Temporal Logic Satisfiability
From Specification Debugging to Benchmark Generation

Kristin Yvonne Rozier

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; 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. SAT has its own long-standing conference series; so does SMT.

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, $\pi$, upholds its specification, $\varphi$. Stream-based RV asks for all $i$ during the mission, whether $\pi, i \models \varphi$: does the trace starting from time $i$ satisfy $\varphi$. 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 $\pi, \varphi$, 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 competition5 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