Satisfiability Checking for Mission-Time LTL

Jianwen Li[1(✉)], Moshe Y. Vardi[2], and Kristin Y. Rozier[1(✉)]

[1] Iowa State University, Ames, IA, USA
lijwen2748@gmail.com, kyrozier@iastate.edu
[2] Rice University, Houston, TX, USA

Abstract. Mission-time LTL (MLTL) is a bounded variant of MTL over naturals designed to generically specify requirements for mission-based system operation common to aircraft, spacecraft, vehicles, and robots. Despite the utility of MLTL as a specification logic, major gaps remain in analyzing MLTL, e.g., for specification debugging or model checking, centering on the absence of any complete MLTL satisfiability checker. We prove that the MLTL satisfiability checking problem is NEXPTIME-complete and that satisfiability checking MLTL_0, the variant of MLTL where all intervals start at 0, is PSPACE-complete. We introduce translations for MLTL-to-LTL, MLTL-to-LTL_f, MLTL-to-SMV, and MLTL-to-SMT, creating four options for MLTL satisfiability checking. Our extensive experimental evaluation shows that the MLTL-to-SMT transition with the Z3 SMT solver offers the most scalable performance.

1 Introduction

Mission-time LTL (MLTL) [34] has the syntax of Linear Temporal Logic with the option of integer bounds on the temporal operators. It was created as a generalization of the variations [3,14,25] on finitely-bounded linear temporal logic, ideal for specification of missions carried out by aircraft, spacecraft, rovers, and other vehicular or robotic systems. MLTL provides the readability of LTL [32], while assuming, when a different duration is not specified, that all requirements must be upheld during the (a priori known) length of a given mission, such as during the half-hour battery life of an Unmanned Aerial System (UAS). Using integer bounds instead of real-number or real-time bounds leads to more generic specifications that are adaptable to model checking at different levels of abstraction, or runtime monitoring on different platforms (e.g., in software vs in hardware). Integer bounds should be read as generic time units, referring to the basic temporal resolution of the system, which can generically be resolved to units such as clock ticks or seconds depending on the mission. Integer bounds also allow generic specification with respect to different granularities of time, e.g., to allow easy updates to model-checking models, and re-usable specifications for the same requirements on different embedded systems that may have different resource limits for storing runtime monitors. MLTL has been used in many industrial case studies [18,28,34,37,42–44], and was the official logic of the 2018 Runtime Verification Benchmark Competition [1]. Many specifications from other case studies, in logics such as MTL [3] and STL [25], can be represented in MLTL. We intuitively relate MLTL to LTL and MTL-over-naturals as follows: (1) MLTL formulas are LTL formulas with bounded intervals over temporal operators, and interpreted over

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11562, pp. 3–22, 2019.
https://doi.org/10.1007/978-3-030-25543-5_1

finite traces. (2) MLTL formulas are MTL-over-naturals formulas without any unbounded intervals, and interpreted over finite traces.

Despite the practical utility of MLTL, no model checker currently accepts this logic as a specification language. The model checker nuXmv encodes a related logic for use in symbolic model checking, where the $\Box$ and $\Diamond$ operators of an `LTLSPEC` can have integer bounds [21], though bounds cannot be placed on the $\mathcal{U}$ or $\mathcal{V}$ (the Release operator of nuXmv) operators.

We also critically need an MLTL satisfiability checker to enable specification debugging. Specification is a major bottleneck to the formal verification of mission-based, especially autonomous, systems [35], with a key part of the problem being the availability of good tools for *specification debugging*. Satisfiability checking is an integral tool for specification debugging: [38,39] argued that for every requirement φ we need to check φ and $\neg\varphi$ for satisfiability; we also need to check the conjunction of all requirements to ensure that they can all be true of the same system at the same time. Specification debugging is essential to model checking [39–41] because a positive answer may not mean there is no bug and a negative answer may not mean there is a bug if the specification is valid/unsatisfiable, respectively. Specification debugging is critical for synthesis and runtime verification (RV) since in these cases there is no model; synthesis and RV are both entirely dependent on the specification. For synthesis, satisfiability checking is the best-available specification-debugging technique, since other techniques, such as vacuity checking (cf. [6,10]) reference a model in addition to the specification. While there are artifacts one can use in RV, specification debugging is still limited outside of satisfiability checking yet central to correct analysis. A false positive due to RV of an incorrect specification can have disastrous consequences, such as triggering an abort of an (otherwise successful) mission to Mars. Arguably, the biggest challenge to creating an RV algorithm or tool is the dearth of benchmarks for checking correctness or comparatively analyzing these [36], where a benchmark consists of some runtime trace, a temporal logic formula reasoning about that trace, and some verdict designating whether the trace at a given time satisfies the requirement formula. A MLTL satisfiability solver is useful for RV benchmark generation [22].

Despite the critical need for an MLTL satisfiability solver, no such tool currently exists. To the best of our knowledge, there is only one available solver (*zot* [8]) for checking the satisfiability of MTL-over-naturals formulas, interpreted over infinite traces. Since MLTL formulas are interpreted over finite traces and there is no trivial reduction from one to another, *zot* cannot be directly applied to MLTL satisfiability checking.

Our approach is inspired by satisfiability-checking algorithms from other logics. For LTL satisfiability solving, we observe that there are multiple efficient translations from LTL satisfiability to model checking, using nuXmv [40]; we therefore consider here translations to nuXmv model checking, both indirectly (as a translation to LTL), and directly using the new KLIVE [13] back-end and the BMC back-end, taking advantage of the bounded nature of MLTL. The bounded nature of MLTL enables us to also consider a direct encoding at the word-level, suitable as input to an SMT solver. Our contribution is both theoretic and experimental. We first consider the complexity of such translations. We prove that the MLTL satisfiability checking problem is NEXPTIME-complete and that satisfiability checking MLTL_0, the variant of MLTL where all intervals start at 0, is PSPACE-complete. Secondly, we introduce translation algorithms for MLTL-to-LTL_f (LTL over finite traces [14]), MLTL-to-LTL, MLTL-to-SMV, and

MLTL-to-SMT, thus creating four options for MLTL satisfiability checking. Our results show that the MLTL-to-SMT transition with the Z3 SMT solver offers the most scalable performance, though the MLTL-to-SMV translation with an SMV model checker can offer the best performance when the intervals in the MLTL formulas are restricted to small ranges less than 100.

2 Preliminaries

A (closed) interval over naturals $I = [a, b]$ ($0 \leq a \leq b$ are natural numbers) is a set of naturals $\{i \mid a \leq i \leq b\}$. I is called *bounded* iff $b < +\infty$; otherwise I is *unbounded*. MLTL is defined using bounded intervals. Unlike Metric Temporal Logic (MTL) [4], it is not necessary to introduce open or half-open intervals over the natural domain, as every open or half-open bounded interval is reducible to an equivalent closed bounded interval, e.g., (1,2) = $\emptyset$, (1,3) = [2,2], (1,3] = [2,3], etc. Let $\mathcal{AP}$ be a set of atomic propositions, then the syntax of a formula in MLTL is

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Box\varphi \mid \Diamond\varphi \mid \varphi\, \mathcal{U}_I\, \psi \mid \varphi \mathcal{R}_I \psi$$

where I is a bounded interval, $p \in \mathcal{AP}$ is an *atom*, and φ and ψ are subformulas.

Given two MLTL formulas φ, ψ, we denote $\varphi = \psi$ iff they are *syntactically equivalent*, and $\varphi \equiv \psi$ iff they are *semantically equivalent*, i.e., $\pi \models \varphi$ iff $\pi \models \psi$ for a finite trace π. In MLTL semantics, we define $\mathsf{false} \equiv \neg\mathsf{true}$, $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\neg(\varphi\, \mathcal{U}_I\, \psi) \equiv (\neg\varphi \mathcal{R}_I \neg\psi)$ and $\neg\Diamond_I\varphi \equiv \Box_I\neg\varphi$. MLTL keeps the standard operator equivalences from LTL, including $(\Diamond_I\varphi) \equiv (true\, \mathcal{U}_I\varphi)$, $(\Box_I\varphi) \equiv (false\, \mathcal{R}_I\, \varphi)$, and $(\varphi\, \mathcal{R}_I\, \psi) \equiv (\neg(\neg\varphi\, \mathcal{U}_I\, \neg\psi))$. Notably, MLTL discards the neXt ($\mathcal{X}$) operator, which is essential in LTL [32], since $\mathcal{X}\varphi$ is semantically equivalent to $\Box_{[1,1]}\varphi$.

The semantics of MLTL formulas is interpreted over finite traces bounded by base-10 (decimal) intervals. Let π be a finite trace in which every position $\pi[i]$ ($i \geq 0$) is over $2^{\mathcal{AP}}$, and $|\pi|$ denotes the length of π ($|\pi| < +\infty$ when π is a finite trace). We use π_i ($|\pi| > i \geq 0$) to represent the suffix of π starting from position i (including i). Let $a, b \in \mathbb{I}, a \leq b$; we define that π models (satisfies) an MLTL formula φ, denoted as $\pi \models \varphi$, as follows:

- $\pi \models p$ iff $p \in \pi[0]$;
- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$;
- $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$;
- $\pi \models \varphi\, \mathcal{U}_{[a,b]}\, \psi$ iff $|\pi| > a$ and, there exists $i \in [a, b]$, $i < |\pi|$ such that $\pi_i \models \psi$ and for every $j \in [a, b]$, $j < i$ it holds that $\pi_j \models \varphi$;

Compared to the traditional MTL-over-naturals[1] [16], the Until formula in MLTL is interpreted in a slightly different way. In MTL-over-naturals, the satisfaction of $\varphi\, \mathcal{U}_I\, \psi$ requires φ to hold from position 0 to the position where ψ holds (in I), while in MLTL φ is only required to hold within the interval I, before the time ψ holds. From the perspective of writing specifications, cf. [34,37], this adjustment is more user-friendly.

[1] In this paper, MTL-over-naturals is interpreted over finite traces.

It is not hard to see that MLTL is as expressive as the standard MTL-over-naturals: the formula $\varphi\ \mathcal{U}_{[a,b]}\ \psi$ in MTL-over-naturals can be represented as $(\Box_{[0,a-1]}\varphi) \wedge (\varphi\ \mathcal{U}_{[a,b]}\ \psi)$ in MLTL; $\varphi\ \mathcal{U}_{[a,b]}\ \psi$ in MLTL can be represented as $\Diamond_{[a,a]}(\varphi\ \mathcal{U}_{[0,b-a]}\ \psi)$ in MTL-over-naturals.

We say an MLTL formula is in *BNF* if the formula contains only $\neg$, $\wedge$ and $\mathcal{U}_I$ operators. It is trivial to see that every MLTL formula can be converted to its (semantically) equivalent BNF with a linear cost. Consider $\varphi = (\neg a) \vee ((\neg b)\mathcal{R}_I(\neg c))$ as an example. Its BNF form is $\neg(a \wedge (b\ \mathcal{U}_I\ c))$. Without explicit clarification, this paper assumes that every MLTL formula is in BNF.

The closure of an MLTL formula φ, denoted as $cl(\varphi)$, is a set of formulas such that: (1) $\varphi \in cl(\varphi)$; (2) $\varphi \in cl(\varphi)$ if $\neg\varphi \in cl(\varphi)$; (3) $\varphi, \psi \in cl(\varphi)$ if $\varphi\ op\ \psi \in cl(\varphi)$, where op can be $\wedge$ or $\mathcal{U}_I$. Let $|cl(\varphi)|$ be the size of $cl(\varphi)$. Since the definition of $cl(\varphi)$ ignores the intervals in φ, $|cl(\varphi)|$ is linear in the number of operators in φ. We also define the closure(*) of an MLTL formula φ, denoted $cl^*(\varphi)$, as the set of formulas such that: (1) $cl(\varphi) \subseteq cl^*(\varphi)$; (2) if $\varphi\ \mathcal{U}_{[a,b]}\ \psi \in cl^*(\varphi)$ for $0 < a \leq b$, then $\varphi\ \mathcal{U}_{[a-1,b-1]}\ \psi$ is in $cl^*(\varphi)$; (3) if $\varphi\ \mathcal{U}_{[0,b]}\ \psi \in cl^*(\varphi)$ for $0 < b$, then $\varphi\ \mathcal{U}_{[0,b-1]}\ \psi$ is in $cl^*(\varphi)$. Let $|cl^*(\varphi)|$ be the size of $cl^*(\varphi)$ and K be the maximal natural number in the intervals of φ. It is not hard to see that $|cl^*(\varphi)|$ is at most $K \cdot |cl(\varphi)|$.

We also consider a fragment of MLTL, namely MLTL_0, which is more frequently used in practice, cf. [18,34]. Informally speaking, MLTL_0 formulas are MLTL formulas in which all intervals start from 0. For example, $\Diamond_{[0,4]}a \wedge (a\ \mathcal{U}_{[0,1]}\ b)$ is a MLTL_0 formula, while $\Diamond_{[2,4]}a$ is not.

Given an MLTL formula φ, the *satisfiability problem* asks whether there is a finite trace π such that $\pi \models \varphi$ holds. To solve this problem, we can reduce it to the satisfiability problem of the related logics LTL and LTL_f (LTL over finite traces [14]), and leverage the off-the-shelf satisfiability checking solvers for these well-explored logics. We abbreviate MLTL, LTL, and LTL_f satisfiability checking as MLTL-SAT, LTL-SAT, and LTL_f-SAT respectively.

LTL_f: Linear Temporal Logic over Finite Traces [14]**.** We assume readers are familiar with LTL (over infinite traces). LTL_f is a variant of LTL that has the same syntax, except that for LTL_f, the dual operator of $\mathcal{X}$ is $\mathcal{N}$ (weak Next), which differs $\mathcal{X}$ in the last state of the finite trace. In the last state of a finite trace, $\mathcal{X}\psi$ can never be satisfied, while $\mathcal{N}\psi$ is satisfiable. Given an LTL_f formula φ, there is an LTL formula ψ such that φ is satisfiable iff ψ is satisfiable. In detail, $\psi = \Diamond Tail \wedge t(\varphi)$ where $Tail$ is a new atom identifying the end of the satisfying trace and $t(\varphi)$ is constructed as follows:

- $t(p) = p$ where p is an atom;
- $t(\neg\psi) = \neg t(\psi)$;
- $t(\mathcal{X}\psi) = \neg Tail \wedge \mathcal{X}t(\psi)$;
- $t(\psi_1 \wedge \psi_2) = t(\psi_1) \wedge t(\psi_2)$;
- $t(\psi_1 U \psi_2) = t(\neg Tail \wedge \psi_1) U t(\psi_2)$.

In the above reduction, φ is in BNF. Since the reduction is linear in the size of the original LTL_f formula and LTL-SAT is PSPACE-complete [45], LTL_f-SAT is also a PSPACE-complete problem [14].

3 Complexity of MLTL-SAT

It is known that the complexity of MITL (Metric Interval Temporal Logic) satisfiability is EXPSPACE-complete, and the satisfiability complexity of the fragment of MITL named $\mathsf{MITL}_{0,\infty}$ is PSPACE-complete [2]. MLTL (resp. MLTL_0) can be viewed as a variant of MITL (resp. $\mathsf{MITL}_{0,\infty}$) that is interpreted over the naturals. We show that MLTL satisfiability checking is NEXPTIME-complete, via a reduction from MLTL to LTL_f.

Lemma 1. *Let φ be an* MLTL *formula, and K be the maximal natural appearing in the intervals of φ (K is set to 1 if there are no intervals in φ). There is an LTL_f formula θ that recognizes the same language as φ. Moreover, the size of θ is in $O(K \cdot |cl(\varphi)|)$.*

Proof (Sketch). For an MLTL formula φ, we define the LTL_f formula $f(\varphi)$ recursively as follows:

- If $\varphi = \mathsf{true}, \mathsf{false}$, or an atom p, $f(\varphi) = \varphi$;
- If $\varphi = \neg\psi$, $f(\varphi) = \neg f(\psi)$;
- If $\varphi = \xi \wedge \psi$, $f(\varphi) = f(\xi) \wedge f(\psi)$;
- If $\varphi = \xi \, \mathcal{U}_{[a,b]} \, \psi$,

$$f(\varphi) = \begin{cases} \mathcal{X}(f(\xi \, \mathcal{U}_{[a-1,b-1]} \, \psi)), & \text{if } 0 < a \leq b; \\ f(\psi) \vee (f(\xi) \wedge \mathcal{X}(f(\xi U_{[a,b-1]} \psi))), & \text{if } a = 0 \text{ and } 0 < b; \\ f(\psi), & \text{if } a = 0 \text{ and } b = 0; \end{cases}$$

$\mathcal{X}$ represents the neXt operator in LTL_f. Let $\theta = f(\varphi)$; we can prove by induction that φ and θ accept the same language. Moreover, the size of θ is at most linear to $K \cdot |cl(\varphi)|$, i.e., in $O(K \cdot |cl(\varphi)|)$, based on the aforementioned construction. □

We use the construction shown in Lemma 1 to explore several useful properties of MLTL. For instance, the LTL_f formula translated from an MLTL formula contains only the $\mathcal{X}$ temporal operator or its dual $\mathcal{N}$, which represents weak Next [19,23], and the number of these operators is strictly smaller than $K \cdot |cl(\varphi)|$. Every $\mathcal{X}$ or $\mathcal{N}$ subformula in the LTL_f formula corresponds to some temporal formula in $cl^*(\varphi)$. Notably, because the natural-number intervals in φ are written in base 10 (decimal) notation, the blow-up in the translation of Lemma 1 is exponential.

The next lower bound is reminiscent of the NEXPTIME-lower bound shown in [31] for a fragment of Metric Interval Temporal Logic (MITL), but is different in the details of the proof as the two logics are quite different.

Theorem 1. *The complexity of* MLTL *satisfiability checking is NEXPTIME-complete.*

Proof (Sketch). By Lemma 1, there is an LTL_f formula θ that accepts the same traces as MLTL formula φ, and the size of θ is in $O(K \cdot |cl(\varphi)|)$. The only temporal connectives used in θ are $\mathcal{X}$ and $\mathcal{N}$, since the translation to LTL_f reduces all MLTL temporal connectives in φ to nested $\mathcal{X}$'s or $\mathcal{N}$'s (produced by simplifying $\neg\mathcal{X}$). Thus, if θ is satisfiable, then it is satisfiable by a trace whose length is bounded by the length of θ.

Thus, we can just guess a trace π of exponential length of θ and check that it satisfies φ. As a result, the upper bound for MLTL-SAT is NEXPTIME.

Before proving the NEXPTIME lower bound, recall the PSPACE-lower bound proof in [45] for LTL satisfiability. The proof reduces the acceptance problem for a linear-space bounded Turing machine M to LTL satisfiability. Given a Turing machine M and an integer k, we construct a formula φ_M such that φ_M is satisfiable iff M accepts the empty tape using k tape cells. The argument is that we can encode such a space-bounded computation of M by a trace π of length c^k for some constant c, and then use φ_M to force π to encode an accepting computation of M. The formula φ_M has to match corresponding points in successive configurations of M, which can be expressed using a $O(k)$-nested $\mathcal{X}$'s, since such points are $O(k)$ points apart.

To prove a NEXPTIME-lower bound for MLTL, we reduce the acceptance problem for exponentially bounded non-deterministic Turing machines to MLTL satisfiability. Given a non-deterministic Turing machine M and an integer k, we construct an MLTL formula φ_M of length $O(k)$ such that φ_M is satisfiable iff M accepts the empty tape in time 2^k. Note that such a computation of a 2^k-time bounded Turing machines consists of 2^k many configurations of length 2^k each, so the whole computation is of exponential length – 4^k, and can be encoded by a trace π of length 4^k, where every point of π encodes one cell in the computation of M. Unlike the reduction in [45], in the encoding here corresponding points in successive configurations are exponentially far (2^k) from each other, because each configuration has 2^k cells, so the relationship between such successive points cannot be expressed in LTL. Because, however, the constants in the intervals of MLTL are written in base-10 (decimal) notation, we can write formulas of size $O(k)$, e.g., formulas of the form $p\,\mathcal{U}_{[0,2^k]}\,q$, that relate points that are 2^k apart.

The key is to express the fact that one Turing machine configuration is a proper successor of another configuration using a formula of size $O(k)$. In the PSPACE-lower-bound proof of [45], LTL formulas of size $O(k)$ relate successive configurations of k-space-bounded machines. Here MLTL formulas of size $O(k)$ relate successive configurations of 2^k-time-bounded machines. Thus, we can write a formula φ_M of length $O(k)$ that forces trace π to encode a computation of M of length 2^k. □

Now we consider MLTL_0 formulas, and prove that the complexity of checking the satisfiability of MLTL_0 formulas is PSPACE-complete. We first introduce the following lemma to show an inherent feature of MLTL_0 formulas.

Lemma 2. *The conjunction of identical* MLTL_0 $\mathcal{U}$*-rooted formulas is equivalent to the conjunct with the smallest interval range:* $(\xi\ \mathcal{U}_{[0,a]}\ \psi) \wedge (\xi\ \mathcal{U}_{[0,b]}\ \psi) \equiv (\xi\ \mathcal{U}_{[0,a]}\ \psi)$*, where* $b > a$*.*

Proof. We first prove that for $i \geq 0$, the equation $(\xi\ \mathcal{U}_{[0,i]}\ \psi) \wedge (\xi\ \mathcal{U}_{[0,i+1]}\ \psi) \equiv (\xi\ \mathcal{U}_{[0,i]}\ \psi)$ holds. When $i = 0$, we have $(\xi\ \mathcal{U}_{[0,0]}\ \psi) \equiv f(\psi)$ and $(\xi\ \mathcal{U}_{[0,1]}\ \psi) \equiv (f(\psi) \vee f(\xi) \wedge \mathcal{X}(f(\psi)))$. So $(\xi\ \mathcal{U}_{[0,0]}\ \psi) \wedge (\xi\ \mathcal{U}_{[0,1]}\ \psi) \equiv f(\psi) \equiv (\xi\ \mathcal{U}_{[0,0]}\ \psi)$ is true. Inductively, assume that $(\xi\ \mathcal{U}_{[0,k]}\ \psi) \wedge (\xi\ \mathcal{U}_{[0,k+1]}\ \psi) \equiv (\xi\ \mathcal{U}_{[0,k]}\ \psi)$ is true for $k \geq 0$. When $i = k + 1$, we have $(\xi\ \mathcal{U}_{[0,k+1]}\ \psi) \equiv (f(\psi) \vee f(\xi) \wedge \mathcal{X}(\xi\ \mathcal{U}_{[0,k]}\ \psi))$ and $(\xi\ \mathcal{U}_{[0,k+2]}\ \psi) \equiv (f(\psi) \vee f(\xi) \wedge \mathcal{X}(\xi\ \mathcal{U}_{[0,k+1]}\ \psi))$. By hypothesis assumption,

$(\xi\,\mathcal{U}_{[0,k]}\,\psi) \wedge (\xi\,\mathcal{U}_{[0,k+1]}\,\psi) \equiv (\xi\,\mathcal{U}_{[0,k]}\,\psi)$ implies that the following equivalence is true:

$$\begin{aligned}
&(\xi\,\mathcal{U}_{[0,k+1]}\,\psi) \wedge (\xi\,\mathcal{U}_{[0,k+2]}\,\psi)\\
&\equiv\ (f(\psi) \vee (f(\xi) \wedge \mathcal{X}(\xi\,\mathcal{U}_{[0,k]}\,\psi))) \wedge (f(\psi) \vee (f(\xi) \wedge \mathcal{X}(\xi\,\mathcal{U}_{[0,k+1]}\,\psi)))\\
&\equiv\ f(\psi) \vee (f(\xi) \wedge \mathcal{X}(\xi\,\mathcal{U}_{[0,k]}\,\psi \wedge \xi\,\mathcal{U}_{[0,k+1]}\,\psi))\\
&\equiv\ f(\psi) \vee (f(\xi) \wedge \mathcal{X}(\xi\,\mathcal{U}_{[0,k]}\,\psi))\\
&\equiv\ (\xi\,\mathcal{U}_{[0,k+1]}\,\psi).
\end{aligned}$$

Since $(\xi\,\mathcal{U}_{[0,i]}\,\psi) \wedge (\xi\,\mathcal{U}_{[0,i+1]}\,\psi) \equiv (\xi\,\mathcal{U}_{[0,i]}\,\psi)$ is true, we can prove by induction that $(\xi\,\mathcal{U}_{[0,i]}\,\psi) \wedge (\xi\,\mathcal{U}_{[0,j]}\,\psi) \equiv (\xi\,\mathcal{U}_{[0,i]}\,\psi)$ is true, where $j > i$. Because $b > a$ is true, it directly implies that $(\xi\,\mathcal{U}_{[0,a]}\,\psi) \wedge (\xi\,\mathcal{U}_{[0,b]}\,\psi) \equiv (\xi\,\mathcal{U}_{[0,a]}\,\psi)$ is true. □

Lemma 3. *$\mathcal{X}$-free* LTL_f-SAT *is reducible to* MLTL_0-SAT *at a linear cost.*

Proof. According to [45], the satisfiability checking of $\mathcal{X}$-free LTL formulas is still PSPACE-complete. This also applies to the satisfiability checking of $\mathcal{X}$-free LTL_f formulas. Given an $\mathcal{X}$-free LTL_f formula φ, we construct the corresponding MLTL formula $m(\varphi)$ recursively as follows:

- $m(p) = p$ where p is an atom;
- $m(\neg\xi) = \neg m(\xi)$;
- $m(\xi \wedge \psi) = m(\xi) \wedge m(\psi)$;
- $m(\xi\,\mathcal{U}\,\psi) = m(\xi)\,\mathcal{U}_{[0,2^{|\varphi|}]}\,m(\psi)$.

Notably for the Until LTL_f formula, we bound it with the interval $[0, 2^{|\varphi|}]$, where φ is the original $\mathcal{X}$-free LTL_f formula, in the corresponding MLTL formula, which is motivated by the fact that every satisfiable LTL_f formula has a finite model whose length is less than $2^{|\varphi|}$ [14]. The above translation has linear blow-up, because the integers in intervals use the decimal notation. Now we prove by induction over the type of φ that φ is satisfiable iff $m(\varphi)$ is satisfiable. That is, we prove that ($\Rightarrow$) $\pi \models \varphi$ implies $\pi \models m(\varphi)$ and ($\Leftarrow$) $\pi \models m(\varphi)$ implies $\pi \models \varphi$, for some finite trace π.

We consider the Until formula $\eta = \xi\,\mathcal{U}\,\psi$ (noting that φ is fixed to the original LTL_f formula), and the proofs are trivial for other types. ($\Rightarrow$) η is satisfiable implies there is a finite trace π such that $\pi \models \eta$ and $|\pi| \leq 2^{|\varphi|}$ [14]. Moreover, $\pi \models \eta$ holds iff there is $0 \leq i$ such that $\pi_i \models \psi$ and for every $0 \leq j < i$, $\pi_j \models \xi$ is true (from LTL_f semantics). By the induction hypothesis, $\pi_i \models \psi$ implies $\pi_i \models m(\psi)$ and $\pi_j \models \xi$ implies $\pi_j \models m(\xi)$. Also, $i \leq 2^{|\varphi|}$ is true because of $|\pi| \leq 2^{|\varphi|}$. As a result, $\pi \models \eta$ implies that there is $0 \leq i \leq 2^{|\varphi|}$ such that $\pi_i \models m(\psi)$ and for every $0 \leq j < i$, $\pi_j \models m(\xi)$ is true. According to the MLTL semantics, $\pi \models m(\eta)$ is true. ($\Leftarrow$) $m(\eta)$ is satisfiable implies there is a finite trace π such that $\pi \models m(\eta)$. According to MLTL semantics, there is $0 \leq i \leq 2^{|\varphi|}$ such that $\pi_i \models m(\psi)$ and for every $0 \leq j < i$ it holds that $\pi_j \models m(\xi)$. By hypothesis assumption, $\pi_i \models m(\psi)$ implies $\pi_i \models \psi$ and $\pi_j \models m(\xi)$ implies $\pi_j \models \xi$. Also, $0 \leq i \leq 2^{|\varphi|}$ implies $0 \leq i$. As a result, $\pi \models m(\eta)$ implies that there is $0 \leq i$ such that $\pi_i \models \psi$ and for every $0 \leq j < i$ it holds that $\pi_j \models \xi$. From LTL_f semantics, it is true that $\pi \models \eta$. □

Theorem 2. *The complexity of checking the satisfiability of* MLTL_0 *is PSPACE-complete.*

Proof. Since Lemma 3 shows a linear reduction from $\mathcal{X}$-free LTL_f-SAT to MLTL_0-SAT and $\mathcal{X}$-free LTL_f-SAT is PSPACE-complete [14], it directly implies that the lower bound of MLTL_0-SAT is PSPACE-hard.

For the upper bound, recall from the proof of Theorem 1 that an MLTL formula φ is translated to an LTL_f formula θ of length $K \cdot |cl(\varphi)|$, which, as we commented, involved an exponential blow-up in the notation for K. Following the automata-theoretic approach for satisfiability, one would translate θ to an NFA and check its non-emptiness [14]. Normally, such a translation would involve another exponential blow-up. We show that this is not the case for MLTL_0. Recalling from the automaton construction in [14] that every state of the automaton is a set of subformulas of θ, the size of a state is at most $K \cdot |cl(\varphi)|$. In the general case, if ψ_1, ψ_2 are two subformulas of θ corresponding to the MLTL formulas $\xi\, \mathcal{U}_{I_1}\, \psi$ and $\xi\, \mathcal{U}_{I_2}\, \psi$, ψ_1 and ψ_2 can be in the same state of the automaton, which implies that the size of the state can be at most $K \cdot |cl(\varphi)|$. When the formula φ is restricted to MLTL_0, we show that the exponential blow-up can be avoided. Lemma 2 shows that either ψ_1 or ψ_2 in the state is enough, since assuming $I_1 \subseteq I_2$, then $(\psi_1 \wedge \psi_2) \equiv \psi_1$, by Lemma 2. So the size of the state in the automaton for a MLTL_0 formula φ is at most $|cl(\varphi)|$. For each subformula in the state, there can be K possible values (e.g., for $\Diamond_I \xi$ in the state, we can have $\Diamond_{[0,1]}\xi$, $\Diamond_{[0,2]}\xi$, etc.). Therefore the size of the automaton is in $O(2^{|cl(\varphi)|} \cdot K^{|cl(\varphi)|}) \approx 2^{O(|cl(\varphi)|)}$. Therefore, MLTL_0 satisfiability checking is a PSPACE-complete problem. □

4 Implementation of MLTL-SAT

We first show how to reduce MLTL-SAT to the well-explored LTL_f-SAT and LTL-SAT. Then we introduce two new satisfiability-checking strategies based on the inherent properties of MLTL formulas, which are able to leverage the state-of-art model-checking and SMT-solving techniques.

4.1 MLTL-SAT via Logic Translation

For a formula φ from one logic, and ψ from another logic, we say φ and ψ are *equi-satisfiable* when φ is satisfiable under its semantics iff ψ is satisfiable under its semantics. Based on Lemma 1 and Theorem 1, we have the following corollary,

Corollary 1 (MLTL-SAT to LTL_f-SAT). MLTL-SAT *can be reduced to* LTL_f-SAT *with an exponential blow-up.*

From Corollary 1, MLTL-SAT is reducible to LTL_f-SAT, enabling use of the off-the-shelf LTL_f satisfiability solvers, cf. aaltaf [23]. It is also straightforward to consider MLTL-SAT via LTL-SAT; LTL-SAT has been studied for more than a decade, and there many off-the-shelf LTL solvers are available, cf. [24,38,40].

Theorem 3 (MLTL **to** LTL)**.** *For an* MLTL *formula* φ*, there is an* LTL *formula* θ *such that* φ *and* θ *are equi-satisfiable, and the size of* θ *is in* $O(K \cdot |cl(\varphi)|)$*, where* K *is the maximal integer in* φ*.*

Proof. Lemma 1 provides a translation from the MLTL formula φ to the equivalent LTL_f formula φ', with a blow-up of $O(K \cdot |cl(\varphi)|)$. As shown in Sect. 2, there is a linear translation from the LTL_f formula φ' to its equi-satisfiable LTL formula θ [14]. Therefore, the blow-up from φ to θ is in $O(K \cdot |cl(\varphi)|)$. □

Corollary 2 (MLTL-SAT **to** LTL-SAT)**.** MLTL-SAT *can be reduced to* LTL-SAT *with an exponential blow-up.*

Since MLTL-SAT is reducible to LTL-SAT, MLTL-SAT can also benefit from the power of LTL satisfiability solvers. Moreover, the reduction from MLTL-SAT to LTL-SAT enables leveraging modern model-checking techniques to solve the MLTL-SAT problem, due to the fact that LTL-SAT has been shown to be reducible to model checking with a linear blow-up [38,39].

Corollary 3 (MLTL-SAT **to** LTL-**Model-checking**)**.** MLTL-SAT *can be reduced to* LTL *model checking with an exponential blow-up.*

In our implementation, we choose the model checker nuXmv [12] for LTL satisfiability checking, as it allows an LTL formula to be directly input as the temporal specification together with a universal model as described in [38,39].

4.2 Model Generation

Using the LTL formula as the temporal specification in nuXmv has been shown, however, to not be the most efficient way to use model checking for satisfiability checking [40]. Consider the MLTL formula $\Diamond_{[0,10]} a \wedge \Diamond_{[1,11]} a$. The translated LTL_f formula is $f(\Diamond_{[0,10]} a) \wedge \mathcal{X}(f(\Diamond_{[0,10]} a))$, where $f(\Diamond_{[0,10]} a)$ has to be constructed twice. To avoid such redundant construction, we follow [40] and encode directly the input MLTL formula as an SMV model (the input model of nuXmv) rather than treating the LTL formula, which is obtained from the input MLTL formula, as a specification.

An SMV [27] model consists of a Boolean transition system $Sys = (V, I, T)$, where V is a set of Boolean variables, I is a Boolean formula representing the initial states of Sys, and T is the Boolean transition formula. Moreover, a specification to be verified against the system is also contained in the SMV model (here we focus on the LTL specification). Given the input MLTL formula φ, we construct the corresponding SMV model M_φ as follows.

- Introduce a Boolean variable for each atom in φ as well as for "$Tail$" (new variable identifying the end of a finite trace).
- Introduce a Boolean variable $\mathcal{X}\_\psi$ for each $\mathcal{U}$ formula ψ in $cl^*(\varphi)$, which represents the intermediate temporal formula $\mathcal{X}\psi$.
- Introduce a temporary Boolean variable[2] T_ψ for each $\mathcal{U}$ formula in $cl^*(\varphi)$.

[2] A temporary variable is introduced in the DEFINE statement rather than the VAR statement of the SMV model, as it will be automatically replaced with those in VAR statements.

- A Boolean formula $e(\psi)$ is used to represent the formula ψ in $cl^*(\varphi)$ in the SMV model, which is defined recursively as follows.
  1. $e(\psi) = \psi$, if ψ is an Boolean atom;
  2. $e(\psi) = \neg e(\psi_1)$, if $\psi = \neg\psi_1$;
  3. $e(\psi) = e(\psi_1) \wedge e(\psi_2)$, if $\psi = \psi_1 \wedge \psi_2$;
  4. $e(\psi) = T\_\psi$, if ψ is an $\mathcal{U}$ formula.
- Let the initial Boolean formula of the system Sys be $e(\varphi)$.
- For each temporary variable T_ψ, create a DEFINE statement according to the type and interval of ψ, as follows.

$$T_{\psi_1\mathcal{U}_{[a,b]}\psi_2} = \begin{cases} \mathcal{X}\_(\psi_1\mathcal{U}_{[a-1,b-1]}\psi_2), & \text{if } 0 < a \leq b; \\ e(\psi_2) \vee (e(\psi_1) \wedge \mathcal{X}\_(\psi_1\mathcal{U}_{[0,b-1]}\psi_2)), & \text{if } a = 0 \text{ and } 0 < b; \\ e(\psi_2), & \text{if } a = 0 \text{ and } b = 0. \end{cases}$$

- Create the Boolean formula $(\mathcal{X}\_\psi \leftrightarrow (\neg Tail \wedge next(e(\psi))))$ for each $\mathcal{X}\_\psi$ in the VAR list (the set V in Sys) of the SMV model.
- Finally, designate the LTL formula $\Box\neg Tail$ as the temporal specification of the SMV model M_φ (which implies that a counterexample trace satisfies $\Diamond Tail$).

Encoding Heuristics for MLTL_0 Formulas. We also encode the rules shown in Lemma 2 to prune the state space for checking the satisfiability of MLTL_0 formulas. These rules are encoded using the INVAR constraint in the SMV model. Taking the $\mathcal{U}$ formula as an example, we encode $T\_(\psi_1\mathcal{U}_{[0,a]}\psi_2) \wedge T\_(\psi_1\mathcal{U}_{[0,a-1]}\psi_2) \leftrightarrow T\_(\psi_1\mathcal{U}_{[0,a-1]}\psi_2)$ $(a > 0)$ for each $\psi_1\mathcal{U}_{[0,a]}\psi_2$ in $cl^*(\varphi)$. Similar encodings also apply to the $\mathcal{R}$ formulas in $cl^*(\varphi)$. Theorem 4 below guarantees the correctness of the translation, and it can be proved by induction over the type of φ and the construction of the SMV model.

Theorem 4. *The* MLTL *formula φ is satisfiable iff the corresponding* SMV *model M_φ violates the* LTL *property $\Box\neg Tail$.*

There are different techniques that can be used for LTL model checking. Based on the latest evaluation of LTL satisfiability checking [24], the KLIVE [13] back-end implemented in the SMV model checker nuXmv [12] produces the best performance. We thus choose KLIVE as our model-checking technique for MLTL-SAT.

Bounded MLTL-SAT. Although MLTL-SAT is reducible to the satisfiability problem of other well-explored logics, with established off-the-shelf satisfiability solvers, a dedicated solution based on inherent properties of MLTL may be superior. One intuition is, since all intervals in MLTL formulas are bounded, the satisfiability of the formula can be reduced to Bounded Model Checking (BMC) [9].

Theorem 5. *Given an* MLTL *formula φ with K as the largest natural in the intervals of φ, φ is satisfiable iff there is a finite trace π with $|\pi| \leq K \cdot |cl(\varphi)|$ such that $\pi \models \varphi$.*

Theorem 5 states that the satisfiability of a given MLTL formula can be reduced to checking for the existence of a satisfying trace. To apply the BMC technique in nuXmv, we compute and set the maximal depth of BMC to be the value of $K \cdot |cl(\varphi)|$ for a given MLTL formula φ. The input SMV model for BMC is still M_φ, as described in Sect. 4.2.

However to ensure correct BMC checking in nuXmv, the constraint "FAIRNESS TRUE" has to be added into the SMV model.[3] The `LTLSPEC` remains $\square\neg Tail$. According to Theorem 5, φ is satisfiable iff the model checker returns a counterexample by using the BMC technique within the maximal depth of $K \cdot |cl(\varphi)|$.

4.3 MLTL-SAT via SMT Solving

Another approach to solve MLTL-SAT is via SMT solving, considering that using SMT solvers to handle intervals in MLTL formulas is straightforward. Since the input logic of SMT solvers is First-Order Logic, we must first translate the MLTL formula to its equi-satisfiable formula in First-Order Logic over the natural domain N. We assume that readers are familiar with First-Order Logic and only focus on the translation. Given an MLTL formula φ and the alphabet Σ, we construct the corresponding formula in First-Order Logic over N in the following way.

1. For each $p \in \Sigma$, define a corresponding function $f_p : Int \rightarrow Bool$ such that $f_p(k)$ is true ($k \in N$) iff there is a satisfying (finite) trace π of φ and p is in $\pi[k]$.
2. The First-Order Logic formula $\mathsf{fol}(\varphi, k, len)$ for φ ($k, len \in N$) is constructed recursively as below:
   - $\mathsf{fol}(\mathsf{true}, k, len) = (len > k)$ and $\mathsf{fol}(\mathsf{false}, k, len) = false$;
   - $\mathsf{fol}(p, k, len) = (len > k) \wedge f_p(k)$ for $p \in \Sigma$;
   - $\mathsf{fol}(\neg\xi, k, len) = (len > k) \wedge \neg\mathsf{fol}(\xi, k, len)$;
   - $\mathsf{fol}(\xi \wedge \psi, k, len) = (len > k) \wedge \mathsf{fol}(\xi, k, len) \wedge \mathsf{fol}(\psi, k, len)$;
   - $\mathsf{fol}(\xi\, \mathcal{U}_{[a,b]}\, \psi, k, len) = (len > a+k) \wedge \exists i.(\ (a+k \leq i \leq b+k) \wedge \mathsf{fol}(\psi, i, len-i) \wedge \forall j.(\ (a+k \leq j < i) \rightarrow \mathsf{fol}(\xi, j, len-j)))$;

In the formula $\mathsf{fol}(\varphi, k, len)$, k represents the index of the (finite) trace from which φ is evaluated, and len indicates the length of the suffix of the trace starting from the index k. Since the formula is constructed recursively, we need to introduce k to record the index. Meanwhile, len is necessary because the MLTL semantics, which is interpreted over finite traces, constrains the lengths of the satisfying traces of the Until formulas. The following theorem guarantees that MLTL-SAT is reducible to the satisfiability of First-Order Logic.

Theorem 6. *For an* MLTL *formula φ, φ is satisfiable iff the corresponding First-Order Logic formula $\exists len.\mathsf{fol}(\varphi, 0, len)$ is satisfiable.*

Proof. Let the alphabet of φ be Σ, and $\pi \in (2^{\Sigma})^*$ be a finite trace. For each $p \in \Sigma$, we define the function $f_p : Int \rightarrow Bool$ as follows: $f_p(k) = \mathsf{true}$ iff $p \in \pi[k]$ if $0 \leq k < |\pi|$. We now prove by induction over the type of φ and the construction of $\mathsf{fol}(\varphi, k, len)$ with respect to φ that $\pi_k \models \varphi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\varphi, k, |\pi|)$: here $|\pi|$ is the length of π. The cases when φ is true or false are trivial.

- If $\varphi = p$ is an atom, $\pi_k \models \varphi$ holds iff $p \in \pi[k]$ (i.e., $\pi_k[0]$) is true, which means $f_p(k) = \mathsf{true}$. As a result, $\{f_p\}$ is a model of $\mathsf{fol}(\varphi, k, |\pi|)$, which implies that $\pi_k \models \varphi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\varphi, k, |\pi|)$.

[3] Based on comments in emails from the nuXmv developers.

- If $\varphi = \neg\xi$, $\pi_k \models \varphi$ holds iff $\pi_k \not\models \xi$ holds. By hypothesis assumption, $\pi_k \models \xi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\xi, k, |\pi|)$, which is equivalent to saying $\pi_k \not\models \xi$ holds iff $\{f_p | p \in \Sigma\}$ is not a model of $\mathsf{fol}(\xi, k, |\pi|)$. As a result, $\pi_k \models \neg\xi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\neg\mathsf{fol}(\xi, k, |\pi|)$.
- If $\varphi = \xi \wedge \psi$, $\pi_k \models \varphi$ holds iff $\pi_k \models \xi$ and $\pi_k \models \psi$. By hypothesis assumption, $\pi_k \models \xi$ (resp. $\pi_k \models \psi$) holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\xi, k, |\pi|)$ (resp. $\mathsf{fol}(\psi, k, |\pi|)$). According to the construction of the fol function, $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\xi \wedge \psi, k, |\pi|)$. As a result, $\pi_k \models \xi \wedge \psi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\xi \wedge \psi, k, |\pi|)$.
- If $\varphi = \xi\, \mathcal{U}_{[a,b]}\, \psi$, $\pi_k \models \varphi$ holds iff there is $a + k \leq i \leq b + k$ such that $\pi_i \models \psi$ and $\pi_j \models \xi$ holds for every $a + k \leq j < i$. By hypothesis assumption, $\pi_i \models \psi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\psi, i, len - i)$ (the length of π_i is $len - i$), and $\pi, j \models \xi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\xi, j, |\pi| - j)$ (the length of π_j is $|\pi| - j$). Moreover, $|\pi| > a + k$ must be true according to the MLTL semantics. As a result, $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\varphi, k, |\pi|)$, which implies that $\pi_k \models \xi\, \mathcal{U}_{[a,b]} \psi$ holds iff $\{f_p | p \in \Sigma\}$ is a model of $\mathsf{fol}(\xi\, \mathcal{U}_{[a,b]}\, \psi, k, |\pi|)$.

This proof holds for all values of k, including the special case where $k = 0$. □

We then encode $\exists len.\mathsf{fol}(\varphi, 0, len)$ into the SMT-LIB v2 format [7], which is the input of most modern SMT solvers; we call the full SMT-LIB v2 encoding $\mathsf{SMT}(\varphi)$. We first use the "declare-fun" command to declare a function $f_a : Int \rightarrow Bool$ for each $p \in \Sigma$. We also define the function $f_\varphi : Int \times Int \rightarrow Bool$ for the First-Order Logic formula $\mathsf{fol}(\varphi, k, len)$. The corresponding SMT-LIB v2 command is "define-fun f_φ $((k\ Int)\ (len\ Int))\ Bool\ S(\mathsf{fol}(\varphi, k, len))$", where $S(\mathsf{fol}(\varphi, k, len))$ is the SMT-LIB v2 implementation of $\mathsf{fol}(\varphi, k, len)$. In detail, $S(\mathsf{fol}(\varphi, k, len))$ is acquired recursively as follows.

- $S(\mathsf{fol}(p, k, len)) \longrightarrow$ (and $(>\ len\ k)$ $(f_p\ k)$)
- $S(\neg\mathsf{fol}(\varphi, k, len)) \longrightarrow$ (and $(>\ len\ k)$ (not $S(\mathsf{fol}(\varphi, k))$))
- $S(\mathsf{fol}(\varphi_1 \wedge \psi, k, len) \longrightarrow$ (and $(>\ len\ k)$ (and $S(\mathsf{fol}(\varphi_1, k, len))$ $S(\mathsf{fol}(\psi, k, len))$))
- $S(\mathsf{fol}(\varphi_1\, \mathcal{U}_{[a,b]}\, \psi, k, len)) \longrightarrow$ (and $(>\ len\ a{+}k)$ (exists $(i\ Int)$ (and $(\leq\ (+\ a\ k)\ i)$ $(\geq\ i\ (+\ b\ k))$ $S(\mathsf{fol}(\psi, i, len - i))$ (forall $(j\ Int)$ $(\Rightarrow$ (and $(\leq\ (+\ a\ k)\ j)$ $(<\ j\ i))$ $S(\mathsf{fol}(\varphi_1, j, len - j)))))))$

Finally, we use the "assert" command "(assert (exists $((len\ Int))$ $(f_\varphi\ 0\ len)$))" together with the "(check-sat)" command to request SMT solvers for the satisfiability of $\exists len.\mathsf{fol}(\varphi, 0, len)$. In a nutshell, the general framework of the SMT-LIB v2 format for $\mathsf{SMT}(\varphi)$ (i.e., $\exists len.\mathsf{fol}(\varphi, 0, len)$) is shown in Table 1, and the correctness is guaranteed by Theorem 7 below.

Table 1. The SMT-LIB v2 template for $\mathsf{SMT}(\varphi)$.

(declare-fun f_a (Int) Bool) //declare corresponding function for $a \in \Sigma$
...
//define function for $\mathsf{fol}(\varphi, k, len)$
(define-fun f_φ ((k Int) (len Int)) Bool $S(\mathsf{fol}(\varphi, k, len))$)
(assert (exists ((len Int)) $(f_\varphi\ 0)$))
(check-sat)

Theorem 7. *The First-Order Logic formula* $\exists len.fol(\varphi, 0, len)$ *is satisfiable iff the* SMT *solver returns SAT with the input* $\mathsf{SMT}(\varphi)$.

An inductive proof for the theorem can be conducted according to the construction of $\mathsf{SMT}(\varphi)$. Notably, there is no difference between the SMT encoding for MLTL formulas and that for MLTL_0 formulas, as the SMT-based encoding does not require unrolling the temporal operators in the formula.

5 Experimental Evaluations

Tools and Platform. We implemented the translator MLTLconverter in C++, including encodings for an MLTL formula as equi-satisfiable LTL and LTL_f formulas, and corresponding SMV and SMT-LIB v2 models. We leverage the extant LTL solver aalta [24], LTL_f solver aaltaf [23], SMV model checker nuXmv [12], and the SMT solver Z3 [29] to check the satisfiability of the input MLTL formula in their respective encodings from MLTLconverter. The solvers, including the runtime flags we used, are summarized in Table 2. We evaluated both BMC and KLIVE [13] model-checking back-ends in nuXmv, and the corresponding commands are shown in Fig. 1. Notably in the figure, the maximal length "*MAX*" to run BMC is computed dynamically for each MLTL formula, based on Theorem 5.

Table 2. List of solvers and their runtime flags.

Encoding	MLTLconverter flag	Solver	Solver flag
LTL	-ltl	aalta	default
LTL_f	-ltlf	aaltaf	default
SMV	-smv	nuXmv	-source bmc.cmd (BMC)
			-source klive.cmd (KLIVE)
SMT-LIB v2	-smtlib	Z3	-smt2

```
read_model
flatten_hierarchy
encode_variables
build_boolean_model
bmc_setup
go_bmc
check_ltlspec_bmc -k MAX
quit
```

```
read_model
flatten_hierarchy
encode_variables
build_boolean_model
check_ltlspec_klive -d
quit
```

Fig. 1. nuXmv commands for BMC (left) and KLIVE (right).

All experiments were executed on Rice University's NOTS cluster,[4] running RedHat 5, with 226 dual socket compute blades housed within HPE s6500, HPE Apollo 2000, and Dell PowerEdge C6400 chassis. All the nodes are interconnected with 10 GigE network. Each satisfiability check over one MLTL formula and one solver was executed with exclusive access to one CPU and 8 GB RAM with a timeout of one hour, as measured by the Linux `time` command. We assigned a time penalty of one hour to benchmarks that segmentation fault or timeout.

Experimental Goals. We evaluate performance along three metrics. (1) Each satisfiability check has two parts: the encoding time (consumed by MLTLconverter) and the solving time (consumed by solvers). We evaluate how each encoding affects the performance of both stages of MLTL-SAT. (2) We comparatively analyze the performance and scalability of end-to-end MLTL-SAT via LTL-SAT, LTL$_f$-SAT, LTL model checking, and our new SMT-based approach. (3) We evaluate the performance and scalability for MLTL$_0$ satisfiability checking using MLTL$_0$-SAT encoding heuristics (Lemma 2).

Benchmarks. There are few MLTL (or even MTL-over-naturals) benchmarks available for evaluation. Previous works on MTL-over-naturals [2–4] mainly focus on the theoretic exploration of the logic. To enable rigorous experimental evaluation, we develop three types of benchmarks, motivated by the generation of LTL benchmarks [38].[5]

(1) *Random* MLTL *Formulas (*R*)*: We generated 10,000 R formulas, varying the formula length L (20, 40, 60, 80, 100), the number of variables N (1, 2, 3, 4, 5), and the probability of the appearance of the $\mathcal{U}$ operator P (0.33, 0,5, 0.7, 0.95); for each (L, N, P) we generated 100 formulas. For every $\mathcal{U}$ operator, we randomly chose an interval $[i, j]$ where $i \geq 0$ and $j \leq 100$.

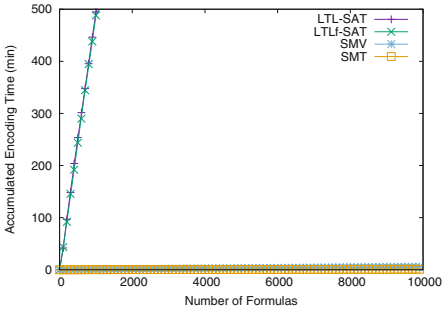

Fig. 2. Cactus plot for different MLTL encodings on R formulas: LTL-SAT and LTL$_f$-SAT lines overlap; SMV and SMT lines overlap.

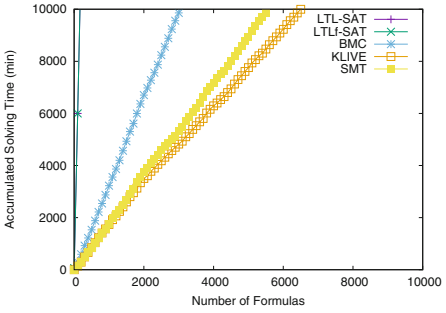

Fig. 3. Cactus plot for different MLTL solving approaches on R formulas: LTL-SAT and LTL$_f$-SAT lines overlap.

[4] https://docs.rice.edu/confluence/display/CD/NOTS+Overview.

[5] All experimental materials are at http://temporallogic.org/research/CAV19/. The plots are best viewed online.

(2) *NASA-Boeing* MLTL *Formulas (*NB*)*: We use challenging benchmarks [15] created from projects at NASA [17,26] and Boeing [11]. We extract 63 real-life LTL requirements from the SMV models of the benchmarks, and then randomly generate an interval for each temporal operator. (We replace each $\mathcal{X}$ with $\Box_{[1,1]}$.) We create 3 groups of such formulas (63 in each) to test the scalability of different approaches, by restricting the maximal number of the intervals to be 1,000, 10,000, and 100,000 respectively.

(3) *Random* MLTL_0 *Formulas (*R0*)*: We generated 500 R0 formulas in the same way as the R formulas, except that every generated interval was restricted to start from 0; we generated sets of five for each (L, N, P). This small set of R benchmarks serve to compare the performance on MLTL_0 formulas whose SMV encodings were created with/without heuristics.

Correctness Checking. We compared the verdicts from all solvers for every test instance and found no inconsistencies, excluding segmentation faults. This exercise aided with verification of our implementations of the translators, including diagnosing the need for including FAIRNESS TRUE in BMC models.

Experimental Results. Figure 2 compares encoding times for the R benchmark formulas. We find that (1) Encoding MLTL as either LTL and LTL_f is not scalable even when the intervals in the formula are small; (2) The cost of MLTL-to-SMV encoding is comparable to that from MLTL to SMT-LIB v2. Although the cost of encoding MLTL as LTL/LTL_f and SMV are in $O(K \cdot |cl(\varphi)|)$, where K is the maximal interval length in φ, the practical gap between the LTL/LTL_f encodings and SMV encoding affirms our conjecture that the SMV model is more compact in general than the corresponding LTL/LTL_f formulas. Also because K is kept small in the R formulas, the encoding cost between SMV and SMT-LIB v2 becomes comparable.

Figure 3 shows total satisfiability checking times for R benchmarks. Recall that the inputs of both BMC and KLIVE approaches are SMV models. The MLTL-SAT via KLIVE is the fastest solving strategy for MLTL formulas with interval ranges of less than 100. The portion of satisfiable/unsatisfiable formulas of this benchmark is approximate 4/1. Although BMC is known to be good at detecting counterexamples with short lengths, it does not perform as well as the KLIVE and SMT approaches on checking satisfiable formulas since only longer counterexamples (with length greater than 1000) exist for most of these formulas. While nuXmv successfully checked all such models, Fig. 4 shows that increasing the interval range constraint results in segmentation faults; more than half of our benchmarks produced this outcome for formulas with allowed interval ranges of up to 600. Meanwhile, the solving solutions via LTL-SAT/LTL_f-SAT are definitely not competitive for any interval range.

The SMT-based approach dominates the model-checking-approaches when considering scalable NB benchmarks, as shown in Fig. 5. Here, e.g., "BMC-1000" means using BMC to check the group of benchmarks with a maximal interval range of 1,000. Due to segmentation faults, "BMC-1000" and "KLIVE-1000" have almost the same performance because the SMV models generated from our translator MLTLconverter are too large for nuXmv to handle. The performance of the model-checking approaches is constrained by the scalability of the model checker (nuXmv). However, the SMT encoding does not face such a bottleneck; see "Z3-1000," "Z3-10000," and "Z3-100000" in Fig. 5. We conclude that the SMT approach is the best available strategy for MLTL satisfiability checking.

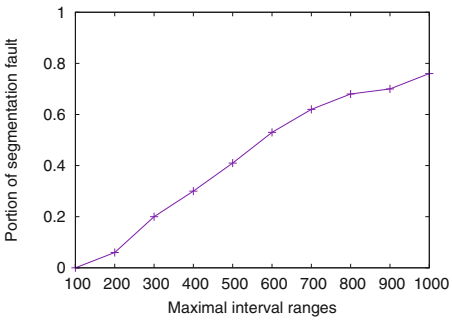

Fig. 4. Proportion of segmentation faults for sets of 200 R formulas with maximal interval ranges varying from 100 to 1000.

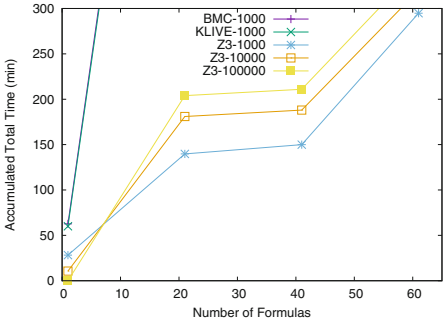

Fig. 5. Cactus plot for BMC,KLIVE and SMT-solving approaches on the NB benchmarks; BMC and KLIVE overlap.

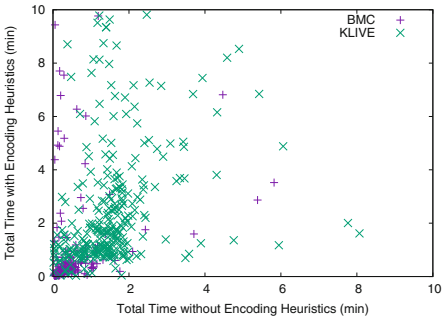

Fig. 6. Scatter plot for both the BMC and KLIVE approaches to checking MLTL_0 formulas ith/without encoding heuristics.

Finally, we evaluated the performance of model-checking-based approaches on the R0 formulas, observing that there is an exponential complexity gap between MLTL-SAT and MLTL_0-SAT. Figure 6 compares the performance of satisfiability solving via the BMC and KLIVE approaches. There is no significant improvement when the SMV encoding heuristics for MLTL_0 are applied. For the BMC solving approach, performance is largely unaffected by encoding heuristics. For the KLIVE solving approach, encoding heuristics decrease solving performance. The results support the well-known phenomenon that the theoretic analysis and the practical evaluations do not always match.

We summarize with three conclusions. (1) For satisfiability checking of MLTL formulas, the new SMT-based approach is best. (2) For satisfiability checking of MLTL formulas with interval ranges less than 100, the MLTL-SAT via KLIVE approach is fastest. (3) The dedicated encoding heuristics for MLTL_0 do not significantly improve the satisfiability checking time of MLTL_0-SAT over MLTL-SAT. They do not solve the nuXmv scalability problem.

6 Discussion and Conclusion

Metric Temporal Logic (MTL) was first introduced in [3], for describing continuous behaviors interpreted over infinite real-time traces. The later variants Metric Interval Temporal Logic (MITL) [5], and Bounded Metric Temporal Logic (BMTL) [30] are also interpreted over infinite traces. Intuitively, MLTL is a combination of MITL and BMTL that allows only bounded, discrete (over natural domain) intervals that are interpreted over finite traces. There are several previous works on the satisfiability of MITL, though their tools only support the infinite semantics. Bounded satisfiability checking for MITL formulas is proposed in [33], and the reduction from MITL to LTL is presented in [20]. Since previous works focus on MITL over infinite traces and there is no trivial way to reduce MLTL over finite traces to MITL over infinite traces, the previous methodologies are not comparable to those presented in this paper. This includes the SMT-based solution of reducing MITL formulas to equi-satisfiable Constraint LTL formulas [8]. Compared to that, our new SMT-based approach more directly encodes MLTL formulas into the SMT language without translation through an intermediate language.

The contribution of a complete, correct, and open-source MLTL satisfiability checking algorithm and tool opens up avenues for a myriad of future directions, as we have now made possible specification debugging MLTL formulas in design-time verification and benchmark generation for runtime verification. We plan to explore alternative encodings for improving the performance of MLTL satisfiability checking and work toward developing an optimized multi-encoding approach, following the style of the previous study for LTL [40]; the current SMT model generated from the MLTL formula uses a relatively simple theory (uninterpreted functions). We also plan to explore lazy encodings from MLTL formulas to SMT models. For example, instead of encoding the whole MLTL formula into a monolithic SMT model, we may be able to decrease overall satisfiability-solving time by encoding the MLTL formula in parts with dynamic ordering similar to [15]. To make the output of SMT-based MLTL satisfiability checking more usable, we plan to investigate translations from the functions returned from Z3 for satisfiable instances into more easily parsable satisfying assignments.

Acknowledgment. We thank anonymous reviewers for their helpful comments. This work is supported by NASA ECF NNX16AR57G, NSF CAREER Award CNS-1552934, NSF grants IIS-1527668, IIS-1830549, and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering."

References

1. Runtime Verification Benchmark Competition (2018). https://www.rv-competition.org/2018-2/
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996)
3. Alur, R., Henzinger, T.A.: Real-time logics: complexity and expressiveness. In: LICS, pp. 390–401. IEEE (1990)
4. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM **41**(1), 181–204 (1994)
5. Alur, R., Henzinger, T.A.: Reactive modules. In: Proceedings of the 11th IEEE Symposium on Logic in Computer Science, pp. 207–218 (1996)
6. Armoni, R., Fix, L., Flaisher, A., Grumberg, O., Piterman, N., Vardi, M.Y.: Enhanced vacuity detection in linear temporal logic. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 368–380. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_35
7. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: version 2.0. In: Workshop on Satisfiability Modulo Theories (2010)
8. Bersani, M., Rossi, M., San Pietro, P.: An SMT-based approach to satisfiability checking of MITL. Inf. Comput. **245**(C), 72–97 (2015)
9. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49059-0_14
10. Bloem, R., Chockler, H., Ebrahimi, M., Strichman, O.: Synthesizing non-vacuous systems. In: Bouajjani, A., Monniaux, D. (eds.) VMCAI 2017. LNCS, vol. 10145, pp. 55–72. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52234-0_4
11. Bozzano, M., et al.: Formal design and safety analysis of AIR6110 wheel brake system. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015, Part I. LNCS, vol. 9206, pp. 518–535. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_36
12. Cavada, R., et al.: The NUXMV symbolic model checker. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 334–342. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_22
13. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: FMCAD, pp. 52–59. IEEE (2012)
14. De Giacomo, G., Vardi, M.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp. 2000–2007. AAAI Press (2013)
15. Dureja, R., Rozier, K.Y.: More scalable LTL model checking via discovering design-space dependencies (D^3). In: Beyer, D., Huisman, M. (eds.) TACAS 2018, Part I. LNCS, vol. 10805, pp. 309–327. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_17
16. Furia, C.A., Spoletini, P.: Tomorrow and all our yesterdays: MTL satisfiability over the integers. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 126–140. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85762-4_9
17. Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., Rozier, K.Y.: Model checking at scale: automated air traffic control design space exploration. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016, Part II. LNCS, vol. 9780, pp. 3–22. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_1
18. Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 215–230. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_18

19. De Giacomo, G., Vardi, M.: Synthesis for LTL and LDL on finite traces. In: IJCAI, pp. 1558–1564 (2015)
20. Hustadt, U., Ozaki, A., Dixon, C.: Theorem proving for metric temporal logic over the naturals. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 326–343. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_20
21. Kessler, F.B.: nuXmv 1.1.0 (2016-05-10) Release Notes (2016). https://es-static.fbk.eu/tools/nuxmv/downloads/NEWS.txt
22. Li, J., Rozier, K.Y.: MLTL benchmark generation via formula progression. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 426–433. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_25
23. Li, J., Zhang, L., Pu, G., Vardi, M.Y., He, J.: LTL_f satisfiability checking. In: ECAI, pp. 91–98 (2014)
24. Li, J., Zhu, S., Pu, G., Vardi, M.Y.: SAT-based explicit LTL reasoning. In: Piterman, N. (ed.) HVC 2015. LNCS, vol. 9434, pp. 209–224. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26287-1_13
25. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
26. Mattarei, C., Cimatti, A., Gario, M., Tonetta, S., Rozier, K.Y.: Comparing different functional allocations in automated air traffic control design. In: Proceedings of Formal Methods in Computer-Aided Design (FMCAD 2015), Austin, Texas, USA. IEEE/ACM, September 2015
27. McMillan, K.: Symbolic model checking: an approach to the state explosion problem. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1992). UMI Order No. GAX92-24209
28. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In: FMSD, pp. 1–31, April 2017
29. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
30. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 1–13. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85778-5_1
31. Pandya, P.K., Shah, S.S.: The unary fragments of metric interval temporal logic: bounded versus lower bound constraints. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 77–91. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_8
32. Pnueli, A.: The temporal logic of programs. In: IEEE FOCS, pp. 46–57 (1977)
33. Pradella, M., Morzenti, A., Pietro, P.: Bounded satisfiability checking of metric temporal logic specifications. ACM Trans. Softw. Eng. Methodol. **22**(3), 20:1–20:54 (2013)
34. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 357–372. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_24
35. Rozier, K.Y.: Specification: the biggest bottleneck in formal methods and autonomy. In: Blazy, S., Chechik, M. (eds.) VSTTE 2016. LNCS, vol. 9971, pp. 8–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48869-1_2
36. Rozier, K.Y.: On the evaluation and comparison of runtime verification tools for hardware and cyber-physical systems. In: RV-CUBES, vol. 3, pp. 123–137. Kalpa Publications (2017)
37. Rozier, K.Y., Schumann, J., Ippolito, C.: Intelligent hardware-enabled sensor and software safety and health management for autonomous UAS. Technical Memorandum NASA/TM-2015-218817, NASA Ames Research Center, Moffett Field, CA 94035, May 2015

38. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 149–167. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73370-6_11
39. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. Int. J. Softw. Tools Technol. Transf. **12**(2), 123–137 (2010)
40. Rozier, K.Y., Vardi, M.Y.: A multi-encoding approach for LTL symbolic satisfiability checking. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 417–431. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21437-0_31
41. Rozier, K.Y., Vardi, M.Y.: Deterministic compilation of temporal safety properties in explicit state model checking. In: Biere, A., Nahir, A., Vos, T. (eds.) HVC 2012. LNCS, vol. 7857, pp. 243–259. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39611-3_23
42. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 233–249. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_15
43. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime analysis with R2U2: a tool exhibition report. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 504–509. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_35
44. Schumann, J., Rozier, K.Y., Reinbacher, T., Mengshoel, O.J., Mbaya, T., Ippolito, C.: Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. IJPHM **6**(1), 1–27 (2015)
45. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logic. J. ACM **32**, 733–749 (1985)