Temporal Logic Satisfiability From Specification Debugging to Benchmark Generation*

Kristin Yvonne Rozier[1]

Iowa State University of Science and Technology, Ames, Iowa, U.S.A.
kyrozier@iastate.edu

1 Extended Abstract

Developing efficient tools and techniques for propositional satisfiability (SAT) solving has long been recognized as impactful pursuit. Advances in SAT directly translate to model checking, planning and scheduling, automatic test-case generation, combinatorial equivalence checking, graph coloring, software verification, and more [3]. Annual SAT competitions rate a plethora of propositional satisfiability tools in a number of different categories [1]; the extension of SAT to include theory solvers that translate the results to a wide variety of additional mathematical domains, Satisfiability Modulo Theories (SMT) enjoys its own annual competition series.[2] SAT has its own long-standing conference series;[3] so does SMT.[4]

Temporal logic satisfiability checking, for example for Linear Temporal Logic (LTL) and its many derivatives such as Mission-time LTL (MLTL) [7, 5] has received considerably less attention. In 2007, we noted that, while there were no devoted LTL satisfiability solvers, we can use LTL model checking against a universal model as a proxy for LTL satisfiability solving [9]. We surveyed all publicly-available translators of LTL to either explicit or symbolic automata that could serve as inputs to LTL model checker and discovered (1) that LTL satisfiability-as-explicit-model-checking does not scale and is highly error prone (with not a single solver generating 100% correct results); and (2) that LTL satisfiability-as-symbolic-model-checking was less error prone and more scalable but still limited [10]. We devised scalable LTL satisfiability benchmarks including encoding binary counters as LTL formulas (e.g., uniquely satisfiable formulas satisfied exactly by an n-bit binary counter with overflow); LTL symbolic satisfiability checking could not scale to handle 12-bit binary counter formulas [10]! From 1997[1] to 2011[11] there was only *one* encoding for LTL to symbolic automata, which then enabled symbolic LTL satisfiability checking. Through defining and benchmarking 29 additional encodings of LTL-to-symbolic-automata, we unleashed performance improvements that were up to exponentially better for some classes of formulas [11].

The historic lack of attention to temporal logic satisfiability is surprising because it is impactful in a wide variety of domains, including those impacted by SAT and SMT. For example, model checking benefits from better encodings of LTL formulas, e.g., derived from satisfiability solving [12]. Planning and scheduling problems are often best-specified by logics like the popular LTLf (LTL over finite traces)[2]; improving satisfiability for that logic has impacted the AI planning domain [6]. Perhaps one of the most wide-reaching applications of temporal logic satisfiability is *specification debugging* [9, 10]. Checking the satisfiability of each specification, its negation, and the conjunction of all specifications for a given system ensures that no requirement is accidentally unsatisfiable or valid, and that all requirements can be true of the same (reactive) system at the same time. Finding specification bugs early in the system design lifecycle, before systems are built off of faulty specifications, can have financial impacts in the millions of dollars.

*Supported by NASA ECF NNX16AR57G and NSF CAREER Award CNS-1552934.

[1] http://www.satcompetition.org/
[2] https://smt-comp.github.io/2020/
[3] http://www.satisfiability.org/
[4] http://smt-workshop.cs.uiowa.edu/2020/index.shtml

The runtime verification (RV) problem asks whether the current system run, π, upholds its specification, φ. Stream-based RV asks for all i during the mission, whether $\pi, i \models \varphi$: does the trace starting from time i satisfy φ. RV algorithms would be improved by faster, more efficient temporal logic satisfiability solvers. Furthermore, such solvers are desperately needed for verification of RV engines, for example, via generating RV benchmarks consisting of π, φ, and an "oracle" verdict stream indicating, for all i, $\pi, i \models \varphi$ [8]. Such a benchmark could be constructed, for example, using a satisfiability solver for a popular RV logic like MLTL [5], and a formula progression algorithm [4].

In current and future work, we look to improve algorithms for satisfiability of LTL and its related logics and encourage competitions centered around this goal, e.g., through further benchmark generation. We look to fuel the fledgling Runtime Verification competition[5] through better benchmark generation and solving tools. In RV in particular, past-time variants of linear temporal logics have had historical appeal: unlike with future-time logics, there is always a verdict (true or false) at the current time and it is only a question of how fast we can find it. Yet there is currently no tool devoted to past-time LTL (or MLTL) satisfiability solving; we look to fill that gap.

References

[1] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design 10*, 10(1):47–71, 1997.

[2] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[3] Doutora Carla Pedro Gomes. *Propositional Satisfiability: Techniques, Algorithms and Applications*. PhD thesis, Universidade Técnica de Lisboa, 2004.

[4] Jianwen Li and Kristin Y Rozier. Mltl benchmark generation via formula progression. In *International Conference on Runtime Verification*, pages 426–433. Springer, 2018.

[5] Jianwen Li, Moshe Y. Vardi, and Kristin Y. Rozier. Satisfiability checking for Mission-time LTL. In *Proceedings of 31st International Conference on Computer Aided Verification (CAV)*, volume 11562 of *LNCS*, pages 3–22, New York, NY, USA, July 2019. Springer.

[6] Jianwen Li, Yueling Zhang, Geguang Pu, Kristin Yvonne Rozier, and Moshe Vardi. SAT-based Explicit LTLf Satisfiability Checking. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, volume 33, pages 2946–2953, Honolulu, Hawaii, February 2019. AAAI Press.

[7] Thomas Reinbacher, Kristin Y. Rozier, and Johann Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 8413 of *Lecture Notes in Computer Science (LNCS)*, pages 357–372. Springer-Verlag, April 2014.

[8] Kristin Yvonne Rozier. On the evaluation and comparison of runtime verification tools for hardware and cyber-physical systems. In *Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES)*, volume 3, pages 123–137, Seattle, WA, USA, September 2017. Kalpa Publications.

[9] K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. In *Model Checking Software (SPIN)*, volume 4595 of *Lecture Notes in Computer Science (LNCS)*, pages 149–167. Springer-Verlag, 2007.

[10] K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(2):123 – 137, March 2010.

[11] K.Y. Rozier and M.Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *17th International Symposium on Formal Methods (FM2011)*, volume 6664 of *Lecture Notes in Computer Science (LNCS)*, pages 417–431. Springer-Verlag, 2011.

[12] K.Y. Rozier and M.Y. Vardi. Deterministic compilation of temporal safety properties in explicit state model checking. In *8th Haifa Verification Conference (HVC2012)*, volume 7857 of *Lecture Notes in Computer Science (LNCS)*, pages 243–259. Springer-Verlag, 2012.

[5] https://www.rv-competition.org/