Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2025) 1–8
Procedia Computer Science

WEST: Interactive Validation of Mission-time Linear Temporal Logic (MLTL)☆

Wang, Zili[a,*], Gamboa Guzman, Laura P.[a], Rozier, Kristin Y.[a]

[a]*Iowa State University, Ames, IA 50011, USA*

Abstract

Mission-time Linear Temporal Logic (MLTL) is a finite, discrete, closed-interval-bounded variant of Metric Temporal Logic (MTL) that formal methods practitioners use to specify requirements for safety-critical systems, such as aircraft and spacecraft. Our tool addresses the specification bottleneck of formal verification by providing an interactive visualization tool for MLTL that allows practitioners to validate that their MLTL specifications do indeed match the intended requirements. We provide an overview of the functionalities of the command-line interface and the graphical user interface of the WEST tool. Additionally, we provide five independent methods used to validate the tool's correctness, as well as experimental results demonstrating the tool's scalability on three suites of randomly generated MLTL formulas.

Keywords: Mission-time Linear Temporal Logic (MLTL), MLTL Validation, Temporal Logic Validation

1. Introduction

Mission-time Linear Temporal Logic (MLTL) adds finite, discrete, closed-interval bounds over integers to the temporal operators of Linear Temporal Logic (LTL). Many specifications from case studies written in other bounded logics such as Metric Temporal Logic (MTL) and Signal Temporal Logic (STL) can be transformed to specifications in MLTL [1, 2]. MLTL was the specification logic for NASA's Robonaut2 verification project [3] as well as the specification logic for both design-time and runtime verification of the NASA Lunar Gateway Vehicle System Manager [4, 5, 6]. There has also been an abundance of verification efforts involving MLTL (see [7, 8, 9, 10, 11, 12]). Although formal methods in practice presume that human designers will translate system specifications to formal logic specifications (i.e., temporal logic), this is known to be challenging, e.g., for LTL [13] and STL [14]. Misconceptions during the translation of English requirements to logical specifications have been a long-standing bottleneck of formal verification [15]. After all, model checking is not very useful when the specification being checked is not the one intended by the system designer. Various published tools address this issue for LTL [16], MTL [17], and STL [18], but WEST [19] is first to address this issue for MLTL.

☆Work supported in part by NSF DMS-1950583, NSF CAREER Award CNS-1552934, NSF:CCRI-2016592, and ISU "Bridging the Divide." Additionally, work is partially supported by the HPC@ISU equipment at Iowa State University, some of which has been purchased through funding provided by NSF under MRI grants number 1726447 and MRI2018594.

*Corresponding author

Email addresses: ziliw1@iastate.edu (Wang, Zili), lpgamboa@iastate.edu (Gamboa Guzman, Laura P.), kyrozier@iastate.edu (Rozier, Kristin Y.)

We present the WEST tool for the validation of MLTL specifications, adapted from the WEST algorithm [19]. Our tool provides visualizations for MLTL formulas: given an MLTL formula, WEST produces the set of trace regular expressions (described in section 2) that succinctly captures the set of satisfying traces for the formula. The tool features a command-line interface (CLI) as well as a graphical user interface (GUI). The CLI writes outputs to console, as well as to output files. The GUI allows the user to interact with traces to explore the behavior of the formula. The CLI is written in C++, while the GUI is written in Python using the PyQt5 library, using the CLI as its backend. A video tutorial on how to build and use the WEST tool as well as the source code can be found at [20].

2. Background

Given a finite set of atomic propositions $\mathcal{AP}$, the syntax of MLTL formulas φ, ψ, and ξ are recursively defined as:

$$\xi := true \mid false \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathtt{F}_{[a,b]}\varphi \mid \mathtt{G}_{[a,b]}\varphi \mid \varphi \mathtt{U}_{[a,b]}\psi \mid \varphi \mathtt{R}_{[a,b]}\psi,$$

where $p \in \mathcal{AP}$, and $a, b \in \mathbb{Z}$ such that $0 \leq a \leq b$. The symbols $\mathtt{F}, \mathtt{G}, \mathtt{U}, \mathtt{R}$ denote the temporal operators Future, Globally, Until, and Release, respectively.

Definition 1. A **trace** π of length m is a finite sequence $\pi = \pi[0], \ldots, \pi[m-1]$ of sets of atomic propositions (i.e., $\pi[i] \subseteq \mathcal{AP}$), where $p \in \pi[i]$ if and only if p is true at time step i.

Traces represent timelines that encode the truth values for each atomic proposition at every time step. We denote the suffix of π starting at i (including i) by π_i. Thus note that $\pi_0 = \pi$. The length of a trace π is denoted by $|\pi|$. We evaluate an MLTL formula over a trace as follows.

Definition 2. The **satisfaction** of an MLTL formula φ by a trace π, denoted $\pi \vDash \varphi$, is defined as follows [19]:

$\pi \vDash p$ iff $|\pi| > 0$ and $p \in \pi[0]$

$\pi \vDash \neg\varphi$ iff $\pi \nvDash \varphi$

$\pi \vDash \varphi \wedge \psi$ iff $\pi \vDash \varphi$ and $\pi \vDash \psi$

$\pi \vDash \varphi \vee \psi$ iff $\pi \vDash \varphi$ or $\pi \vDash \psi$

$\pi \vDash \mathtt{F}_{[a,b]}\ \varphi$ iff $|\pi| > a$ and $\exists i \in [a, b]$ such that $\pi_i \vDash \varphi$

$\pi \vDash \mathtt{G}_{[a,b]}\ \varphi$ iff $|\pi| \leq a$ or $\forall i \in [a, b]\ \pi_i \vDash \varphi$

$\pi \vDash \varphi\ \mathtt{U}_{[a,b]}\ \psi$ iff $|\pi| > a$ and $\exists i \in [a, b]$ such that $\pi_i \vDash \psi$ and $\forall j \in [a, i-1]\ \pi_j \vDash \varphi$

$\pi \vDash \varphi\ \mathtt{R}_{[a,b]}\ \psi$ iff $|\pi| \leq a$ or $(\forall i \in [a, b]\ \pi_i \vDash \psi)$ or $(\exists j \in [a, b]$ such that $\pi_j \vDash \varphi$ and $\forall k \in [a, j]\ \pi_k \vDash \psi)$

Illustrations of the intuitive meanings of the temporal operators appear in Figure 1. A formula φ is in **negation normal form** (NNF) if negations only appear in front of atomic propositions. Any MLTL formula not in NNF can be converted to an equivalent formula linear in the size of the original formula by pushing negations inwards using the logical duals: $\neg(\varphi\ \mathtt{U}_{[a,b]}\ \psi)$ is equivalent to $(\neg\varphi\ \mathtt{R}_{[a,b]}\ \neg\psi)$, and $\neg\mathtt{G}_{[a,b]}\ \varphi$ is equivalent to $\mathtt{F}_{[a,b]}\ \neg\varphi$.

Since the set of atomic propositions is finite, we may assume without loss of generality that $\mathcal{AP} = \{p_0, p_1, \ldots, p_{n-1}\}$. This imposes a natural ordering on the atomic propositions, which we use to define the string encoding of a trace.

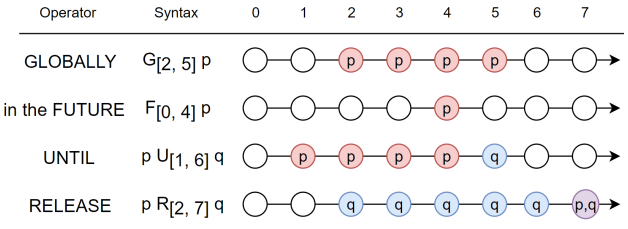


Figure 1: "Intuitive" semantics of the temporal operators. The presence of a variable indicates that it is true at that time step, and the absence indicates that it is false.

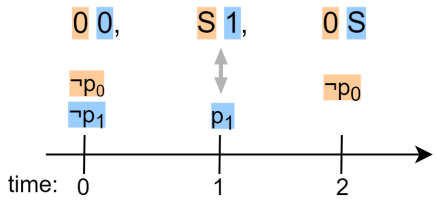


Figure 2: Example of the correspondence between a trace regular expression and the types of traces it represents. The absence of a variable indicates that its truth value is not specified at that time step.

Definition 3. The **string encoding** of a finite trace π of length $m = |\pi|$ over $n = |\mathcal{AP}|$ atomic propositions is the string $w_\pi \in \{\mathbf{0}, \mathbf{1}\}^{mn}$, i.e., a binary string of length mn, such that $p_k \in \pi[i]$ if and only if the $(i * n + k)$-th character (indexing from 0) of w_π is $\mathbf{1}$. We will refer to the string encoding of a trace as simply the trace.

Definition 4. For fixed $m, n \in \mathbb{N}$, a **trace regular expression** is a string $r \in \{\mathbf{0}, \mathbf{1}, \mathbf{s}\}^{mn}$ representing a set of traces of length m over n atomic propositions. This set is the language of the trace regular expression, denoted $\mathcal{L}(r) \subseteq \{\mathbf{0}, \mathbf{1}\}^{mn}$, where a trace π of length m belongs to $\mathcal{L}(r)$ iff for every $0 \leq i < mn$, the i-th character of r is either **s** or is equal to the i-th character of π. In other words, **0** and **1** represent the truth assignments *true* and *false* for an atomic proposition at the corresponding time step, respectively, and **s** represents that the truth assignment can be either *true* or *false*.

The **computation length** of an MLTL formula φ is the minimum length needed for a trace so that no interval in φ is out of bounds, as defined in [19], also known as the worst-case propagation delay in [21]. Given an MLTL formula φ with $n = |\mathcal{AP}|$ atomic propositions and computation length $m = \mathit{complen}(\varphi)$, the corresponding trace regular expressions $\mathit{reg}(\varphi)$ represents the set of all traces of length m that satisfy φ [19]. That is, $\mathcal{L}(\mathit{reg}(\varphi)) = \{\pi : \pi \vDash \varphi \text{ and } |\pi| = m\}$; [19] proves this representation is sound and complete. As an example, the trace regular expression $r =$ **00,s1,0s** (illustrated in Figure 2) represents a set of 4 traces, each of length 3, over atomic propositions $\mathcal{AP} = \{p_0, p_1\}$. For any trace $\pi \in \mathcal{L}(r)$, p_0 and p_1 are false at time 0, p_1 is false at time 1, and p_0 is false at time 2.

3. WEST Algorithm and Tool

We provide an overview of the WEST algorithm. The algorithm takes as input an MLTL formula φ in negation normal form, and recursively computes a set of trace regular expressions, $\mathit{reg}(\varphi)$, each of length $\mathit{complen}(\varphi)$ that captures the set of traces satisfying φ. The base cases of the recursion are if φ is a literal, meaning an atomic proposition or its negation; if φ is the Boolean constant *true*, in which case the trace regular expression is simply the string encoding of the trace of length 1 of all `s`; or if φ is the Boolean constant *false*, in which case the algorithm returns the empty trace. The recursive case for conjunction ($\vee$) computes the union of the sets of trace regular expressions, while the recursive case for disjunction ($\wedge$) computes the intersection of the sets of trace regular expressions. The recursive cases for the temporal operators `F`, `G`, `U`, `R` are computed solely using $\wedge$ and $\vee$ operations, and the base cases of the recursion. The details for the remaining recursive cases appear in [19], as well as proofs for the soundness and completeness of the algorithm. A summary of the WEST algorithm appears in Figure 3.

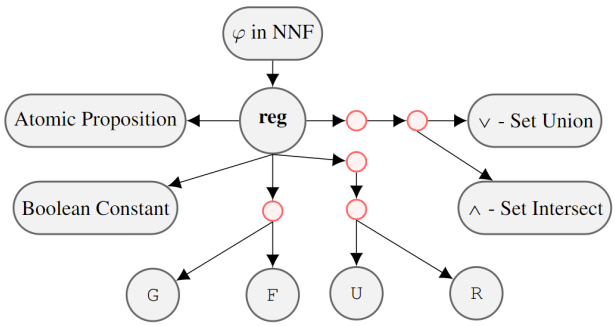


Figure 3: Abstracted flowchart of the WEST algorithm on input formula φ in negation normal form. Red nodes indicate recursive calls to *reg* on subformulas of the input formula. Inputs to recursive calls are MLTL formulas, and the outputs are trace regular expressions, each handled by the appropriate case of the algorithm.

```
$ ./west "(p0 & !(F[0,3]!p1)) -> p2"
Input formula: ((p0&!(F[0,3]!p1))->p2)
nnf: ((p0&G[0,3]p1)->p2)
Number of propositonal variables: 3
Computation Length: 4
Use -OPTIMIZED flag to optimize bits
(Warning: This has a few seconds overhead)
========================================
0ss,sss,sss,sss
s0s,sss,sss,sss
sss,s0s,sss,sss
sss,sss,s0s,sss
sss,sss,sss,s0s
ss1,sss,sss,sss
========================================
Time taken: 0 milliseconds
Number of regular expressions: 6
Output written to ./output/output.txt
Subformulas written to ./output/subformulas.txt
```

Figure 4: Example output of the WEST command-line tool on the formula $(p_0 \wedge \neg(\mathtt{F}[0,3]\neg p_1)) \rightarrow p_2$. WEST prints the set of trace regular expressions for the formula, along with various other statistics.

3.1. Command-line Tool

The command-line tool runs by executing the command `./west <formula>`. The formula input syntax is detailed in the repository's README file [20]. The tool will output the set of trace regular expressions that represent the set of satisfying traces for the formula to the console, along with various other information such as the negation normal form of the formula, the number of atomic propositions, the number of trace regular expressions computed, and the time taken. For each subformula of the input formula (including the input formula itself), the tool writes the corresponding

set of trace regular expressions to the `output` directory. An example of running the command-line tool on the formula $(p_0 \wedge \neg(\mathrm{F}[0,3]\neg p_1)) \rightarrow p_2$ [1] appears in Figure 4.

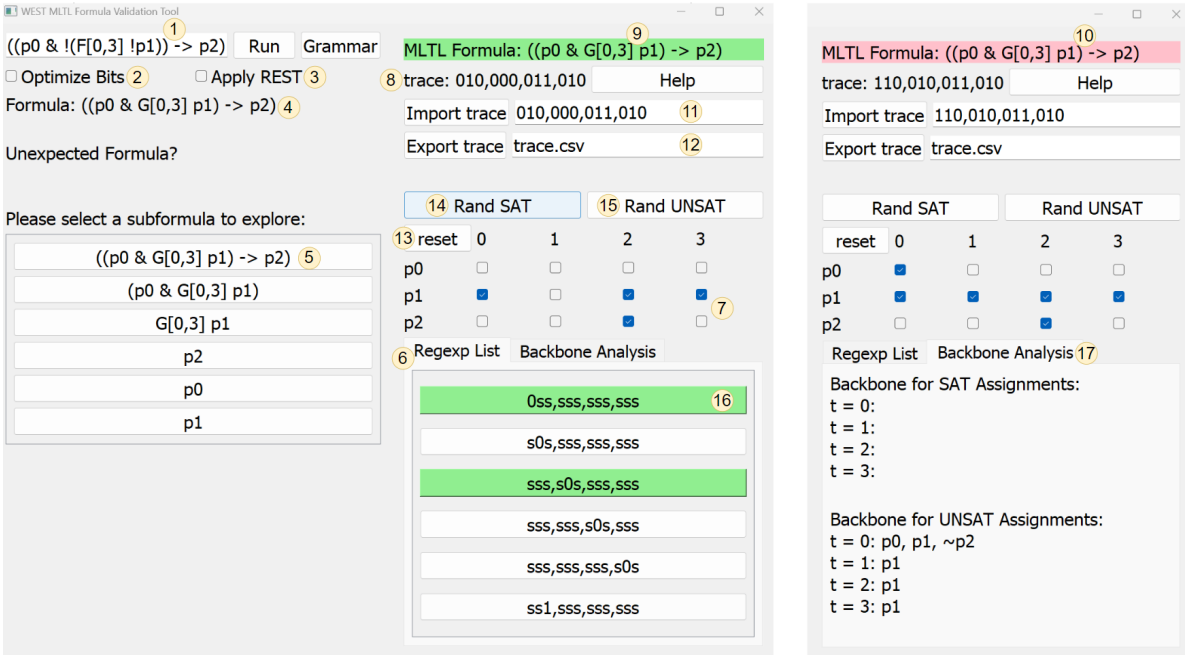


Figure 5: The left screenshot shows the WEST GUI when the input formula $(p_0 \wedge \neg(\mathrm{F}[0,3]\neg p_1)) \rightarrow p_2$ is satisfied by the toggled trace, indicated by the formula highlighted in green. The formula highlighted in red (right screenshot) indicates that the toggled trace does not satisfy the formula.

3.2. User Interface

The GUI is written in Python using the PyQt5 library, using the command-line tool as a backend. The GUI prompts the user to input an MLTL formula (label **1** in Figure 5), with options to optimize bits (label **2**), apply the Regular Expression Simplification Theorem (label **3**), which is stated and proven in [19], and display the negation normal form of the formula (label **4**). The user is then prompted to select a subformula to visualize (label **5**). Upon selecting a subformula, the GUI will display the set of trace regular expressions for the subformula (label **6**), and the user can interact with the trace by toggling the truth values of the atomic propositions at each time step (label **7**). The trace updates in real time (label **8**), and the formula gets highlighted in green if the trace satisfies the subformula (label **9**), and highlighted in red if the trace does not satisfy the subformula (label **10**). The trace can also be specified by typing the string encoding of the trace into the GUI, or importing it from a CSV file (label **11**), and the trace can be exported to a specified CSV file (label **12**). The `reset` button resets the trace to the trace with all atomic propositions false (label **13**). The `Rand SAT` button randomly generates a satisfying trace (label **14**), while the `Rand UNSAT` button randomly generates an unsatisfying trace (label **15**). Clicking on one of the listed trace regular expressions randomly selects a trace from the set of traces represented by that trace regular expression (label **16**).

Lastly, the backbone assignments for the formula and the negation of formula appear under the backbone tab (label **17**). The backbone of a formula is the assignment of atomic propositions that must hold in every satisfying trace of the formula. Accordingly, the backbone of the negation of the formula is the assignment of atomic propositions that must hold in every unsatisfying trace of the formula. The backbone analysis is useful for understanding the necessary conditions for satisfaction and unsatisfaction of the formula.

3.3. Implementation

Our previously published artifact [22] is a prototype that represents string trace encodings as C++ strings. The artifact accompanying this paper [23] more efficiently represents trace encodings as C++ bit sets, to take advantage

[1] Negations ($\neg$) are represented by the `!` character, and conjunctions ($\wedge$) by the `&` character. A full grammar for the input syntax pops up when clicking on the `Grammar` button in the GUI.

of the speed of vectorized bitwise operations. C++ bit sets must have their sizes specified at compile time, which is why the "optimize bits" option is available to recompile with the most optimal number of bits for an input formula. Technical details of the implementation are available in the online supplementary materials [20].

4. Validation

Although the WEST algorithm is sound and complete, we must validate the tool to ensure that the algorithm is implemented correctly. Previously, the prototype WEST tool was validated over a set of 1662 MLTL formulas, and the details of the validation effort and test suite generation were published in [19]. We validate the WEST tool over the same test suite with the prototype WEST tool, a Binary Decision Diagram (BDD)-based AllSAT solver applied to Boolean encodings of MLTL formulas[24], the R2U2 tool[25], and an MLTL evaluator [20]. On each of the 1662 MLTL formulas, we check that the the output of the WEST tool is equivalent to the output of the other four tools, as indicated by the output of the Python verification scripts in our artifact [22]. The FPROGG tool [26] also independently validates the WEST tool. Figure 6 summarizes the WEST tool validation process.

4.1. WEST Validation

We check that the bit set implementation of WEST is equivalent to the prototype WEST tool. On each formula φ of the test suite, the set of satisfying traces produced by the WEST tool is compared to the set of satisfying traces produced by the prototype WEST tool, by checking that $\mathcal{L}(r) = \mathcal{L}(r')$ where r and r' are the trace regular expressions produced by the WEST tool and the prototype WEST tool on the same input, respectively.

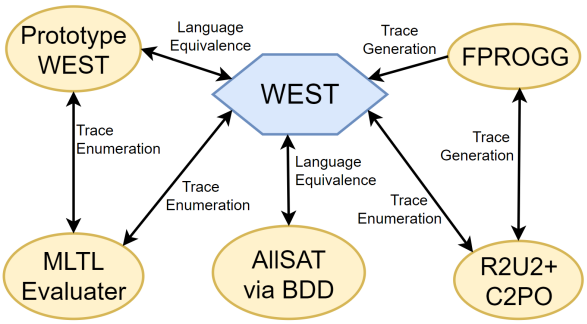


Figure 6: Summary of the WEST tool validation process. WEST is validated to a high degree of confidence through five different methods. Bidirectional arrows indicate that the validation process ensures that the set of satisfying traces produced by the two tools are equivalent, while the unidirectional arrow indicates a subset relationship.

4.2. AllSAT via BDD Engine

Recent work in MLTL MaxSAT solving [24] provides a translation from an MLTL formula to a Boolean formula that is not only equisatisfiable, but also logically equivalent if the additional variables introduced in the translation are disregarded. For each φ in the test suite, we apply the translation and encode the Boolean constraints in a binary decision diagram (BDD) engine. Specifically, we used the Python library `dd`, and the `pick_iter` function to enumerate all satisfying assignments for the Boolean encoding. Lastly, we translate the Boolean assignments into traces, and ensure that the set of traces produced this way for φ is the same as the set of traces produced by WEST.

4.3. R2U2

The Realizable, Responsive, Unobtrusive Unit (R2U2) is a runtime monitoring tool designed for MLTL specifications [27]. One byproduct of R2U2's observer-based monitoring engine is that one can easily use it as an MLTL evaluator when combined with the C2PO compiler [25]. That is, given a formula φ and a trace π, C2PO compiles the formula into assembly-like imperative evaluation instructions for R2U2 to determine if $\pi \vDash \varphi$. For each formula φ in the test suite, we iterate over all 2^{mn}, where $m = \mathit{complen}(\varphi)$ the computation length of φ and $n = |\mathcal{AP}|$ the number of atomic propositions, traces and check that the set of satisfying traces produced by WEST is equal to the set of satisfying traces produced by R2U2.

4.4. Additional Validation

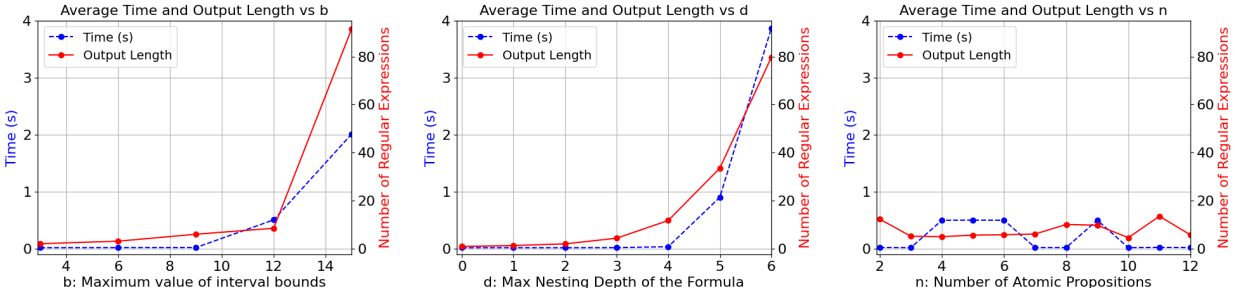
A direct implementation of the MLTL semantics in C++ also validates the WEST tool. The MLTL evaluator [20] determines $\pi \vDash \varphi$ for trace π and MLTL formula φ. The validation process is the same brute-force enumeration of traces used in the R2U2 validation process. Lastly, the FPROGG tool [26] implements the MLTL formula progression algorithm [28] developed for MLTL satisfiability checking. The tool takes as input an MLTL formula and produces a single trace that satisfies the formula by iteratively calling a SAT solver. The FPROGG validation exercise in [26] independently checked that the trace produced by FPROGG was amongst the traces produced by WEST, over their test suite of 55 MLTL formulas.

5. Experimental Benchmarking

We demonstrate the performance of the WEST tool on randomly generated MLTL formulas. Our random formula generator takes as parameters the number of atomic propositions n, the maximum nesting depth of the formula d, the maximum value of interval bounds b, and the probability p of a temporal operator at each node of the formula's syntax tree. We show the results of three experiments, in which we vary n, d, and b, while keeping the other parameters fixed. p is fixed at 0.5 for all experiments, in order to generate formulas with nontrivial Boolean and temporal structures. For each value of the parameter being varied, we generate 250 random MLTL formulas, and record the average time and number of trace regular expressions produced by WEST for each formula. We ran these experiments on the Iowa State Nova Cluster [29], on a node with an Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz with 384GB of RAM.

The results shown in Figures 7a and 7b demonstrate that b and d have a significant impact on the run time and number of trace regular expressions produced by WEST. This is expected, since a longer computation length b can be equivalently captured by appropriate nesting of temporal operators. The results shown in Figure 7c demonstrate that the number of atomic propositions n has a less significant impact on the tool's performance.

The case study presented in [30] affirms that the WEST tool is sufficiently performant for visualizing MLTL formulas that can feasibly be written and comprehended by humans during the design and specification process. Across four separate aerospace systems, over 200 person-hours were spent on the elicitation of 422 MLTL specifications matching one of four patterns: $^{1.}\ \mathtt{G}_{[0,M]}a_0$ $^{2.}\ \mathtt{G}_{[0,M]}\ (\mathtt{F}_{[0,N]}a_0)$ $^{3.}\ \mathtt{G}_{[0,M]}(a_0 \rightarrow \mathtt{F}_{[0,N]}a_1)$ or $^{4.}\ \mathtt{G}_{[0,M]}(a_0 \rightarrow (a_0\ \mathtt{U}_{[0,N]}\ a_1))$. N and M are integers, and each a_i is a propositional logic formulas that can be as simple as just p_0, but no more complicated than $\neg p_1 \rightarrow (p_2 \vee p_3)$, for example. Even in the fourth pattern, the most complex one, nesting depth d does not exceed 5 and the number of variables n does not exceed 6, which are within reasonable bounds for the WEST tool. Lastly, the value of b is dependent on how the designer chooses to model the system. Modeling a real system often necessitates higher-level abstraction of the system, and limiting the maximum value of interval bounds b to 10, for example, still gives reasonable freedom to specify the system's behavior in different stages of its target environment (such as in the launching, boosting, coasting, and descent phases of a sounding rocket [9]).



(a) Maximum interval bound b varies, fixed number of atomic propositions $n = 4$, fixed maximum nesting depth $d = 2$.

(b) Maximum nesting depth d varies, fixed number of atomic propositions $n = 4$, fixed maximum interval bound $b = 3$.

(c) Number of atomic propositions n varies, fixed maximum nesting depth $d = 2$, fixed maximum interval bound $b = 8$.

Figure 7: Performance of the WEST tool on randomly generated MLTL formulas. Each experiment varies over different values of one parameter while keeping all other variables fixed. Each datapoint is an average over 250 randomly generated MLTL formulas.

6. Impact and Future work

The WEST tool is a useful visualizer for MLTL formulas that provides the first interactive tool for validating MLTL specifications. This will aid system designers in the elicitation of specifications from English requirements. The improved explanability of formulas through visualizing and interacting with traces will make it easier for designers to demonstrate the correctness of their specifications. Additionally, WEST is now tied into the continuous integration testing of the R2U2 tool, which now reruns our validation test suite after significant updates to R2U2. The development of future tools for MLTL can similarly leverage WEST for validation testing.

Future work to the GUI includes additional visualization options, such as signal representations of traces, displaying automata representations of the input MLTL formula, and alternative visualizations. The WEST algorithm gives a translation of MLTL into regular expressions, which have a canonical translation to finite automata.

References

[1] R. Alur, T. A. Henzinger, Real-time logics: complexity and expressiveness, Information and Computation 104 (1) (1993) 35–77.

[2] O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: FORMATS, Springer, 2004, pp. 152–166.

[3] B. Kempa, P. Zhang, P. H. Jones, J. Zambreno, K. Y. Rozier, Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2, in: Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), Vol. 12288 of Lecture Notes in Computer Science (LNCS), Springer, Vienna, Austria, 2020, pp. 196–214. doi:10.1007/978-3-030-57628-8_12.

[4] J. B. Dabney, J. M. Badger, P. Rajagopal, Adding a verification view for an autonomous real-time system architecture, in: AIAA Scitech 2021 Forum, 2021, p. 0566.

[5] J. B. Dabney, Using assume-guarantee contracts in autonomous spacecraft, Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=zrtyiyNf674 (February 2021).

[6] J. B. Dabney, P. Rajagopal, J. M. Badger, Using assume-guarantee contracts for developmental verification of autonomous spacecraft, Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=HFnn6TzblPg (February 2022).

[7] N. Okubo, Using R2U2 in JAXA program, Electronic correspondence, series of emails and zoom call from JAXA with technical questions about embedding MLTL formula monitoring into an autonomous satellite mission with a provable memory bound of 200KB (November–December 2020).

[8] A. Hammer, M. Cauwels, B. Hertz, P. Jones, K. Y. Rozier, Integrating Runtime Verification into an Automated UAS Traffic Management System, Innovations in Systems and Software Engineering: A NASA Journaldoi:10.1007/s11334-021-00407-5.

[9] B. Hertz, Z. Luppen, K. Y. Rozier, Integrating runtime verification into a sounding rocket control system, in: Proceedings of the 13th NASA Formal Methods Symposium (NFM 2021), LNCS, Springer International Publishing, 2021, pp. 151–159. doi:10.1007/978-3-030-76384-8_10.

[10] Z. A. Luppen, D. Y. Lee, K. Y. Rozier, Correction: A case study in formal specification and runtime verification of a cubesat communications system, in: AIAA Scitech 2021 Forum, American Institute of Aeronautics and Astronautics, Nashville, TN, USA, 2021, p. 0997. doi:10.2514/6.2021-0997.c1.

[11] Z. Luppen, M. Jacks, N. Baughman, B. Hertz, J. Cutler, D. Y. Lee, K. Y. Rozier, Elucidation and Analysis of Specification Patterns in Aerospace System Telemetry, in: Proceedings of the 14th NASA Formal Methods Symposium (NFM 2022), Vol. 13260 of Lecture Notes in Computer Science (LNCS), Springer, Cham, Caltech, California, USA, 2022, pp. 527–537. doi:10.1007/978-3-031-06773-0_28.

[12] A. Aurandt, P. H. Jones, K. Y. Rozier, Runtime Verification Triggers Real-Time, Autonomous Fault Recovery on the CySat-I, in: Proceedings of the 13th NASA Formal Methods Symposium (NFM), LNCS, Springer International Publishing, 2022, pp. 816–825. doi:10.1007/978-3-031-06773-0_45.

[13] B. Greenman, S. Saarinen, T. Nelson, S. Krishnamurthi, Little tricky logic: Misconceptions in the understanding of ltl, The Art, Science, and Engineering of Programming 7 (2). doi:10.22152/programming-journal.org/2023/7/7.
URL http://dx.doi.org/10.22152/programming-journal.org/2023/7/7

[14] H. C. Siu, K. Leahy, M. Mann, Stl: Surprisingly tricky logic (for system validation) (2023). arXiv:2305.17258.

[15] K. Y. Rozier, Specification: The biggest bottleneck in formal methods and autonomy, in: Proceedings of 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2016), Vol. 9971 of LNCS, Springer-Verlag, Toronto, ON, Canada, 2016, pp. 1–19. doi:10.1007/978-3-319-48869-1_2.

[16] R. Li, K. Gurushankar, M. H. Heule, K. Rozier, What's in a name? linear temporal logic literally represents time lines, in: 2023 IEEE Working Conference on Software Visualization (VISSOFT), IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 73–83. doi:10.1109/VISSOFT60811.2023.00018.
URL https://doi.ieeecomputersociety.org/10.1109/VISSOFT60811.2023.00018

[17] L. Lima, A. Herasimau, M. Raszyk, D. Traytel, S. Yuan, Explainable online monitoring of metric temporal logic, in: S. Sankaranarayanan, N. Sharygina (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Nature Switzerland, Cham, 2023, pp. 473–491.

[18] D. Nickovic, O. Lebeltel, O. Maler, T. Ferrère, D. Ulus, Amt 2.0: qualitative and quantitative trace analysis with extended signal temporal logic, International Journal on Software Tools for Technology Transfer 22 (2020) 1–18. doi:10.1007/s10009-020-00582-z.

[19] J. Elwing, L. Gamboa-Guzman, J. Sorkin, C. Travesset, Z. Wang, K. Y. Rozier, Mission-time LTL (MLTL) formula validation via regular expressions, in: P. Herber, A. Wijs (Eds.), iFM 2023, Springer Nature Switzerland, Cham, 2024, pp. 279–301.

[20] Z. Wang, C. Travesset, S. Jeremy, J. Elwing, L. Gamboa-Guzman, K. Y. Rozier, WEST: Website, https://temporallogic.org/research/WEST/, accessed: 2023-10-09.

[21] C. Boufaied, C. Menghi, D. Bianculli, L. Briand, Y. I. Parache, Trace-checking signal-based temporal properties: A model-driven approach, in: 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2020, pp. 1004–1015.

[22] Z. Wang, C. Travesset, S. Jeremy, J. Elwing, L. Gamboa-Guzman, K. Y. Rozier, Artifact for "Mission-time LTL (MLTL) Formula Validation Via Regular Expressions", Zenodo (2023).
URL https://doi.org/10.5281/zenodo.8339325

[23] Z. Wang, L. Gamboa-Guzman, K. Y. Rozier, Artifact for "WEST: Interactive Validation of Mission-time Linear Temporal Logic (MLTL)", Zenodo (2024).
URL https://zenodo.org/records/14649154

[24] G. Hariharan, P. H. Jones, K. Y. Rozier, T. Wongpiromsarn, Maximum satisfiability of misstion-time linear temporal logic, in: Formal Modeling and Analysis of Timed Systems: 21st International Conference, FORMATS 2023, Antwerp, Belgium, September 19–21, 2023, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2023, p. 86–104. doi:10.1007/978-3-031-42626-1_6.
URL https://doi.org/10.1007/978-3-031-42626-1_6

[25] C. Johannsen, P. Jones, B. Kempa, K. Y. Rozier, P. Zhang, R2U2 Version 3.0: Re-Imagining a Toolchain for Specification, Resource Estimation, and Optimized Observer Generation for Runtime Verification in Hardware and Software, in: C. Enea, A. Lal (Eds.), Computer Aided Verification, Springer Nature Switzerland, Cham, 2023, pp. 483–497.

[26] A. Rosentrater, K. Y. Rozier, FPROGG: A Formula Progression-Based MLTL Benchmark Generator, under submission. (2024).
[27] K. Y. Rozier, J. Schumann, R2U2: Tool Overview, in: Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES), Vol. 3, Kalpa Publications, Seattle, WA, USA, 2017, pp. 138–156. doi:10.29007/5pch.
URL https://easychair.org/publications/paper/Vncw
[28] J. Li, M. Y. Vardi, K. Y. Rozier, Satisfiability checking for Mission-time LTL (MLTL), Information and Computation (2022) 104923doi:https://doi.org/10.1016/j.ic.2022.104923.
[29] Iowa State University, Iowa State University HPC, https://www.hpc.iastate.edu/, accessed: 2023-10-09.
[30] R. Dureja, K. Y. Rozier, Incremental Design-space Model Checking Via Reusable Reachable State Approximations, Formal Methods in System Design (FMSD) Journal online (1). doi:{https://doi.org/10.1007/s10703-022-00389-5}.