RLRom: Monitoring and Training Reinforcement Learning Agents using Signal Temporal Logic

Ana María Gómez-Ruiz, Alexandre Donzé and Thao Dang

VERIMAG, Bâtiment IMAG, Université Grenoble Alpes, 150 Pl du Torrent, 38401 Grenoble

Abstract

Reinforcement learning (RL) excels at solving complex tasks, but its application presents challenges in interpretability, safety and reliability. This is in part due to the dynamic nature of the RL problems, for which timing interactions are crucial. RLRom is a novel tool that integrates Signal Temporal Logic (STL) with common RL frameworks. STL, a formal language for time-dependent properties, enables RLRom to monitor agent behavior against specified requirements, identifying both intended and unexpected behaviors. STL properties can be robustly monitored online, generating quantitative satisfaction values that can serve as new observations or reward components during training. This improves the design, interpretability and development of RL problems of safe, reliable and efficient agents. We demonstrate RLRom's utility using the highway-env environment, showing its ability to influence driving agent behavior, such as speed, safety and rule adherence.

Keywords

Reinforcement Learning, Signal Temporal Logic, Monitoring, Formal Methods

1. Introduction

Reinforcement learning (RL) [1] is a powerful and versatile unsupervised learning method used to teach artificial systems how to solve complex tasks. The advent of deep learning and the development of mature RL algorithms have made it applicable to a variety of situations, from advanced robotics to fine-tuning large language models. However, applying RL to a given problem remains a challenging endeavor. This is because, as with most modern artificial intelligence (AI) tasks, difficult issues of interpretability and uncertainty of the results arise during and after the learning process. Moreover, RL is concerned with dynamical or sequential problems, where timing between events is important, making it even more difficult to design and debug solutions. As a consequence, several authors have looked into temporal logics to reason formally about the characteristics of traces, i.e., sequences of observations of episodes realised by trained agent.

One such logic is Signal Temporal Logic (STL) introduced in [2] to specify properties of signals with continuous times and values. STL syntax is minimalist, yet quite expressive and intuitive. Atomic predicates are inequalities over quantities of interest (for example the speed is greater than 50, height less than 5, etc) and formulas are constructed using standard Boolean operators and temporal operator, e.g., $\Diamond_{[0,30]}\ p_1$ is "eventually in 30 seconds, p1 is true" or "if p1 is true then p2 is always true from 5 seconds to 10 seconds" can be written as $p_1 \Rightarrow \Box_{[5,10]}\ p_2$. One crucial advantage of STL is that it admits a *quantitative* semantics [3] which can be very efficiently evaluated (monitored) even on long traces [4] and online [5], i.e., when a trace is still under computation. Sometimes called "robustness", such semantics estimate the degree of satisfaction of a formula by a trace not only as true or false, but as a number which amplitude represents how "robust" or fragile the true or false evaluation is. This number can then be used for various purposes. For instance by trying to minimize it, one can find a trace that violates a given requirement, potentially exhibiting a bug [6]. Conversely, by maximizing it, one can find a controller or policy that robustly satisfy a design requirement [7, 8]. The idea to use RL to train a policy satisfying STL requirements was then naturally explored in the past decade ([9, 10, 11, 12])

7th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (OVERLAY 2025), October 26, 2025, Bologna, Italy

ana.gomez@univ-grenoble-alpes.fr (A. M. Gómez-Ruiz); alexandre.donze@univ-grenoble-alpes.fr (A. Donzé); thao.dang@univ-grenoble-alpes.fr (T. Dang)

though few tools have been implemented.

In most of the works around RL and STL, the goal is that of *controller synthesis*, i.e., the problem assumes that a set of STL requirements is given and the goal is to train the best policy satisfying them. However, even if STL is a relatively simple language, specifying the right goals can be difficult and such approach can be prone to the reward hacking problem, wherein the trained agent satisfy the requirements but by deploying unexpected and often undesirable strategies.

Our view is that in practice, STL requirements (and requirements in general, not necessarily expressed in STL) are part of the design problem as much as the system or its controller itself. Hence, one should be able to manipulate them in a flexible way, within an iterative process converging towards a satisfactory design with a clearly and formally defined set of requirements.

For these reasons, we implemented RLRom [1], which integrates a rich STL parser with efficient online monitoring methods (taken from [4, 5]) with popular RL frameworks. Our tool is similar to stl-gym[2] based on [13], which allows to replace the reward function with the robustness of an STL formula. RLROM goes beyond by allowing richer formulas to be written and monitored, as well as to modify and augment not only the reward functions but also the observation vector.

2. Tool Overview

2.1. Implementation

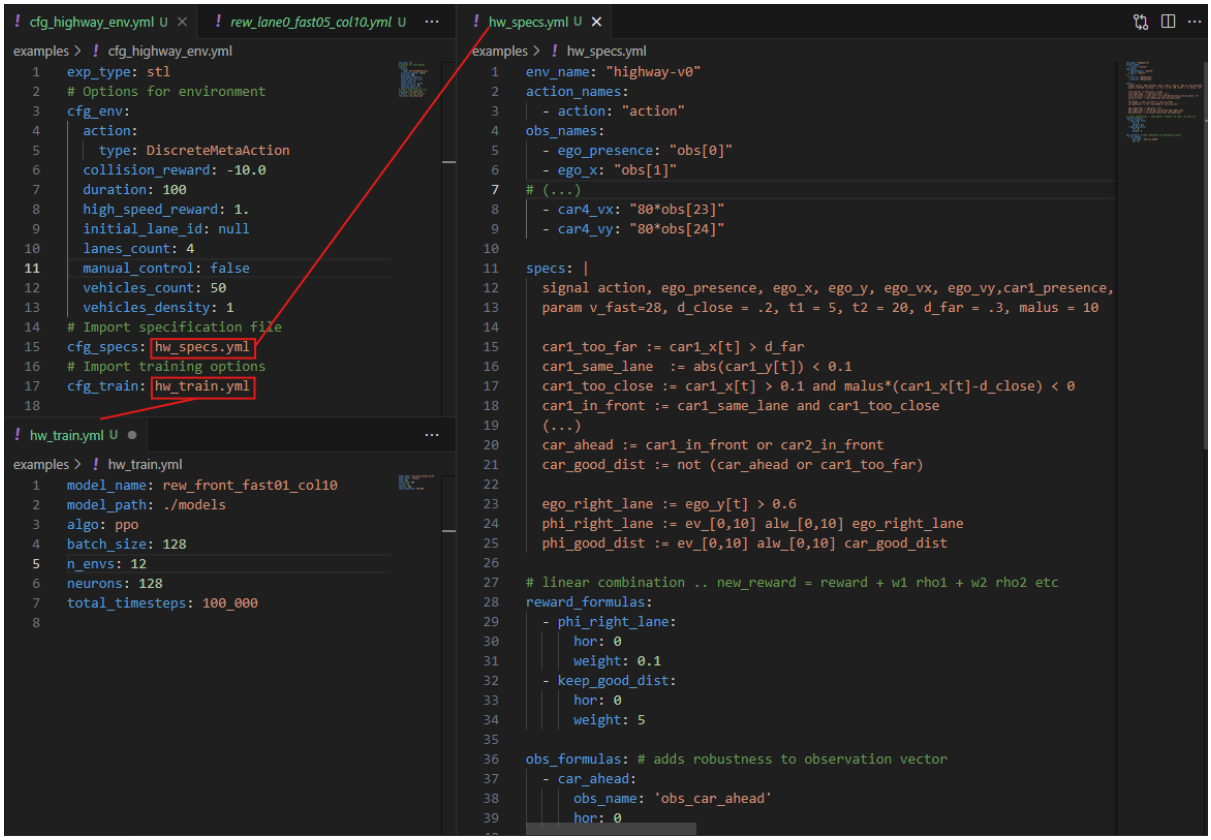

Figure 1: Configuration files for the highway-env environment. At the top level (`cfg_highway_env.yml`, there are three main sections for environment options, specifications and training options. These sections can be included directly, or imported from other files. When training occurs, the configuration file is flattened into a single file and copied along with the trained model so that the training can be reproduced more easily.

RLRom is built over three main existing components, namely:

[1] https://github.com/decyphir/rlrom
[2] https://github.com/nphamilton/stl-gym

- **Gymnasium** [14]: A Python library offering a standardized API for developing and comparing reinforcement learning algorithms by providing a variety of pre-built and customizable environments. It serves as a widely used platform for RL research and development;
- **Stable-Baselines3** [15]: A Python library that provides well-documented implementations of model-free reinforcement learning algorithms (such as PPO, DQN, SAC, etc) in PyTorch. It offers a consistent interface for training and evaluating RL agents.
- **STLRom**[3]:A C++ library with Python bindings for robust online monitoring of Signal Temporal Logic (STL). It enables users to define temporal properties of signals and compute their robustness value online.

The main implementation idea is based on the wrapper mechanism of Gymnasium, which takes an existing RL environment with a reset(), step() and reward() methods and observation and action spaces, and returns a new environment that changes one or all of these components. In RLRom, we implemented an stl_wrapper that does the following:

1. It adds an stl_driver object from STLRom which can read data and monitors STL formulas;
2. Within the step() function, a new observation of the wrapped environment is added to the stl_driver and the satisfaction of the formulas is updated; the resulting robustness values are then used to augment the observation and/or influence the reward.

The resulting augmented environment can then be used for extended evaluation of a trained agent (monitoring) or for training. RLRom is designed to be modular and as interpretable as possible. Modularity is achieved by separating the configurations of environment, training algorithm and specifications. This is illustrated in Figure 1. Monitoring and visualization are the basics for interpretability. RLRom implements very flexible monitoring methods, inherited from STLRom, and plotting, as illustrated in Figure 2.1.

2.2. Workflows and features

When working with a new RL problem, assuming we are given a RL environment implementing a Gymnasium-compatible API, RLRom is intended to support the following steps, with workflows of increasing complexities.

1. **(WF1) Testing**: RLRom provides a simple way to run multiple episodes varying initial seed. Models can be imported from files or from popular repos, such as Hugging Face [4] Eventually, we plan on implementing more advanced simulation-based testing methods, in the spirit of [16].
2. **(WF2) Training**: new models can be trained with RLRom interfacing to existing stable-baselines3 algorithms - existing hyper-parameters can be quickly fetched and tried to obtain some baseline model for the base problem;
3. **(WF3) Monitoring**: Once a baseline model has been obtained, STL specifications can be written and monitored. This requires deciding which feature of the observation space should be monitored as a signal, and what properties should be checked. A typical first step is to formalize the initial goal of the environment, and check that the baseline model satisfy this formalization. E.g., for the classic cart pole environment, the goal is to keep the pole up, which can be translated into the simple predicate `pole_up := abs(theta[t]) < epsi` for some small value of `epsi`. The goal specification can then be:

   ```
   phi_goal_cartpole := ev_[0,400] alw_[0,100] mu
   ```

   that is, before 400 steps, the pole should stay up for at least 100 steps.
4. **(WF4) Training with specifications**: Depending on the intentions and potential desirable or undesirable behaviors discovered in (WF3), STL specifications can be incorporated into the model to solve a problem with more complex objectives.

[3] https://github.com/decyphir/stlrom
[4] https://huggingface.com

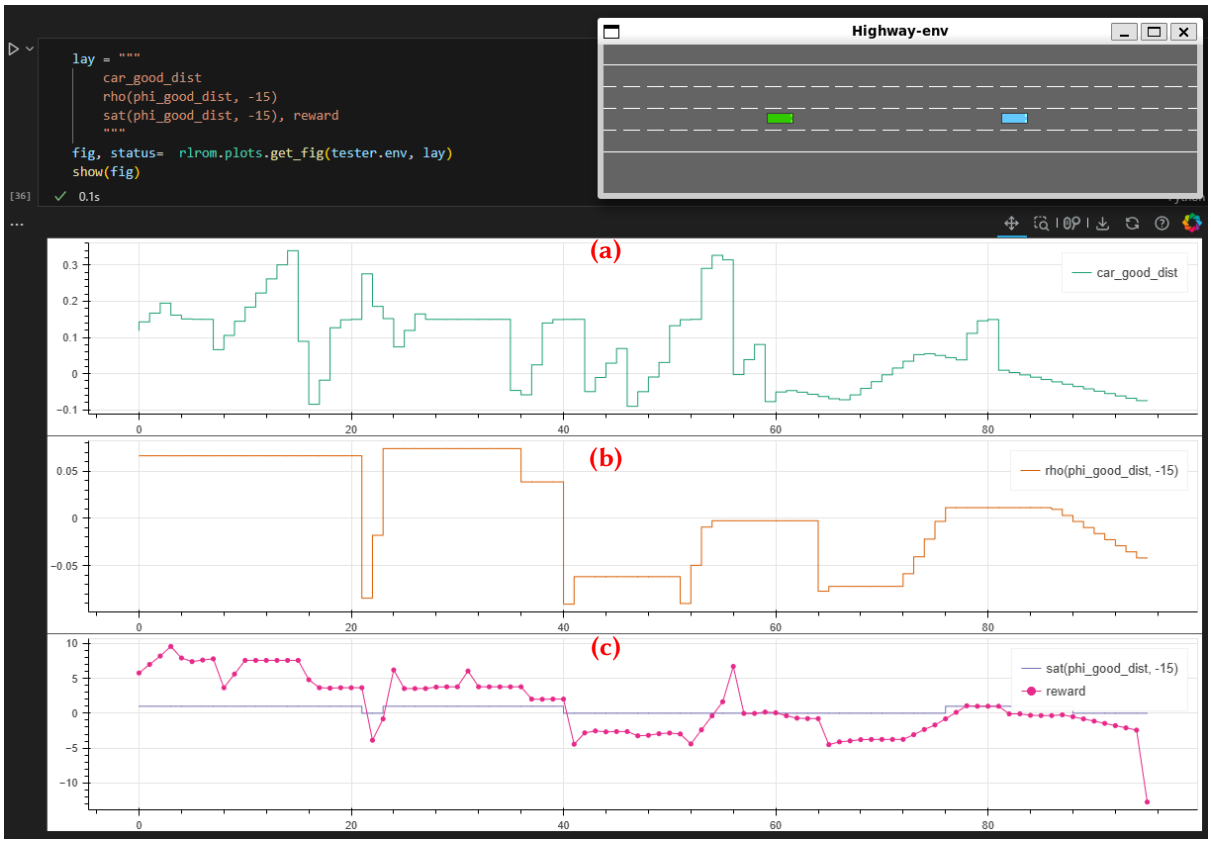

Figure 2: Monitoring example. We run an episode using the configuration file of Figure 1, then produced a figure from the resulting trace of the episode. On the top axis (a), we plot the evaluation of the `car_good_dist` formula, which has no temporal operator, at every time step. On the middle axis (b), we plot the robustness of `phi_good_dist` evaluated between t and $t - 15$. We can see that, e.g., it takes negative values after step 20 because `car_good_dist` was negative (false) during several steps betwen steps 10 and 20. On the bottom axis (c), we plot the Boolean satisfaction of `phi_good_dist` and the reward. We can observe that the reward is indeed low when `phi_good_dist` is not true.

(WF1) and (WF2) are standard workflows in the RL community, implemented for example in the RL-Zoo framework [5]. (WF3) and (WF4) are more unique to RLRom. Although various integrations of STL specifications into RL problems have been proposed before, our intention is not to propose an elaborate solution to one well posed difficult problem, such as finding the optimal agent satisfying some complex STL formula, but rather to propose flexible ways to incorporate and exploit STL monitoring in a RL framework, providing a base for many different implementations of elaborate solutions to well posed problems.

To illustrate this, our framework already allows the following transformations:

- Augment the observation space with the robustness of arbitrary formulas, say

$$\texttt{new\_obs} := [\texttt{old\_obs}, \texttt{obs\_rob1}, \texttt{obs\_rob2}, \ldots]$$

- Add the robustness of arbitrary formulas to the reward:

$$\texttt{new\_reward} := \texttt{old\_reward} + w1\ \texttt{rob1} + w2\ \texttt{rob2} + \ldots$$

We believe that this can already allow a large panel of applications, while remaining interpretable transformations. However we are still exploring their theorical and practical implications in a cautious way. For instance, since the satisfaction of STL formula obviously depends on history, the Markov

[5] https://rlzoo.readthedocs.io

assumption can easily be violated by these transformations. Hence, in our early experiments, we started with zero or short horizon formulas to include in observation and/or rewards. We found that in order to satisfy a global specification, a simple approach is to identify local sub-formulas or predicates that affects monotonically the global specifications, and add them in the reward and/or the observation accordingly. We believe RLRom will make it simpler to validate such intuitions and derive from them more systematic approaches.

3. Conclusion and future work

RLRom goal is to provide an experimental platform for first testing RL problems against STL requirements, then augment these problems with adjusted, formally defined goals. Basic functionalities such as writing STL formulas, monitoring and training with augmented rewards and observations have already been implemented in a way that provides great flexibility and potential of exploration. This already lead to promising results in terms of training agents in a controlled and monitored way to satisfy certain behaviors. Future work include consolidating the existing features and apply to more RL problems, as well as exploring the integration and implementation of formal frameworks, such as in particular, reward machines, not as an alternative but as a complement to STL monitoring.

4. Acknowledgments

This work is partially supported by the joint French-Japanese ANR-JST project CyPhAI, the Auvergne-Rhône-Alpes Region Project DetAI, and by a French state grant managed by the National Research Agency as part of France 2030, with the reference ANR-23-IACL-0006.

Declaration on Generative AI

The authors certify that no Generative AI tool was used during the production of this work.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, second ed., The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[2] O. Maler, D. Nickovic, Monitoring Temporal Properties of Continuous Signals, in: Y. Lakhnech, S. Yovine (Eds.), Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Springer, Berlin, Heidelberg, 2004, pp. 152–166. doi:10.1007/978-3-540-30206-3_12.

[3] A. Donzé, O. Maler, Robust Satisfaction of Temporal Logic over Real-Valued Signals, in: K. Chatterjee, T. A. Henzinger (Eds.), Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2010, pp. 92–106. doi:10.1007/978-3-642-15297-9_9.

[4] A. Donzé, T. Ferrère, O. Maler, Efficient Robust Monitoring for STL, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, volume 8044 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 264–279. URL: https://doi.org/10.1007/978-3-642-39799-8_19. doi:10.1007/978-3-642-39799-8_19.

[5] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, S. A. Seshia, Robust Online Monitoring of Signal Temporal Logic, in: E. Bartocci, R. Majumdar (Eds.), Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings, volume 9333 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 55–70. URL: https://doi.org/10.1007/978-3-319-23820-3_4. doi:10.1007/978-3-319-23820-3_4.

[6] T. Dreossi, A. Donzé, S. A. Seshia, Compositional Falsification of Cyber-Physical Systems with Machine Learning Components, in: C. W. Barrett, M. Davies, T. Kahsai (Eds.), NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings, volume 10227 of *Lecture Notes in Computer Science*, 2017, pp. 357–372. URL: https://doi.org/10.1007/978-3-319-57288-8_26. doi:10.1007/978-3-319-57288-8_26.

[7] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, S. A. Seshia, Model predictive control with signal temporal logic specifications, in: 53rd IEEE Conference on Decision and Control, 2014, pp. 81–87. doi:10.1109/CDC.2014.7039363, iSSN: 0191-2216.

[8] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, S. A. Seshia, Reactive synthesis from signal temporal logic specifications, in: A. Girard, S. Sankaranarayanan (Eds.), Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015, ACM, 2015, pp. 239–248. doi:10.1145/2728606.2728628.

[9] X. Li, C.-I. Vasile, C. Belta, Reinforcement Learning With Temporal Logic Rewards, 2017. URL: http://arxiv.org/abs/1612.03471. doi:10.48550/arXiv.1612.03471, arXiv:1612.03471 [cs].

[10] H. Venkataraman, D. Aksaray, P. Seiler, Tractable Reinforcement Learning of Signal Temporal Logic Objectives, in: Proceedings of the 2nd Conference on Learning for Dynamics and Control, PMLR, 2020, pp. 308–317. URL: https://proceedings.mlr.press/v120/venkataraman20a.html, iSSN: 2640-3498.

[11] L. Berducci, E. A. Aguilar, D. Ničković, R. Grosu, Hierarchical Potential-based Reward Shaping from Task Specifications, 2022. URL: http://arxiv.org/abs/2110.02792. doi:10.48550/arXiv.2110.02792, arXiv:2110.02792 [cs].

[12] P. Kapoor, K. Mizuta, E. Kang, K. Leung, Stlcg++: A masking approach for differentiable signal temporal logic specification, 2025. URL: https://arxiv.org/abs/2501.04194. arXiv:2501.04194.

[13] D. Nickovic, T. Yamaguchi, RTAMT: Online Robustness Monitors from STL, arXiv:2005.11827 [cs] (2020). URL: http://arxiv.org/abs/2005.11827, arXiv: 2005.11827.

[14] J. Towers, A. Sowinski, L. Happ, A. Gleave, M. Pinto, E. Grefenstette, Gymnasium: An API for Reinforcement Learning Research, Journal of Machine Learning Research 25 (2024) 1–11. URL: http://jmlr.org/papers/v25/24-0010.html.

[15] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Park, N. Piot, M. Ernestus, Q. Dormann, N. Plouffe, Stable Baselines3: Reliable Reinforcement Learning Implementations, in: Journal of Machine Learning Research, 2024, pp. 25 (284): 1–11. URL: https://stable-baselines3.readthedocs.io/.

[16] A. Donzé, Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems, in: T. Touili, B. Cook, P. B. Jackson (Eds.), Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings, volume 6174 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 167–170. URL: https://doi.org/10.1007/978-3-642-14295-6_17. doi:10.1007/978-3-642-14295-6_17.