Impossible Made Possible: Encoding Intractable Specifications via Implied Domain Constraints*

Chris Johannsen, Brian Kempa, Phillip H. Jones, Kristin Y. Rozier, Tichakorn Wongpiromsarn

Iowa State University, {cgjohann,bckempa,phjones,kyrozier,nok}@iastate.edu

Abstract. We take another look at intractable temporal logic specifications, where the intractability stems from self-reference, unboundedness, or the need for explicit counting. A classic example is the specification, "Every file that gets opened eventually gets closed." In all cases, we show that we can capitalize on realistic constraints implied by the operating environment to generate Mission-time Linear Temporal Logic (MLTL) encodings with reasonably-sized memory signatures. We derive a new set of rewriting rules for MLTL, accompanied by proofs of correctness for each rule, and memory optimizations. We utilize these in creating MLTL encodings for all three patterns of "intractability," proving correctness, time complexity, and space complexity for each type of specification encoding.

Keywords: Mission-time Linear Temporal Logic (MLTL) · MLTL Satisfiability · Temporal Logic Specification.

1 Introduction

Since it was named specifically in 2014 [32] as a particularly popular subset of the logics MTL [29] and STL [26] for industrial practice, Mission-time Linear Temporal Logic (MLTL) has become increasingly utilized as a specification logic for industrial applications. For a couple of examples, MLTL was utilized in formal verification on-board Robonaut2 [18], and the NASA Lunar Gateway project is currently using this logic for requirements capture, design-time testing, and online runtime verification [7,8,9]. While we can use established algorithms for evaluating more expressive logics of which MLTL is a subset, previous work has shown that there are substantial advantages to working in MLTL directly, for example in satisfiability checking [24], model checking [19], and runtime verification [16]. We know from these studies that MLTL brings the advantages of being easier to validate and more efficient to evaluate than more expressive, higher-order Logics [23]. However, these advantages come with an expressability trade off; the value of MLTL is limited to the specifications we can accurately capture in it.

There are common specifications that industrial practitioners tend to naturally express using, e.g., first order (FO) logic, but doing so precludes using their established, efficient evaluation tools developed for lower-order temporal logics like MLTL and LTL; satisfiability of first-order logic is undecidable [3]. This has spurred many advances in more-efficiently evaluating more-expressive extensions of linear-time temporal logics that capture higher-order sentiments. Using Petri nets as an intermediate

* Work supported in part by NSF:CPS Award 2038903, NSF:CAREER Award 1664356, and NASA Cooperative Agreement Grant 80NSSC21M0121

specification language, [14] defined a full FO-LTL for specifying liveness in concurrent systems. Quantified Propositional Temporal Logic (QPTL) [34] extends MLTL with limited quantification while only incurring non-elementary complexity [34]. Quantified Linear Temporal Logic (QLTL) [30] specifically defines complexity with respect to Markov processes as a function of number of alternating quantifiers, thus providing a middle-ground restriction on both expressiveness and complexity. Variable-LTL (VLTL) [37] studies which quantifier patterns do and do not forfeit decidability and restricts specifications to those. FO-LTL [22] looks for finite models of infinite domains, relying on the structure of FO fragments and quantifier ordering to bound complexity.

Capitalizing on realistic limits on the evaluation domain led to several tractable algorithms for linear-time logics with first-order-like extensions. First order LTL over Finite Time Structures FO-LTL$^{\text{fin}}$ defined a procedure for checking validity given a finite time horizon [5]. Finite domains enabled reducing first-order properties to a satisfiability check on the sequential circuit representation of a program for model checking [27]. Finite Quantified LTL (FQLTL) [6] extends LTL with quantifiers over finite domains, targeting infinite time evaluation by generating LTL formulas during execution rather than enforcing restrictions in preprocessing.

With this in mind, we take another look at common "first order" specifications like "every file that gets opened eventually gets closed." Depending on the logical encoding of this specification, its evaluation could involve unboundedness (e.g., unbounded number of files), self-reference (e.g., closing a file refers back to the specific file that was opened), or counting, all of which are provably outside the expressability of linear-time logics including MLTL and LTL [12,38]. We note that industrial domains, i.e., the domain where we will evaluate this specification in performing some verification task, naturally impose restrictions that we can use to encode this specification in lower-order logic without changing its meaning. For example, we are tempted to start the encoding with "for all files," yet the default open-file limit in Linux is 1024[36], quite a small number in practice. This leads to an important realization: by parameterizing specification patterns over reasonable limits we can expect to be imposed by the evaluation domain, we can design a set of MLTL encodings of many common "intractable" specifications without changing their meaning, extending the logic, or precluding the use of efficient existing tools for MLTL evaluation.

Having an extensive set of categorized rewriting rules for LTL enables the current state-of-the-art encodings of that logic for a variety of evaluation algorithms; see, e.g., SPOT [10,11]. Therefore, we contribute such a set for MLTL in Section 3 including their proofs of correctness and scalability in terms of reducing the memory signature of the MLTL formula; these utilize the MLTL semantics included in our preliminaries Section 2. We contribute MLTL specification patterns covering common statements that involve self-reference in Section 4, unboundedness in Section 5 and counting in Section 6, including proofs of correctness, time complexity, and memory scalability for the last two. Section 7 contributes an illustrative example to show the effectiveness of pairing the presented techniques to reduce memory requirements for realistic industrial domain parameters. Section 8 concludes with impacts and future work.

2 Preliminaries: Mission-time LTL and Formula-wise Encoding

Mission-time Linear Temporal Logic (MLTL) [24] is a bounded variant of MTL [1] where each temporal operator has an associated closed natural number interval bound.

Definition 1 (*MLTL Syntax*). *The syntax of an MLTL formula φ over a set of atomic propositions $\mathcal{AP}$ is recursively defined as:*

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

where $p \in \mathcal{AP}$, ψ and ξ are MLTL formulas, and I is an interval $[l, u]$ such that $l, u \in \mathbb{N}$ and $l \leqslant u$.

We evaluate MLTL formulas over finite traces. Let π be a finite trace where an element at timestamp $i \in \mathbb{N}_0$ is $\pi[i] \subseteq \mathcal{AP}$ such that $|\pi|$ is the length of π where $i < |\pi| < +\infty$ and π_i is the suffix of π starting at and including i.

Definition 2 (*MLTL Semantics*). *The satisfaction of an MLTL formula by a trace π is defined recursively as:*

- $\pi \models p$ *iff* $p \in \pi[0]$
- $\pi \models \neg\varphi$ *iff* $\pi \not\models \varphi$
- $\pi \models \varphi \wedge \psi$ *iff* $\pi \models \varphi$ *and* $\pi \models \psi$
- $\pi \models \varphi \, \mathcal{U}_{[l,u]} \, \psi$ *iff* $|\pi| \geqslant l$ *and there exists a* $j \in [l, u]$ *such that* $\pi_j \models \psi$ *and* $\pi_k \models \varphi$ *for all* $k \in [l, u]$ *such that* $k < j$.

We say two MLTL formulas φ, ψ are *semantically equivalent* (denoted as $\varphi \equiv \psi$) if and only if $\pi \models \varphi \Leftrightarrow \pi \models \psi$ for all traces π over $\mathcal{AP}$. To complete the 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 $\Diamond_I \varphi \equiv (\mathsf{true} \, \mathcal{U}_I \, \varphi)$, $\Box_I \varphi \equiv (\mathsf{false} \, \mathcal{R}_I \, \varphi)$. Notably, MLTL discards the next ($\mathcal{X}$) operator, since $\mathcal{X}\varphi \equiv \Box_{[1,1]}\varphi$.

2.1 MLTL Formula-wise AST Encoding Structure

We focus on memory usage optimization techniques that use an Abstract Syntax Tree-based (AST) representation for MLTL formulas. Each node in the AST of an MLTL formula φ computes and stores a result-timestamp pair $T_\varphi = (v, t)$ for the corresponding sub-formula with respect to an input trace where $v \in \{\mathsf{true}, \mathsf{false}\}$ and $t \in \mathbb{N}_0$. We call result-timestamp pairs *verdicts*.

Propagation Delay To compute the required memory for each node in an AST-encoded MLTL formula, we must first compute the earliest and latest timestamps when we may have sufficient information to evaluate the formula. We bound these timestamps using the upper and lower interval bounds of the temporal operators in a given formula.

Definition 3 (*Propagation Delay*). *[17] The propagation delay of an MLTL formula φ is the time between when a set of propositions $\pi[i]$ arrives and when it is possible to know if $\pi_i \models \varphi$. A node's* worst-case propagation delay *(**wpd**) is its maximum propagation delay, and the minimum value is its* best-case propagation delay *(**bpd**).*

Definition 4 (*Propagation Delay Semantics*). *[17] Let $\varphi, \psi, \psi_1, \psi_2$ be well-formed MLTL formulas where $\varphi.bpd$ and $\varphi.wpd$ are the best- and worst-case propagation delays of formula φ respectively:*

$$\text{if } \varphi \in \mathcal{AP} : \begin{cases} \varphi.wpd = 0 \\ \varphi.bpd = 0 \end{cases} \qquad \text{if } \varphi = \neg\psi : \begin{cases} \varphi.wpd = \psi.wpd \\ \varphi.bpd = \psi.bpd \end{cases}$$

$$\text{if } \varphi = \Box_{[l,u]}\psi \text{ or } \varphi = \Diamond_{[l,u]}\psi : \begin{cases} \varphi.wpd = \psi.wpd + u \\ \varphi.bpd = \psi.bpd + l \end{cases}$$

$$\text{if } \varphi = \psi_1 \vee \psi_2 \text{ or } \varphi = \psi_1 \wedge \psi_2 : \begin{cases} \varphi.wpd = max(\psi_1.wpd, \psi_2.wpd) \\ \varphi.bpd = min(\psi_1.bpd, \psi_2.bpd) \end{cases}$$

$$\text{if } \varphi = \psi_1 \,\mathcal{U}_{[l,u]}\, \psi_2 \text{ or } \varphi = \psi_1 \,\mathcal{R}_{[l,u]}\, \psi_2 : \begin{cases} \varphi.wpd = max(\psi_1.wpd, \psi_2.wpd) + u \\ \varphi.bpd = min(\psi_1.bpd, \psi_2.bpd) + l \end{cases}$$

The values of $\varphi.wpd$ and $\varphi.bpd$ are based solely on the structure of the given formula φ and do not take into account interactions between sub-formulas. For example, the formula $\Box_{[0,5]}\varphi \vee \Box_{[0,10]}\varphi$ has a structural worst-case propagation delay of 10 but the relationship between the two $\Box_I$ operators always allows evaluation of the formula by time step 5. In other words, we can simplify this formula to $\Box_{[0,5]}\varphi$.

2.2 MLTL AST Encoding Memory Requirements [17]

Consider an AST node g and its set of sibling nodes $\mathcal{B}_g$ (not including g). The minimum required memory (with respect to the number of verdicts) for g is:

$$mem_{node}(g) = max(max\{b.wpd \mid b \in \mathcal{B}_g\} - g.bpd, 0) + 1 \tag{1}$$

We can recursively compute the memory requirements of an AST rooted at g, where $\mathcal{C}_g$ is the set of child nodes of g, as follows:

$$mem_{AST}(g) = mem_{node}(g) + \sum\{mem_{AST}(c) \mid c \in \mathcal{C}_g\} \tag{2}$$

Formula 1 accounts for the worst-case input with respect to evaluating the parent of g. Consider a trace π, time $0 \leqslant i < |\pi|$, and a node g such that g's value is known at index $i + g.bpd$ but the value of a sibling node b_{max} is known at index $i + b_{max}.wpd$ where $b_{max}.wpd = max\{b.wpd \mid b \in \mathcal{B}_g\}$. In order to evaluate g's parent at i, we must know the evaluations of both g and b_{max} at i and therefore buffer the values of g from indices $i + g.bpd$ to $i + b_{max}.wpd$. If $b_{max}.wpd - g.bpd \geqslant 0$, this requires a buffer of size $(i + b_{max}.wpd) - (i + g.bpd) = b_{max}.wpd - g.bpd$, otherwise we do not need to buffer values at node g.

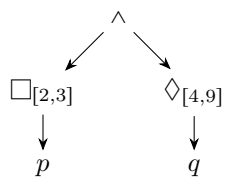

Fig. 1: AST for the MLTL formula $(\Box_{[2,3]}p) \wedge (\Diamond_{[4,9]}q)$ where $p, q \in \mathcal{AP}$.

As an example, consider the AST in Figure 1. We see that $mem_{node}(\wedge) = 1$ since the node has no siblings. Now, for each temporal node we have

$$mem_{node}(\Box_{[2,3]}) = max(max\{\Diamond_{[4,9]}.wpd\} - \Box_{[2,3]}.bpd, 0) + 1 = 8$$
$$mem_{node}(\Diamond_{[4,9]}) = max(max\{\Box_{[2,3]}.wpd\} - \Diamond_{[4,9]}.bpd, 0) + 1 = 1$$

Finally, each $mem_{node}(p) = mem_{node}(q) = 1$ bit. Putting this all together:

$$mem_{AST}(\wedge) = mem_{node}(\wedge) + mem_{node}(\Box_{[2,3]}) + mem_{node}(\Diamond_{[4,9]}) + mem_{node}(p) + mem_{node}(q) = 12$$

3 MLTL Encoding Optimizations

We present MLTL rewriting rules for reducing the AST encoding size of MLTL formulas that can be applied automatically during MLTL formula encoding. This type of optimization is similar to SPOT's [11] optimizations for LTL, with the primary difference being SPOT minimizes the size of an LTL formula's automata representation.

Figure 2 contains the MLTL rewrite rules. We first prove that both sides of each rewrite rule are equivalent using trivially-derived equivalences from LTL and the semantic definitions of MLTL operators. We then show how each rewriting rule maintains or reduces the memory of a given MLTL formula. First, recall that MLTL does not include a Next-time operator ($\mathcal{X}$) as in LTL because it is equivalent to $\Box_{[1,1]}$. More

$\Box_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi \mapsto \Box_{[l_1+l_2,u_1+u_2]}\varphi$ $\quad$ $\Diamond_{[l_1,u_1]}\Diamond_{[l_2,u_2]}\varphi \mapsto \Diamond_{[l_1+l_2,u_1+u_2]}\varphi$	(R1)
$\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi \mapsto \Box_{[l_3,u_3]}(\Box_{[l_1-l_3,u_1-u_3]}\varphi \wedge \Box_{[l_2-l_3,u_2-u_3]}\psi)$ $\Diamond_{[l_1,u_1]}\varphi \vee \Diamond_{[l_2,u_2]}\psi \mapsto \Diamond_{[l_3,u_3]}(\Diamond_{[l_1-l_3,u_1-u_3]}\varphi \vee \Diamond_{[l_2-l_3,u_2-u_3]}\psi)$ where $l_3 = min(l_1, l_2), u_3 = l_3 + min(u_1 - l_1, u_2 - l_2), l_3 < u_3$	(R2)
$\Box_{[a,a]}\Diamond_{[l,u]}\varphi \mapsto \Diamond_{[l+a,u+a]}\varphi$ $\quad$ $\Diamond_{[l,u]}\Box_{[a,a]}\varphi \mapsto \Diamond_{[l+a,u+a]}\varphi$ $\Diamond_{[a,a]}\Box_{[l,u]}\varphi \mapsto \Box_{[l+a,u+a]}\varphi$ $\quad$ $\Box_{[l,u]}\Diamond_{[a,a]}\varphi \mapsto \Box_{[l+a,u+a]}\varphi$	(R3)
$\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\varphi \mapsto \Box_{[l_1,u_3]}\varphi$ $\quad$ $\Diamond_{[l_1,u_1]}\varphi \vee \Diamond_{[l_2,u_2]}\varphi \mapsto \Diamond_{[l_1,u_2]}\varphi$ where $l_1 \leqslant l_2 \leqslant u_1 + 1, u_3 = max(u_1, u_2)$	(R4)
$\Box_{[l_1,u_1]}\varphi \vee \Box_{[l_2,u_2]}\varphi \mapsto \Box_{[l_2,u_2]}\varphi$ $\quad$ $\Diamond_{[l_1,u_1]}\varphi \wedge \Diamond_{[l_2,u_2]}\varphi \mapsto \Diamond_{[l_2,u_2]}\varphi$ where $l_1 \leqslant l_2 \leqslant u_2 \leqslant u_1$	(R5)
$\Box_{[a,a]}(\varphi\, \mathcal{U}_{[l,u]}\, \psi) \mapsto \varphi\, \mathcal{U}_{[l+a,u+a]}\, \psi$ $\quad$ $(\Box_{[a,a]}\varphi)\, \mathcal{U}_{[l,u]}\, (\Box_{[a,a]}\psi) \mapsto \varphi\, \mathcal{U}_{[l+a,u+a]}\, \psi$	(R6)
$(\varphi_1\, \mathcal{U}_{[l,u_1]}\, \varphi_2) \wedge (\varphi_3\, \mathcal{U}_{[l,u_2]}\, \varphi_2) \mapsto (\varphi_1 \wedge \varphi_3)\, \mathcal{U}_{[l,u_1]}\, \varphi_2$ where $l \leqslant u_1, l \leqslant u_2, u_1 \leqslant u_2$	(R7)
$\varphi\, \mathcal{U}_{[l_1,u_1]}\, \Box_{[0,u_2]}\varphi \mapsto \Box_{[l_1,l_1+u_2]}\varphi$ $\quad$ $\varphi\, \mathcal{U}_{[l_1,u_1]}\, \Diamond_{[0,u_2]}\varphi \mapsto \Diamond_{[l_1,l_1+u_2]}\varphi$	(R8)

Fig. 2: Table of MLTL rewrite rules where $\varphi, \psi, \varphi_1, \varphi_2, \varphi_3$ are well-formed MLTL formulas and $a, l, u, l_1, u_2, l_2, u_2, l_3, u_3 \in \mathbb{N}_0$ such that $l \leqslant u$, $l_1 \leqslant u_1$, $l_2 \leqslant u_2$, $l_3 \leqslant u_3$. Each group of rules has identical constraints on their interval bounds.

generally, we can express $a \in \mathbb{N}_0$ nested $\mathcal{X}$ operations with a singleton interval such as in $\Box_{[a,a]}$. We therefore observe the following equivalences:

$$\Box_{[a,a]}\varphi \equiv \Diamond_{[a,a]}\varphi \equiv \psi \,\mathcal{U}_{[a,a]}\, \varphi \tag{3}$$

The following directly follow from the semantics of $\mathcal{U}_I$:

$$\mathsf{false}\,\mathcal{U}_{[l,u]}\,\varphi \equiv \Box_{[l,l]}\varphi \qquad \mathsf{true}\,\mathcal{U}_{[l,u]}\,\varphi \equiv \Diamond_{[l,u]}\varphi \qquad \varphi\,\mathcal{U}_{[l,u]}\,\varphi \equiv \Box_{[l,l]}\varphi \tag{4}$$

With these basic equivalences in hand, we can show that each rewrite rule preserves the MLTL semantics and thus is also an equivalence relation.

Theorem 1 (Equivalence of MLTL Rewrite Rules). *Let $\varphi, \psi, \varphi_1, \varphi_2, \varphi_3$ be well-formed MLTL formulas and a, l, u, l_1, u_2, l_2, u_2, l_3, $u_3 \in \mathbb{N}_0$ such that $l \leqslant u$, $l_1 \leqslant u_1$, $l_2 \leqslant u_2$, $l_3 \leqslant u_3$. Then each rewrite relation ($\mapsto$) in Fig 2 is also an equivalence relation.*

Proof Sketch. [1] We case split over each rule and prove the left- and right-hand sides of each $\mapsto$ relation to be equivalent. Most rules follow directly from the semantics of MLTL (e.g., (R1)) and we present the more complicated proof for (R2) to illustrate:

(R1): Let π be a finite trace such that $\pi \models \Box_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi$. We show that $\pi \models \Box_{[l_1+l_2,u_1+u_2]}\varphi$. By the semantics of $\Box_I$, we know that $\pi_i+ \models \Box_{[l_2,u_2]}\varphi$ for each $i \in [l_1, u_1]$. Intuitively, this means that π satisfies φ at timestamps $[l_2, u_2]$ *relative to* i i.e., $\pi_i \models \varphi$ starting at the timestamp $i + l_2$ and ending at $i + u_2$. So, applying the semantics of $\Box_I$ again, we have that $\pi_{i+j} \models \varphi$ for each $j \in [l_2, u_2]$. By the definition of trace suffixes, this means that $\pi_k \models \varphi$ for each $k \in [l_1 + l_2, u_1 + u_2]$. Therefore $\pi \models \Box_{[l_1+l_2,u_1+u_2]}\varphi$. The converse proof follows from a similar argument.

(R2): Using (R1), we see that

$$\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi \equiv \Box_{[l_3,l_3]}\Box_{[l_1-l_3,u_1-l_3]}\varphi \wedge \Box_{[l_3,l_3]}\Box_{[l_2-l_3,u_2-l_3]}\psi.$$

This follows if both intervals $[l_1 - l_3, u_1 - l_3]$, $[l_2 - l_3, u_2 - l_3]$ are valid i.e., (a) $l_1 - l_3 \geqslant 0$, (b) $l_2 - l_3 \geqslant 0$, and (c) $l_1 - l_3 \leqslant u_1 - l_3$, (d) $l_2 - l_3 \leqslant u_2 - l_3$.
(a) Recall that $l_3 = l_1$, then $l_3 \leqslant l_1$.
(b) Recall that $l_3 = l_1 \leqslant l_2$, then $l_2 - l_3 \geqslant 0 \Rightarrow l_2 \geqslant l_3 \Rightarrow l_3 \leqslant l_2$ holds.
(c) Since $l_1 \leqslant u_1$, we see that $l_1 - l_3 \leqslant u_1 - l_3 \Rightarrow l_1 \leqslant u_1$ holds.
(d) Since $l_2 \leqslant u_2$, we see that $l_2 - l_3 \leqslant u_2 - l_3 \Rightarrow l_2 \leqslant u_2$ holds.
Now, let $u_3 = l_3 + min(u_1 - l_1, u_2 - l_2)$. Applying (R1) once more, we have

$$\Box_{[l_3,l_3]}\Box_{[l_1-l_3,u_1-l_3]}\varphi \wedge \Box_{[l_3,l_3]}\Box_{[l_2-l_3,u_2-l_3]}\psi \equiv \Box_{[l_3,u_3]}\Box_{[l_1-l_3,u_1-u_3]}\varphi \wedge \Box_{[l_3,u_3]}\Box_{[l_2-l_3,u_2-u_3]}\psi.$$

Since this only affects the upper bounds of the inner $\Box$ operators, we show that (a) $l_1 - l_3 \leqslant u_1 - u_3$ and (b) $l_2 - l_3 \leqslant u_2 - u_3$.
(a) Consider the two cases of $u_1 - l_1 \leqslant u_2 - l_2$ and $u_2 - l_2 < u_1 - l_1$:

[1] The full proof for the entire set of rules can be found at https://temporallogic.org/research/FMICS2023/.

(i) Assume $u_1 - l_1 \leqslant u_2 - l_2$, then $u_3 = u_1 - l_1 + l_3$. Replacing this in the target inequality, we have $l_1 - l_3 \leqslant u_1 - (u_1 - l_1 + l_3) \Rightarrow l_1 - l_3 \leqslant l_1 - l_3$.
(ii) Otherwise, $u_2 - l_2 < u_1 - l_1$. Then $u_3 = u_2 - l_2 + l_3$, and replacing this in the target inequality, we have $l_1 - l_3 \leqslant u_1 - (u_2 - l_2 + l_3) \Rightarrow 0 \leqslant u_1 - l_3 - (u_2 - l_2) \Rightarrow u_2 - l_2 \leqslant u_1 - l_3$. Now, since $l_3 = l_1$, we have $u_2 - l_2 \leqslant u_1 - l_1$ which is true from our assumption.

(b) Consider the two cases of $u_1 - l_1 \leqslant u_2 - l_2$ and $u_2 - l_2 < u_1 - l_1$:
(i) Assume $u_1 - l_1 \leqslant u_2 - l_2$, then $u_3 = u_1 - l_1 + l_3$. Replacing this in the target inequality, we have $l_2 - l_3 \leqslant u_2 - (u_1 - l_1 + l_3) \Rightarrow l_2 - l_3 \leqslant u_2 - u_1 + l_1 - l_3 \Rightarrow l_2 \leqslant u_2 - u_1 + l_1 \Rightarrow u_1 - l_1 \leqslant u_2 - l_2$, which is true from our assumption.
(ii) Otherwise, $u_2 - l_2 < u_1 - l_1$, then $u_3 = u_2 - l_2 + l_3$. Replacing this in the target inequality, we have $l_2 - l_3 \leqslant u_2 - (u_2 - l_2 + l_3) \Rightarrow l_2 - l_3 \leqslant u_2 - u_2 + l_2 - l_3 \Rightarrow l_2 \leqslant u_2 - u_2 + l_2 \Rightarrow l_2 \leqslant l_2$.

Finally, we prove that

$$\Box_{[l_3,u_3]}\Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi \wedge \Box_{[l_3,u_3]}\Box_{[l_2-l_3,u_2-l_3-u_3]}\psi \equiv \Box_{[l_3,u_3]}(\Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi \wedge \Box_{[l_2-l_3,u_2-l_3-u_3]}\psi).$$

Let π be a finite trace such that

$$\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2-l_3,u_2-l_3-u_3]}\psi).$$

We apply the semantic definitions of $\wedge$ and $\Box_I$ to see that $\pi_i \models \Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi$ and $\pi_i \models \Box_{[l_2-l_3,u_2-l_3-u_3]}\psi$ for all $i \in [l_3, u_3]$. Combining these relations using the semantics of $\wedge$ once more, we see that

$$\pi_i \models \Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi \wedge \Box_{[l_2-l_3,u_2-l_3-u_3]}\psi$$

for all $i \in [l_3, u_3]$. Using the semantics of $\Box_I$ again, we see that

$$\pi \models \Box_{[l_3,u_3]}(\Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi \wedge \Box_{[l_2-l_3,u_2-l_3-u_3]}\psi).$$

The converse proof follows from a similar argument.

□

Inapplicable LTL Equivalences While the equivalences discussed so far have corresponding equivalence relations in LTL, there are some LTL equivalences without such a relation in MLTL. For instance, consider the LTL formula $\Diamond(\varphi \mathcal{U} \psi) \equiv \Diamond\psi$. Intuitively, so long as ψ holds at some timestamp i in a given trace, then it is trivially true that $\varphi \mathcal{U} \psi$ holds at i in that trace. However, once we add interval bounds to each temporal operator as in $\Diamond_{[l_1,u_1]}(\varphi \, \mathcal{U}_{[l_2,u_2]} \, \psi)$ there is now a constraint on when ψ can hold in a trace with respect to φ and still satisfy the formula. For example, if $\pi_{l_1+l_2} \not\models \psi$ for some trace π that models this MLTL formula, then necessarily $\pi_{l_1+l_2} \models \varphi$ i.e., the satisfaction of φ is still relevant for some satisfying traces.

Similarly, consider the LTL equivalence $\Diamond\Box\varphi \wedge \Diamond\Box\psi \equiv \Diamond\Box(\varphi \wedge \psi)$. Again, intuitively, the LTL operators $\Diamond$ and $\Box$ do not specify *when* their operands must hold, just

that they both eventually always hold. When we add bounds to the left-hand side as in $\Diamond_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi \wedge \Diamond_{[l_3,u_3]}\Box_{[l_4,u_4]}\psi$, both φ and ψ have constraints on *when* they must hold in order for a trace to satisfy this formula, and when this is exactly may differ for either φ or ψ. Speaking generally, MLTL places more constraints on the set of traces that satisfy a given formula than LTL.

Memory Effects of Rewriting Rules on MLTL AST Encodings Applying these rewriting rules strategically can reduce the overall memory requirements of the AST encoding of the MLTL formula. These rules reduce memory requirements in one of two ways: (1) by tightening propagation delays or (2) by reducing formula length.

From Equation 2, an AST node g's required memory is the difference between that $g.bpd$, and the maximum wpd of its siblings. Therefore, reducing $max\{b.wpd \mid b \in \mathcal{B}_g\}$ for a set of sibling nodes $\mathcal{B}_g$ can reduce the memory requirements of all other sibling nodes in $\mathcal{B}_g$. Furthermore, reducing g's wpd can reduce its ancestors' wpd, which in turn could reduce the memory requirements for the ancestors' set of siblings in the same manner. In the following, we use $\varphi(\psi_1 \mapsto \psi_2)$ to denote an MLTL formula that is identical to a formula φ where a sub-formula ψ_1 of φ is replaced with ψ_2.

Lemma 1 (*Memory Effect of Tighter BPD*). *Let φ, ψ_1, ψ_2 be well-formed MLTL formulas where ψ_1 is a sub-formula of φ, ψ_2 is the sub-formula in $\varphi(\psi_1 \mapsto \psi_2)$, and $\psi_1.bpd \leqslant \psi_2.bpd$. Then*

$$mem_{node}(\psi_1) \geqslant mem_{node}(\psi_2).$$

Proof. We first note that ψ_1, ψ_2 have the same set of siblings i.e., $\mathcal{B}_{\psi_1} = \mathcal{B}_{\psi_2} = \mathcal{B}$. Then from Equation 1 we see that

$$\begin{aligned} mem_{node}(\psi_1) =& max(max\{b.wpd \mid \mathcal{B}\} - \psi_1.bpd, 0) + 1 \\ \geqslant& max(max\{b.wpd \mid \mathcal{B}\} - \psi_2.bpd, 0) + 1 \\ =& mem_{node}(\psi_2) \end{aligned}$$

□

Lemma 2 (*Memory Effect of Tighter WPD*). *Let φ, ψ_1, ψ_2 be well-formed MLTL formulas where ψ_1 is a sub-formula of φ, ψ_2 is the sub-formula in $\varphi(\psi_1 \mapsto \psi_2)$, and $\psi_2.wpd \leqslant \psi_1.wpd$. Then*

$$mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leqslant mem_{AST}(\varphi) - mem_{AST}(\psi_1)$$

Proof. As in the proof for Lemma 1, ψ_1, ψ_2 have the same set of siblings i.e., $\mathcal{B}_{\psi_1} = \mathcal{B}_{\psi_2} = \mathcal{B}$.

First, assume $\psi_1.wpd \leqslant max\{b_{\psi_1}.wpd \mid b_{\psi_1} \in \mathcal{B}_{\psi_1}\}$ i.e., ψ_1 does not have the maximum wpd of all of its sibling nodes. Then

$$max\{b_{\psi_1}.wpd \mid b_{\psi_1} \in \mathcal{B}_{\psi_1}\} = max\{b_{\psi_2}.wpd \mid b_{\psi_2} \in \mathcal{B}_{\psi_2}\}$$

since rewriting φ from ψ_1 to ψ_2 does not affect the sibling nodes for either ψ_1, ψ_2. Therefore $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) = mem_{AST}(\varphi) - mem_{AST}(\psi_1)$.

Otherwise, ψ_1 has the maximum wpd of all of its sibling nodes i.e.,

$$\psi_1.wpd > max\{b_{\psi_1}.wpd \mid b_{\psi_1} \in \mathcal{B}_{\psi_1}\}$$

Then the amount of memory required for each node $b_{\psi_1} \in \mathcal{B}_{\psi_1}$ is $mem_{node}(b_{\psi_1}) = max(\psi_1.wpd - b_{\psi_1}.bpd, 0)$. Importantly, each node $b_{\psi_1} \in \mathcal{B}_{\psi_1}$ has a structurally identical counterpart in $\mathcal{B}_{\psi_2}$ since we defined φ as identical to $\varphi(\psi_1 \mapsto \psi_2)$, except where ψ_1 is replaced with ψ_2. We define a mapping $Sib: \mathcal{B}_{\psi_1} \to \mathcal{B}_{\psi_2}$ such that $Sib(b_{\psi_1}) = b_{\psi_2}$. This implies that $b_{\psi_1}.bpd = Sib(b_{\psi_1}).bpd$ for each $b \in \mathcal{B}_{\psi_1}$. Therefore, we see that each sibling node of ψ_2 has a lower memory requirement than the corresponding sibling node of ψ_1 for all $b \in \mathcal{B}_{\psi_1}$:

$$mem_{node}(b_{\psi_1}) = max(\psi_1.wpd - b_{\psi_1}.bpd, 0) \leqslant max(\psi_2.wpd - Sib(b_{\psi_1}).bpd, 0)$$

Further, the propagation delay semantics (Def. 4) dictate that the wpd of a node is greater than or equal to the maximum wpd of all its children. Since we assumed that ψ_1 has the maximum wpd of its parent's children (i.e., ψ_1's siblings) and $\psi_2.wpd \leqslant \psi_1.wpd$, it follows that the ψ_2's parent would have a lower wpd than ψ_1's parent. We can apply this argument recursively to each ancestor of ψ_2 such that every ancestor of ψ_2 will have a lower or equal wpd than the corresponding ancestor of ψ_1, where the preceding relation holds if the wpd is lowered and the first assumption of the proof holds otherwise.

Then the sibling nodes of each ancestor of ψ_2 will have a lower or equal memory requirement than the sibling nodes of each ancestor of ψ_1. Therefore $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leqslant mem_{AST}(\varphi) - mem_{AST}(\psi_1)$. ▫

Intuitively, Lemma 1 and Lemma 2 express the notion that a semantically equivalent formula with a tighter propagation delay results in reduced required memory. A tighter propagation delay provides more information as to *when* the formula will be evaluated in the best and worst cases, requiring less memory for storing intermediate results.

Theorem 2 (Memory Reduction of Rewriting Rules). *Let φ, ψ_1, ψ_2 be well-formed MLTL formulas where ψ_1 is a sub-formula of φ. Then applying a valid rewrite rule in Figure 2 to ψ_1 will result in a new formula $\varphi(\psi_1 \mapsto \psi_2)$ such that $\varphi \equiv \varphi(\psi_1 \mapsto \psi_2)$ and*

$$mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) \leqslant mem_{AST}(\varphi)$$

Proof Sketch. [2] The proof case splits over each rewrite rule and applies Lemma 1 and Lemma 2 to show that each rule causes a memory reduction, where we know from Theorem 1 that each rule maintains semantics. We show the proof for (R2) to illustrate, since the rule increases the resulting formula size by 1 and does not tighten the propagation delays of the top-level node:

First, let $\psi_1 = \square_{[l_1,u_1]}\varphi_1 \wedge \square_{[l_2,u_2]}\varphi_2$ and $\psi_2 = \square_{[l_3,u_3]}(\square_{[l_1-l_3,u_1-u_3]}\varphi_1 \wedge \square_{[l_2-l_3,u_2-u_3]}\varphi_2)$ where $l_1 \leqslant u_1$, $l_2 \leqslant u_2$, $l_3 = min(l_1, l_2)$, $u_3 = l_3 + min(u_1 -$

[2] The full proof can be found at https://temporallogic.org/research/FMICS2023/.

$l_1, u_2 - l_2)$, and $l_3 < u_3$. We show that

$$\begin{aligned}\psi_1.wpd =& max(\varphi_1.wpd + u_1, \varphi_2.wpd + u_2)\\ =& max(\varphi_1.wpd + u_1 + (u_3 - u_3), \varphi_2.wpd + u_2 + (u_3 - u_3))\\ =& max(\varphi_1.wpd + u_3 + (u_1 - u_3), \varphi_2.wpd + u_3 + (u_2 - u_3))\\ =& u_3 + max(\varphi_1.wpd + (u_1 - u_3), \varphi_2.wpd + (u_2 - u_3))\\ =& \psi_2.wpd.\end{aligned}$$

Therefore we have $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leqslant mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2. A similar derivation is used to show that $\psi_1.bpd = \psi_2.bpd$, so $mem_{node}(\wedge_1) \geqslant mem_{node}(\Box_{[l_3,u_3]})$ by Lemma 1 where $\wedge_1, \wedge_2$ denote the $\wedge$-nodes in ψ_1, ψ_2 respectively.

Next we show that $mem_{AST}(\psi_1) \geqslant mem_{AST}(\psi_2)$. First, we see that because $l_3 < u_3$:

$$\begin{aligned}mem_{node}(\Box_{[l_1,u_1]}) =& ((\varphi_2.wpd + u_2) - (\varphi_1.bpd + l_1) + 1)\\ >& ((\varphi_2.wpd + (u_2 - u_3)) - (\varphi_1.bpd + l_1 - l_3) + 1)\\ =& ((\varphi_2.wpd + u_2) - (\varphi_1.bpd + l_1) + 1) + (l_3 - u_3)\\ =& mem_{node}(\Box_{[l_1-l_3,u_1-u_3]}).\end{aligned}$$

Similarly, $mem_{node}(\Box_{[l_2,u_2]}) \geqslant mem_{node}(\Box_{[l_1-l_3,u_1-u_3]})$. Then

$$\begin{aligned}mem_{AST}(\psi_1) =& mem_{node}(\wedge_1) + mem_{node}(\Box_{[l_1,u_1]}) + mem_{node}(\Box_{[l_2,u_2]})\\ & mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2)\\ \geqslant& mem_{node}(\Box_{[l_3,u_3]}) + mem_{node}(\Box_{[l_1,u_1]}) + mem_{node}(\Box_{[l_2,u_2]})\\ & mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2)\\ \geqslant& mem_{node}(\Box_{[l_3,u_3]}) + mem_{node}(\Box_{[l_1-l_3,u_1-u_3]}) + 1 +\\ & mem_{node}(\Box_{[l_2-l_3,u_2-u_3]}) + mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2)\\ =& mem_{node}(\Box_{[l_3,u_3]}) + mem_{node}(\Box_{[l_1-l_3,u_1-u_3]}) + mem_{node}(\wedge_2)\\ & mem_{node}(\Box_{[l_2-l_3,u_2-u_3]}) + mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2)\\ =& mem_{AST}(\psi_2).\end{aligned}$$

Combining Lemma 2 and the previous result, we have that $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) \leqslant mem_{AST}(\varphi)$ for (R2). The rest of the rules rely on either tightening the propagation delay or reducing the number of nodes in the rewritten AST in order to reduce memory requirements. □

4 Realizing Self-Reference Via Slot-Based MLTL Encoding

When monitoring formulas over sets, one area of concern deals with self-reference within a formula. The requirement "Every file that gets opened eventually gets closed" is a simple classical example that illustrates this concern [20].

One formalization of this requirement using First-order Logic is

$$\forall f.open(f) \rightarrow \Diamond close(f) \tag{5}$$

where the predicate $close(f)$ refers to an object that is referenced earlier in the specification (i.e., f in the predicate $open(f)$). While some have attempted to resolve this issue in runtime monitoring, wherein reference variables are dynamically instantiated during system execution [13], there is no way to ensure that these predicates refer to the same f in general without infinite space to accommodate an unbounded number of dynamically instantiated monitors. We present slot-based monitoring as an alterative technique to ensure consistency of an object's identity over the course of a formula evaluation within finite space.

To this end, we shift from reasoning about objects themselves to reasoning about an underlying data structure and introduce the notion of a "slot" that tracks the data necessary for evaluation of an MLTL formula.

Definition 5 (*Slot*). *A* slot S_m *is a set of atomic proposition symbols, has a slot ID* $m \in \mathbb{N}$*, has an object ID* $id(S_m) \in \mathbb{N} \cup \perp$*, and has a "no change" proposition* nc_m *that is true at time* i *if and only if* $id(S_m)$ *has not changed between time* $i-1$ *and* i*.*

We include the object ID number to track the identity of the contents in the slot as most real-world objects will have some identifier that can be encoded as a natural number. For a finite set of slots $\mathcal{S}$ and $m \in [1, |\mathcal{S}|]$, we rename each proposition $p \in S_m$ to p_m and add it to $\mathcal{AP}$. If S_m is empty at time i, then $id(S_m) = \perp$ and none of its propositions p_m are in $\pi[i]$ for any trace π.

As an example, consider a slot $S_1 = \{open_1, close_1, nc_1\}$ and formula $open_1 \rightarrow \Diamond_{[0,t]} close_1$. Intuitively, this formula should evaluate to true at time step j if S_1 is empty, $open$ is not true for the object in S_1 at time j, or $open$ is true for the object in S_1 at time j and $close$ is true for the object S_1 for some time in $[j, j+t]$.

To formalize this notion of enforcing the consistency of an object in a slot S, we define a function h that adds this constraint to a "self-referential" MLTL formula using the available proposition nc_m. Let $l, u \in \mathbb{N}_0$ such that $l \leqslant u$ and $0 < u$ and φ be an MLTL formula. Then h is defined recursively by:

- $h(m, p) = p$
- $h(m, \neg\varphi) = \neg h(m, \varphi)$
- $h(m, \psi \vee \xi) = h(m, \psi) \vee h(m, \xi)$
- $h(m, \Box_{[l,u]}\varphi) = \Box_{[1,u]} nc_m \wedge \Box_{[l,u]} h(m, \varphi)$
- $h(m, \Diamond_{[l,u]}\varphi) = \Box_{[1,l-1]} nc_m \wedge nc_m \,\mathcal{U}_{[l,u]}\, (nc_m \wedge h(m, \varphi))$ if $l \geqslant 2$, otherwise
- $h(m, \Diamond_{[l,u]}\varphi) = nc_m \,\mathcal{U}_{[l,u]}\, (nc_m \wedge h(m, \varphi))$
- $h(m, \psi \,\mathcal{U}_{[l,u]}\, \xi) = \Box_{[1,l-1]} nc_m \wedge ((nc_m \wedge h(m, \psi)) \,\mathcal{U}_{[l,u]}\, (nc_m \wedge h(m, \xi)))$ if $l \geqslant 2$, otherwise
- $h(m, \psi \,\mathcal{U}_{[l,u]}\, \xi) = (nc_m \wedge h(m, \psi)) \,\mathcal{U}_{[l,u]}\, (nc_m \wedge h(m, \xi))$

Theorem 3 (*Slot-based MLTL Encoding Correctness*). *Let* S_m *be a slot,* φ *be an MLTL formula, and* π *be a finite trace. Then* $\pi \models h(m, \varphi)$ *if and only if* $\pi \models \varphi$ *and the object ID of slot* S_m *does not change while* φ *is evaluated.*

Proof Sketch. We prove via induction on the form of φ that each sub-formula of $h(m, \varphi)$ maintains the semantics of φ and enforces that the object in S_m does not change until φ is evaluated. For a finite trace π:

- If $\varphi = p$, $\varphi = \neg\psi$ or $\varphi = \varphi_1 \vee \varphi_2$, then $h(m, \varphi)$ does not alter the φ's semantics.
- If $\varphi = \Box_{[l,u]}\psi$, then $h(m, \varphi)$ evaluates whether both $\pi \models \Box_{[l,u]}\psi$ and the object ID in S_m does not change between the current timestamp and u.
- If $\varphi = \Diamond_{[l,u]}\psi$, then $h(m, \varphi)$ evaluates whether both $\pi \models \Diamond_{[l,u]}\psi$ (implied by the semantics of $\mathcal{U}_I$) and the object ID in S_m does not change between the current timestamp and $l - 1$ as well as until either $\pi \models \psi$ or $\pi \not\models \Diamond_{[l,u]}\psi$.
- If $\varphi = \psi \, \mathcal{U}_{[l,u]} \, \xi$, then $h(m, \varphi)$ evaluates whether both $\pi \models \psi \, \mathcal{U}_{[l,u]} \, \xi$ and the object ID in S_m does not change between the current timestamp and $l - 1$ as well as until either $\pi \models \psi$ or $\pi \not\models \Diamond_{[l,u]}\xi$.

Note the constraints h places on l and u, where we can remove any temporal operator with an upper bound of 0. Further, we only reason about nc_m starting at time 1 since nc_m tracks whether the object has changed since the *previous* time step, so it is valid for the object to have changed before we start monitoring φ. □

5 Realizing Unboundedness Via Dynamic Set Specification Unrolling

Another area of concern for set-based reasoning is unboundedness. For the file requirement mentioned in Section 4, the set of files being monitored is, as currently stated, unbounded. We argue that this set is never truly unbounded on a real-world system – every system has a bound on the number of files that can be open at one time, for instance, and in most cases these bounds are reasonable. To leverage these bounds, we formalize the notion of a set that changes during system execution then encode MLTL specifications over such "dynamic" sets.

Definition 6 (*Domain-bounded Dynamic Sets*). *A* domain-bounded dynamic set *(DBDS) is a set whose membership may change over time and has a maximum size $n \in \mathbb{N}_0$, for which n is derived from the application-domain in which the dynamic set is being used.*

In the context of our illustrative file system example, this definition captures both that the set of files open on a system may change over time as well as that there exists a bound on the maximum number of files that can be open at once (either because the bound exists in the system or the system runs out of memory, where the latter of which likely points to a design flaw).

We can then leverage this construct to efficiently encode properties of DBDSs in MLTL.

Definition 7 (*DBDS Specifications*). *A* DBDS Specification *is an MLTL formula φ applied to a DBDS where π models the specification if and only if $\pi \models \varphi$ for each object in the DBDS and each each object in the DBDS does not leave the DBDS until after φ is evaluated for that object for a trace π and time i.*

Naturally, for a DBDS D with max size n, we introduce n slots to track the objects in D over time. This allows us to express MLTL formulas over the set of all objects in D at a given time.

Definition 8 (*MLTL Encoding of DBDS Specifications*). *Let D be a DBDS with maximum size n, $\mathcal{S} = \{S_1, \cdots, S_n\}$ be a set of n slots such that $id(d) = id(S_m)$ for each $d \in D$ and some $m \in [1, n]$, and φ be an MLTL formula. The* MLTL Encoding *of φ applied to D is defined as:*

$$\bigwedge_{i \in [1,n]} h(i, \varphi).$$

To continue the file system example, if we assume that there can only be a maximum of n files open at once and the underlying system arbitrarily places files into empty slots as files open and removes them once they close, then we can encode the requirement that an open file must close within t time steps as:

$$\bigwedge_{i \in [1,n]} (h(i, open \rightarrow \Diamond_{[0,t]} close)) = \bigwedge_{i \in [1,n]} (open_i \rightarrow (nc_i \, \mathcal{U}_{[0,t]} \, close_i)).$$

Theorem 4 (*DBDS Specification Correctness*). *Let D be a DBDS with maximum size n, $\mathcal{S} = \{S_1, \cdots, S_n\}$ be a set of n slots such that $id(d) = id(S_m)$ for each $d \in D$ and some $m \in [1, n]$, and φ be an MLTL formula, and π be a finite trace. Then π models φ applied to D (Def. 7) if and only if $\pi \models \bigwedge_{i \in [1,n]} h(i, \varphi)$ (Def. 8).*

Proof Sketch. We apply Theorem 3 over each $S_m \in \mathcal{S}$ to show that $h(m, \varphi)$ accurately monitors φ and ensures a consistent object across timestamps during evaluation of φ. Because we assume the objects in D are correctly reflected in the set of slots $\mathcal{S}$, π models the conjunction of each $h(m, \varphi)$ over $m \in [1, n]$ if and only if each object $d \in D$ satisfies φ at time i and d does not leave D until φ is evaluated. □

Theorem 5 (*DBDS Specification Time Complexity*). *The evaluation of the MLTL encoding of a DBDS specification φ has a time complexity of $\mathcal{O}(\log_2 \log_2 max(p, t) \cdot d \cdot n)$ where p is the maximum worst-case propagation delay of all nodes in $AST(\varphi)$, d is the depth of $AST(\varphi)$, $t \in \mathbb{N}_0$ is the timestamp φ is evaluated for (i.e., the trace π_t), and n is the maximum size of the set.*

Proof. We know that the time complexity for evaluating a given MLTL formula is $\mathcal{O}(\log_2 \log_2 max(p, t) \cdot d)$ [32]. Therefore, since the encoding has n conjunctive/disjunctive clauses and each clause has a time complexity of $\mathcal{O}(\log_2 \log_2 max(p, t) \cdot d)$, the proof follows. □

Theorem 6 (*DBDS Specification Space Complexity*). *The MLTL encoding of a dynamic set specification φ has a space complexity of $\mathcal{O}((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p) \cdot n)$ where p is the maximum worst-case propagation delay of all nodes in $AST(\varphi)$, m is the number of binary operations in the AST, $t \in \mathbb{N}_0$ is the timestamp φ is evaluated for (i.e., the trace π_t), and n is the maximum size of the set.*

Proof. We know that the space complexity for encoding an MLTL formula is $\mathcal{O}((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p))$ [32]. Therefore, since the encoding has n conjunctive/disjunctive clauses and each clause has a space complexity of $\mathcal{O}((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p))$, the proof follows. □

6 Realizing Counting Via Domain-Bounded Dynamic Sets

It has been shown that LTL cannot express "counting" properties [39] e.g., that an event must occur every n time steps and may or may not occur during any other time step. Counting in temporal logics allow specifications to reason over buffers, queues, and FIFOs [35] as well as perform multi-agent planning [33]. Variants of LTL with counting have been proposed that sacrifice decidability [31], but it has been shown that Metric Temporal Logic on finite words is elementarily decidable with counting and interval bounds on Until [21]. By leveraging system constraints, we can tractably encode specifications such as "No more than k tasks are active in the scheduler at once" in MLTL.

We use DBDSs to encode counting specifications of bounded systems, particularly specifications of the form "exactly k of the elements in D satisfy φ."

Definition 9 (*Counting DBDS Specifications*). *A* Counting DBDS Specification *with parameter k is an MLTL formula φ applied to a DBDS where π models the specification if and only if $\pi \models \varphi$ for exactly k objects in the DBDS and each object in the DBDS does not leave the DBDS until after φ is evaluated for that object in π.*

These specifications can be encoded in MLTL by enumerating all $\binom{n}{k}$ possible ways in which k φ-clauses hold at one time (called an *enumeration clause*).

Definition 10 (*MLTL Encoding of Counting DBDS Specifications*). *Let D be a DBDS with maximum size n, $\mathcal{S} = \{S_1, \cdots, S_n\}$ be a set of n slots such that $id(d) = id(S_m)$ for each $d \in D$ and some $m \in [1, n]$, φ be an MLTL formula, and $\mathcal{P}_k(X)$ be the subset of the powerset of X such that $|P| = k$ for each $P \in \mathcal{P}_k(X)$. The* MLTL Encoding *of "exactly k" φ applied to D is defined as:*

$$\bigvee_{X \in \mathcal{P}_k([1,n])} \left(\bigwedge_{i \in X} h(i, \varphi) \wedge \bigwedge_{i \in [1,n] \setminus X} \neg h(i, \varphi) \right).$$

Theorem 7 (*Counting Specification Correctness*). *Let D be a DBDS with maximum size n, $\mathcal{S} = \{S_1, \cdots, S_n\}$ be a set of n slots such that $id(d) = id(S_m)$ for each $d \in D$ and some $m \in [1, n]$, φ be an MLTL formula, π be a finite trace, and $\mathcal{P}_k(X)$ be the subset of the powerset of X such that $|P| = k$ for each $P \in \mathcal{P}_k(X)$. Then π models φ applied to D with parameter k (Def. 9) if and only if*

$$\pi \models \bigvee_{X \in \mathcal{P}_k([1,n])} \left(\bigwedge_{i \in X} h(i, \varphi) \wedge \bigwedge_{i \in [1,n] \setminus X} \neg h(i, \varphi) \right) \textit{(Def. 10).}$$

Proof Sketch. The proof sketch follows a similar structure to the proof for Theorem 4: we apply Theorem 3 over each $S_m \in \mathcal{S}$ to show that $h(m, \varphi)$ accurately monitors φ and ensures a consistent object across timestamps during evaluation of φ. Because we assume the objects in D are correctly reflected in the set of slots $\mathcal{S}$, π models the disjunction of enumeration clauses $\bigwedge_{i \in X} h(i, \varphi) \wedge \bigwedge_{i \in [1,n] \setminus X} \neg h(i, \varphi)$ over $X \in \mathcal{P}_k([1, n])$ if and only if exactly k objects in D satisfy φ and each object does not leave D until φ is evaluated. □

We can then further encode counting specifications of the form "at least k of the elements in D satisfy φ" using the encoding:

$$\bigvee_{X \in \mathcal{P}_k([1,n]) \cup \mathcal{P}_{k+1}([1,n]) \cup \cdots \cup \mathcal{P}_n([1,n])} \left(\bigwedge_{i \in X} h(i,\varphi) \wedge \bigwedge_{i \in [1,n] \setminus X} \neg h(i,\varphi) \right)$$

where we enumerate all possible ways in which at least k enumeration-clauses are true at one time. While these encodings cause a factorial blowup in the number of terms in the MLTL formula (i.e., $\binom{n}{k}$ enumeration clauses for "exactly k" specifications), the value of $\binom{n}{k}$ is tractable in practice for previously published collections of real-world MLTL specifications [7,8,16,2,25,15,4].

Theorem 8 (*Counting Specification Time Complexity*). *The MLTL encoding of a counting specification φ has a space complexity of $\mathcal{O}\left((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p) \cdot \binom{n}{k}\right)$ where p is the maximum worst-case propagation delay of all nodes in $AST(\varphi)$, m is the number of binary operations in the AST, $t \in \mathbb{N}_0$ is the timestamp φ is evaluated for (i.e., the trace π_t), n is the maximum size of the set, and k is number of objects counted in φ.*

Proof. We know that the time complexity for evaluating a given MLTL formula is $\mathcal{O}(\log_2 \log_2 max(p,t) \cdot d)$ [32]. Therefore, since the encoding has $\binom{n}{k}$ clauses and each clause has a time complexity of $\mathcal{O}\left(\log_2 \log_2 max(p,t) \cdot d\right)$, the proof follows. □

Theorem 9 (*Counting Specification Space Complexity*). *The MLTL encoding of a counting specification φ has a space complexity of $\mathcal{O}\left((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p) \cdot \binom{n}{k}\right)$ where p is the maximum worst-case propagation delay of all nodes in $AST(\varphi)$, m is the number of binary operations in the AST, $t \in \mathbb{N}_0$ is the timestamp φ is evaluated for (i.e., the trace π_t), n is the maximum size of the set, and k is number of objects counted in φ.*

Proof. We know that the space complexity for encoding an MLTL formula is $\mathcal{O}((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p))$ [32]. Therefore, since the encoding has $\binom{n}{k}$ conjunctive/disjunctive clauses and each clause has a space complexity of $\mathcal{O}\left((2 + \lceil \log_2(t) \rceil) \cdot (2 \cdot m \cdot p)\right)$, the proof follows. □

Count Operator Encoding It is worth noting that these specifications can be encoded much more efficiently in practice by adding a *count* operator. We define

$$count(\varphi_1, \cdots, \varphi_n, k)$$

to take n MLTL formulas and a natural k such that $count$ is true if and only if exactly k of the φ_i formulas for $i \in [1, n]$ are true. Then, we can encode counting specifications in MLTL as: $count(h(1,\varphi), \cdots, h(n,\varphi), k)$. This encoding is similar to *dynamic set specifications* in that it grows linearly with the maximum size of the dynamic set, though is not pure MLTL.

7 Applying MLTL Rewrite Rules to DBDS Specifications

Memory encoding size is a important metric for monitoring MLTL formulas. This is especially true for real-time, resource-constrained systems, which are natural targets for MLTL monitoring [17]. The techniques presented in Section 5,6 place a hard bound on the memory requirements for specifications over sets, and the rules in Section 3 reduce the overall memory signature of an encoded MLTL formula. These approaches are especially powerful when used in tandem, rewriting MLTL-encoded DBDS specifications.

To illustrate, consider a request arbiter that receives requests and either grants, rejects, or delays them and assume that the arbiter can only ever handle a maximum of 100 requests at once. Then say we want to monitor the requirement that for each active request R, R shall be either granted (g) or rejected (r) within 20 seconds otherwise R will be delayed (d) within 10 seconds and be granted or rejected within 20 seconds of being delayed. We formalize this in MLTL by introducing slots $S_1, \cdots, S_{100}$ such that active requests are arbitrarily assigned an empty slot during system execution:

$$\bigwedge_{i \in [1,100]} h(i, \Diamond_{[0,20]}(g \vee r) \vee \Diamond_{[0,10]}(d \wedge \Diamond_{[0,20]}(g \vee r))). \tag{6}$$

According to Equation 2, encoding the MLTL formula given as an argument to h as a monitor requires space for at least 82 verdicts. We can apply (R2) to rewrite this MLTL formula:

$$\bigwedge_{i \in [1,100]} h(i, \Diamond_{[0,10]}(\Diamond_{[0,10]}(g \vee r) \vee (d \wedge \Diamond_{[0,20]}(g \vee r)))). \tag{7}$$

Evaluating this formula requires space for only 62 verdicts. If we assume a verdict size of 33 bits (32 for the timestamp, 1 for the result), Equation 6 requires memory on the order of $(1 + 32) \times 100 \times 82 = 270,600$ bits (33.825 KB). Automatically applying (R2) therefore saves memory on the order of $(1 + 32) \times 100 \times (82 - 62) = 66,000$ bits (8.25 KB). This is a meaningful improvement for some resource-constrained systems, which only have on the order of KBs to dedicate towards runtime monitoring [28].

A non-expert specification author is unlikely to recognize that Equation 6 is more memory-intensive to monitor than Equation 7, so automating the application of rewrite rules aids in such optimizations. Additionally, applying rewrite rules to DBDS specifications saves n-times the memory as compared to applying the rules to a single MLTL formula, showing the power of pairing these two techniques in practice.

8 Impacts and Future Work

We argue that a large class of "intractable" specifications have corresponding temporal logic patterns based off of our novel constructions. By leveraging MLTL rewriting rules and encodings for dynamic set specifications, we enable the expression and memory optimization of "intractable" specification patterns. Systems that feature domain constraints (e.g., real-time systems) can leverage these encodings to express specifications that otherwise may have required higher-order logics.

Further investigation into optimizations for MLTL encodings is warranted, for instance, by deriving tighter bounds for the propagation delays of AST nodes. While we presented an instance of removing vacuous sub-formulas that can tighten these bounds (R5), generalized techniques for detecting vacuity may help tighten these bounds further. Similarly, an algorithm for detecting the optimal ordering of rewrite rules may reduce memory requirements further.

References

1. Alur, R., Henzinger, T.A.: Real-time logics: complexity and expressiveness. Information and Computation **104**(1), 35–77 (1993)
2. 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/10.1007/978-3-031-06773-0_45
3. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Lectures on Runtime Verification, pp. 1–33. Springer (2018)
4. Cauwels, M., Hammer, A., Hertz, B., Jones, P., Rozier, K.Y.: Integrating Runtime Verification into an Automated UAS Traffic Management System. In: Proceedings of DETECT: international workshop on moDeling, vErification and Testing of dEpendable CriTical systems. Communications in Computer and Information Science (CCIS), Springer, L'Aquila, Italy (September 2020). https://doi.org/10.1007/978-3-030-59155-7_26
5. Cerrito, S., Mayer, M.C., Praud, S.: First order linear temporal logic over finite time structures. In: Logic for Programming and Automated Reasoning: 6th International Conference, LPAR'99 Tbilisi, Georgia, September 6–10, 1999 Proceedings 6. pp. 62–76. Springer (1999)
6. Chen, Y., Zhang, X., Li, J.: Finite quantified linear temporal logic and its satisfiability checking. In: Artificial Intelligence Logic and Applications: The 2nd International Conference, AILA 2022, Shanghai, China, August 26–28, 2022, Proceedings. pp. 3–18. Springer (2022)
7. Dabney, J.B., Badger, J.M., Rajagopal, P.: Adding a verification view for an autonomous real-time system architecture. In: Proceedings of SciTech Forum. p. Online. 2021-0566, AIAA (January 2021). https://doi.org/https://doi.org/10.2514/6.2021-0566
8. Dabney, J.B.: Using assume-guarantee contracts in autonomous spacecraft. Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=zrtyiyNf674 (February 2021)
9. Dabney, J.B., Rajagopal, P., Badger, J.M.: Using assume-guarantee contracts for developmental verification of autonomous spacecraft. Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=HFnn6TzblPg (February 2022)
10. Duret-Lutz, A.: Manipulating LTL formulas using Spot 1.0. In: Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13). Lecture Notes in Computer Science, vol. 8172, pp. 442–445. Springer, Hanoi, Vietnam (Oct 2013). https://doi.org/10.1007/978-3-319-02444-8_31
11. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Gbaguidi Aisse, A., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., et al.: From spot 2.0 to spot 2.10: What's new? In: Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part II. pp. 174–187. Springer (2022)
12. Havelund, K., Reger, G.: Runtime verification logics a language design perspective. Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday pp. 310–338 (2017)
13. Havelund, K., Reger, G., Thoma, D., Zălinescu, E.: Monitoring events that carry data. Lectures on Runtime Verification: Introductory and Advanced Topics pp. 61–102 (2018)
14. He, X., Lee, J.A.N.: Integrating predicate transition nets with first order temporal logic in the specification and verification of concurrent systems. Form. Asp. Comput. **2**(1), 226–246 (mar 1990). https://doi.org/10.1007/BF01888226, https://doi.org/10.1007/BF01888226
15. Hertz, B., Luppen, Z., Rozier, K.Y.: Integrating runtime verification into a sounding rocket control system. In: Proceedings of the 13th NASA Formal Methods Symposium (NFM 2021) (May 2021), available online at http://temporallogic.org/research/NFM21/
16. Kempa, B., Johannsen, C., Rozier, K.Y.: Improving Usability and Trust in Real-Time Verification of a Large-Scale Complex Safety-Critical System. Ada User Journal **September** (2022)

17. 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). pp. 196–214. Lecture Notes in Computer Science (LNCS), Springer, Vienna, Austria (September 2020), http://research.temporallogic.org/papers/KZJZR20.pdf
18. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: Embedding online runtime verification for fault disambiguation on robonaut2. In: Under Submission. TBD (2021)
19. Kessler, F.B.: nuXmv 1.1.0 (2016-05-10) Release Notes. https://es-static.fbk.eu/tools/nuxmv/downloads/NEWS.txt (2016)
20. Khoury, R., Halle, S.: Tally keeping-ltl: An ltl semantics for quantitative evaluation of ltl specifications. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI). pp. 495–502. IEEE Computer Society, Los Alamitos, CA, USA (jul 2018). https://doi.org/10.1109/IRI.2018.00079, https://doi.ieeecomputersociety.org/10.1109/IRI.2018.00079
21. Krishna, S.N., Madnani, K., Pandya, P.K.: Metric temporal logic with counting. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures. pp. 335–352. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
22. Kuperberg, D., Brunel, J., Chemouil, D.: On finite domains in first-order linear temporal logic. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis. pp. 211–226. Springer International Publishing, Cham (2016)
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 2019). LNCS, Springer, New York, NY, USA (July 2019)
24. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability checking for mission-time ltl (mltl). Information and Computation **289**, 104923 (2022)
25. 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 (May 2022). https://doi.org/10.1007/978-3-031-06773-0_28
26. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pp. 152–166. Springer (2004)
27. Noureddine, M.A., Zaraket, F.A.: Model checking software with first order logic specifications using aig solvers. IEEE Transactions on Software Engineering **42**(8), 741–763 (2016). https://doi.org/10.1109/TSE.2016.2520468
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. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: Cassez, F., Jard, C. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
30. Piribauer, J., Baier, C., Bertrand, N., Sankur, O.: Quantified linear temporal logic over probabilistic systems with an application to vacuity checking. In: CONCUR 2021-32nd International Conference on Concurrency Theory. pp. 1–18 (2021)
31. Regis, G., Degiovanni, R., D'Ippolito, N., Aguirre, N.: Specifying event-based systems with a counting fluent temporal logic. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 733–743 (2015). https://doi.org/10.1109/ICSE.2015.86

32. 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 (April 2014)
33. Sahin, Y.E., Nilsson, P., Ozay, N.: Multirobot coordination with counting temporal logics. IEEE Transactions on Robotics **36**(4), 1189–1206 (2020). https://doi.org/10.1109/TRO.2019.2957669
34. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for büchi automata with applications to temporal logic. Theoretical Computer Science **49**(2-3), 217–237 (1987)
35. Sistla, A., Clarke, E., Francez, N., Meyer, A.: Can message buffers be axiomatized in linear temporal logic? Information and Control **63**(1), 88–112 (1984). https://doi.org/https://doi.org/10.1016/S0019-9958(84)80043-1, https://www.sciencedirect.com/science/article/pii/S0019995884800431
36. Software, F.: Setting the Open File Limit (Linux/Unix). Online: https://docs.revenera.com/fnci6133/Content/helplibrary/Setting_the_Open_File_Limit__Linux_Unix_.html (December 2019)
37. Song, F., Wu, Z.: Extending temporal logics with data variable quantifications. In: 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2014)
38. Vardi, M.Y.: Branching vs. linear time: Final showdown. In: Tools and Algorithms for the Construction and Analysis of Systems: 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001 Proceedings. pp. 1–22. Springer (2001)
39. Wolper, P.: Temporal logic can be more expressive. Information and control **56**(1-2), 72–99 (1983)