Runime Monitoring of Temporal Logic Specifications using streamLAB

David Kooi

June 29, 2020

1 Motivation

Temporal logic is a formal method to describe the behavior requirements of Cyber Physical Systems(CPS). A CPS may use a temporal logic monitor to sample signals and execute a verification algorithm to check if those signals satisfy the formal requirements. Temporal logic requirements are often categorized as "Safety"(Always) and "Liveness"(Eventually) requirements. While monitors can be run offline during the integration and testing phase, it can be useful to perform online-time(run-time) monitoring during the operation.

Runtime monitoring is the verification approach of using information about a running system to detect (un)satisfactory behaviors. This information can be used for fault diagnosis and control.

2 Outline

This project looks into run-time monitoring of cyber-physical systems. First a background on real-valued logics is given. Then an example is setup to demonstrate run-time monitoring though periodic sampling. The streamLAB tool is used for run-time monitoring of a dynamical simulation. The example shows how if the sample rate can affect monitor correctness.

3 Background

3.1 Signal Temporal Logic

Signal temporal logic(STL) was introduced in 2004 by [9] as a way to quantify real-valued signals. The original STL was an extension of MITL(Metric Interval Temporal Logic)[2] for real valued signals. Later, in 2010, STL was extended in [1] to support quantitative measures of satisfaction. It is interesting to note that in in 2006 MTL(Metric Temporal Logic) was extended for the same quantitative measurement feature[4].

The progenitor of these real-time logics was MTL developed in 1990. [8]

3.1.1 Notation

- $s = s_0, s_1, s_2...$ is a stream of states where $s_k \in \mathbb{R}^n \times \mathbb{B}^m$
- $\tau = \tau_0, \tau_1, \tau_2...$ is a stream of time stamps where $\tau_k \in \mathbb{R}_{\geq 0}$
- $\mathcal{T} = \mathcal{T}_0\mathcal{T}_1\mathcal{T}_2... = (\tau_0, s_0)(\tau_1, s_1)(\tau_2, s_2)...$ is a timed state sequence
- AP is a finite set of atomic propositions with cardinality $|AP|$
- ϕ is the top level formula containing $|\varphi|$ sub-formulae
- φ_i is the i^{th}sub-formula of ϕ

We consider the set past-time $\mathbb{STL}$ formulas defined inductively as

$$\varphi := p_i \in AP \mid \neg\varphi \mid \varphi_i \wedge \varphi_j \mid \varphi_i \mathcal{S}_{\mathcal{I}} \varphi_j \mid \blacklozenge_{\mathcal{I}}\varphi \mid \blacksquare_{\mathcal{I}}\varphi$$

Each atomic predicate $p_i \in AP$ takes the form $p_i = f_i(s_k) > 0$

(Past-Time) $\mathbb{STL}$ Robustness Semantics

$$\begin{aligned}
\rho(\top, s, k) &= +\infty \\
\rho(\varphi_i \in AP, s, k) &= f_i(s_k) \\
\rho(\neg\varphi_i, s, k) &= -\rho(\varphi_i, s_k) \\
\rho(\varphi_i \wedge \varphi_j, s, k) &= \min(\rho(\varphi_i, s_k), \rho(\varphi_j, s_k)) \\
\rho(\blacklozenge_{\mathcal{I}}\varphi_i, s, k) &= \max_{\tau_n \in \tau_k - \mathcal{I}} \rho(\varphi_i, s_n) \\
\rho(\blacksquare_{\mathcal{I}}\varphi_i, s, k) &= \min_{\tau_n \in \tau_k - \mathcal{I}} \rho(\varphi_i, s, n) \\
\rho(\varphi_i \mathcal{S}_{\mathcal{I}} \varphi_j, s, k) &= \max_{\tau_n \in \tau_k - \mathcal{I}} \left[\min \left(\rho(\varphi_i, s, n), \min_{\tau_m \in [\tau_k, \tau_n]} \rho(\varphi_j, s, m) \right) \right]
\end{aligned}$$

(Past-Time) $\mathbb{STL}$ Boolean Satisfaction Semantics

$$\begin{aligned}
(s, k) \models p_i &\iff f_i(s_k) > 0 \\
(s, k) \models \neg\varphi_i &\iff (s, k) \not\models \varphi_i \\
(s, k) \models \varphi_i \wedge \varphi_j &\iff (s, k) \models \varphi_i) \wedge (s, k) \models \varphi_j) \\
(s, k) \models \blacklozenge_{\mathcal{I}}\varphi_i &\iff \exists t_j \in t_k - \mathcal{I} \text{ s.t } x(t_j) \models \varphi \\
(s, k) \models \blacksquare_{\mathcal{I}}\varphi &\iff \forall t_j \in t_k - \mathcal{I}, s(t_j) \models \varphi
\end{aligned}$$

3.2 Time Domain Specifications

Common time domain specifications for dynamical systems include rise-time, overshoot, and settling time requirements. t_r is the time for a state to reach 90% of the set-point, M_p is the over-shoot (maximum peak) and the settling time t_s is the time it takes for the system to each 1% of the set-point.

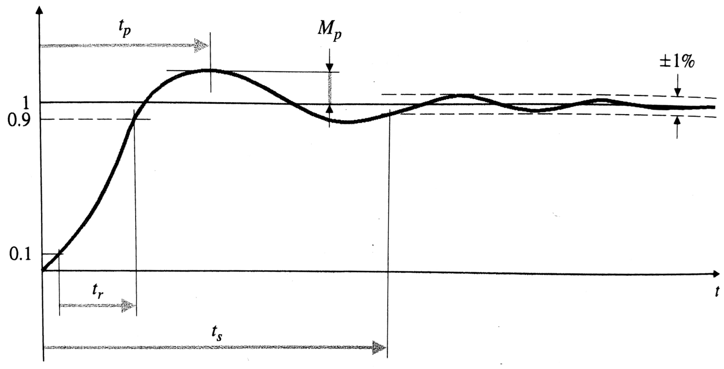


Figure 1: Time Domain Specifications[7]

4 Example: Step Response Requirements for a Satellite

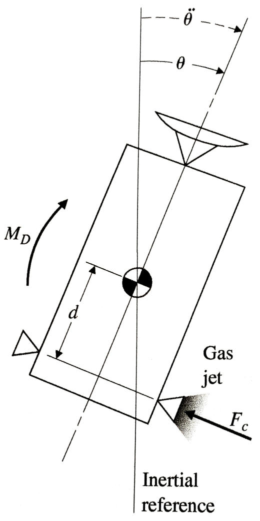


Figure 2: Satellite Model[7]

Consider a satellite with physical states $\vec{\theta} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$ and dynamics

$$\dot{\vec{\theta}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \vec{\theta} + \begin{bmatrix} 0 \\ \frac{d}{I} \end{bmatrix} u \tag{1}$$

Where d is the length between center of mass and the thruster, $I = \frac{M_D l^2}{12}$ is the moment of inertia. $u = -Kx$ is state feedback.

Suppose the step response requirements are formalized by the STL formula

$$\phi = \Box\,(\varphi_{RT} \wedge \varphi_{OS}) \tag{2}$$

φ_{RT} and φ_{OS} are rise-time and over-shoot requirements, respectively. They are defined as

$$\varphi_{RT} = \Diamond_{[0,1.5s]}\ |\theta - SP| < 0.1 \tag{3}$$

$$\varphi_{OS} = \Box\ \frac{M_p}{SP} < 25\% \tag{4}$$

4.1 The problem of causality

The above formulas are intuitively written in future time logic. But future time logic is non-causal: It requires knowledge of the future.

Future Time	$x(t) \models \Box_{[a,b]}\psi \iff \forall t' \in [t+a, t+b], x(t') \models \psi$
Past Time	$x(t) \models \blacksquare_{[a,b]}\psi \iff \forall t' \in [t-a, t-b], x(t') \models \psi$

Table 1: Example of (non)causality

This is okay if one records into memory the system's entire operation and and performs verification offline. But recall that the focus is on run-time monitoring.

There are several options to overcome the problem of causality:

(a) Restrict the temporal logic to it's past time fragment. This approach was followed in [11].

(b) Delay evaluation until a future time interval has passed. ϕ_f can be properly evaluated after b time has passed.

(c) Return indeterminate outputs until a determination can be made. This approach is taken in [3] where authors develop the notion of a robust satisfaction interval over partial signals and [10] where authors use an synchronous and asynchronous monitor to output real-time updates and delayed updates dependent on future data.

4.2 Translation to Past Time Temporal Logic

The approach this report takes is to use past-time logic for the monitor. The formal specification ϕ must be translated to a past-time equivalent. Equivalence between future-time and past-time temporal logic formulas can be proven using SMT or automata methods, but this is not pursued here. Consider the past-time translations:

Top Level Requirement

$$\phi = \Box\,(\varphi_{RT} \wedge \varphi_{OS})$$

to

$$\phi = \blacksquare\,(\varphi_{RT} \wedge \varphi_{OS})$$

Rise Time Requirement

$$\varphi_{RT} = \Diamond_{[0,1.5s]} \ \ |\theta - SP| < 0.1 \cdot SP$$

to

$$\varphi_{RT} = \blacklozenge_{[0,1.5s]}(t_k - 1.5 < 0 \ \ \wedge \ \ |\theta - SP| < 0.1 \cdot SP)$$

Remark: The rise-time requirement states that $|\theta - SP|$ has to be within 10% of SP within 1.5 seconds. To support the past-time formula the system state is augmented with time. The semantics of the formula are expanded for clarification:

$$(s, k) \models \blacklozenge_{[0,1.5s]}\varphi_{RT} \iff \exists t_j \in [t_k - 1.5, t_k] \text{ s.t } (s, j) \models (t_k - 1.5 < 0 \ \wedge \ |\theta_j - SP| < 0.1 \cdot SP)$$

$|\theta - SP| < 0.1 \cdot SP$ must be true at least once between $t_k = 0$ and $t_k = 1.5s$. If $|\theta - SP| < 0.1 \cdot SP$ occurs after $t_k > 1.5s$ then the system has missed the deadline. Simply put, φ_{RT} has to be true within the first $1.5s$.

Overshoot Requirement

$$\varphi_{OS} = \Box \ \frac{M_p}{SP} < 25\%$$

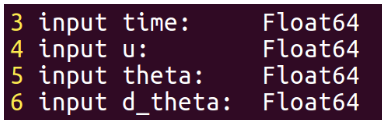
to

$$\varphi_{OS} = \blacksquare \ \frac{M_p}{SP} < 25\%$$

5 Monitor Implementation

Now that the formulas can be evaluated only using values in the past the next step is implementation of a run-time monitor. The tool streamLAB[5] is a real-time monitoring engine that uses rtLOLA[6] as temporal specifications.

streamLAB takes inputs as CSV formated strings and processes outputs according to the rt-LOLA specification. The data of the system consists of time, control input, and the system states: $t, u, \theta, \dot{\theta}$.



```
input time:     Float64
input u:        Float64
input theta:    Float64
input d_theta:  Float64
```

Figure 3: Monitor Inputs

Next, we need to decompose the formal specifications and define for each formula the robustness metric. For berevity $\rho(\phi, s, k) \equiv \rho(\phi)$.

5.1 Top Level Requirements and Robustness Metric

$$\phi = \blacksquare \ (\varphi_{RT} \wedge \varphi_{OS})$$
$$\rho(\phi) = \min(\rho(\varphi_R), \rho(\varphi_{OT}))$$

5.2 Rise Time Requirements

$$\begin{aligned}\varphi_{RT} &= \blacklozenge_{[0,r_t]}(t == 0 \quad \wedge \quad |\theta - SP| < 0.1)\\ &= \blacklozenge_{[0,r_t]}(\varphi_1 \quad \wedge \quad \varphi_2)\\ &= \blacklozenge_{[0,r_t]}\varphi_0\end{aligned}$$

Robustness Metrics

$$\begin{aligned}\rho(\varphi_{RT}) &= \max_{t'_k \in t_k - [0,r_t]} \rho(\varphi_0)\\ \rho(\varphi_0) &= \min(\rho(\varphi_1), \rho(\varphi_2))\\ \rho(\varphi_1) &= \begin{cases}\infty & \text{if } t < 1500ms\\ -\infty & \text{if } t \geq 1500ms\end{cases}\\ \rho(\varphi_2) &= 0.1 - |\theta - SP|\end{aligned}$$

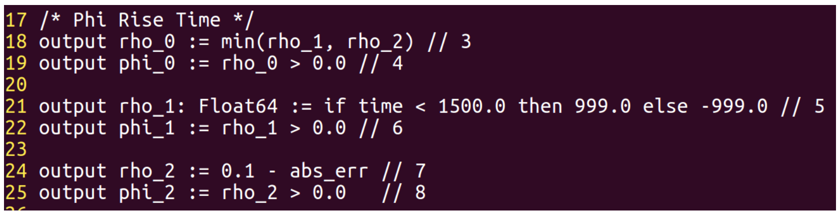
5.3 Overshoot Requirements

$$\begin{aligned}\varphi_{OS} &= \blacksquare \ \frac{M_p}{SP} < 25\%\\ &= \blacksquare \ \varphi_3\end{aligned}$$

Robustness Metrics

$$\begin{aligned}\rho(\varphi_{OS}) &= \min(\rho(\varphi_3))\\ \rho(\varphi_3) &= 0.25 - \frac{M_p}{SP}\end{aligned}$$

5.4 rtLOLA Specification Example



```
/* Phi Rise Time */
output rho_0 := min(rho_1, rho_2) // 3
output phi_0 := rho_0 > 0.0 // 4

output rho_1: Float64 := if time < 1500.0 then 999.0 else -999.0 // 5
output phi_1 := rho_1 > 0.0 // 6

output rho_2 := 0.1 - abs_err // 7
output phi_2 := rho_2 > 0.0   // 8
```

Figure 4: Risetime Requirements in rtLOLA

6 Periodic Monitoring Results

The system described by (1) was simulated and it's output fed to the streamLAB monitor. Results show present a good example of monitor (in)correctness.

6.1 Incorrect Monitor: Sample Rate 2Hz

Figure 8 highlights the rise-time requirements. At 2Hz the monitor misses any points that satisfy the requirement: There must be a sample within the overlapping green regions for the rise-time monitor capture a point that satisfies the rise-time specification. As a result $\rho(\varphi_{RT})$ is false and the top level formula is false.

Remark: streamLAB does not make any assumptions about sample times. One feature of streamLAB is the ability to handle asynchronous events and inputs. This is enabled by providing a time-stamp at each input. [5] That is, streamLAB guarantees that inputs will be evaluated with respect to their time-stamp.

Accordingly, it is the user's responsibility to provide time-stamped inputs at periods that correctly capture the system behavior.

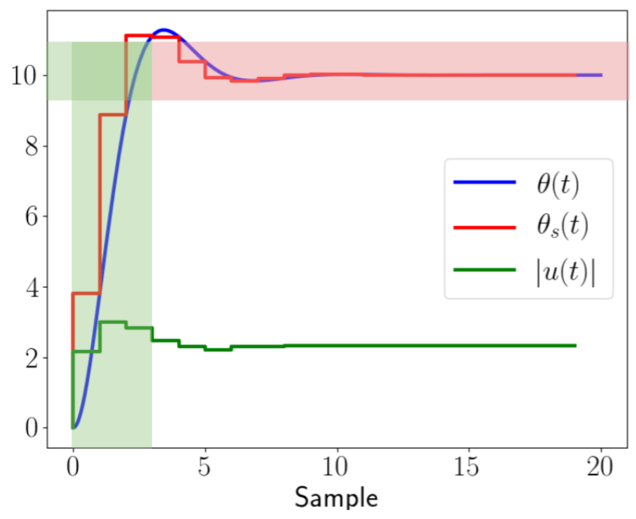


Figure 5: Monitoring at 2Hz.
$\theta(t)$: Physical signal; $\theta_s(t)$: Sampled signal

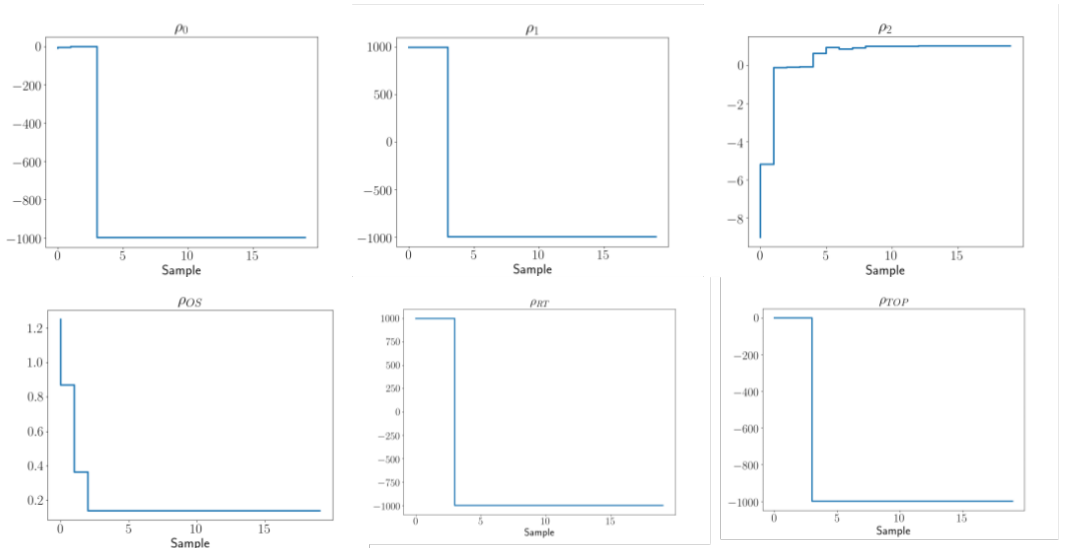


Figure 6: STL Robustness Metrics at 2Hz

6.2 Correct Monitor: Sample Rate 4Hz

Note that there is a sample point within the overlapping green region. This shows that the monitor was able to capture a point that satisfies the rise-time specification.

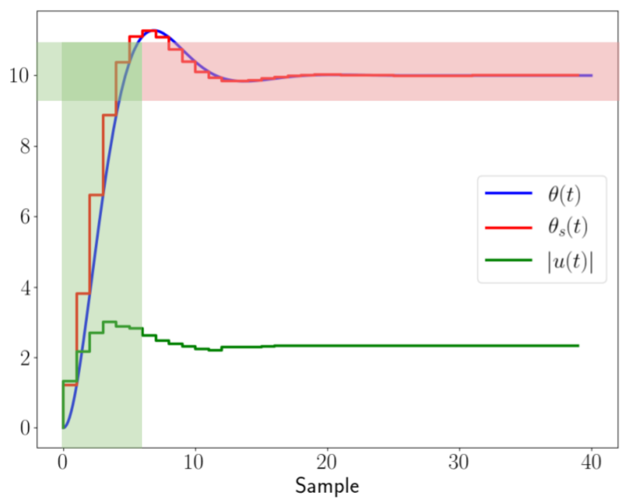


Figure 7: Monitoring at 4Hz.
$\theta(t)$: Physical signal; $\theta_s(t)$: Sampled signal

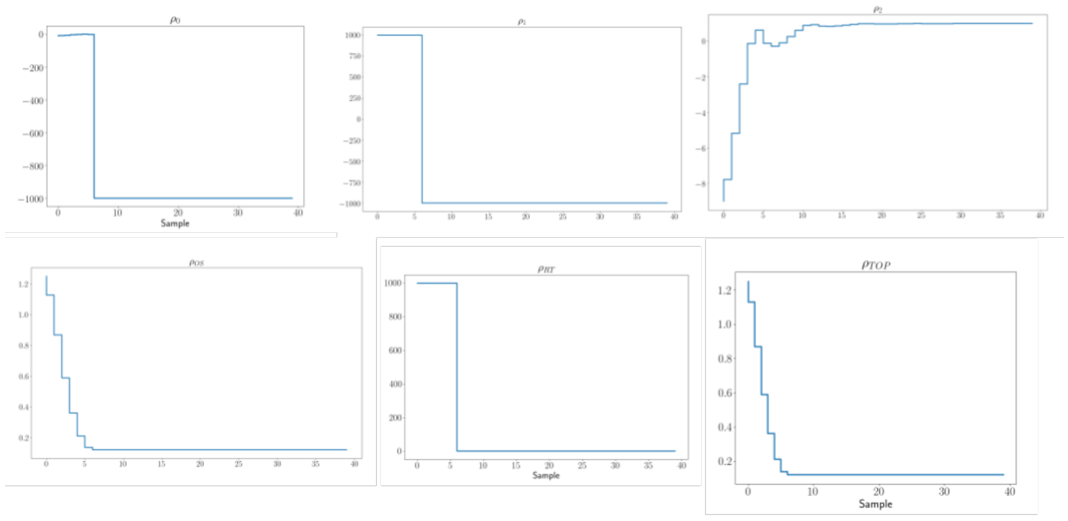


Figure 8: STL Robustness Metrics at 4Hz

7 Sample Rate Selection

The above example shows how a badly chosen sample rate can result in incorrect monitor conclusions. One way is to pick a 'high enough' sample rate. One heuristic is to sample at 40 times the system bandwidth[7].

However, since monitors can be computationally and memory intensive, is it possible to monitor when needed? That is: How should a monitor be executed in order that

1. The monitor is correct
2. The number of executions is minimized

7.1 Sample Rate Using Bounded Velocity

Following from the work on [4] the sample period T_S at state s_k can be chosen adaptively as

$$T_k < \frac{\rho(\phi, s, k)}{V} \tag{5}$$

where V is the maximum velocity expected by the system. This is intuitive: If $\rho(\cdot)$ is large there is more margin. But this approach uses a global bound and does not consider the direction of flow and may be overly conservative.

7.2 Sample Rate Using Direction of Dynamics

The next sample period can be chosen

$$T_k \leq \frac{\rho(\phi, s, k)}{M_s(s_k))} \tag{6}$$

with

$$M_s(s_k) = \sup_{s' \in R(\bar{T}, s_k)} \langle \nabla \rho(x'), f(x') \rangle \tag{7}$$

$R(\cdot)$ is the reachable set from s_k for $\bar{T}$ seconds. The kth sample now depends on the direction of robustness measure and the direction of the dynamics.

8 Conclusion and Future Work

This report gave and example of the tool streamLAB[5] for run-time monitoring. A the dynamics of a satellite was simulated checked against formal requirements. An illustration of incorrect monitor conclusion was shown.

Future work involves further work with adaptive sample rate and creating examples of it's use. Additional work involves creating runtime monitors from Lustre specifications. Lustre is a specification language like rtLOLA and can be used to generate reactive programs in C.

8.1 Challenges

Once challenge involved working with the cutting edge technology of streamLAB. There was a number of communications with the developers at Saarland University and a couple new releases in response to identified bugs. Certain work arounds were done to arrive at simulation results. (See specification code in appendix)

Another challenge was manual decomposition of the formal specification into it's subformulae.

There were also implementation challenges in interprocess communication between the python simulator and the streamLAB program. I initially wanted to illustrate a simple adaptive sample rate, but that requires feedback from the monitor. By the time the simulation pipeline was built it was difficult to pipe feedback from the monitor back to the simulator. This can be overcome by a smarter software design.

8.2 Acknowledgements

I would like thank Hazem Torfah from UC Berkely and the streamLAB team at the Saarland University for communications and new software releases.

References

[1] Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems*, volume 6246, pages 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[2] Rajeev Alur. The Benefits of Relaxing Punctuality.

[3] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromon a Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. Robust Online Monitoring of Signal Temporal Logic. *arXiv:1506.08234 [cs]*, June 2015.

[4] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42), September 2009.

[5] Peter Faymonville, Bernd Finkbeiner, Mal te Schledjewski, Maximilian Schwenger, Marvin Stenger, Leander Tentrup, and Hazem Torfah. StreamLAB: Stream-based Monitoring of Cyber-Physical Systems. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, volume 11561, pages 421–431. Springer International Publishing, Cham, 2019.

[6] Peter Faymonville, Bernd Finkbeiner, Maximi lian Schwenger, and Hazem Torfah. Real-time Stream-based Monitoring. *arXiv:1711.03829 [cs]*, June 2019. arXiv: 1711.03829.

[7] Gene F Franklin, J David Powell, Abbas Emami-Naeini, and J David Powell. *Feedback control of dynamic systems*, volume 3. Addison-Wesley Reading, MA, 1994.

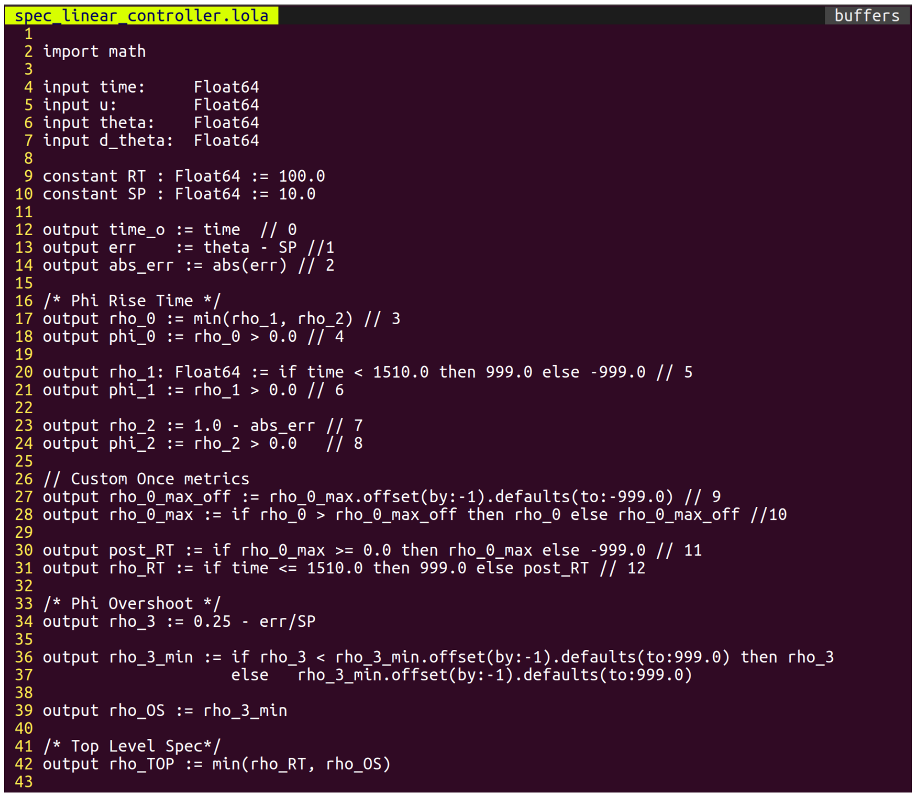
[8] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, pages 255–299, November 1990.

[9] Oded Maler and Dejan Nickovic. Monitoring Temporal Properties of Continuous Signals. volume 3253. Berlin, Heidelberg, 2004.

[10] Thomas Reinbacher, Kristin Yvonne Rozier, and Jo hann Schumann. Temporal-Logic Based Runtime Observer Pairs for Syst em Health Management of Real-Time Systems. volume 8413, pages 357–372. Berlin, Heidelberg, 2014.

[11] Dogan Ulus. Online Monitoring of Metric Temporal Logic using Sequential Networks. January 2019.

Appendices



```

import math

input time:     Float64
input u:        Float64
input theta:    Float64
input d_theta:  Float64

constant RT : Float64 := 100.0
constant SP : Float64 := 10.0

output time_o := time  // 0
output err     := theta - SP //1
output abs_err := abs(err) // 2

/* Phi Rise Time */
output rho_0 := min(rho_1, rho_2) // 3
output phi_0 := rho_0 > 0.0 // 4

output rho_1: Float64 := if time < 1510.0 then 999.0 else -999.0 // 5
output phi_1 := rho_1 > 0.0 // 6

output rho_2 := 1.0 - abs_err // 7
output phi_2 := rho_2 > 0.0   // 8

// Custom Once metrics
output rho_0_max_off := rho_0_max.offset(by:-1).defaults(to:-999.0) // 9
output rho_0_max := if rho_0 > rho_0_max_off then rho_0 else rho_0_max_off //10

output post_RT := if rho_0_max >= 0.0 then rho_0_max else -999.0 // 11
output rho_RT := if time <= 1510.0 then 999.0 else post_RT // 12

/* Phi Overshoot */
output rho_3 := 0.25 - err/SP

output rho_3_min := if rho_3 < rho_3_min.offset(by:-1).defaults(to:999.0) then rho_3
                    else   rho_3_min.offset(by:-1).defaults(to:999.0)

output rho_OS := rho_3_min

/* Top Level Spec*/
output rho_TOP := min(rho_RT, rho_OS)

```

Figure 9: rtLOLA Specification Code