RTAMT: Online Robustness Monitors from STL

Dejan Ničković[1(✉)] and Tomoya Yamaguchi[2]

[1] AIT Austrian Institute of Technology, Vienna, Austria
dejan.nickovic@ait.ac.at
[2] Toyota Research Institute - North America, Ann Arbor, Michigan, USA

Abstract. We present rtamt, an online monitoring library for Signal Temporal Logic (STL) and its interface-aware variant (IA-STL), providing both discrete- and dense-time interpretation of the logic. We also introduce rtamt4ros, a tool that integrates rtamt with Robotic Operating System (ROS), a common environment for developing robotic applications. We evaluate rtamt and rtamt4ros on two robotic case studies.

1 Introduction

Robotic applications are complex autonomous cyber-physical systems (CPS). Robotic Operating System (ROS) [1] provides a meta-operating system that helps development of robotic applications. Verification remains a bottleneck, as existing techniques do not scale to this level of complexity, thus making static safety assurance a very costly, if not impossible, activity. Run-time assurance (RTA) is an alternative approach for ensuring the safe operation of robotic CPS that cannot be statically verified. RTA allows the use of untrusted components in a system that implements a safe fallback mechanism for (1) detecting anomalies during real-time system operations and (2) invoking a recovery mechanism that brings the system back to its safe operation. Runtime verification (RV) provides a reliable and rigorous way for finding violations in system executions and consequently represents a viable solution for the monitoring RTA component.

Formal specifications play an important role in RV and enable formulating system properties. Signal Temporal Logic (STL) [2] is a formal specification language used to describe CPS properties. It admits *robustness* semantics that measure how far is an observed behavior from satisfying/violating a specification.

We introduce rtamt[1], an online STL monitoring library. rtamt supports standard STL and its *interface-aware* extension (IA-STL) [3] as specification languages. It provides automated generation of *online robustness* monitors from specifications under both *discrete* and *continuous* interpretation of time. We also present rtamt4ros[2], an extension that integrates rtamt to ROS, thus enabling the use of specification-based RV methods in robotic applications. We assess the library on two robotic applications.

[1] https://github.com/nickovic/rtamt.
[2] https://github.com/nickovic/rtamt4ros.

D. V. Hung and O. Sokolsky (Eds.): ATVA 2020, LNCS 12302, pp. 564–571, 2020.
https://doi.org/10.1007/978-3-030-59152-6_34

Related Work. Several tools support *offline* monitoring of STL with qualitative (AMT2.0 [4]) and quantitative semantics (S-TaLiRo [5] and Breach [6]). Reelay [7] implements past Metric Temporal Logic (MTL) monitors over discrete and continuous time and with qualitative and quantitative semantics. PyMTL [8] is a library for quantitative offline evaluation of MTL specifications. R2U2 tool [9] combines runtime observers for the discrete *mission-time linear temporal logic* (mtLTL), with Bayesian networks, sensor filters and Boolean testers. MONTRE [10] implements monitoring algorithms for *timed regular expressions* (TRE). MONAA [11] implements an automata-based matching algorithms for TREs. StreamLAB [12] and TeSSLa [13] are tools for evaluating real-time CPS streams. The problem of online robustness monitoring was studied in [14], where the authors propose an interval-based approach of online evaluation that allows estimating the minimum and the maximum robustness with respect to both the observed prefix and unobserved trace suffix. RVROS [15] is a specification-agnostic monitoring framework for improving safety and security of robots using ROS. To the best of our knowledge, rtamt/rtamt4ros is the only tool that implements online robustness STL monitors with both future and past operators and ROS support.

2 RTAMT Design and Functionality

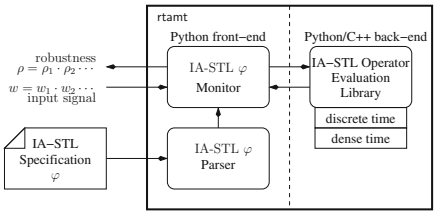

Fig. 1. RTAMT architecture.

The main functionality of rtamt is the automatic generation of online robustness monitors from declarative specifications. Given an input signal in the form of a sequence of (time, value) pairs and a specification, rtamt computes at different points in time how robust is the observed signal to the specification, i.e. how far is it from satisfying or violating it. The library consists of 3 major parts: (1) *specifications* expressed in a declarative specification language, (2) a *front-end* with an Application Programming Interface (API) to parse specifications and generate the monitor, and (3) a *back-end* that implements the actual evaluation algorithm used by the monitor. The rtamt library uses a modular architecture depicted in Fig. 1. It uses ANTLR4 parser generator to translate textual (IA-)STL specifications into an abstract parse tree (APT) data structure used to build the actual monitor. The front-end implements the Application Programming Interface (API) and the pre-processing steps such as the translation of bounded-future (IA-)STL to past (IA-)STL in Python. The back-end implements the monitoring algorithms in Python (for discrete-time and dense-time interpretation) and C++ (for discrete-time interpretation). The library is compatible with both Python 2.7 and 3.7.

Specification language in rtamt is STL with infinity-norm quantitative semantics [16]. The library supports four variants of the specification language – standard STL and interface-aware STL [3] interpreted over discrete and dense time.

IA-STL extends STL with an input/output signature of the variables and provides two additional semantic interpretations: (1) output robustness and (2) input vacuity. Output robustness measures robustness of output signals with respect to some fixed input. Input vacuity measures how vacuously is a specification satisfied with input signals only. rtamt accepts as input *bounded-future* STL (bfSTL) that restricts the use of the future temporal operators (eventually, always and until) to bounded intervals.

Parsing and preprocessing follows a two-step procedure. The first step consists in translating the specification given in a textual form to an abstract parse tree (APT). The translation uses ANTLR4 to generate a Python parser for the (IA-)STL grammar. This translation is still not suitable for online monitors – the specification may have future temporal operator that would require clair-voyant monitoring capability. Hence, we implement the *pastification* procedure [17] that translates the bfSTL formula ϕ into an *equi-satisfiable* past STL formula ψ, which uses only past temporal operators and postpones the formula evaluation from time index t, to the end of the (bounded) horizon $t + h$ where all the inputs necessary for computing the robustness degree are available.

Monitoring consists of evaluating in online fashion the past STL specification according to its quantitative semantics, interpreted in discrete or dense time [3].

Discrete-time monitors follow a time-triggered approach in which sensing of inputs and output generation are done at a periodic rate. This choice is motivated by [18], which shows that by weakening/strengthening real-time specifications, discrete-time evaluation of properties preserves important properties of dense-time interpretation. This approach admits an upper bound on the use of computation resources. rtamt implements two back-ends for STL monitors – one in Python (for rapid prototyping) and one in C++ (for efficiency). rtamt uses Boost.Python library to integrate the Python front-end with the C++ backend.

Dense-time monitors follow an event-driven approach. Their implementation combines the incremental evaluation approach from [19] with the optimal streaming algorithm to compute the min and max of a numeric sequence over a sliding window from [20]. Unlike their discrete-time counterparts, continuous-time monitors do not have bounds on memory requirements.

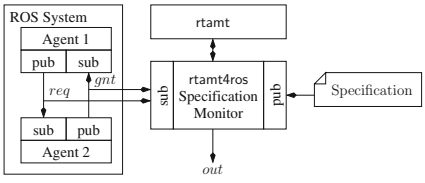

Fig. 2. Integration of RTAMT to ROS.

Integration of RTAMT to ROS ROS supports several messaging approaches, including the *subscriber* and *publisher* pattern. A *publisher* categorizes a message into a class (called *topic* in ROS) and sends it without knowing who will read the message. A *subscriber* subscribes to a topic and receives its associated messages, without knowing who sent the message[4]. The

[3] Due to pastification, rtamt only needs to evaluate past temporal operators.

[4] Unless the publisher encodes its identity into the message itself.

messages are received and processed in callback() functions. Common ROS applications associate a callback() function per subscribed variable.

rtamt4ros, illustrated in Fig. 2, integrates rtamt into ROS using *rospy*. The integration is non-intrusive and provides the user with a generic and transparent monitoring solution for (IA-)STL specifications. The ROS system under observation is implemented with ROS nodes, which interact by publishing and receiving ROS messages on dedicated topics. To publish values of a variable x of type T on a topic t, ROS node associates x and T to t. Similarly, we declare in the STL specification variables that we want to monitor, declare their types and associate them to ROS subscription/publication topics using annotations. Variable names, their types and associated topics are specification-dependent. rtamt4ros implements a *dynamic* subscription/publication mechanism that uses the concepts of *introspection* and *reflection* (the ability to passively infer the type of an object and actively change its properties at runtime). Given a (IA-)STL specification, rtamt4ros infers all the specification variables and dynamically creates their associated subscribers and publishers. The use of reflection allows us to associate a single callback() function to all specification variables, by passing the variable object as an argument to the function. We use the callback() function only to collect input data and the main ROS loop to make robustness monitor updates.

3 Experiments and Use Scenario

We now present experiments performed using rtamt and rtamt4ros. We apply rtamt and rtamt4ros on two ROS case studies: Simple Two Dimensional Simulator (STDR) and Toyota's Human Support Robot (HSR) platform [21]. We use the STDR example to show step-by-step usage of the rtamt and rtamt4ros for online monitoring of robotic applications. We note that rtamt is versatile and could be used for instance for offline monitoring and non-robotic applications. We then evaluate the computation time requirements of the library. The experiments were performed on a Dell Latitude 7490 with an i7-8650U processor and 16 GB of RAM, running Ubuntu 16.04 on a virtual machine.

Online Monitoring of Robotic Applications: STDR is a ROS-compliant environment for easy multi-robot 2D simulation (see Fig. 3). We use a simple robot controller with commands encoded as ROS Twist messages that expresses velocity in free space consisting of its linear and angular parts. The robot state is encoded as a ROS Odometry message that represents an estimate of the position (pose) and velocity (twist) in free space. We then use the rtamt4ros and rtamt to monitor its low-level requirement stating that every step in the command must be followed by the observed response. The specification `spec.stl` requires that at all times the distance between the linear velocity on the x dimension of the command and the robot is smaller than 0.5. The user first needs to import data types used in the specification (lines 1–3). Then, it declares variables used in specification, with their data type and (optionally) their input/output signature (lines 4, 6 and 8). Special comments in lines 5 and 7 are annotations that provide

additional information about variables - in this case they associate variables to ROS topics. Finally, line 9 defines the IA-STL property.

```
1 from geometry_msgs.msg import Twist
2 from nav_msgs.msg import Odometry
3 from rtamt_msgs.msg import FloatMessage
4 input Twist cmd
5 @ topic(cmd, robot0/cmd_vel)
6 output Odometry robot
7 @ topic(res, robot0/odom)
8 output FloatMessage out
9 out.value = always(abs(cmd.linear.x - robot.twist.twist.
    linear.x) <= 0.5)
```

To monitor the IA-STL specification `spec.stl` with rtamt/rtamt4ros , it suffices to run the following command in the ROS environment.

```
1 roscore rtamt4ros ros_stl_monitor.py --stl spec.stl --
    period 100 --unit ms
```

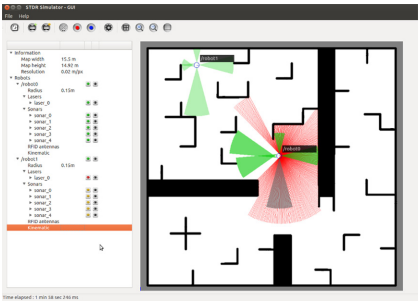

Fig. 3. STDR simulator.

Fig. 4. HSR service robotics application.

HSR is a robot with 8 degrees of freedom (DoF), combining 3 DoF of its mobile base, 4 DoF of the arm and 1 DoF of the torso lift (see Fig. 4). The robot is equipped with ROS modules for localization, path planning and obstacle avoidance. We used this example to experiment with *system-level properties* in a multi-agent environment. We were interested in particular in monitoring the following requirements: (1) *no-collision* requirement stating that two robots are never closer than some d_{min} distance from each other, and (2) when robot 2 is closer than d distance from robot 1, then robot 2 two goes in at most T seconds within d' distance of some location L. For this industrial application, we present an abstracted formalization of the above requirements.

```
1  out1 = always (abs(rob1.pos - rob2.pos) < d)
2  out2 = abs(rob1.pos - rob2.pos) < d implies
3           eventually[0:T](rob1.pos - L) < d'
```

This experiment demonstrates the use of the library in a sophisticated ROS/Gazebo environment in an industrial case study. The addition of monitors is orthogonal to the development of the application and the monitors are non-intrusive.

Table 1. Timing requirement per single monitor update.

k bound	C++ (s)	Python (s)
100	0.00014	0.00023
1k	0.0002	0.00085
10k	0.0008	0.029
100k	0.0047	0.31
1M	0.046	72

Timing Figures: For online monitors, the most important quantitative measure is the computation time of a single monitoring update step. We compared the difference in timing requirements between the C++ and the Python implementation of the discrete-time monitoring algorithm. We used for the experiment the STL specification `out = always[0:k] (a + b > -2)` where k is the upper bound on the timing modality of the always operator that we varied between 100 and 1 million. Table 1 summarized the results of the experiment. The outcomes clearly demonstrate the efficiency of the C++ back-end, especially for large upper bounds in temporal modalities.

4 Conclusions

In this paper, we presented rtamt a library for generating online monitors from declarative specifications and rtamt4ros, its ROS extension, demonstrating their usability and versatility two robotic case studies.

Acknowledgements. This work was partially supported by iDev40 project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783163. The JU receives support from the European Union's Horizon 2020 research and innovation programme. It is co-funded by the consortium members, grants from Austria, Germany, Belgium, Italy, Spain and Romania.

References

1. Quigley, M., et al.: ROS: an open-source robot operating system. In: ICRA workshop on open source software, vol. 3, p. 5. Kobe, Japan (2009)
2. 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
3. Ferrère, T., Nickovic, D., Donzé, A., Ito, H., Kapinski, J.: Interface-aware signal temporal logic. In: HSCC, pp. 57–66 (2019)
4. Nickovic, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. In: TACAS 2018, pp. 303–319 (2018)

5. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
6. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
7. Ulus, D.: Online monitoring of metric temporal logic using sequential networks. CoRR, abs/1901.00175 (2019)
8. Vazquez-Chanlatte, M.: mvcisback/py-metric-temporal-logic: v0.1.1 (2019)
9. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime analysis with R2U2: a tool exhibition report. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 504–509. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_35
10. Ulus, D.: Montre: a tool for monitoring timed regular expressions. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 329–335. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_16
11. Waga, M., Hasuo, I., Suenaga, K.: MONAA: A tool for timed pattern matching with automata-based acceleration. In: 3rd Workshop on Monitoring and Testing of Cyber-Physical Systems, MT-CPSWeek 2018, 10 April 2018, Porto, Portugal, pp. 14–15 (2018)
12. Faymonville, P.: StreamLAB: stream-based monitoring of cyber-physical systems. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 421–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_24
13. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: Tessla: runtime verification of non-synchronized real-time streams. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, 09–13 April 2018, pp. 1925–1933 (2018)
14. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Form. Methods Syst. Des. **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7
15. Huang, J.: ROSRV: runtime verification for robots. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 247–254. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_20
16. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
17. Maler, O., Nickovic, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 95–107. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_12
18. Henzinger, T.A., Manna, Z., Pnueli, A.: What good are digital clocks? In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 545–558. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55719-9_103

19. Nickovic, D., Maler, O.: AMT: a property-based monitoring tool for analog systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75454-1_22
20. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_19
21. Yamamoto, T., Terada, K., Ochiai, A., Saito, F., Asahara, Y., Murase, K.: Development of human support robot as the research platform of a domestic mobile manipulator. ROBOMECH J. **6**(1), 1–15 (2019). https://doi.org/10.1186/s40648-019-0132-3