Maximum Satisfiability of Mission-time Linear Temporal Logic

Gokul Hariharan[0000−0002−3447−2183], Phillip H. Jones[0000−0002−8220−7552], Kristin Y. Rozier[0000−0002−6718−2828], and Tichakorn Wongpiromsarn[0000−0002−3977−122X]

Iowa State University, Ames, IA, 50010, USA
{gokul,phjones,kyrozier,nok}@iastate.edu

Abstract. Mission-time Linear Temporal Logic (MLTL) is a variant of Linear Temporal Logic (LTL) with finite interval bounds on temporal operators, and is a popular formal specification language for safety-critical cyber-physical systems. Given a set of specifications, the maximum satisfiability problem (MaxSAT) asks to find the maximum number of simultaneously satisfiable specifications. MaxSAT is useful for system design and feature prioritization that are integral to designing complex systems. Considering the significant advances in MaxSAT for Boolean logic, we develop translations from MLTL to Boolean logic to solve the MLTL MaxSAT problem. Given an MLTL formula φ of length $|\varphi|$ with maximum interval length m, our first translator runs in $\mathcal{O}(m^{|\varphi|})$ time. Our second, improved translator runs in $\mathcal{O}(|\varphi|^2 m^2)$ time. Performance tests of satisfiability checks on MaxSAT instances illustrate that these Boolean translations perform significantly better than the best satisfiability checking approaches reported recently in the literature on real and random instances. Furthermore, the second translator is embarrassingly parallelizable to a factor of $|\varphi| m$. We contribute to (1) an easy-to-implement translation from MLTL to Boolean logic that runs in $\mathcal{O}(m^{|\varphi|})$ time, and (2) an efficient translation that runs in $\mathcal{O}(|\varphi|^2 m^2)$ time, and prove their correctness and runtime. Lastly, (3) we consider examples of using Boolean MaxSAT solvers to solve the MLTL MaxSAT problem.

1 Introduction

Mission-time LTL (MLTL) is a variant of LTL with integer-bounded temporal operators [21,23,31]. Whereas LTL specifications reason over an infinite computation [5], MLTL specifications reason over a finite computation. The latter is better suited to cyber-physical systems that inherently have a finite computation (e.g., due to battery life). Furthermore, the finite bounds in MLTL specifications open avenues for efficient implementation in resource-constrained hardware like drones, UAVs, and robots [21]. Indeed, MLTL is used in specifications for flight missions, robotics, NASA drone aircraft, and other applications in the industry [4,18,20,24,28,30,32].

Extensive use of MLTL in safety-critical cyber-physical systems necessitates solving related problems in system specifications, such as satisfiability, realizability, and resolving conflicting objectives. MLTL satisfiability was extensively considered by Li et al. [23] and was motivated to aid debugging runtime verification specifications. Solving for the maximum requirements that satisfy a system simultaneously (MaxSAT for MLTL) is a related problem for resolving conflicting objectives. Safety-critical cyber-physical systems are often subject to multiple regulatory requirements, and therefore MaxSAT for MLTL is important for system design, feature prioritization, and cost analysis. Whereas most specifications in runtime verification are satisfiable (a specification has a bug if it can not be satisfied), the MaxSAT problem is more relevant to sets of formulas that are simultaneously unsatisfiable (due to conflicting objectives). We find that the best methods for satisfiability checking by Li et al. [23] do not show scalable performance for sets of formulas that are simultaneously unsatisfiable (Section 5). Therefore, although their approach [23] is suitable for specification debugging, better approaches are necessary to solve the MLTL MaxSAT problem.

We advocate translations from MLTL to Boolean logic to improve the performance of satisfiability checks for unsatisfiable formulas, and to solve the MLTL MaxSAT problem. Boolean SAT solvers have made unprecedented advancement in the last decade [3,8,9,10,15,26]. Most solvers also report the (nonminimal) unsatisfiable core for unsatisfiable formulas. What is the best possible time complexity of a translator from MLTL to Boolean logic? Li et al. [23] show that the MLTL satisfiability problem is NEXP-Complete, i.e., the best translation to Boolean logic is at least exponential in time. We ask how fast this translation can be accomplished in practice. We will first review available translators from MLTL to other logics investigated by Li et al. [23], and then answer this question. Note that MLTL satisfiability for a fragment of formulas that have intervals starting at 0 is PSPACE-Complete [23].

Let m denote the maximum interval length of an MLTL formula φ of length $|\varphi|$. Li et al. [23] provide translations to LTL and LTLf, which run in $\mathcal{O}(m^{|\varphi|})$ time, and a translation to first-order logic (which we will call the SMT translation) that runs in $\mathcal{O}(|\varphi|)$ time. Li et al. [23] found that the SMT translation is the only scalable approach for the satisfiability checking of long formulas. We confirm that this approach works well, but the Boolean translation performs better (Section 5). Moreover, a Boolean translation can solve MLTL MaxSAT instances using off-the-shelf Boolean MaxSAT solvers.

First, we consider an extension from Li et al.'s [23] MLTL to LTL translation to make an MLTL to Boolean translator (Section 3). This approach takes the same runtime as the LTL translation, $\mathcal{O}(m^{|\varphi|})$. Next, we propose a more efficient translation to Boolean logic to improve the overall performance for sets of unsatisfiable formulas as in MLTL MaxSAT instances. Our efficient translation to Boolean logic runs in $\mathcal{O}(m|\varphi|)^2$ time (Section 4). Furthermore, the algorithm is embarrassingly parallelizable to a factor of $|\varphi|m$. Even without parallelizing the implementation of this algorithm, the total runtime (translation + satisfiability check) is better than the best approaches considered by Li et al. [23] (Section 5).

Although an $O(m|\varphi|)^2$ runtime algorithm will outperform an $\mathcal{O}(m^{|\varphi|})$ algorithm, both translations run in exponential time when the interval length is exponential in the formula length, e.g., $m = 2^{|\varphi|}$. This is expected as MLTL satisfiability is NEXP-Complete [23]. However, exponentially long interval lengths are unlikely in most practical situations. For example, the largest interval length considered by Li et al. [23] for a formula of length 100 is 10^5 (for extreme situations), which is much less than $2^{|\varphi|}$ $(= 2^{100} \sim 10^{30})$. Moreover, as MLTL is used in applications like robots and drones, the maximum interval length is at most the mission-time (an a priori known constant), and is generally smaller than the mission-time for efficient implementation in resource-constrained hardware. If the interval ranges were as large as 2^{100}, one would consider using alternative logic like LTL or a timescale-based logic for specifications [14,19,22].

The summary of contributions is as follows:

1. (Section 3) We develop a translation from MLTL to Boolean logic that runs in $\mathcal{O}(m^{|\varphi|})$ time (which we will call the "slow" translation) and prove its correctness and runtime.
2. (Section 4) We contribute to an efficient translation that runs in $\mathcal{O}(m|\varphi|)^2$ time (the "fast" translation) and prove its correctness and runtime.
3. (Section 5) We benchmark satisfiability using our approaches with the best approaches of Li et al. [23] on real instances from the NASA Air Traffic Controller specifications [13,16], as well as random MaxSAT instances.
4. (Section 6) We present preliminary results of solving MLTL MaxSAT problems using off-the-shelf Boolean MaxSAT solvers.

2 Preliminaries

MLTL is interpreted over a finite trace [31]. We formally define a trace, the MLTL syntax, and semantics.

Definition 1. *(Finite Computation/Trace) A finite computation/trace, denoted by π, is a finite sequence of sets of atomic propositions, and the ith set is denoted by $\pi[i]$. The trace length is denoted by $|\pi|$.*

Given a trace $\pi = \pi[0]\pi[1],...,\pi[N]$, where $N = |\pi| - 1$, we let π_i, where $i \in \{0,1,...,N\}$, denote the suffix of π starting at the ith set, i.e., $\pi_i = \pi[i]\pi[i+1],...,\pi[N]$. We note that $\pi_0 = \pi$.

Definition 2. *(MLTL Syntax) A formula φ in MLTL is recursively defined over the set of atomic propositions $\mathcal{AP}$ as:*

$$\varphi := \text{true} \mid p \in \mathcal{AP} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2, \tag{1}$$

where φ_1 and φ_2 are MLTL formulas, and lb and ub are finite natural numbers such that $0 \leq lb \leq ub$.

The MLTL syntax in Eq (1) gives the abstract syntax tree (AST) of an MLTL formula φ. The number of nodes in the AST gives the length of the formula and is denoted as $|\varphi|$. The height of the AST is between $\log(|\varphi|)$ and $|\varphi|$. The number of leaves (atomic propositions) is bounded by the number of nodes in the tree [12]. The set of atomic propositions in a formula φ is denoted as $alp(\varphi)$.

Definition 3. *(MLTL Semantics) The semantics of MLTL is recursively defined over a trace π starting at a position i as follows:*

– $\pi_i \models \text{true}$,

- *for any* $p \in \mathcal{AP}$, $\pi_i \models p$ *iff* $p \in \pi[i]$,
- $\pi_i \models \neg\varphi$ *iff* $\pi_i \not\models \varphi$,
- $\pi_i \models \varphi_1 \wedge \varphi_2$ *iff* $\pi_i \models \varphi_1$ *and* $\pi_i \models \varphi_2$,
- $\pi_i \models \varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2$ *iff* $|\pi| \geq (i + lb)$ *and* $\exists j \in [i + lb, i + ub]$ *and* $\pi_j \models \varphi_2$ *and* $\forall k \in [i+lb,i+ub], k < j, \pi_k \models \varphi_1$.

Other useful operators can be derived from the semantics as in LTL: $\Diamond_{[lb,ub]}\varphi := \text{true}\mathcal{U}_{[lb,ub]}\varphi$ and $\Box_{[lb,ub]}\varphi := \neg\Diamond_{[lb,ub]}(\neg\varphi)$. The next operator is expressed as $\Box_{[1,1]}\varphi$.

Definition 4. *(MLTL Satisfiability) An MLTL formula* φ *is satisfiable if a finite computation* π *exists, such that* $\pi_0 \models \varphi$.

Definition 5. *(MLTL MaxSAT) Given a set of MLTL formulas A, find the subset* $B \subseteq A$ *of the largest cardinality such that* $\bigwedge_{a\in B} a$ *is satisfiable.*

In our notation, $\bigwedge_{a\in B} a$ denotes a formula by taking a conjunction of all formulas in B. We abbreviate Boolean satisfiability as SAT, MLTL satisfiability as MLTL-SAT, and LTL satisfiability as LTL-SAT. Li et al. [23] showed that the following function $f : \Phi \to \Psi$, where Φ and Ψ are the sets of MLTL and LTL formulas respectively, translates an MLTL formula to an equisatisfiable LTL formula (we simplified the until operator case):

$$f(\varphi) := \begin{cases} \text{true} & \text{if } \varphi = \text{true}, \\ p & \text{if } \varphi = p \text{, for some } p \in \mathcal{AP}, \\ \neg f(\varphi_1) & \text{if } \varphi = \neg\varphi_1, \\ f(\varphi_1) \wedge f(\varphi_2) & \text{if } \varphi = \varphi_1 \wedge \varphi_2, \\ f(\varphi_2) & \text{if } \varphi = \varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2 \text{ and } (ub = 0), \\ X^{lb} f(\varphi_1 \mathcal{U}_{[0,ub-lb]} \varphi_2) & \text{if } \varphi = \varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2 \text{ and } (lb > 0), \\ f(\varphi_2) \vee (f(\varphi_1) \wedge X f(\varphi_1 \mathcal{U}_{[0,ub-1]} \varphi_2)) & \text{if } \varphi = \varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2 \text{ and } (lb = 0), \end{cases} \tag{2}$$

where X^i represents that X is repeated i times. The runtime of this translation is evident from the number of recursive calls to the function f in the case of the Until operator in Eq. (2). The number of recursive calls is $\mathcal{O}(ub - lb + 1)$. Let m be the maximum interval length in an MLTL formula φ, then the runtime of this translation is bounded by $m^{|\varphi|}$. A formula reaches this upper bound if it has all nodes as the Until operator, with all intervals as the maximum interval length.

Li et al. [23] also translate to first-order logic that can be conveniently encapsulated as an SMT formula. The first-order translation introduces an uninterpreted function for each $p \in \mathcal{AP}$, $f_p : \mathbb{N} \to \{\text{true,false}\}$, where $\mathbb{N}$ is the set of natural numbers. The function f_p is such that $f_p(k)$ is true if $p \in \pi[k]$, where π is the satisfying trace (see [7] for more information on uninterpreted functions). The function to convert to a first-order formula fol(φ,k,len) was recursively defined as

- $\text{fol}(\text{true},k,len) = (len > k)$ and $\text{fol}(\text{false},k,len) = \text{false}$,
- $\text{fol}(p,k,len) = (len > k) \wedge f_p(k)$,
- $\text{fol}(\varphi \wedge \xi,k,len) = (len > k) \wedge \text{fol}(\varphi,k,len) \wedge \text{fol}(\xi,k,len)$,

– $\text{fol}(\varphi\mathcal{U}_{[a,b]}\xi,k,len)=(len>k)\wedge(\exists i.((a+k)\leq i\leq(b+k))\wedge(\text{fol}(\xi,k,len-i)\wedge(\forall j((a+k)\leq j\leq i))\Rightarrow\text{fol}(\varphi,k,len-j))$.

The runtime of this translation is bounded by $|\varphi|$, as the function visits every node once in the AST of the formula. Moreover, this runtime is independent of the interval length, unlike the LTL translation in Eq. (2). Interval dependence is mitigated in the runtime using first-order quantifiers at the cost of interpreting quantifiers during satisfiability checking.

3 The slow translation to Boolean logic

The LTL translation of Li et al. [23] almost immediately gives a method to translate to Boolean logic. Let $\mathbb{N}$ be the set of natural numbers, Φ be the set of MLTL formulas, and Φ_B be the set of Boolean formulas on an extended set of atomic propositions $\mathcal{AP}_{Ex}=\{p^i \mid i\in\mathbb{N} \text{ and } p\in\mathcal{AP}\}$. Consider the following recursive function, $f:\Phi\times\mathbb{N}\rightarrow\Phi_B$,

$$f(\varphi,i):=$$
$$\begin{cases} \text{true} & \text{if } \varphi=\text{true},\\ p^i & \text{if } \varphi=p, \text{ for some } p\in\mathcal{AP},\\ \neg f(\varphi_1,i) & \text{if } \varphi=\neg\varphi_1,\\ f(\varphi_1,i)\wedge f(\varphi_2,i) & \text{if } \varphi=\varphi_1\wedge\varphi_2,\\ f(\varphi_2,i) & \text{if } \varphi=\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2 \text{ and } (ub=0),\\ f(\varphi_1\mathcal{U}_{[0,ub-lb]}\varphi_2),i+lb) & \text{if } \varphi=\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2 \text{ and } (lb>0),\\ f(\varphi_2,i)\vee(f(\varphi_1,i)\wedge f(\varphi_1\mathcal{U}_{[0,ub-1]}\varphi_2),i+1) & \text{if } \varphi=\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2 \text{ and } (lb=0). \end{cases} \tag{3}$$

Lemma 1. *An MLTL formula φ is satisfiable iff the Boolean formula $f(\varphi,0)$ is satisfiable, i.e., (φ is MLTL-SAT) $\Leftrightarrow$ ($f(\varphi,0)$ is SAT).*

Proof sketch. Let $\mathcal{G}=\{g \mid g:\mathcal{AP}_{Ex}\rightarrow\{\top,\bot\}\}$ denote the set of functions that evaluate the extended atomic propositions to a Boolean value (true/false), and Φ_B be the set of Boolean formulas. Lastly, let $Eval:\Phi_B\times\mathcal{G}\rightarrow\{\bot,\top\}$ take a Boolean formula $\varphi\in\Phi_B$, and a $g\in\mathcal{G}$, and return the evaluation of the Boolean formula, where the atomic propositions are assigned truth values according to g. We prove that:

$$(\exists\pi, s.t., \pi\models\varphi) \Leftrightarrow (\exists g\in\mathcal{G}, s.t., Eval(f(\varphi,0),g)=\top). \tag{4}$$

Assume the left side of Eq. (4). For a given trace π, consider a $g\in\mathcal{G}$ defined by $g(p^i)=\top$ if $p\in\pi[i]$ and $g(p^i)=\bot$ otherwise. We prove that if $(\pi\models\varphi)$ holds, then $Eval(f(\varphi,0),g)=\top$ by induction on the structure of φ. Similarly, assume the right side of Eq. (4), then we can construct a trace π, such that $p\in\pi[i]$ if $g(p^i)=\top$ for all $p\in alp(\varphi)$, such that $(\pi\models\varphi)$. The full proof is detailed in the Appendix.

Lemma 2. *(Runtime and encoding length) Let m be the maximum interval length in an MLTL formula φ. Then, the runtime and encoding length of the slow Boolean translation (Eq. (3)) is $\mathcal{O}(m^{|\varphi|})$.*

Proof. Follows directly from the $\mathcal{O}(m)$ recursive calls to the same function in the until operator cases of Eq. (3). □

4 The fast translation to Boolean logic

Our ideas stemmed from the equivalence relationships used in the next normal form in the Black solver for LTL-SAT [17]. In addition, the proof to Lemma 1 gave insights into making an iterative version of Eq. (3) to remove redundant function calls. In the following, we first illustrate the main ideas of the algorithm using one example MLTL formula, and then generalize the idea formally.

4.1 Algorithm overview

Consider a formula, $\Box_{[a,b]}\Box_{[c,d]}\Box_{[e,f]}p$ and use slack atomic propositions $t_1, t_2 \in \mathcal{AP} \backslash alp(\varphi)$ to make the substitutions: $t_1 := \Box_{[c,d]}\Box_{[e,f]}p$ and $t_2 := \Box_{[e,f]}p$. The formula can be expressed using the slack atomic proposition t_1 according to the MLTL semantics (Def. 3) as

$$f_{t_1}(a) \wedge f_{t_1}(a+1) \wedge \cdots \wedge f_{t_1}(b), \tag{5}$$

where $f_p : \mathbb{N} \to \{\text{true,false}\}$ is an uninterpreted function for an atomic proposition $p \in \mathcal{AP}$ (see [7,23]). Note that an uninterpreted function does not evaluate to an expression but is rather a data structure to represent an extended proposition with two attributes: its name and its extended index (the extended atomic proposition p^2, would have name "p", and index of 2, and is represented as $f_p(2)$). The SAT solver determines the value that the uninterpreted function returns; if the SAT solver finds a model, it will report assignments for these functions (e.g., $(f_p(2) = \text{true}) \equiv (p^2 = \text{true})$.)

Now the slack atomic propositions (with their extended set, i.e., $f_{t_1}(a) \ldots f_{t_1}(b)$ in Eq. (5)) are free variables that need to be defined. They are defined in the next step in terms of t_2 using the semantics of MLTL (Def. 3),

$$\begin{aligned} f_{t_1}(a) &\leftrightarrow f_{t_2}(a+c) \wedge f_{t_2}(a+c+1) \wedge \cdots \wedge f_{t_2}(a+d), \\ f_{t_1}(a+1) &\leftrightarrow f_{t_2}(a+c+1) \wedge f_{t_2}(a+c+2) \wedge \cdots \wedge f_{t_2}(a+d+1), \\ &\vdots \\ f_{t_1}(b) &\leftrightarrow f_{t_2}(b+c) \wedge f_{t_2}(b+c) \wedge \cdots \wedge f_{t_2}(b+d). \end{aligned} \tag{6}$$

Next, we define the slack variables corresponding to $f_{t_2}(i)$ in terms of the extended propositions of p (i.e., $f_p(i)$),

$$\begin{aligned} f_{t_2}(a+c) &\leftrightarrow f_p(a+c+e) \wedge f_p(a+c+e+1) \wedge \cdots \wedge f_p(a+d+f), \\ f_{t_2}(a+c+1) &\leftrightarrow f_p(a+c+e+1) \wedge f_p(a+c+e+2) \wedge \cdots \wedge f_p(a+d+f+1), \\ &\vdots \\ f_{t_2}(b+d) &\leftrightarrow f_p(b+d+e) \wedge f_p(b+d+e+1) \wedge \cdots \wedge f_p(b+d+f). \end{aligned} \tag{7}$$

No new slack variables are introduced on the right-hand sides of Eq. (7). Each step expanded the operator based on the semantics (Def. 3), using slack variables. The bounds range from a until b on the left-hand sides of the equivalence relationships in

Eq. (6), whereas on the right-hand sides, the bounds go from $a{+}c$ to $b{+}d$ ($f_{t_2}(a{+}c)$ and $f_{t_2}(b{+}d)$ in the first and last equivalence expressions of Eq. (6)). On proceeding to the next step in Eq. (7), the bounds on the left go from $a{+}c$ to $b{+}d$ and on the right from $a{+}c{+}e$ to $b{+}d{+}f$. Thus, on going down each step, we observe that the number of slack variables introduced are related to the accumulated bounds on the temporal operators of $\Box_{[a,b]}\Box_{[c,d]}\Box_{[e,f]}p$. Having provided an overview, we present the general algorithm and prove its correctness.

4.2 The main algorithm

Definition 6. *(Accumulated Bounds) The root of the AST of a formula has accumulated upper and lower bounds $\varphi.alb{=}\varphi.aub{=}0$. For all other nodes, accumulated bounds are defined for children nodes as follows:*

- *If $\varphi{=}\neg\varphi_1$, then $\varphi_1.alb{=}\varphi.alb$ and $\varphi_1.aub{=}\varphi.aub$.*
- *If $\varphi{=}\varphi_1{\wedge}\varphi_2$, then $\varphi_1.alb{=}\varphi.alb$, $\varphi_1.aub{=}\varphi.aub$, $\varphi_2.alb{=}\varphi.alb$, $\varphi_2.aub{=}\varphi.aub$.*
- *If $\varphi=\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2$, then $\varphi_1.alb=\varphi.alb+lb$, $\varphi_1.aub=\varphi.aub+ub-1$, $\varphi_2.alb=\varphi.alb{+}lb$, $\varphi_2.aub{=}\varphi.aub{+}ub$.*

The accumulated bounds can be computed in $\mathcal{O}(|\varphi|)$ time by visiting each node once. We associate each subformula in a given MLTL formula φ with a unique slack variable denoted as $\varphi.t^i$, where i is the index of the extended atomic proposition corresponding to the atomic proposition $\varphi.t$. Furthermore, we define that $(t^i)^j := t^{i+j}$.

Definition 7. *(Associated Clauses) The associated clause of the root of the AST of an MLTL formula φ is defined as follows:*

- *If $\varphi{=}\neg\varphi_1$, then the associated clause is $(\neg\varphi_1.t^0)$,*
- *If $\varphi{=}\varphi_1{\wedge}\varphi_2$, then the associated clause is $(\varphi_1.t^0{\wedge}\varphi_2.t^0)$,*
- *If $\varphi{=}\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2$, then the associated clause is $f(\varphi_1.t^0\,\mathcal{U}_{[lb,ub]}\,\varphi_2.t^0{,}0)$.*

The associated clause for all other nodes is defined as follows:

- *If $\varphi{=}\neg\varphi_1$, then the associated clause is $\bigwedge_{j=[\varphi.alb,\varphi.aub]}(\varphi.t^j{\Leftrightarrow}\neg\varphi_1.t^j)$,*
- *If $\varphi{=}\varphi_1{\wedge}\varphi_2$, then the associated clause is $\bigwedge_{j=[\varphi.alb,\varphi.aub]}(\varphi.t^j{\Leftrightarrow}\varphi_1.t^j{\wedge}\varphi_2.t^j)$,*
- *If $\varphi{=}\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2$, then the associated clause is $\bigwedge_{j=[\varphi.alb,\varphi.aub]}(\varphi.t^j{\Leftrightarrow}f(\varphi_1.t^j\mathcal{U}_{[lb,ub]}\varphi_2.t^j{,}0))$.*

Definition 8. *A slack variable is "defined" in an equivalence relation if it alone appears on the left-hand side of the equivalence expression, and if the right-hand side of the expression contains only new slack variables or extended atomic propositions $p^i{\in}\mathcal{AP}_{Ex}$, s.t., $p{\in}alp(\varphi)$.*

Lemma 3. *(Loop Invariance) Let φ be an MLTL formula. All the slack variables in the associated clause of the root node, and the right-hand sides of the equivalence relations in the associated clauses of other nodes in the AST of φ, are defined in the associated clause(s) of their children nodes.*

Algorithm 1: Function initialize(φ)

```
Input: The AST of $\varphi$
Output: (V,Q)
Q = [] ;                                          // A queue
V = [] ;                                          // A vector of assertions
Q.insert($\varphi$)
v = Q.pop()
switch $v$ do                                     // Initialization begins
    case $p \in \mathcal{AP}$, true do
        continue
    case $\neg\varphi_1$ do
        V.add(!$\varphi_1.t^0$);                  // $\mathcal{O}(1)$
    case $\varphi_1 \wedge \varphi_2$ do
        V.add(($\varphi_1.t^0 \wedge \varphi_2.t^0$))        // $\mathcal{O}(1)$
    case $\varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2$ do
        temp = $\varphi_2.t^{ub}$;
        for ($i$=$ub-1$; $i \geq lb$; $i$=$i-1$) do
            temp = $\varphi_2.t^i \vee (\varphi_1.t^i \wedge$ temp);
        V.add(temp)                               // $\mathcal{O}(ub-lb+1)$
If $v$ has children, add them to Q.
return (V,Q)
```

Proof. The operator at the root node is expressed in terms of slack variables of its immediate children, starting at index 0 (Def. 7).

– If the parent is a conjunction or negation, then the associated clause of the parent has the slack variables $\varphi_1.t^i$ and $\varphi_2.t^i$ for all $i \in [v.alb, v.aub]$. However, the accumulated upper and lower bounds do not change for the children nodes from Def. 6.
– If the parent is $\mathcal{U}_{[lb,ub]}$, the slack variables introduced are $\varphi_1.t^i$ and $\varphi_2.t^j$ for all $i \in [lb + alb, ub + aub - 1]$ and $j \in [lb + alb, ub + aub]$. From Def. 6, $[lb{+}alb,ub{+}aub{-}1] = [\varphi_1.alb,\varphi_1.aub]$ and $[lb{+}alb,ub{+}aub] = [\varphi_2.alb,\varphi_2.aub]$.

Hence from Def. 7, all the slack variables on the left-hand side of the parent node's associated clause are defined in the children nodes's associated clause.

Termination: If a child of a parent node is an atomic proposition, then no slack variables are introduced corresponding to that child. □

Theorem 1. *Let V be the set of all associated clauses of an MLTL formula φ. Then, (φ is MLTL-SAT) $\Leftrightarrow$ (($\bigwedge_{v \in V} v$) is SAT).*

Proof. In Lemma 3, we showed that all the slack variables introduced are defined subsequently in the subexpression evaluations. To show that the conjunction of all the elements in the vector of V is an equisatisfiable formula, it suffices to show that the slack variables are defined according to the semantics of MLTL in Def. 3. To see this, notice that Def. 7 directly uses the MLTL semantics (Def. 3). Hence proved. □

Algorithm 2: Function main(φ)

```
   Input: The AST of $\varphi$
   Output: Vector of constraints V
1  accumulatedBounds($\varphi$)      // Assign $alb$ and $aub$ on each node of $\varphi$
     according to Def. 6, $\mathcal{O}(|\varphi|)$
2  (V,Q) = initialize($\varphi$)                          // Alg. 1, $\mathcal{O}(ub-lb+1)$
3  while $!Q.isEmpty()$ do                                 // The main loop, $\mathcal{O}(|\varphi|)$
4      $v$ = Q.pop()
5      switch $v.type$ do
6          case $p \in \mathcal{AP}$, true do
7              continue
8          case $\neg\varphi_1$ do
9              foreach $i \in [v.alb, v.aub]$ do                  // $\mathcal{O}(v.aub-v.alb+1)$
10                 V.add($v.t^i \Leftrightarrow !\varphi_1.t^i$);
11         case $\varphi_1 \wedge \varphi_2$ do
12             foreach $i \in [v.alb, v.aub]$ do                  // $\mathcal{O}(v.aub-v.alb+1)$
13                 V.add($\varphi.t^i \Leftrightarrow (\varphi_1.t^i \wedge \varphi_2.t^i)$ )
14         case $\varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2$ do
15             foreach $j \in [v.alb, v.aub]$ do      // $\mathcal{O}(v.aub-v.alb+1)(ub-lb+1)$
16                 temp = $\varphi_2.t^{ub+j}$
17                 for $(i = ub-1; i \geq lb; i = i-1)$ do
18                     temp = $\varphi_2.t^{i+j} \vee (\varphi_1.t^{i+j} \wedge$ temp)
19                 V.add($\varphi.t^j \Leftrightarrow$ temp)
20     If $v$ has children, add them to Q.
21 return $V$
```

Algs. 1 and 2 give a breadth-first implementation to derive a vector of associated clauses according to Def. 7. We describe the algorithms, and then analyze the runtime. **Description of Alg. 1, initialize(φ)**: The algorithm takes as input the AST of an MLTL formula φ, and returns a tuple (V,Q), where V is a vector of associated clauses, and Q is a queue. This is the initialization step of Alg. 2. Depending on the formula's root node type, an associated clause is inserted into V according to Def. 7. **Description of Alg. 2, main(φ)**: The algorithm takes as input the AST of an MLTL formula φ, and returns a vector of associated clauses of φ, V.

Lemma 4. *The worst case runtime and encoding length of Alg. 2 is $\mathcal{O}(|\varphi|m)^2$, where m is the maximum interval length of the formula.*

Proof. The complexity is evaluated from line 15 in Alg. 2 which has the worst-case runtime. To get an upper bound, we take the interval length $m = (ub-lb+1)$ that is the maximum among all the temporal operators of φ. The accumulated interval bounds keep increasing with every visit to a temporal operator, as in m, $2m$, $3m$, and so on. Accounting for the additional factor of $m = (ub-lb+1)$ (see line 15 of

Alg. 2), the total runtime is bounded by $m^2+2m^2+3m^2\cdots|\varphi|m^2$ (as the maximum height of the AST is the formula length). This is bounded by $|\varphi|^2m^2$.

Lemma 5. *Alg. 2 is embarrassingly parallelizable to a factor of* $|\varphi|m$.

Proof. In each of lines 10, 13 and 19 of Alg. 2, the for loop over the accumulated bounds $[v.alb, v.aub]$ are fully independent, and if embarrassingly parallelized will amount to an $\mathcal{O}(1)$ runtime (i.e., $\mathcal{O}(v.alb-v.aub+1)(ub-lb+1)$ becomes $\mathcal{O}(ub-lb+1)$ when fully parallelized, in line 15 of Alg. 2). Thus, when embarrassingly parallelized, the overall runtime is the maximum interval length, m, times the formula length, i.e., $\mathcal{O}(|\varphi|m)$.

5 Benchmarks

We benchmark MLTL-SAT using the Boolean translations in Sections 3 and 4 with the translations considered by Li et al. [23]. **Real instances:** The LTL formulas from NASA Air Traffic Controller [13,16] are assigned random intervals of maximum lengths 100, 1000, and 10000. Li et al., used a part of this data set (63 formulas with random intervals), whereas we consider the full set with 756 formulas. **Random instances:** All the real instances were satisfiable formulas, whereas, in MaxSAT, instances are expected to be unsatisfiable. Our random formulas have a good mix of satisfiable and unsatisfiable instances. The set is generated by taking the conjunction of 16 smaller formulas, each of length 10, with a normal distribution over operators $\neg$, $\vee$, $\wedge$, F, G, and U. Hence the overall length is 160 for each formula in this set. The formulas are generated over 4 atomic propositions, and the maximum random interval length is set to 100. A similar procedure is used to generate random MaxSAT instances in Boolean logic [1,11].

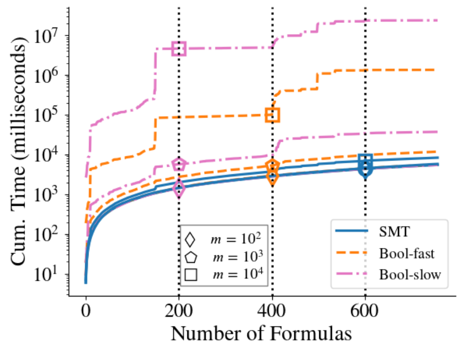
The final composition of the formulas is shown in Table 1. We do not use the random instances in Li et al. as they are again mostly satisfiable and may not yield the best perspective for MaxSAT instances (see Table 1; about 94% of formulas whose answers are known are satisfiable in Li et al.'s [23] random instances).

Source	sat	unsat	unknown	timeout	total
Li et al. [23]	6404	438	1377	2222	10438
Our random formulas	38	62	0	0	100

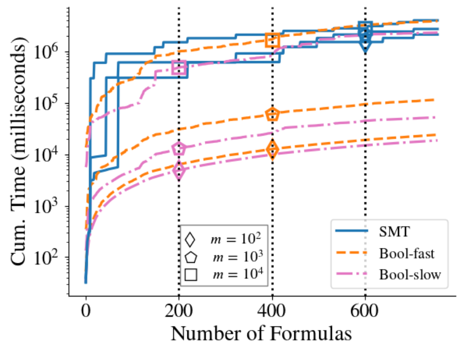
Table 1: Random formula composition.

All translations were implemented in C++, and they generate Boolean expressions in the SMT-LIB v2 format [6]. The SMT translation of Li et al. [23], was taken from their artifacts and fit into our C++ codes with minimal syntactic changes. Z3 was used as the SAT solver. In addition, we also consider two SMV-based MLTL-SAT approaches that performed well in Li et al.'s benchmarks in [23]: klive and bmc. NuXmv (version 2.0.0) updated the command ncheck_ltlspec_klive to ncheck_ltlspec_ic3, and this was the only change we made to the commands used by Li et al. [23, Figure

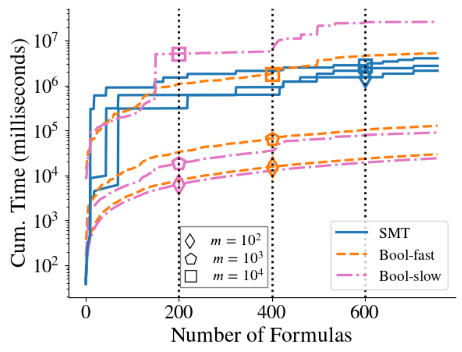
1]. The SMV-translations are only considered on random formulas and not on the real formula set as the latter contained expressions such as the ternary operator that aren't accommodated by the parsers of Li et al. [23]. Benchmarks were run on an isolated node with a single core and 128 GB memory on the NOVA supercomputer of Iowa State University. A timeout of 5 minutes was used, similar to the benchmarks presented for the Black solver for LTL [17] so that results presented here can be reproduced in a reasonable time frame (the 100 random instances took 17 hours, and the 756 real instances took about 11 hours with this timeout).



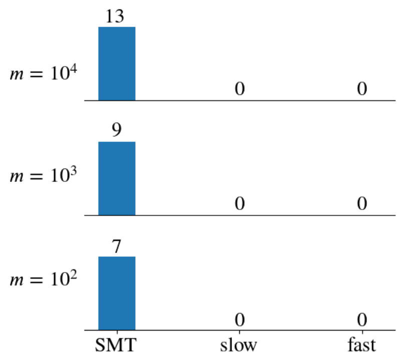
(a) Cummulative translation time



(b) Cummulative solving time

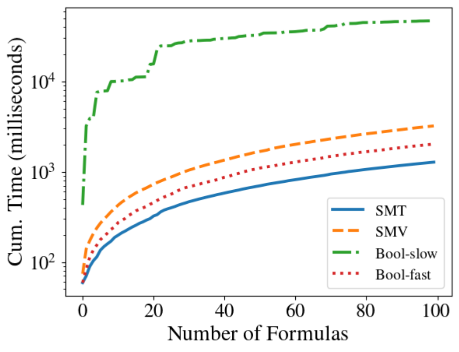


(c) Total time

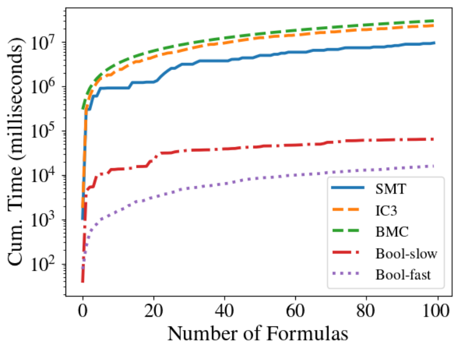


(d) # formula timeouts

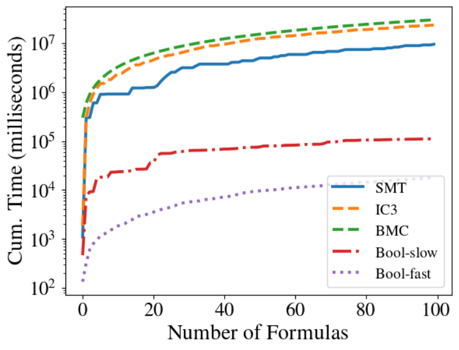
Fig. 1: NASA Air Traffic Controller benchmarks [13,16] with random intervals. The cummulative (a) translation, (b) solving, and (c) total (translation + solving) times, and (d) the number of timeouts. SMT denotes the SMT translation in Li et al. [23], Bool-slow and Bool-fast correspond to the slow (Section 3) and fast (Section 4) Boolean translations, and m is the maximum interval length. The vertical dotted lines at 200, 400, and 600 are used for Bool-slow, Bool-fast, and SMT respectively. In (d), Bool-slow and Bool-fast are abbreviated as "slow" and "fast" respectively.



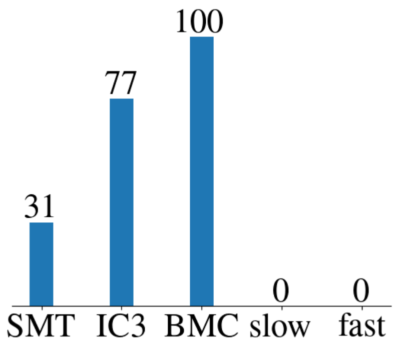
(a) Cummulative translation time



(b) Cummulative runtime



(c) Total time



(d) # formula timeouts

Fig. 2: Benchmarks of random MaxSAT instances. The cummulative (a) translation, (b) solving, and (c) total (translation + solving) times, and (d) the number of timeouts. SMT denotes the SMT translation in Li et al. [23], and IC3 and BMC correspond to the two SMV approaches in Li et al. Bool-slow and Bool-fast correspond to the slow (Section 3) and fast (Section 4) Boolean translations. In (d), Bool-slow and Bool-fast are abbreviated as "slow" and "fast" respectively.

In all figures, high-up lines correspond to poor performance (longer times), and low-lying lines correspond to good performance (shorter times). Figure 1a shows the translation time for the real NASA instances. Li et al.'s [23] SMT translation is labeled as SMT, and the slow (Section 3), and fast (Section 4) Boolean translations are labeled as Bool-slow and Bool-fast respectively. The symbols on the vertical dotted lines are used to distinguish curves corresponding to the different maximum interval lengths (m) of the random intervals; $\Diamond$, ⬠ and $\Box$ correspond to $m = 10^2$, 10^3 and 10^4 respectively, and the vertical dotted lines at $x = 200$, 400, and 600 are used for Bool-slow, Bool-fast, and SMT respectively.

From the vertical dotted line at $x = 600$ in Figure 1a, we see that Li et al.'s SMT translation has very little effect on increasing m owing to its linear translation time. In contrast, the slow and fast translators (vertical dotted lines at 200 and 400, respectively) are affected by an increase in m, and the slow translation is affected the most because of its exponential runtime (Lemma 2). At $m = 10^2$ (marked by $\Diamond$), the translation times are nearly the same for the three translators.

Figure 1b shows the solving time. Both the Boolean translators show superior performance at all m that we considered. Specifically, at $m = 10^2$ and 10^3, the Boolean translators outperform Li et al.'s [23] SMT translation. Figure 1c shows the total time (adding times from Figures 1a and 1b) which is mostly dominated by the time in the solving phase for the Bool-fast and the SMT translations, and by the translation time for Bool-slow. From the total time perspective (Figure 1c), both the Boolean translations outperform the SMT translation at $m = 10^2$ and $m = 10^3$, and the SMT translation seems to perform slightly better at $m = 10^4$. However, this is not the full picture; Figure 1d shows the number of formulas that timed out (in 5 minutes) for each m. None of the Boolean translations timed out, whereas in each case several SMT translations timed out. Although we haven't experimented, for $m > 10^4$, we expect a similar trend where Li et al.'s approach has many timeouts, and the boolean translations succeed without timeouts, with an overall better performance of the SMT translation due to a linear encoding time. The timeouts in Li et al.'s approach may be reduced by employing more recent SMT algorithms [2,27].

Figure 2a shows the translation time for the random formulas. Again, Bool-slow takes the longest time, and the SMT translation of Li et al. takes the shortest time. Both Boolean translations perform better than other approaches in the solving phase (Figure 2b). Figure 2c shows the total time dominated by the solving time in Figure 2b. Finally, Figure 2d shows the number of timeouts with each approach. Again, none of the Boolean translations timed out. Notice that the larger number of timeouts indicates that these are harder instances (with a mix of unsatisfiable and satisfiable formulas; see Table 1).

6 Preliminary results of MLTL MaxSAT

Direct Boolean translations enable using off-the-shelf Boolean MaxSAT solvers to solve the MLTL MaxSAT problem. We summarize the preliminary results of using this approach for MLTL MaxSAT. This problem is very hard (NEXP-Complete), and, to the best of our knowledge, we are the first to present a feasible approach for MLTL MaxSAT. We reconsider the random instances in Section 5. Recall that they were made by a conjunction of 16 subformulas, each of length 10. Thus, each instance is a MaxSAT problem with 16 clauses. (For simplicity, we assume that all the clauses have the same weight.) We ran Z3's Boolean MaxSAT solver on all instances to solve for the maximum number of satisfiable clauses. Most random instances in the benchmarks were solvable in time in the order of seconds on our laptops, and at most, took 2 minutes to provide the MaxSAT solution. We then considered tougher formula sets to test the extent of this approach. We carried out two types of tests; in the first one, we kept the maximum interval length constant and increased the length of each clause ranging from 10 to 50 in steps of 10. We used the same setting as the MLTL-SAT tests in Section 5. Tests with clause lengths greater than 20 segment faulted in the MaxSAT phase.

We then considered cases by holding the clause length to 10 and the number of atoms to 5, and tested maximum interval lengths 100, 1000, 10000, and 100000. All tests with $m > 1000$ segment faulted in the solving phase. Table 2 summarizes the extreme cases we could reach using the Z3 MaxSAT solver, with a limit of 128 GB memory and a 24-hour timeout. This section is due further investigation, e.g., by increasing the memory and using a large instance MaxSAT solver like Volt [25] instead of Z3, and will be considered in future work.

# Atoms	Clause length	Max. interval length	Sol.	Translation (ms)	SAT (ms)
5	10	100	9	370	2148
11	20	100	3	17916	748211
5	10	1000	7	80297	1725711

Table 2: MLTL MaxSAT on random instances. In all cases, the number of clauses was 16. The fourth column gives the MaxSAT solution, i.e., the number of clauses that are simultaneously satisfiable.

7 Conclusion and future work

MLTL MaxSAT is useful for system design and specification prioritization, and will find great value in engineering aircraft, drones, robots, etc. However, the MLTL-SAT problem is NEXP-Complete and needs good problem-solving approaches. This work extensively considered the approach involving a translation to Boolean logic that leverage off-the-shelf SAT solvers. We developed an efficient parallelizable translation algorithm to Boolean logic, and when complemented with modern MaxSAT solvers, our approach has potential to solve real MLTL MaxSAT instances.

A parallel version of the fast translation can be readily implemented and will be considered in future work. It will also be interesting to incorporate GPU parallelized Boolean pre-processing in the translation phase to simplify (and shorten) the final Boolean expression [29]. Another interesting direction would be to try to reduce the translation time of $\mathcal{O}(|\varphi|^2 m^2)$ to at least linear in the interval length which will certainly improve the overall performance of MLTL MaxSAT. Lastly, we would also consider using MaxSAT solvers better suited for large formulas like Volt [25] to test the extent of this approach on large interval lengths.

We presented preliminary results on MLTL MaxSAT on random instances. The approach can be readily applied to real instances like the NASA Air Traffic Controller design objectives (real MaxSAT instances) [13,16], but this needs additional work to parse the hard and soft clauses separately to form the partial MLTL MaxSAT problem. These will be considered in future work. Note that the real instances were solvable more easily in the SAT phase compared to the random instances in Section 5. Therefore, we expect real MaxSAT instances to be more easily solvable (compared to random instances).

Appendix

A Proof to Lemma 1

An MLTL formula φ is satisfiable iff the Boolean formula $f(\varphi,0)$ is satisfiable, i.e., (φ is MLTL-SAT) $\Leftrightarrow$ ($f(\varphi,0)$ is SAT).

Proof. Let $\mathcal{G}=\{g \mid g:\mathcal{AP}_{Ex}\rightarrow\{\top,\bot\}\}$ denote the set of functions that evaluate the extended atomic propositions to a Boolean value (true/false), and Φ_B be the set of Boolean formulas. Lastly, let $Eval:\Phi_B\times\mathcal{G}\rightarrow\{\bot,\top\}$ take a Boolean formula $\varphi\in\Phi_B$, and a $g\in\mathcal{G}$, and return the evaluation of the Boolean formula, where the atomic propositions are assigned truth values according to g. We prove that

$$(\exists\pi, s.t., \pi\models\varphi) \Leftrightarrow (\exists g\in\mathcal{G}, s.t., Eval(f(\varphi,0),g)=\top). \tag{8}$$

Forward:

Assume the left side of Eq. (8). For a given trace π, consider a $g\in\mathcal{G}$ defined by:

$$g(p^i)=\begin{cases}\top & \text{if } p\in\pi[i],\\ \bot & \text{otherwise.}\end{cases} \tag{9}$$

We prove that if $(\pi\models\varphi)$ holds, then $Eval(f(\varphi,0),g)=\top$ by induction on the structure of φ.

Base Case: When $\varphi=p\in\mathcal{AP}$, then if $\pi_i\models p$, then $f(\varphi,i)=f(p,i)=p^i$, and from Eq. (9), $Eval(p^i,g)=\top$.

Now let φ be a formula other than an atomic proposition, and let $(\pi_i\models\varphi_m)\Rightarrow$ $(Eval(f(\varphi_m,i),g)=\top)$ hold for all formulas φ_m at the mth level of the AST of φ. We prove that $(\pi_i\models\varphi_{m-1})\Rightarrow(Eval(f(\varphi_{m-1},i),g)=\top)$, that is, whatever $(\pi_i\models\varphi_{m-1})$ evaluates to (i.e., true or false), $(Eval(f(\varphi_{m-1},i),g)=\top)$ evaluates to the same value, (note that $(Eval(f(\varphi_{m-1},i),g)=\top)$ is false iff $(Eval(f(\varphi_{m-1},i),g)=\bot)$ is true).

1. Case when $\varphi_{m-1}=\neg\varphi_m$. From the hypothesis,

$$\begin{aligned}&(\pi_i\models\varphi_m)\Rightarrow(Eval(f(\varphi_m,i),g)=\top),\\ \Leftrightarrow&(\pi_i\not\models\varphi_m)\Rightarrow(Eval(f(\varphi_m,i),g)=\bot),\\ \Leftrightarrow&(\pi_i\models\neg\varphi_m)\Rightarrow(Eval(\neg f(\varphi_m,i),g)=\top),\\ \Leftrightarrow&(\pi_i\models\varphi_{m-1})\Rightarrow(Eval(f(\neg\varphi_m,i),g)=\top),\\ \Leftrightarrow&(\pi_i\models\varphi_{m-1})\Rightarrow(Eval(f(\varphi_{m-1},i),g)=\top).\end{aligned}$$

2. Case when $\varphi_{m-1}=\varphi_m\wedge\varphi'_m$. From the hypothesis, we have for both φ_m and φ'_m,

$$\begin{aligned}&(\pi_i\models\varphi_m)\Rightarrow(Eval(f(\varphi_m,i),g)=\top),\\ &(\pi_i\models\varphi'_m)\Rightarrow(Eval(f(\varphi'_m,i),g)=\top),\\ \Leftrightarrow&(\pi_i\models\varphi_m\wedge\varphi'_m)\Rightarrow(Eval(f(\varphi_m\wedge\varphi'_m,i),g)=\top),\\ \Leftrightarrow&(\pi_i\models\varphi_{m-1})\Rightarrow(Eval(f(\varphi_{m-1},i),g)=\top).\end{aligned}$$

3. Case when $\varphi_{m-1} = \varphi_m \mathcal{U}_{[lb,ub]} \varphi'_m$. Subcase:
   (a) When $ub = 0$, we have that,

$$\begin{aligned}
&(\pi_i \models \varphi'_m) \Rightarrow (Eval(\varphi'_m, i), g) = \top),\\
\Leftrightarrow &(\pi_i \models \varphi_m \mathcal{U}_{[lb,ub]} \varphi'_m) \Rightarrow (Eval(f(\varphi_m \mathcal{U}_{[lb,ub]} \varphi'_m, i), g) = \top),\\
\Leftrightarrow &(\pi_i \models \varphi_{m-1}) \Rightarrow (Eval(f(\varphi_{m-1}, i), g) = \top).
\end{aligned}$$

   (b) When $lb \geq 0$, for each $k \in [i+lb, i+ub]$, we have from hypothesis that, $(\pi_k \models \varphi_m) \Rightarrow (Eval(f(\varphi_m, k), g) = \top)$. Hence it follows that:

$$\begin{array}{c} (\pi_{i+lb} \models \varphi'_m) \\ \vee((\pi_{i+lb} \models \varphi'_m) \wedge (\pi_{i+lb+1} \models \varphi'_m) \\ \vdots \\ \vee((\pi_{i+ub-1} \models \varphi'_m) \wedge (\pi_{i+ub} \models \varphi'_m)) \\ \cdots)))) \end{array} \Rightarrow \begin{array}{c} (Eval(f(\varphi'_m, i+lb) \\ \vee(f(\varphi_m, i+lb) \wedge (f(\varphi'_m, i+lb+1) \\ \vdots \\ \vee(f(\varphi_m, i+ub-1) \wedge f(\varphi'_m, i+ub)) \\ \cdots))))))), g) = \top) \end{array},$$

$$\begin{aligned}
\Leftrightarrow &(\exists j \in [i+lb, i+ub].\, (\pi_j \models \varphi'_m \wedge \forall k \in [i+lb, i+ub], k < j, \pi_k \models \varphi_m))\\
&\Rightarrow (Eval(f(\varphi_m \mathcal{U}_{[0,ub-lb]} \varphi'_m, i+lb), g) = \top),\\
\Leftrightarrow &(\pi_i \models \varphi_m \mathcal{U}_{[lb,ub]} \varphi'_m) \Rightarrow (Eval(f(\varphi_m \mathcal{U}_{[lb,ub]} \varphi'_m, i), g) = \top),\\
\Leftrightarrow &(\pi_i \models \varphi_{m-1}) \Rightarrow (Eval(f(\varphi_{m-1}, i), g) = \top).
\end{aligned}$$

Backward

Given a function g, we can construct a trace π, such that $p \in \pi[i]$ if $g(p^i) = \top$ for all $p \in alp(\varphi)$. The proof is the same as the forward proof with the opposite direction of implication.

Lastly, if a formula φ is MLTL-SAT, then there is a trace such that $\pi \models \varphi$ (by Def. 4), then there is a g such that $Eval(f(\varphi, 0), g) = \top$, and hence $f(\varphi, 0)$ is SAT. Similarly, the converse holds as well.

References

1. Achlioptas, D., Naor, A., Peres, Y.: On the maximum satisfiability of random formulas. Journal of the Association of Computing Machinery **54**(2), 10 (2007). https://doi.org/10.1145/1219092.1219098, https://doi.org/10.1145/1219092.1219098
2. Arif, M.F., Larraz, D., Echeverria, M., Reynolds, A., Chowdhury, O., Tinelli, C.: SYSLITE: Syntax-guided synthesis of PLTL formulas from finite traces. In: 2020 Formal Methods in Computer Aided Design (FMCAD). pp. 93–103 (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_16
3. Audemard, G., Simon, L.: On the glucose SAT solver. International Journal on Artificial Intelligence Tools **27**(01), 1840001 (2018)
4. Aurandt, A., Jones, P., Rozier, K.Y.: Runtime verification triggers real-time, autonomous fault recovery on the CySat-I. In: Proceedings of the 14th NASA Formal Methods Symposium (NFM 2022). Lecture Notes in Computer Science (LNCS), vol. 13260. Springer, Cham, Caltech, California, USA (May 2022). https://doi.org/\url{https://doi.org/10.1007/978-3-031-06773-0_45}
5. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
6. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)
7. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK). vol. 13, p. 14 (2010)
8. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
9. Bjørner, N., Phan, A.D., Fleckenstein, L.: νz-an optimizing SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21. pp. 194–199. Springer (2015)
10. Cherif, M.S., Habet, D., Terrioux, C.: Proceedings of SAT competition 2021 : Solver and benchmark descriptions. SAT competition p. 15 (2021)
11. Coppersmith, D., Gamarnik, D., Hajiaghayi, M., Sorkin, G.B.: Random MAX SAT, random MAX CUT, and their phase transitions. Random Structures and Algorithms **24**(4), 502–545 (2004). https://doi.org/https://doi.org/10.1002/rsa.20015, https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.20015
12. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press, 4th edn. (2022)
13. Dureja, R., Rozier, K.Y.: More scalable LTL model checking via discovering design-space dependencies D^3. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 309–327. Springer International Publishing, Cham (2018)
14. Franceschet, M., Montanari, A., Peron, A., Sciavicco, G.: Definability and decidability of binary predicates for time granularity. Journal of Applied Logic **4**(2), 168–191 (2006). https://doi.org/10.1016/j.jal.2005.06.004
15. Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT competition 2020. Artificial Intelligence **301**, 103572 (2021)

16. 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.) Computer Aided Verification. pp. 3–22. Springer International Publishing, Cham (2016)
17. Geatti, L., Gigante, N., Montanari, A.: A SAT-based encoding of the one-pass and tree-shaped tableau system for LTL. In: Cerrito, S., Popescu, A. (eds.) Automated Reasoning with Analytic Tableaux and Related Methods. pp. 3–20. Springer International Publishing, Cham (2019)
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: Proceedings of the 14th International Conference on Runtime Verification (RV14). vol. 8734, pp. 215–230. Springer-Verlag (2014)
19. Hariharan, G., Kempa, B., Wongpiromsarn, T., Jones, P.H., Rozier, K.Y.: MLTL Multi-type (MLTLM): A logic for reasoning about signals of different types. In: Proceedings of the 15th International Workshop on Numerical Software Verification (NSV). LNCS, vol. 13466. Springer (2022)
20. Hertz, B., Luppen, Z., Rozier, K.Y.: Integrating runtime verification into a sounding rocket control system. In: Dutle, A., Moscato, M.M., Titolo, L., Muñoz, C.A., Perez, I. (eds.) NASA Formal Methods. pp. 151–159. Springer International Publishing, Cham (2021)
21. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: 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). Lecture Notes in Computer Science (LNCS), vol. 12288, pp. 196–214. Springer, Vienna, Austria (2020). https://doi.org/10.1007/978-3-030-57628-8_12, http://research.temporallogic.org/papers/KZJZR20.pdf
22. Lago, U.D., Montanari, A., Puppis, G.: On the equivalence of automaton-based representations of time granularities. In: 14th International Symposium on Temporal Representation and Reasoning (TIME'07). pp. 82–93 (2007). https://doi.org/10.1109/TIME.2007.56
23. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability checking for Mission-time LTL. In: Proceedings of 31st International Conference on Computer Aided Verification (CAV). LNCS, vol. 11562, pp. 3–22. Springer, New York, NY, USA (2019)
24. Luppen, Z., Jacks, M., Baughman, N., Hertz, B., Cutler, J., Lee, D.Y., Rozier, K.Y.: Elucidation and analysis of specification patterns in aerospace system telemetry. In: Proceedings of the 14th NASA Formal Methods Symposium (NFM 2022). Lecture Notes in Computer Science (LNCS), vol. 13260. Springer, Cham, Caltech, California, USA (2022)
25. Mangal, R., Zhang, X., Nori, A.V., Naik, M.: Volt: A lazy grounding framework for solving very large MaxSAT instances. In: Heule, M., Weaver, S. (eds.) Theory and Applications of Satisfiability Testing – SAT 2015. pp. 299–306. Springer International Publishing, Cham (2015)
26. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of satisfiability, pp. 133–182. IOS press (2021)
27. Neider, D., Gavran, I.: Learning linear temporal properties. In: 2018 Formal Methods in Computer Aided Design (FMCAD). pp. 1–10 (2018). https://doi.org/10.23919/FMCAD.2018.8603016
28. Okubo, N.: Using R2U2 in JAXA program. Electronic correspondence (November–December 2020), series of emails and zoom call from JAXA to PI with technical questions about embedding R2U2 into an autonomous satellite mission with a provable memory bound of 200KB

29. Osama, M., Wijs, A.: GPU acceleration of bounded model checking with ParaFROST. In: Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part II 33. pp. 447–460. Springer (2021)
30. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science (LNCS), vol. 8413, pp. 357–372. Springer-Verlag (2014)
31. Rozier, K.Y., Schumann, J.: R2U2: Tool overview. In: Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES). vol. 3, pp. 138–156. Kalpa Publications, Seattle, WA, USA (2017). https://doi.org/10.29007/5pch, https://easychair.org/publications/paper/Vncw
32. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime analysis with R2U2: A tool exhibition report. In: Proceedings of the 16th International Conference on Runtime Verification (RV15). Springer-Verlag, Madrid, Spain (2016)