customization options as shown in Figure 1. First, properties that contain some kind of recurrence have additional parameters to specify the minimum and maximum number of recurrence inside a single period. By the arguments `--min-recur` and `--max-recur` , we can specify a range and actual number of recurrences is randomly selected to be within this range. Secondly, generated temporal behaviors may involve the repetition of the same value over long periods called stuttering periods. We provide an option to condense such behaviors by omitting a time point in the trace file if the next time point also

```
usage: timescales [-h] [-d INT] [-l INT] [-u INT] [--min-
    recur INT] [--max-recur INT] [--name STRING] [--condense
    INT] [--failing-end] [--future] [--output-dir DIR]
    property

positional arguments:
property
    absent_aq             UBOUND
    absent_br             UBOUND
    absent_bqr    LBOUND  UBOUND

    always_aq             UBOUND
    always_br             UBOUND
    always_bqr    LBOUND  UBOUND

    recur_glb             UBOUND
    recur_bqr             UBOUND  MIN_RECUR  MAX_RECUR

    resp_glb      LBOUND  UBOUND
    resp_bqr      LBOUND  UBOUND  MIN_RECUR  MAX_RECUR
```

Fig. 1: Timescales Command Line Interface

has the same value. By the argument `--condense`, we can control the amount of condensation, which caps the duration of such omitted periods. Choosing a large value (such as larger than the total duration) would eliminate any stuttering in the trace file while choosing zero means there would be no condensation and the trace file explicitly includes every time point. Finally, we provide another option to append a sequence that makes the property fail at the end of the trace. This trick often serves a sanity check for that the monitoring algorithm does actually find an error (thus not silently fails) during the benchmarking. The option `--failing-end` enables this behavior.

4.2 Output Formats

Timescales produces a specification file in the standard YAML format and a temporal behavior in the standard CSV format as its output. These are simple human-readable text files and several implementations to read and write these formats already exist in virtually all major programming languages. Hence, Timescales can be used directly for RV tools that support these formats or else requires little implementation effort for the rest.

At the top of Figure 2, we show an example specification file that we have generated using Timescales. The name attribute indicates the name of property, which can be overwritten via the command line, and the spec attribute denotes the actual MTL formula. We provide our MTL grammar as an ANTLR grammar file so that an ANTLR parser can be automatically generated to parse our MTL formulas. The bottom row of Figure 2 demonstrates three example CSV files

```
---
name : "always_bqr_300_600"
spec : "(r && !q && once q ) -> (p since[300:600] q)"
```

```
time,q,p,r
1,   0,0,0
2,   1,1,0
3,   0,1,0
4,   0,1,0
5,   0,1,0
6,   0,1,0
7,   0,1,1
8,   0,0,0
9,   0,0,0
```

```
time,q,p,r
1,   0,0,0
2,   1,1,0
6,   0,1,0
7,   0,1,1
16,  0,0,0
17,  1,0,0
18,  0,1,0
27,  0,1,1
29,  0,0,0
```

```
time,q,p,r
10,  0,0,0
11,  1,1,0
111, 0,1,0
211, 0,1,0
311, 0,1,0
411, 0,1,0
454, 0,1,0
455, 0,1,1
500, 0,0,0
```

Fig. 2: (Top) An example specification file generated for the property with timing values 300 and 600. (Bottom Left) An example discrete time behavior in CSV format. (Bottom Middle) A dense time behavior with limited 10. (Bottom Right) A dense time behavior with limited 100. Note that whitespaces in CSVs are for the aesthetics.

generated using Timescales. The leftmost CSV file contains a row of propositional values for each time point in the time domain and represents a discrete time behavior. The CSV file show a condensed trace where some time points are omitted in the file if the next time has the same value of propositions. Finally the rightmost CSV is generated with the option `--condense 100` and therefore the duration of time jumps in the file is capped by 100 time units.

4.3 Default Benchmark Suites

In the source code distribution, we also include a Makefile script to generate three predefined sets of benchmarks named as small, large, and full suites using the generator. First, the small suite is intended for initial testing and demonstration purposes. It contains one benchmark for each supported property (10 in total) with small timing bounds over short traces with a duration of 1000 time units. Secondly, the large suite targets discrete time MTL monitoring tools and contains three benchmarks for each property (30 in total) with increasingly larger time bounds (1x, 10x, and 100x) over traces with a duration of one million time units. These numbers are often sufficient to check whether a runtime verification tool scales towards large time bounds in the specification. The total size of the large suite is about 400MB and not included in the distribution. Finally, the full suite extends the large suite by varying the length of generated traces (10K, 100K, 1M) and the amount of condensation (1, 10, 100). These benchmarks can be used to check the scalability of the tool with respect to the trace length as well as compare discrete and dense time implementations if the tool supports both settings.

5 Conclusion

In this paper, we presented Timescales benchmark generator to help testing the performance and scalability of MTL monitoring tools over the most common types of timed properties encountered in real designs. In particular, we have been interested in checking such tools when the specification contains large values of timing constraints and ideally expect that large timing constraints do not deteriorate the performance. This is especially important in runtime verification of cyber-physical systems, which involves various cooperating components operating in different timescales. Hence, our main motivation has been to generate benchmarks of the same property with the varying timing constraints and measuring the performance of the tool over them. In our tool, we have considered 10 typical properties for the benchmark generation and provided various customization options, which can be accessed easily via the command line interface. We adhered standard file formats for the implementation and tried to conform current practices in runtime verification as much as possible.

References

[1] David Basin, Felix Klaedtke, and Eugen Zalinescu. "The MonPoly Monitoring Tool". In: *Proceedings of the Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES)*. Vol. 3. 2017, pp. 19–28.

[2] David Basin, Srdjan Krstic, and Dmitriy Traytel. "AERIAL: Almost Event-Rate Independent Algorithms for Monitoring Metric Regular Properties." In: *Proceedings of the Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES)*. 2017, pp. 29–36.

[3] Matthew B Dwyer, George S Avrunin, and James C Corbett. "Property specification patterns for finite-state verification". In: *Proceedings of the Workshop on Formal Methods in Software Practice (FMSP)*. 1998, pp. 7–15.

[4] Volker Gruhn and Ralf Laue. "Patterns for timed property specifications". In: *Electronic Notes in Theoretical Computer Science* 153.2 (2006), pp. 117–133.

[5] Sascha Konrad and Betty HC Cheng. "Real-time specification patterns". In: *Proceedings of the International Conference on Software engineering (ICSE)*. 2005, pp. 372–381.

[6] Ron Koymans. "Specifying Real-Time Properties with Metric Temporal Logic". In: *Real-Time Systems* 2.4 (1990), pp. 255–299.

[7] Dejan Ničković, Olivier Lebeltel, Oded Maler, Thomas Ferrère, and Dogan Ulus. "AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic". In: *Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2018, pp. 303–319.

[8] Johann Schumann, Patrick Moosbrugger, and Kristin Y Rozier. "R2U2: monitoring and diagnosis of security threats for unmanned aerial systems". In: *Proceedings of the Conference on Runtime Verification (RV)*. 2015, pp. 233–249.

[9] Dogan Ulus. "Online Monitoring of Metric Temporal Logic using Sequential Networks". In: *arXiv preprint arXiv:1901.00175* (2019).