











# Scarlet: Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic

Ritam Raha <sup>1,2\*</sup>, Rajarshi Roy <sup>3\*</sup>, Nathanaël Fijalkow <sup>2</sup>, and Daniel Neider <sup>4,5</sup>

1 University of Antwerp, Antwerp, Belgium 2 CNRS, LaBRI and Université de Bordeaux, France 3 Max Planck Institute for Software Systems, Kaiserslautern, Germany 4 TU Dortmund University, Dortmund, Germany 5 Center for Trustworthy Data Science and Security, University Alliance Ruhr, Germany   
Corresponding author \* These authors contributed equally.

DOI: [10.21105/joss.05052](https://doi.org/10.21105/joss.05052)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Adi Singh](#) 

## Reviewers:

- [@JDRomano2](#)
- [@Smattr](#)

Submitted: 10 August 2022

Published: 08 January 2024

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

In the past decade, incorporating data-driven AI techniques in system design has become mainstream in almost all branches of science and technology. Typically, systems powered by AI tend to be rather complex, far beyond human understanding. Naturally, one cannot always develop trust in such complex, so-called black-box systems, restricting their widespread use in safety-critical domains.

To generate trust in systems, a standard approach in Explainable AI is to build simple explanations using human-understandable models. A number of recent works ([Neider & Gavran, 2018](#)) ([Camacho & McIlraith, 2019](#)) ([Roy et al., 2020](#)) have identified models in temporal logic to be both formal and explainable. Among temporal logics—Linear Temporal Logic (LTL)—arguably the most widely used temporal logic, has received particular focus due to its resemblance to natural language. Moreover, LTL is a de-facto standard in several fields of computer science, including model-checking, program analysis, and motion planning for robotics.

Scarlet is one of the most competitive tools for learning explainable models in LTL ([Raha et al., 2022](#)). To learn such models, it relies on positive (or desirable) and negative (or undesirable) executions of the system under consideration. Based on the executions, it learns a concise model in LTL that is consistent with the given executions.

Let us consider a concrete example to understand Scarlet's functioning. Consider a robot that has been designed to collect wastebin contents in an office-like environment. For the sake of the example, in this environment, let there be an office  $o$  with a wastebin, a hallway  $h$ , and a container  $c$  to accumulate the waste. The following could be the possible executions of the robot:

$h, h, h, h, o, h, c, h$   
 $h, h, h, h, h, c, h, o, h, h$

Let the first execution be positive since the robot first collects waste from the office and then accumulates in the container. Further, let the second execution be negative since the robot tries to accumulate the waste in the container even before it collects from the office. From these executions, Scarlet learns a model  $F(o \text{ and } FX\ c)$ , where the  $F$ -operator stands for “finally” and  $X$ -operator stands for “next”. This model, in simple words, expresses that: eventually, the robot should visit the office  $o$  and then, at a later point, should visit the container  $c$ .

## Statement of need

The fundamental problem solved by Scarlet is to build an explainable model in the form of an LTL formula from system executions, formally termed as traces.

Scarlet is a tool built entirely in Python 3. It can be run using its command-line features or its Python API hosted in PyPi. Its main capabilities include:

- construction of an LTL formula from execution traces,
- generation of execution traces from an LTL formula for testing LTL learning algorithms (using automata-based techniques, such as LTLf2DFA and MONA, and random sampling).

Scarlet additionally supports noisy data: the user can specify a noise threshold (between zero and one, zero for perfect classification) and the algorithm returns an almost separating formula with respect to that threshold.

## Key insights

A paper presenting the algorithms behind Scarlet was published in TACAS'2022: Tools and Algorithms for the Construction and Analysis of Systems ([Raha et al., 2022](#)).

We believe that the path to scalability for learning models in LTL is to leverage normal forms for LTL formulas and derive efficient enumeration algorithms from them. Scarlet combines two insights:

- An efficient enumeration algorithm for directed LTL formulas, which are formulas that can be evaluated only moving forward in traces,
- An algorithm solving the Boolean set cover problem, which constructs Boolean combinations of already constructed formulas in order to separate positive and negative traces.

For experimental results, refer to the full paper ([Raha et al., 2022](#)).

## Related works

For learning models in LTL, several approaches have been proposed, leveraging SAT solvers ([Neider & Gavran, 2018](#)), automata ([Camacho & McIlraith, 2019](#)), and Bayesian inference ([Kim et al., 2019](#)). In fact, there are approaches for many temporal logics such as Property Specification Language (PSL) ([Roy et al., 2020](#)), Computational Tree Logic (CTL) ([Ehlers et al., 2020](#)) ([Roy & Neider, 2023](#)), Metric Temporal Logic (MTL) ([Raha et al., 2023](#)), etc.

Applications of LTL learning include program specification ([Lemieux et al., 2015](#)), anomaly and fault detection ([Bombara et al., 2016](#)), robotics ([Chou et al., 2020](#)), and many more: we refer to ([Camacho & McIlraith, 2019](#)) for a list of practical applications. An equivalent point of view on LTL learning is as a specification mining question. The ARSENAL ([Ghosh et al., 2016](#)) and FRET ([Giannakopoulou et al., 2020](#)) projects construct LTL specifications from natural language; see ([Li, 2013](#)) for an overview.

The two state-of-the-art tools for learning logic formulas from examples are:

- FLIE ([Neider & Gavran, 2018](#)) infers minimal LTL formulas using a learning algorithm that is based on constraint solving (SAT solving).
- SYSLITE ([Arif et al., 2020](#)) infers minimal past-time LTL formulas using an enumerative algorithm implemented in a tool called CVC4SY ([Reynolds et al., 2019](#)).

Existing methods do not scale beyond formulas of small size, making them hard to deploy for industrial cases. A second serious limitation is that they often exhaust computational resources

without returning any results. Indeed theoretical studies (Fijalkow & Lagarde, 2021) have shown that constructing the minimal LTL formula is NP-hard already for very small fragments of LTL, explaining the difficulties found in practice.

## Acknowledgments

This project was funded by the FWO G030020N project SAILor and Deutsche Forschungsgemeinschaft (DFG) grant no. 434592664.

## References

- Arif, M. F., Larraz, D., Echeverria, M., Reynolds, A., Chowdhury, O., & Tinelli, C. (2020). SYS-LITE: Syntax-guided synthesis of PLTL formulas from finite traces. *Formal Methods in Computer Aided Design, FMCAD*. [https://doi.org/10.34727/2020/ISBN.978-3-85448-042-6\\_16](https://doi.org/10.34727/2020/ISBN.978-3-85448-042-6_16)
- Bombara, G., Vasile, C. I., Penedo Alvarez, F., Yasuoka, H., & Belta, C. (2016). *A Decision Tree Approach to Data Classification using Signal Temporal Logic*. <https://doi.org/10.1145/2883817.2883843>
- Camacho, A., & McIlraith, S. A. (2019). Learning interpretable models expressed in linear temporal logic. *International Conference on Automated Planning and Scheduling, ICAPS*. <https://doi.org/10.1609/icaps.v29i1.3529>
- Chou, G., Ozay, N., & Berenson, D. (2020). Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations. *Robotics: Science and Systems*. <https://doi.org/10.15607/RSS.2020.XVI.097>
- Ehlers, R., Gavran, I., & Neider, D. (2020). Learning properties in  $LTL \cap ACTL$  from positive examples only. *Formal Methods in Computer Aided Design, FMCAD*, 104–112. [https://doi.org/10.34727/2020/isbn.978-3-85448-042-6\\_17](https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_17)
- Fijalkow, N., & Lagarde, G. (2021). The complexity of learning linear temporal formulas from examples. *International Conference on Grammatical Inference, ICGI*. <https://proceedings.mlr.press/v153/fijalkow21a.html>
- Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., & Steiner, W. (2016). ARSENAL: Automatic requirements specification extraction from natural language. *NASA Formal Methods, NFM*. [https://doi.org/10.1007/978-3-319-40648-0\\_4](https://doi.org/10.1007/978-3-319-40648-0_4)
- Giannakopoulou, D., Pressburger, T., Mavridou, A., Rhein, J., Schumann, J., & Shi, N. (2020). Formal requirements elicitation with FRET. *International Conference on Requirements Engineering: Foundation for Software Quality, REFSQ*. <http://ceur-ws.org/Vol-2584/PT-paper4.pdf>
- Kim, J., Muise, C., Shah, A., Agarwal, S., & Shah, J. (2019). Bayesian inference of linear temporal logic specifications for contrastive explanations. *International Joint Conference on Artificial Intelligence, IJCAI*. <https://doi.org/10.24963/ijcai.2019/776>
- Lemieux, C., Park, D., & Beschastnikh, I. (2015). General LTL specification mining. *International Conference on Automated Software Engineering, ASE*. <https://doi.org/10.1109/ASE.2015.71>
- Li, W. (2013). *Specification mining: New formalisms, algorithms and applications* [PhD thesis, University of California, Berkeley, USA]. <http://www.escholarship.org/uc/item/4027r49r>
- Neider, D., & Gavran, I. (2018). Learning linear temporal properties. *Formal Methods in Computer Aided Design, FMCAD*. <https://doi.org/10.23919/FMCAD.2018.8603016>

- Raha, R., Roy, R., Fijalkow, N., & Neider, D. (2022). Scalable anytime algorithms for learning fragments of linear temporal logic. *TACAS (1)*, 13243, 263–280. [https://doi.org/10.1007/978-3-030-99524-9\\_14](https://doi.org/10.1007/978-3-030-99524-9_14)
- Raha, R., Roy, R., Fijalkow, N., Neider, D., & Pérez, G. A. (2023). Synthesizing efficiently monitorable formulas in metric temporal logic. *CoRR*, *abs/2310.17410*. <https://doi.org/10.48550/ARXIV.2310.17410>
- Reynolds, A., Barbosa, H., Nötzli, A., Barrett, C. W., & Tinelli, C. (2019). cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. *Computer-Aided Verification, CAV*. [https://doi.org/10.1007/978-3-030-25543-5\\_5](https://doi.org/10.1007/978-3-030-25543-5_5)
- Roy, R., Fisman, D., & Neider, D. (2020). Learning interpretable models in the property specification language. *International Joint Conference on Artificial Intelligence, IJCAI*, 2213–2219. <https://doi.org/10.24963/ijcai.2020/306>
- Roy, R., & Neider, D. (2023). Inferring properties in computation tree logic. *CoRR*, *abs/2310.13778*. <https://doi.org/10.48550/ARXIV.2310.13778>