Kalpa Publications in Computing
Volume 3, 2017, Pages 29–36
RV-CuBES 2017. An International Workshop on
Competitions, Usability, Benchmarks, Evaluation,
and Standardisation for Runtime Verification Tools

AERIAL: Almost Event-Rate Independent Algorithms for Monitoring Metric Regular Properties

David Basin, Srđan Krstić, and Dmitriy Traytel

Institute of Information Security, Department of Computer Science, ETH Zürich, Switzerland

Abstract

We present AERIAL, a tool for the online monitoring of metric regular properties. AERIAL supports both the standard metric temporal logic (MTL) and the more expressive metric dynamic logic (MDL) as its property specification language. Unlike MTL, which is restricted to star-free properties, MDL can express all metric regular properties by generalizing MTL's temporal operators to arbitrary regular expressions. AERIAL's distinguishing feature is its ability to monitor an event stream using memory logarithmic in the event rate. This space efficiency is achieved by altering how AERIAL outputs its monitoring verdicts.

1 Introduction

We consider the *online monitoring problem*: Given a stream of time-stamped data, called events, and a property formulated in a specification language, determine whether the property is satisfied *for every* prefix of the stream. The input stream in the problem formulation is typically an abstraction of a running system. We do not consider any further details of the abstracted system: neither its implementation nor the way the events were generated and collected from it. An (*online*) *monitor* processes the input stream event-wise and reports violations as soon as it identifies them. This online processing is in contrast with *offline monitoring*, which processes a finite number of events given in their entirety, for example as a log file.

The main challenge in offline monitoring is to efficiently handle a large *volume* of events in a log. In contrast, in the online case, the *event rate* (or *velocity*) of the event stream, i.e., the rate at which the events arrive [3], poses an additional challenge. Monitors that can efficiently handle large volumes of events are historically called *trace-length independent* [7, 9] because their memory consumption is independent of the size of the event log or the prefix of the stream observed by the monitor. The stronger notion of *event-rate independence* [3] is paramount for monitors that need to handle streams with large and varying event rate. An event-rate independent monitor has memory usage that does not depend on the event rate.

In this paper we showcase our tool AERIAL [1], which is an *almost* event-rate independent online monitor. AERIAL's unique feature is that its memory consumption is logarithmic in the event rate of the input stream. AERIAL is open-source and publicly available under the LGPL free software license. Initially, AERIAL supported metric temporal logic (MTL) as its property specification language. However, it now also supports a strictly more expressive metric dynamic logic (MDL) [6]. MDL extends linear dynamic logic (LDL) [10], which uses regular expressions

as its temporal operators, with metric constraints and past temporal operators. Unlike MTL, MDL can express all regular properties, in particular also non-star-free properties.

This paper is structured as follows. Section 2 gives a high-level overview of AERIAL's input, output, and operation. Section 3 provides more details from the user's perspective, in particular its input and output syntax and various tool configurations. Section 4 summarizes our earlier evaluation of AERIAL [6], and Section 5 concludes the paper.

2 Overview

AERIAL [1] is an OCaml implementation of our almost event-rate independent monitoring algorithms for MTL [3] and MDL [6]. In the following, we provide more details on AERIAL's input, its output, and we give some insights about its internal architecture and operation. AERIAL takes an MTL or MDL formula and a stream of time-stamped events as input. It outputs a verdict stream.

MTL is an extension of LTL that can be used for specifying both qualitative and quantitative temporal properties. An instance of a qualitative property is that *a publish event is always preceded by a matching approval event*, which can be formalized in MTL as $\Box(\mathsf{publish} \rightarrow \bullet(\neg\mathsf{publish}\ \mathcal{S}\ \mathsf{approve}))$. We denote with $\Box$ the temporal operator "always in the future", with $\bullet$ the "previous" temporal operator, and with $\mathcal{S}$ the past temporal operator "since". We call a temporal formula any formula that has a top-level temporal operator (e.g., $\neg\mathsf{publish}\ \mathcal{S}\ \mathsf{approve}$) and an atomic formula a formula representing basic event occurrence (e.g., $\mathsf{publish}$).

Quantitative (or metric) temporal properties, in contrast, express time constraints between events, such as a publish event is always preceded by a matching approval event *within an hour*. Such a property can also be formalized in MTL by introducing time intervals to the temporal operators. The formula $\Box(\mathsf{publish} \rightarrow (\bullet_{[0,1hour]}\mathsf{approve} \vee (\blacklozenge_{[0,1hour]}\mathsf{approve} \wedge \bullet(\neg\mathsf{publish}\ \mathcal{S}\ \mathsf{approve}))))$ expresses precisely this property by associating the time interval $[0, 1hour]$ with the "previous" and "eventually in the past" (denoted with $\blacklozenge$) operators. A time interval's bounds must be non-negative integers (or infinity for the right bound). Typically, online monitors require all intervals associated with future operators to be finite, whereas AERIAL does not have this restriction.

It is well known that LTL cannot express all regular languages. For example, one cannot state that *an event occurs at every other position in a stream*. MTL inherits this limitation [8], which is often a problem in practice both for qualitative [13], and quantitative [8] properties. An example of a quantitative property that MTL cannot express is that *at any point in time, an action is approved and executed within a day and the approval happens before the execution*. To overcome this limitation, AERIAL supports MDL, which can express the above quantitative property as $\Box(\langle \top^* \cdot \mathsf{approve} \cdot \top^* \rangle_{[0,1day]}\ \mathsf{execute})$. An MDL formula of the form $\langle r \rangle_I\ \varphi$ has the metric future diamond top-level operator, which expresses that the formula φ is true at some future time-point with a time difference bounded by the interval I and that the regular expression r matches the portion of the event stream from the current point up to that future time-point. The atoms (or letters) of the regular expression r are MDL formulas themselves. Similarly, one can also use the past diamond operator: $\varphi\ {}_I\langle r \rangle$. A formula with a top-level diamond operator is considered a temporal formula. The diamond operators strictly generalize MTL's temporal operators. For example, the formula $\varphi\ \mathcal{U}_I\ \psi$ can be written in MDL as $\langle \varphi^* \rangle \psi$.

We interpret MTL and MDL formulas on (infinite) event streams (rather than on finite event logs), which are better suited abstractions for online monitoring. An event stream consists of an infinite sequence of time-points each containing an integer time-stamp and a set of events. We assume that the sequence is (non-strictly) ordered according to the value of the time-stamps.

For the formal definitions of MTL's and MDL's semantics we refer to our earlier work [3, 6].

At the high level, our monitoring algorithm iteratively processes the event stream, updates its internal state based on each processed event, and then potentially outputs one or more verdicts. Its internal state can be seen as a collection of verdicts for all *components* of the monitored formula at the current and previous time-point. The definition of a formula component depends on the particular input logic. In the case of MTL, components are *interval-skewed subformulas*, which include all the standard subformulas as well as all the variants of the temporal subformulas that have their intervals skewed (or shifted) down by some constant [3]. For example, the formula $\Diamond_{[2,4]}\varphi$ has four additional interval-skewed variants: $\Diamond_{[1,3]}\varphi$, $\Diamond_{[0,2]}\varphi$, $\Diamond_{[0,1]}\varphi$, and $\Diamond_{[0,0]}\varphi$. In the case of MDL, components additionally include all the variants of temporal subformulas that have regular expressions replaced with their *partial derivatives* [6]. For example, the formula $\langle b \rangle_I\ \varphi$ is a partial derivative variant of $\langle a \cdot b \rangle_I\ \varphi$. The algorithm updates the verdicts using dynamic programming [11], starting from the atomic formulas and reusing their verdicts to update their superformulas. Each verdict is updated based on a recursive definition of the satisfiability relation for temporal formulas. For instance, when monitoring a since formula $\varphi\ \mathcal{S}_{[a,b]}\ \psi$, AERIAL can conclude that it holds at some time-point i in the event stream if ψ holds at i and interval $[a, b]$ includes 0. Otherwise, it can check if φ holds at i and, if it does not hold, conclude that the formula is violated. If φ holds at i the satisfaction of the formula becomes dependent on the time-point $i-1$. Let Δ_i denote the difference between time-stamps at time-points i and $i-1$, then the formula is violated if $b < \Delta_i$. If $b \geq \Delta_i$, the formula $\varphi\ \mathcal{S}_{[a,b]}\ \psi$ holds at time-point i if and only if the interval-skewed formula $\varphi\ \mathcal{S}_{[a,b]-\Delta_i}\ \psi$ holds at the time point $i-1$. Since AERIAL maintains verdicts at the previous time-point for all the components in its internal state, it can reuse them to update the verdicts at the current time-point.

Future formulas complicate the picture. In that case, a formula's satisfiability may depend on the satisfiability of a component at the *next* time-point. Since the latter is unknown at the current time-point, the verdicts can be seen as Boolean expressions over truth values of the various components at the next time-point (rather than just Booleans). If the Boolean expression representing a verdict for the monitored formula Φ is a tautology, the monitor can output a positive verdict. If the expression is unsatisfiable, the monitor can output a negative verdict. If however the Boolean expression depends on the truth value of another component, AERIAL keeps the expression and the corresponding time-point, until the truth value is resolved.

The trick to achieving almost event-rate independence is in the way AERIAL handles these unresolved Boolean expressions. It eagerly finds all pairwise equivalent expressions and for each such pair it removes the expression with the larger time-point and outputs an equivalence verdict. This guarantees that only semantically different Boolean expressions are maintained by AERIAL at any given time along with their corresponding time-points. The number of such expressions is independent of the event rate. The stored time-points are represented as the current value of the time-stamp and the offset between the current and the first time-point with the same time-stamp. The size of the offset is logarithmic in the event rate; hence AERIAL is *almost* event-rate independent.

Overall, our implementation consists of about 2000 lines of OCaml code. Most of the code is purely functional and structured into modules. There are separate modules for manipulating MTL and MDL formulas, as well as a module that generates random formulas of a given size. The module that implements the almost event-rate independent monitoring algorithm is made generic to accommodate various input languages. Introducing a new input language is a matter of implementing a new module that handles the syntax representation and parsing, and implements the core dynamic programming function that updates the Boolean expressions stored in the monitor according to the next event for all operators in the language.

Operator	Primary syntax	Alternative syntax
True	$\top$	`true`
False	$\bot$	`false`
Negation	$\neg\, f$	`!` *f*, `NOT` *f*
Conjunction	$f1 \wedge f2$	*f1* `&` *f2*, *f1* `AND` *f2*
Disjunction	$f1 \vee f2$	*f1* `|` *f2*, *f1* `OR` *f2*
Implication	$f1 \rightarrow f2$	*f1* `->` *f2*, *f1* `=>` *f2*
Equivalence	$f1 \leftrightarrow f2$	*f1* `<->` *f2*, *f1* `<=>` *f2*
Next	$\bigcirc$ *int f*	`X` *int f*, `NEXT` *int f*
Previous	$\bullet$ *int f*	`Y` *int f*, `PREV` *int f*, `PREVIOUS` *int f*, `X`$^-$ *int f*
Weak Until	*f1* `W` *int f2*	*f1* `WEAK_UNTIL` *f2*
Until	*f1* `U` *int f2*	*f1* `UNTIL` *f2*
Since	*f1* `S` *int f2*	*f1* `SINCE` *f2*, *f1* `U`$^-$ *f2*
Release	*f1* `R` *int f2*	*f1* `RELEASE` *f2*
Trigger	*f1* `T` *int f2*	*f1* `TRIGGER` *f2*, *f1* `R`$^-$ *f2*
Eventually (future)	$\Diamond$ *int f*	`F` *int f*, `EVENTUALLY` *int f*, `FINALLY` *int f*
Eventually (past)	$\blacklozenge$ *int f*	`F`$^-$ *int f*, `ONCE` *int f*, `FINALLY_PAST` *int f*
Always (future)	$\Box$ *int f*	`G` *int f*, `ALWAYS` *int f*, `GLOBALLY` *int f*
Always (past)	$\blacksquare$ *int f*	`G`$^-$ *int f*, `HISTORICALLY` *int f*, `GLOBALLY_PAST` *int f*
Diamond (future)	`<`*r*`>` *int f*	
Diamond (past)	*f int* `<`*r*`>`	
Box (future)	`[`*r*`]` *int f*	
Box (past)	*f int* `[`*r*`]`	
Empty language	$\varnothing$	`empty`, `{}`
Empty word	ε	`epsilon`, λ
Wildcard	$\star$	`.`
Formula letter	*f*	
Test	*f* `?`	
Alternation	*r1* `+` *r2*	*r1* `|` *r2*
Concatenation	*r1 r2*	
Kleene star	*r* `*`	

Figure 1: AERIAL's formula (upper part) and regular expression (lower part) syntax where *f*, *f1*, and *f2* are formulas, *r*, *r1*, and *r2* are regular expressions, and *int* is an interval. The operators highlighted in gray are only allowed to occur in MDL formulas.

3 User Interface

In this section, we explain AERIAL's input syntax as well as some useful configuration options. AERIAL is a command line tool with various input parameters. For example, the invocation

```
$ aerial -fmla formula.txt -log events.stream -out verdicts.stream
```

tells AERIAL to read a formula from the `formula.txt` text file, then monitor the event stream accessible through the `events.stream` file descriptor, and write the resulting verdict stream in the `verdicts.stream` file descriptor. MDL is the assumed default input logic, since it subsumes MTL, and unless explicitly specified with a flag `-mtl`, AERIAL will convert any MTL formula into an equivalent MDL formula. Since our implementation of the MTL monitoring algorithm

```
@1307522571 approve execute
@1307532861 publish
@1307955600 publish
@1308477599 approve
@1308477599
@1308477599 execute
@1308477600 publish
 ...
```

(a) Input stream

```
1307522571:0 true
1307532861:0 false
1307955600:0 false
1308477599:0 true
1308477599:1 true
1308477599:2 true
1308477600:0 true
...
```

(b) Output for Φ

```
1307522571:0 false
1307532861:0 false
1307955600:0 false
1308477599:0 true
1308477599:2 = 1308477599:1
...
```

(c) Output for Ψ

Figure 2: Input stream and AERIAL's output

exhibits a slightly better performance (Section 4), one should carefully consider if the monitored property can be expressed in MTL. If the input or the output streams are omitted from the invocation, then standard input and output streams will be used instead.

To write MTL or MDL formulas in the AERIAL syntax, one can start from atomic formulas represented by the standard identifier syntax and combine them using the operators summarized in Figure 1. AERIAL accepts different ASCII and Unicode notations for most temporal and logical operators; for each operator, it supports at least one syntax using only ASCII characters. AERIAL supports open (e.g., `(0,42)`), half-open (e.g., `(1,3]`, `[6,∞)`, or `[0,INFINITY)`) and closed time intervals (e.g., `[1,6]`). Omitted intervals are interpreted as the interval `[0,∞)`. For instance, for the MTL formula $\Phi = \Box(\mathsf{publish} \rightarrow (\bullet_{[0,1hour]}\mathsf{approve} \vee (\blacklozenge_{[0,1hour]}\mathsf{approve} \wedge \bullet(\neg\mathsf{publish}\ \mathcal{S}\ \mathsf{approve})))$ both

```
publish → (●[0,3600] approve) ∨ (⧫[0,3600] approve) ∧ ●(¬publish S approve)
```

and

```
publish -> (PREV[0,3600] approve) OR
           (ONCE[0,3600] approve) AND PREV (!publish S approve)
```

are valid concrete textual inputs to AERIAL. Here, we assume that the unit of time for time-stamps used in the event stream is a second. The outermost temporal operator $\Box$ is always implicit, since AERIAL checks if properties hold at every time-point in the event stream by default.[1] Similarly, the MDL formula $\Psi = \Box(\langle \top^* \cdot \mathsf{approve} \cdot \top^* \rangle_{[0,1day]}\ \mathsf{execute})$ can be written as

```
<⊤* approve ⊤*> [0, 86400] execute
```

To represent event streams, we adopt the UNIX formatting for time-stamps and prefix them with the `@` symbol. Events at a time-point can be seen as occurring at the time denoted by the corresponding time-stamp. Events that are assumed to have happened simultaneously are grouped together and share the same time-stamp. Alternatively, an event stream can also contain several time-points with the same time-stamp. These events are considered to occur in the order defined by the event stream and it is assumed that the time difference between their occurrence is smaller than the granularity of the clock used when sampling the events. For example, for the event stream shown in Figure 2a, the time-point 0 contains two events `approve` and

[1] Making the outermost $\Box$ implicit allows AERIAL to report to the user at which point the property φ was violated. This makes the monitor's output more useful than a single Boolean verdict, which some monitors would report for $\Box\ \varphi$.

`execute` occurring simultaneously at the time 1307522571 (which is 2011-06-08, 08:42:51). The event `approve` at time-point 3 occurs strictly before the event `execute` at time-point 5, although their time-stamps have the same value. Empty events (time-point 4) are permitted, too.

If we monitor our example MTL formula Φ over the event stream from Figure 2a, AERIAL produces the output shown in Figure 2b, where each line represents a time-point and a verdict. As mentioned earlier, a time-point is represented as a pair consisting of the time-stamp, written before the `:` symbol and the offset written after the `:` symbol. The output shows that the occurrences of the first two `publish` events violate the property: the first one was approved too early while the second one was never approved. For the MDL formula Ψ, AERIAL produces the output shown in Figure 2c. The formula is violated at the time-point 0 because the approval and the execution happen simultaneously while the property requires the approval to happen strictly before the executions (and later executions are too far away in the future). Moreover, after processing the event at time-point 5 (the second execute), the monitor outputs the equivalence verdict `1308477599:2 = 1308477599:1`. This means that the monitor does not know the Boolean verdict for both time-points 5 (`1308477599:1`) and 6 (`1308477599:2`) but it knows that those verdicts will be equal regardless of the future events to come.

As another example consider the MTL formula $\Box(\Diamond \mathsf{alive})$. Such future properties with unbounded intervals are considered to be out-of-scope or "not monitorable" by traditional monitors [5]. AERIAL, however, can handle them: For the input formula $\Diamond$`alive` and the event stream in which no `alive` event occurs, AERIAL will output for every time-point $i \neq 0$ the equivalence of i to the time-point 0. If `alive` occurs for the first time at time-point j, AERIAL outputs `true` for the time-points 0 and j.

Recall from the previous section that AERIAL ensures almost event rate independence by keeping track of the earliest unresolved verdict, while outputting equivalence verdicts for any time-point that has a verdict semantically equivalent to a presently tracked one. A user can influence the way AERIAL outputs verdicts by specifying the flag `-mode` followed by `naive`, `global`, or `local`, which selects the NAIVE, GLOBAL, or LOCAL mode of operation, respectively. The NAIVE mode does not perform any equivalence checks and keeps all the unresolved verdicts as pending. This mode is not almost event-rate independent. The GLOBAL mode (default setting) adds the new expression only if there is no semantically equivalent expression. The LOCAL mode adds the new expression only if there is no semantically equivalent expression labeled with the same time-point.

Our monitor also offers the choice between two internal representations of Boolean expressions. Either they can be represented syntactically using the `-expr` flag (default setting), or as binary decision diagrams (BDDs) using the `-bdd` flag. For the former, it is important to keep the expressions small and thus they are eagerly normalized with respect to the associativity, commutativity, and idempotence of $\wedge$ and $\vee$, as well as Boolean tautologies such as $\top \wedge c = c$. Boolean expressions offer a low-cost substitution operation (used when updating the expressions), but are expensive to check for equivalence. The BDD representation, in contrast, offers easier equivalence checking, but substitution is more costly.

By default (or with the flag `-noflush`) AERIAL allows the operating system to buffer the verdicts and optimize the writing to the output stream. This behavior can be changed with the `-flush` flag, which is especially useful when using AERIAL interactively.

4 Evaluation Summary

In the past, we have evaluated AERIAL's time and memory efficiency using synthetic event streams and random formulas of various sizes as input [6]. We have also compared its monitoring

performance on MTL formulas to MONPOLY [4], which was the best performing tool that supports MTL in the First International Competition on Software for Runtime Verification [2] (CSRV 2014). For monitoring MDL properties, we have compared AERIAL to MONTRE [12], a state-of-the-art pattern matcher for *timed regular expressions* (*TRE*). We used streams with the event rate (i.e., number of time-points per time-stamp) ranging from 100 to 100,000 on average and random formulas with size (i.e., number of subformulas) ranging from 5 to 100.

The time performance of AERIAL's MTL monitoring algorithm was the best overall, whereby the MDL version of the algorithm was slower only by a small margin. For instance, when monitoring a formula $p\,\mathcal{U}_{[0,5]}\,(q\,\mathcal{U}_{[2,6]}\,r)$ over an event stream with event rate 100,000, both versions of the algorithm took about 40 seconds to process the first 10,000,000 events, while MONPOLY and MONTRE did not finish even after more than 100 seconds. As for memory consumption, AERIAL used at most 12MB of RAM for the monitoring tasks it performed during the evaluation. For the largest formulas, MONTRE used up to 250MB, while MONPOLY used 2GB. For both MONPOLY and MONTRE, the evaluation clearly showed that they were not implementing event-rate independent algorithms.

We also compared the three modes of operation of our tool: NAIVE, LOCAL, and GLOBAL. Our intuition about almost event-independence was confirmed in practice: for all formulas and streams, the space consumption in the NAIVE mode of our tool increases linearly in the event rate, while for LOCAL and GLOBAL modes it stays almost constant. From our evaluation, we also conclude that it is more efficient to translate Boolean expressions to BDDs when checking for verdict equivalence, than to maintain the BDD representation and perform the substitution operation on BDDs directly. Avoiding the direct BDD representation when monitoring very large MDL formulas rendered our monitor 50% faster, while the difference in memory consumption was negligible. For further details on the setup, AERIAL's performance, and our findings see [6].

5 Discussion and Future Work

We presented AERIAL, our almost event-rate independent monitor for metric temporal logic and metric dynamic logic. The tool is efficient and particularly well suited for monitoring high-velocity event streams. Although AERIAL's equivalence verdict output is nonstandard, we are convinced that it is useful. The output provides sufficient information to reconstruct all violations. Moreover, often the monitor's users are only interested in the existence of violations or in identifying the first (earliest) violation. When outputting equivalences, we ensure that the equivalence is output for the later time-points, while the earliest time-point stays in the monitor's memory and is eventually output with a Boolean verdict. Thus, users will always see a truth value at the earliest violating event.

As future work, we want to validate our tool and test its scalability in realistic scenarios using case studies supported by industrial partners. We will also explore if more expressive logics can be monitored in an almost event-rate independent way. Interesting logics to study are first-order extensions of MTL [5] (or MDL), in which events include data and formulas may quantify over the data domain.

Acknowledgment. The MTL algorithm behind AERIAL was developed jointly with Bhargav Bhatt. We thank four anonymous RV-CuBES reviewers for many insightful comments. Srđan Krstić is supported by the Swiss National Science Foundation grant Big Data Monitoring (167162). The authors are listed alphabetically.

References

[1] Aerial: An almost event-rate independent monitor for metric temporal logic and metric dynamic logic. https://bitbucket.org/traytel/aerial, 2017.

[2] E. Bartocci, Y. Falcone, B. Bonakdarpour, C. Colombo, N. Decker, K. Havelund, Y. Joshi, F. Klaedtke, R. Milewicz, G. Reger, G. Rosu, J. Signoles, D. Thoma, E. Zalinescu, and Y. Zhang. First international competition on runtime verification: rules, bint. j. softw. tools technol. transf.enchmarks, tools, and final results of CRV 2014. *Int. J. Softw. Tools Technol. Transf.*, 2017.

[3] D. Basin, B. Bhatt, and D. Traytel. Almost event-rate indepedent monitoring of metric temporal logic. In A. Legay and T. Margaria, editors, *TACAS 2017*, volume 10206 of *LNCS*, pages 94–112. Springer, 2017.

[4] D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. MONPOLY: Monitoring usage-control policies. In *RV 2011*, volume 7186 of *LNCS*, pages 360–364. Springer-Verlag, 2011.

[5] D. Basin, F. Klaedtke, S. Müller, and E. Zălinescu. Monitoring metric first-order temporal properties. *J. ACM*, 62(2):15:1–15:45, May 2015.

[6] D. Basin, S. Krstić, and D. Traytel. Almost event-rate indepedent monitoring of metric dynamic logic. In S. Lahiri and G. Reger, editors, *RV 2017*, volume 10548 of *LNCS*, pages 85–102. Springer, 2017.

[7] A. Bauer, J. Küster, and G. Vegliach. From propositional to first-order monitoring. In A. Legay and S. Bensalem, editors, *RV 2013*, volume 8174 of *LNCS*, pages 59–75. Springer, 2013.

[8] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. *Inf. Comput.*, 208(2):97–116, 2010.

[9] B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. Lola: Runtime monitoring of synchronous systems. In *TIME'05*, pages 166–174. IEEE Computer Society, 2005.

[10] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In F. Rossi, editor, *IJCAI-13*, pages 854–860. AAAI Press, 2013.

[11] K. Havelund and G. Roşu. Synthesizing monitors for safety properties. In J. Katoen and P. Stevens, editors, *TACAS 2002*, volume 2280 of *LNCS*, pages 342–356. Springer, 2002.

[12] D. Ulus. Montre: A tool for monitoring timed regular expressions. *arXiv preprint arXiv:1605.05963*, 2016.

[13] M. Y. Vardi. From Church and Prior to PSL. In O. Grumberg and H. Veith, editors, *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 150–171. Springer, 2008.