MLTL Benchmark Generation via Formula Progression

Jianwen Li(✉) and Kristin Y. Rozier

Iowa State University, Ames, IA, USA
{jianwen,kyrozier}@iastate.edu

Abstract. Safe cyber-physical system operation requires runtime verification (RV), yet the burgeoning collection of RV technologies remain comparatively untested due to a dearth of benchmarks with oracles enabling objectively evaluating their performance. Mission-time LTL (MLTL) adds integer temporal bounds to LTL to intuitively describe missions of such systems. An MLTL benchmark for runtime verification is a 3-tuple consisting of (1) an MLTL specification φ; (2) a set of finite input streams representing propositional system variables (call this computation π) over the alphabet of φ; (3) an oracle stream of $\langle v, t \rangle$ pairs where verdict v is the result (true or false) for time t of evaluating whether $\pi_t \models \varphi$ (computation π at time t satisfies formula φ). We introduce an algorithm for reliably generating MLTL benchmarks via formula progression. We prove its correctness, demonstrate it executes efficiently, and show how to use it to generate a variety of useful patterns for the evaluation and comparative analysis of RV tools.

1 Introduction

Runtime Verification (RV) provides the essential check that a system upholds its requirements during execution. Tools performing *online* or *stream-based* verification run on-board safety-critical systems, checking the current execution against the system's requirements in real time. RV is often expected, or even required, on-board modern human-interactive systems as it provides the essential capability to detect, and possibly mitigate, failures that could cause harm to people, property, or the environment. RV on-board an aircraft can provide the crucial trigger to abandon a mission or switch to safe mode in the face of the failure of a critical sensor. However it is essential that the RV tool be *correct*; a false-positive could trigger an abort unnecessarily and a false-negative would be equivalent to not running RV at all.

RV requirements are frequently expressed in Mission-time LTL (MLTL) [11], one of the many variations on Metric Temporal Logic [9], which has the syntax of Linear Temporal Logic with the option of integer bounds on the temporal operators. It provides the readability of LTL while assuming, when a different

Work supported by NASA ECF NNX16AR57G and NSF CAREER Award CNS-1552934.

C. Colombo and M. Leucker (Eds.): RV 2018, LNCS 11237, pp. 426–433, 2018.
https://doi.org/10.1007/978-3-030-03769-7_25

duration is not specified, that all requirements must be upheld during the (a priori known) length of a given mission, such as during the half-hour battery life of an Unmanned Aerial System (UAS). Using integer bounds instead of real-number or real-time bounds leads to more generic specifications that are adaptable to monitoring on different platforms (e.g., in software vs in hardware) with different granularities of time (e.g., because monitoring on-board an embedded system with more limited resources for storing the monitors may necessitate a wider granularity of time to fit the monitor encodings). We choose MLTL because it has been used for the Runtime Verification Benchmark Challenge [10] and in many industrial case studies [5,8,11,13–16]. Many specifications from other case studies, in logics such as MTL [1] and STL [7], can be represented in MLTL.

Arguably the biggest challenge facing the RV community today is the dearth of benchmarks for checking the correctness of RV tools and comparatively analyzing them [12]. An RV benchmark has three parts: (a) an input stream or *computation* π representing the values of the system variables over time; (b) an MLTL requirement φ; (c) an oracle $\mathcal{O}$, or output stream of tuples $\langle v, t\rangle$ where v is the valuation of φ (true or false) at time t for all $0 \leq t \leq M$ where M is the mission bound, or the number of time steps in the benchmark instance. The oracle is crucially required to evaluate correctness of RV algorithms but checking whether computation π satisfies requirement φ at each timestep in M is hard. Therefore, we create RV benchmarks by generating an MLTL requirement φ and deciding what pattern we'd like to see in our oracle (e.g., to achieve goals of code coverage for the RV tool under test). We utilize a new algorithm based on *formula progression* [3] over MLTL formulas to generate a π that satisfies φ at each timestep accordingly.

The contributions of this paper include a definition of formula progression for MLTL along with proofs of decomposibility and correctness. We design an RV benchmark generation algorithm based on MLTL formula progression, argue for its correctness, and show how to use it to generate different interesting benchmark patterns. Section 2 provides base definitions of MLTL semantics and benchmarks. We define MLTL formula progression in Sect. 3 and use it for benchmark generation algorithms in Sect. 4. Section 5 concludes.

2 Mission-Time Linear Temporal Logic (MLTL)

MLTL was first introduced in [11] as a variation on LTL with closed, finite integer intervals on the temporal operators that translate to practical concepts, such as mission bounds. A closed interval over naturals $I = [a, b]$ ($0 \leq a \leq b$ are natural numbers) is a set of naturals $\{i \mid a \leq i \leq b\}$. We focus on *bounded* intervals such that $b < +\infty$. All MLTL intervals I are closed because every open or half-open interval, e.g., in Metric Temporal Logic (MTL) [2], is reducible to an equivalent closed bounded interval. For example, (1,2) = $\emptyset$, (1,3) = [2,2], (1,3] = [2,3], etc. Let $\mathcal{P}$ be a set of atomic propositions, then the syntax of a formula in Mission-Time LTL (abbreviated as MLTL) is:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi U_I \varphi \mid \varphi R_I \varphi,$$

where I is a bounded interval, and $p \in \mathcal{P}$ is an *atom*. We use the abbreviations $F_I\varphi$ for $\mathsf{true}U_I\varphi$, $G_I\varphi$ for $\mathsf{false}R_I\varphi$, and $F_{[1,1]}\varphi$ for the equivalent of the LTL formula $X\varphi$.

The semantics of MLTL formulas are interpreted over finite traces. Let π be a finite trace in which every timestamp $\pi[i : i \geq 0]$ is over $2^{\mathcal{P}}$, and $|\pi|$ denotes the length of π; $|\pi| < +\infty$ because π is a finite trace. We use π^i (where $i \geq 1$) to represent the prefix of π ending at timestamp i (excluding i), and π_i (where $i \geq 0$) to represent the suffix of π starting from timestamp i (including i). Note that $\pi_i = \epsilon$ (empty trace) if $i \geq |\pi|$. Let $a, b : a \leq b$ be two natural numbers; we define that π models (satisfies) an MLTL formula φ, denoted as $\pi \models \varphi$, as follows:

- $\pi \models p$ iff $p \in \pi[0]$;
- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$;
- $\pi \models \varphi_1 \vee \varphi_2$ iff $\pi \models \varphi_1$ or $\pi \models \varphi_2$;
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$;
- $\pi \models \varphi_1 U_{[a,b]} \varphi_2$ iff $|\pi| > a$ and, there exists $i \in [a, b]$ such that $\pi_i \models \varphi_2$ and for every $j \in [a, b] : j < i$, it holds that $\pi_j \models \varphi_1$;
- $\pi \models \varphi_1 R_{[a,b]} \varphi_2$ iff $|\pi| \leq a$ or for every $i \in [a, b]$, either $\pi_i \models \varphi_2$ holds or there exists $j \in [a, b]$ s.t. $\pi_j \models \varphi_1$ and $\forall i, a \leq i \leq j, \pi_j \models \varphi_2$.

The Until and Release operators are interpreted slightly differently in MLTL than in the traditional MTL-over-naturals[1] [4]. In MTL-over-naturals, the satisfaction of $\varphi_1 U_I \varphi_2$ requires φ_1 to hold from position 0 to the position where φ_2 holds (in I), while in MLTL φ_1 is only required to hold within the interval I, before the time φ_2 holds. The same applies to the Release operator. From our experience in writing specifications, cf. [5,11,13–15], this adjustment is more user-friendly. Meanwhile, it is not hard to see that MLTL is as expressive as MTL-over-naturals: the formula $\varphi_1 U_{[a,b]} \varphi_2$ in MTL-over-naturals can be represented as $(G_{[0,a-1]}\varphi_1) \wedge (\varphi_1 U_{[a,b]} \varphi_2)$ in MLTL; $\varphi_1 U_{[a,b]} \varphi_2$ in MLTL can be represented as $F_{[a,a]}(\varphi_1 U_{[0,b-a]} \varphi_2)$ in MTL-over-naturals.

MLTL*Benchmarks.* One *benchmark instance* is a triple $\langle \pi, \varphi, \mathcal{O} \rangle$, where π is a finite trace of length $|\pi|$ over $(2^{\Sigma})^{|\pi|}$ representing the propositional variable input streams, φ is the MLTL requirement being monitored, and $\mathcal{O}$ is an oracle, itself a stream of pairs $\langle v, t \rangle$ such that verdict $v = true$ if $\pi_t \models \varphi$ and $v = false$ if not. An RV tool takes as input the formula φ and the finite trace set π and uses φ to generate a monitor; $\mathcal{O}$ is required to verify that the monitor operates correctly.

3 Formula Progression on MLTL

We introduce the concept of *formula progression* [3] over MLTL formulas.

Definition 1. *Given an* MLTL *formula* φ *and a finite trace* π*, let* φ' *be one formula progression of* φ*. We define the* progression function $prog(\varphi, \pi) = \varphi'$ *recursively:*

[1] In this paper, MTL-over-naturals is interpreted over finite traces.

– *if* $|\pi| = 1$, *then*
  - $prog(\mathsf{true}, \pi) = \mathsf{true}$ *and* $prog(\mathsf{false}, \pi) = \mathsf{false}$;
  - *if* $\varphi = p$ *is an atom,* $prog(\varphi, \pi) = \mathsf{true}$ *iff* $p \in \pi[0]$;
  - *if* $\varphi = \neg\psi$, $prog(\varphi, \pi) = \neg prog(\psi, \pi)$;
  - *if* $\varphi = \psi_1 \vee \psi_2$, $prog(\varphi, \pi) = prog(\psi_1, \pi) \vee prog(\psi_2, \pi)$;
  - *if* $\varphi = \psi_1 \wedge \psi_2$, $prog(\varphi, \pi) = prog(\psi_1, \pi) \wedge prog(\psi_2, \pi)$;
  - *if* $\varphi = \psi_1 U_{[a,b]} \psi_2$,
$$prog(\varphi, \pi) = \begin{cases} \psi_1 U_{[a-1,b-1]} \psi_2 & \text{if } 0 < a \leq b; \\ prog(\psi_2, \pi) \vee (prog(\psi_1, \pi) \wedge \psi_1 U_{[0,b-1]} \psi_2) & \text{if } 0 = a < b; \\ prog(\psi_2, \pi) & \text{if } 0 = a = b; \end{cases}$$
  - *if* $\varphi = F_{[a,b]} \psi_2$,
$$prog(\varphi, \pi) = \begin{cases} F_{[a-1,b-1]} \psi_2 & \text{if } 0 < a \leq b; \\ prog(\psi_2, \pi) \vee F_{[0,b-1]} \psi_2 & \text{if } 0 = a < b; \\ prog(\psi_2, \pi) & \text{if } 0 = a = b; \end{cases}$$
  - *if* $\varphi = \psi_1 R_{[a,b]} \psi_2$, $prog(\varphi, \pi) = \neg prog((\neg\psi_1) U_{[a,b]} (\neg\psi_2), \pi)$;
  - *if* $\varphi = G_{[a,b]} \psi_2$, $prog(\varphi, \pi) = \neg prog(F_{[a,b]}(\neg\psi_2), \pi)$;
– *else* $prog(\varphi, \pi) = prog(prog(\varphi, \pi[0]), \pi_1)$.

The procedure *prog* takes an MLTL formula φ and finite trace π as the input, and returns another MLTL formula by progressing π over φ. Figure 1 exemplifies formula progression over $\varphi = F_{[2,3]} a$ with respect to the trace $\pi = \{\neg a\}\{\neg a\}\{a\}$. From the figure, we have $prog(\varphi, \pi^1 (= \{\neg a\})) = F_{[1,2]} a$, $prog(\varphi, \pi^2 (= \{\neg a\}\{\neg a\})) = F_{[0,1]} a$, and $prog(\varphi, \pi^3 (= \{\neg a\}\{\neg a\}\{a\})) = \mathsf{true}$. Based on Definition 1, we have the following theorems.

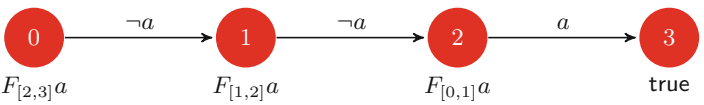

Fig. 1. The schema of $prog(F_{[2,3]} a, \pi = \{\neg a\}\{\neg a\}\{\neg a\}\{a\})$.

Theorem 1. Formula Progression Decomposition. *Let* φ *be an* MLTL *formula and* π *be a finite trace. Then formula progression on* π *can be decomposed into two progressions on the sub-traces (i.e.* π^k, π_k*) of* π *for an arbitrary* k *in the range* $1 \leq k \leq |\pi|$. *Formally,* $prog(\varphi, \pi) = prog(prog(\varphi, \pi^k), \pi_k)$ *for every* $1 \leq k \leq |\pi|$.

Proof. When $k = 1$, $prog(\varphi, \pi) = prog(prog(\varphi, \pi^1), \pi_1)$ is true based on Definition 1. Assume $prog(\varphi, \pi) = prog(prog(\varphi, \pi^k), \pi_k)$ is true for $1 \leq k < |\pi|$. Since $prog(prog(\varphi, \pi^k), \pi_k) = prog(prog(prog(\varphi, \pi^k), \pi[k]), \pi_{k+1})$ and $prog(prog(\varphi, \pi^k), \pi[k]) = prog(\varphi, \pi^{k+1})$ are true by Definition 1, we have the following is also true: $prog(\varphi, \pi) = prog(prog(\varphi, \pi^{k+1}), \pi_{k+1}) = prog(prog(\varphi, \pi^{k+1}), \pi_{k+1})$. □

Theorem 1 generalizes the recursive part of Definition 1. To perform formula progression over φ with respect to the finite trace π, it is equivalent first perform formula progression over φ with respect to the prefix of π up to k, i.e. π^k, and then perform formula progression over $prog(\varphi, \pi^k)$ with respect to the subfix of π from k, i.e., π_k.

Theorem 2. Satisfiability Preservation. *Let φ be an* MLTL *formula and π be a finite trace. Then π satisfies φ iff the suffix of π, i.e., π_k for some k, satisfies the formula obtained from formula progression over φ with respect to the prefix of π, i.e., π^k. Formally, $\pi \models \varphi$ iff $\pi_k \models prog(\varphi, \pi^k)$ for every $1 \leq k \leq |\pi|$.*

Proof. (Sketch.) When $k = 1$, the proof can be done by an induction over the construction of $prog(\varphi, \pi^1)$ (the base case in Definition 1). Inductively, assume $\pi \models \varphi$ iff $\pi_k \models prog(\varphi, \pi^k)$ is true for $1 \leq k < |\pi|$. From the hypothesis assumption, $\pi_k \models prog(\varphi, \pi^k)$ iff $\pi_{k+1} \models prog(prog(\varphi, \pi^k), \pi[k]) = prog(\varphi, \pi^{k+1})$ holds. As a result, we have that $\pi \models \varphi$ iff $\pi_{k+1} \models prog(\varphi, \pi^{k+1})$ holds. □

Theorem 2 states that the formula progression is able to preserve the satisfaction of π in terms of the MLTL formula φ.

Theorem 3. Correctness. *Let φ be an* MLTL *formula and π be a finite trace. Then $\pi \models \varphi$ holds iff $prog(\varphi, \pi) = \mathsf{true}$ holds.*

Proof. (Sketch.) For the base case when $|\pi| = 1$, the inductive proof can be done over the construction of $prog(\varphi, \pi)$ (the base case in Definition 1). When $|\pi| > 1$, we have $\pi \models \varphi$ iff $\pi_{|\pi|-1} \models prog(\varphi, \pi^{|\pi|-1})$ according to Theorem 2. Moreover, since $|\pi_{|\pi|-1}| = 1$ and we have proved that $\pi_{|\pi|-1} \models prog(\varphi, \pi^{|\pi|-1})$ iff $prog(prog(\varphi, \pi^{|\pi|-1}), \pi[|\pi| - 1]) = prog(\varphi, \pi) = \mathsf{true}$ holds (from Theorem 1), it is true that $\pi \models \varphi$ holds iff $prog(\varphi, \pi) = \mathsf{true}$ holds when $|\pi| > 1$. □

Theorem 3 is a direct conclusion from Theorem 2, considering the particular situation when formula progression has been performed on all timestamps of π.

Corollary 1. *For the* MLTL *formula φ and finite trace π, $\pi \models \varphi$ implies $\pi \cdot \pi' \models \varphi$ for any arbitrary finite trace π'.*

Proof. From Theorem 3, $\pi \models \varphi$ implies that $prog(\varphi, \pi) = \mathsf{true}$ holds. Since $\pi' \models \mathsf{true}$ and $prog(\varphi, \pi) = \mathsf{true}$ hold, it is true that $\pi \cdot \pi' \models \varphi$ based on Theorem 2. □

We use the theorems and corollary introduced above as the theoretic correctness guarantee of our benchmark construction algorithms in the following section.

4 Benchmark Generation

4.1 Random Pattern

We use the MLTL generation tool released in [6] to construct random MLTL formulas. Once the formula φ is generated, we create a finite trace over the

alphabet of the formula, i.e., Σ, with a random length (≥ 1) and assign a random assignment $P \in 2^{|\Sigma|}$ to each timestamp of the trace. We use the algorithm *prog* and Theorem 3 to generate π such that $\pi_k \models \varphi$ holds iff $prog(\varphi, \pi_k) = \mathsf{true}$ for $0 \leq k < |\pi|$. In this way, we can efficiently generate large sets of always-satisfiable benchmark instances, representing the case where the system always upholds its requirements.

4.2 Almost-Satisfiable Pattern

For an instance $\langle \varphi, \pi, \mathcal{O} \rangle$ under the *Almost-Satisfiable Pattern*, $\pi_k \models \varphi$ must be true for as many k as possible ($1 \leq k < |\pi|$). To generate such instances, we leverage both the MLTL-SAT [6] and formula progression techniques in the following procedure:

- Use the MLTL-SAT solver to generate a model (satisfying finite trace) π for the given formula φ. If no such model exists, φ is unsatisfiable and we discard it. Otherwise, $\pi(= \pi_0) \models \varphi$ and we push the pair $\langle 0, \mathsf{true} \rangle$ into $\mathcal{O}$;
- To pursue $\pi_k \models \varphi$ ($1 \leq k < |\pi|$) also being true, we extend π as follows:
  - First apply the formula progression technique to obtain the formula $prog(\varphi, \pi_k)$;
  - Use the MLTL-SAT solver to generate a model π' for $prog(\varphi, \pi_k)$. It may be possible that such model π' does not exist, in which case we push $\langle k, \mathsf{false} \rangle$ into $\mathcal{O}$ and terminate our attempt to make $\pi_k \models \varphi$;
  - If π' exists, update π with $\pi \cdot \pi'$. Theorem 2 guarantees that $\pi_k \models \varphi$ holds for the updated π. Push $\langle k, \mathsf{true} \rangle$ into $\mathcal{O}$;
  - The updated π also preserves the fact that $\pi_{k-1} \models \varphi$, i.e., the extension of π does not affect the previous truth evaluations, according to Corollary 1.
- To ensure termination, we set a mission length bound for the finite trace π.

The procedure $\mathsf{SAT}(\varphi)$ calls the MLTL-SAT solver to check the satisfiability of φ. Taking the MLTL formula φ and fixed length bound K for the generated trace in the instance, the procedure returns an instance of an Almost-Satisfiable Pattern.

Theorem 4 (Correctness). *Let $\langle \varphi, \pi, \mathcal{O} \rangle$ be the instance generated from Algorithm 1. Then we have $\pi_k \models \varphi$ iff $\langle k, \mathsf{true} \rangle \in \mathcal{O}$ for $1 \leq k \leq |\varphi|$.*

4.3 Almost-Unsatisfiable and Median-Satisfiable Patterns

We also consider the dual of Almost-Satisfiable Pattern, namely Almost-Unsatisfiable Pattern, each instance under which requires that $\pi_k \not\models \varphi$ be true for as many k as possible ($1 \leq k < |\pi|$). First we create an Almost-Satisfiable Pattern instance $\langle \varphi, \pi, \mathcal{O} \rangle$ as shown in the previous section. Then we negate the formula in the instance and set $\mathcal{O}' = \{\langle k, \mathsf{true} \rangle | \langle k, \mathsf{false} \rangle \in \mathcal{O}\} \cup \{\langle k, \mathsf{false} \rangle | \langle k, \mathsf{true} \rangle \in \mathcal{O}\}$. As a result, the instance $\langle \neg\varphi, \pi, \mathcal{O}' \rangle$ is under the Almost-Unsatisfiable Pattern.

Algorithm 1. The Pseudo-code to generate the Almost-Satisfiable Pattern instances

Require: An MLTL formula φ, and the length bound K for the generated finite trace.
Ensure: An instance $\langle \varphi, \pi, \mathcal{O} \rangle$ that is under Almost-Satisfiable Pattern.

if SAT(φ) return UNSAT **then**
  return $\langle \varphi, \epsilon, \mathcal{O} \rangle$ (ϵ is the empty trace);
end if
Let π be the model returned from SAT(φ);
while $1 \leq k < |\pi|$ **do**
  if $|\pi| > K$ **then**
    return $\langle \varphi, \pi, \mathcal{O} \rangle$;
  end if
  Let $\varphi' = prog(\varphi, \pi_k)$;
  if SAT(φ') return UNSAT **then**
    Push the pair $\langle k, \mathsf{false} \rangle$ into $\mathcal{O}$;
  else
    Let π' be the model returned from SAT(φ');
    Update $\pi = \pi \cdot \pi'$;
    Push the pair $\langle k, \mathsf{true} \rangle$ into $\mathcal{O}$;
  end if
end while
return π;

The Median-Satisfiable Pattern is a combination of the Almost-Satisfiable and Almost-Unsatisfiable Patterns; in each instance the number of timestamps on which the formula are satisfied is almost the same as that of timestamps on which the formula are falsified. To generate such an instance, we simply create an Almost-Satisfiable and Almost-Unsatisfiable Pattern instance respectively, i.e. $\langle \varphi, \pi_1, \mathcal{O}_1 \rangle$ and $\langle \varphi, \pi_2, \mathcal{O}_2 \rangle$, which have the same MLTL formula. Then the instance $\langle \varphi_1, \pi_1 \cdot \pi_2, \mathcal{O} \rangle$, where $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$, is under the Median-Satisfiable Pattern.

5 Conclusions and Future Work

By introducing algorithms for generating several crafted patterns of RV benchmarks, we have paved the way for the creation of a benchmark generation tool and the ability to create a large set of publicly-available benchmarks. Next, we plan to implement and experimentally evaluate the performance of our benchmark generation algorithms. After we optimize the performance to enable efficient generation of large sets of each type of benchmark, we plan to release our code and a database of generated instances.

References

1. Alur, R., Henzinger, T.: Real-time logics: complexity and expressiveness. In: Proceedings 5th IEEE Symposium on Logic in Computer Science, pp. 390–401 (1990)
2. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM **41**(1), 181–204 (1994)
3. Bacchus, F., Kabanza, F.: Planning for temporally extended goals. Ann. Math. Artif. Intell. **22**, 5–27 (1998)
4. Furia, C.A., Spoletini, P.: Tomorrow and all our yesterdays: MTL satisfiability over the integers. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 126–140. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85762-4_9
5. Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 215–230. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_18
6. Li, J., Rozier, K.Y., Vardi, M.Y.: Evaluating the satisfiability of mission-time LTL: a bounded MTL over naturals. Under submission (2018)
7. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
8. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. FMSD **51**, 1–31 (2017)
9. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 1–13. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85778-5_1
10. Reger, G., Rozier, K.Y., Stolz, V.: Runtime verification benchmark challenge, November 2018. https://www.rv-competition.org/2018-2/
11. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 357–372. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_24
12. Rozier, K.Y.: On the evaluation and comparison of runtime verification tools for hardware and cyber-physical systems. In: RV-CUBES, vol. 3, pp. 123–137. Kalpa Publications (2017)
13. Rozier, K.Y., Schumann, J., Ippolito, C.: Intelligent hardware-enabled sensor and software safety and health management for autonomous UAS. In: Technical Memorandum NASA/TM-2015-218817, NASA Ames Research Center, Moffett Field, CA 94035, May 2015
14. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2U2: Monitoring and diagnosis of security threats for unmanned aerial systems. In: RV. Springer-Verlag (2015)
15. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime Analysis with R2U2: A Tool Exhibition Report. In: RV. Springer-Verlag (2016)
16. Schumann, J., Rozier, K.Y., Reinbacher, T., Mengshoel, O.J., Mbaya, T., Ippolito, C.: Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. IJPHM **6**(1), 1–27 (2015)