Scalable MLTL Runtime Monitoring and Satisfiability via Bit-Vector Encoding

Christopher Johannsen
Iowa State University
Ames, United States
cgjohann@iastate.edu

Phillip H. Jones
Iowa State University
Ames, United States
phjones@iastate.edu

Kristin Yvonne Rozier
Iowa State University
Ames, United States
kyrozier@iastate.edu

Tichakorn Wongpiromsarn
Iowa State University
Ames, United States
nok@iastate.edu

***Abstract*—We introduce a novel encoding for Mission-time Linear Temporal Logic (MLTL), inspired by the recently-proposed Temporal Exponential Propagation method for learning finite-trace Linear Temporal Logic (LTLf) formulas. Our encoding translates MLTL formulas into bit-vector expressions whose size is $\mathcal{O}(nlog^2(K))$ with n and K being proportional to the formula size and largest temporal interval present in the formula respectively. We demonstrate how this encoding can be leveraged for two core applications: satisfiability checking and runtime monitoring. For satisfiability, our approach complements existing techniques by efficiently solving instances that are challenging for traditional encodings. For runtime monitoring, we show that our method achieves significantly higher throughput than current state-of-the-art tools, enabling faster and more scalable runtime verification in time-sensitive systems. Experimental results confirm that the proposed encoding and monitoring algorithm provide a practical and effective foundation for reasoning about MLTL properties, achieving 10.52 times higher throughput compared to state-of-the-art online techniques over a sample of pattern-instantiated formulas.**

I. Introduction

Modern high-throughput systems, such as those found in autonomous vehicles, networking, and large-scale industrial automation, must process and react to vast data streams with minimal latency. In these settings, correctness is often time-sensitive, and verification must occur at the system's speed. Traditional techniques for specifying and monitoring temporal properties frequently become performance bottlenecks, unable to scale with the volume and velocity of data [1], [2]. As systems grow more complex and their responsiveness more critical, the need for lightweight, efficient reasoning frameworks that can keep up with high-speed streams becomes increasingly urgent.

Mission-time Linear Temporal Logic (MLTL) [3] is well-suited for specifying and monitoring the behavior of real-time systems, offering the ability to express constraints over bounded time intervals. Its expressiveness has led to its adoption in several applications, including cyber-physical systems and safety-critical embedded software [4], [5], [6], [7], [8]. Two fundamental reasoning tasks in MLTL are satisfiability checking, determining whether a formula can be satisfied by any trace; and runtime monitoring, checking whether a given trace satisfies a formula. While there has been progress on MLTL satisfiability, existing methods still struggle to handle all instances effectively [9], highlighting the need for more scalable and efficient techniques for MLTL reasoning.

Stream-based monitoring of temporal logics such as MLTL, Metric Temporal Logic (MTL) [10], [11], Metric Dynamic Logic (MDL) [12], and Metric First-Order Temporal Logic [13] have led to the development of various tools. MonPoly [14] and DejaVu [15] target high-throughput settings, leveraging first-order temporal logics to monitor system behavior through database-like event streams. However, their reliance on a first-order framework makes them less suitable for lightweight, non-first-order specifications. In contrast, tools such as AERIAL [16] and Hydra [17] focus on monitoring MTL and MDL formulas, but Hydra operates exclusively offline. While numerous stream-based monitoring frameworks exist, including Lola/RTLola [18], [19], CoPilot [20], and TeSSLa [21], these support a range of tool-specific, stream-based specification languages that favor expressiveness over performance. In contrast, R2U2 optimizes for real-time performance in tightly-resource-constrained environments [3], [22], [23], [24]. We narrow our focus on the sweet spot in-between: stream-based monitors that target temporal logic specifications, aiming to improve performance in high-throughput, real-time scenarios.

We present a new approach to encoding MLTL formulas inspired by Temporal Exponential Propagation, a recently-proposed technique for learning finite-trace Linear Temporal Logic (LTL_f) formulas [25]. Our approach translates MLTL formulas into bit-vector expressions whose size is logarithmic in the size of the original formula. Our encoding uses the SMT theory of quantifier-free bit-vectors for satisfiability checking, offering a complementary alternative to existing quantified, linear integer arithmetic methods. This encoding enables a portfolio of solvers to solve 62% of pseudo-randomly generated instances faster than previous approaches. For runtime monitoring, the use of branch-free bit-vector operations results in significant throughput gains, making it well-suited for systems with high-frequency data streams. Experimental results show that our approach consistently outperforms the throughput of state-of-the-art MLTL online monitoring techniques, on average by 10.52 times over a set of five pattern-instantiated formulas from literature [26].

Contributions We contribute a novel encoding for MLTL formulas into bit-vector expressions, prove its correctness, and

https://doi.org/

show its complexity is $\mathcal{O}(n \log^2(K))$ in the total number of bit-vector operations with n and K being proportional to the formula size and largest temporal interval present in the formula respectively. We also provide an open-source implementation of both an MLTL satisfiability checker and runtime monitor, named SABRE (Stream Analysis via Bit-wise Reasoning), and an experimental evaluation empirically demonstrating the performance of each [27][1].

Organization Section II introduces preliminaries, including important previously-proved properties of the logic MLTL and the bit-vector operations upon which we build. In Section III, we present the new bit-vector encoding for MLTL formulas along with correctness and complexity proofs. Section IV applies this encoding to the MLTL satisfiability problem and includes corresponding experimental results. In Section V, we apply the approach to stream-based MLTL runtime monitoring, describing the monitoring algorithm and evaluating its performance. Finally, Section VI concludes the paper and outlines directions for future work.

II. Preliminaries

First, let $[l, u] = \{n \in \mathbb{N}_0 \mid l \leq n \leq u\}$ be an inclusive interval where l, u are zero-inclusive naturals ($\mathbb{N}_0$). The size of an interval $[l, u]$ is defined as $|[l, u]| = u - l + 1$. Let T and F denote logical truth and false respectively.

A. Mission-time Linear Temporal Logic (MLTL)

We proceed with defining MLTL [3].

Definition 1 (MLTL Syntax)**.** The syntax of an MLTL formula over a set of atomic propositions $\mathcal{AP}$ where $p \in \mathcal{AP}$ and φ, ψ are MLTL formulas is defined as:

$$p \mid \neg\varphi \mid \varphi \wedge \xi \mid \Diamond_{[l,u]}\varphi \mid \varphi \; \mathcal{U}_{[l,u]} \; \psi$$

A useful metric for bounding the size of transformed formula is the *closure* [9] of an MLTL formula φ, denoted $cl(\varphi)$, which captures the set of syntactic elements of φ (i.e., elements in the DAG of φ), ignoring temporal intervals. We formally define this set as follows:

$$cl(p) = \{p\}, \qquad cl(\neg\varphi) = \{\neg\varphi\} \cup cl(\varphi)$$
$$cl(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup cl(\varphi) \cup cl(\psi)$$
$$cl(\Diamond_{[l,u]}\varphi) = \{\Diamond_{[l,u]}\varphi\} \cup cl(\varphi)$$
$$cl(\varphi \; \mathcal{U}_{[l,u]} \; \psi) = \{\varphi \; \mathcal{U}_{[l,u]} \; \psi\} \cup cl(\varphi) \cup cl(\psi)$$

We define a related notion of temporal closure, denoted $cl_T(\varphi)$, to capture only the temporal operators in the DAG of φ:

$$cl_T(\varphi) = \{\psi \in cl(\varphi) \mid \psi = \Diamond_{[l,u]}\xi \text{ or } \psi = \xi \; \mathcal{U}_{[l,u]} \; \vartheta\}$$

MLTL formulas are interpreted over finite traces. A trace π is a finite sequence of states with length $|\pi| \in \mathbb{N}_0$, where each state defines a set of atomic propositions that are true in that state. We denote the state at timestamp $0 \leq \tau < |\pi|$ by $\pi[\tau] \subseteq \mathcal{AP}$ and π_τ the trace π starting at and including the state at τ. The concatenation of two traces π_1 and π_2 is denoted $\pi_1 \cdot \pi_2$.

[1]Artifact is available at https://doi.org/10.5281/zenodo.16884213

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

$\pi, \tau \models p$ iff $p \in \pi[\tau]$ $\qquad$ $\pi, \tau \models \neg\varphi$ iff $\pi, \tau \not\models \varphi$

$\pi, \tau \models \varphi \wedge \psi$ iff $\pi, \tau \models \varphi$ and $\pi, \tau \models \psi$

$\pi, \tau \models \Diamond_{[l,u]}\varphi$ iff $\tau + l < |\pi|$ and $\pi, i \models \varphi$ for some $i < |\pi|$ such that $l + \tau \leq i \leq u + \tau$

$\pi, \tau \models \varphi \; \mathcal{U}_{[l,u]} \; \psi$ iff $\tau + l < |\pi|$ and $\pi, i \models \psi$ for some $i < |\pi|$ such that $l + \tau \leq i \leq u + \tau$ and $\pi, j \models \varphi$ for all j such that $l + \tau \leq j < i$

We say two MLTL formulas φ, ψ are *semantically equivalent* iff $\pi, \tau \models \varphi \leftrightarrow \pi, \tau \models \psi$ for all finite traces π over $\mathcal{AP}$ and timestamps $0 \leq \tau < |\pi|$. MLTL keeps the standard operator equivalences from LTL, including $\Diamond_{[l,u]}\varphi = (\mathsf{T} \; \mathcal{U}_{[l,u]} \; \varphi)$, $\varphi \; \mathcal{R}_{[l,u]} \; \psi = \neg(\neg\varphi \; \mathcal{U}_{[l,u]} \; \neg\psi)$, and $\Box_{[l,u]}\varphi = (\mathsf{F} \; \mathcal{R}_{[l,u]} \; \varphi)$. To complete the MLTL semantics, we define $\mathsf{F} = \neg\mathsf{T}$, $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, and $\neg\Diamond_{[l,u]}\varphi = \Box_{[l,u]}\neg\varphi$.

We also define the *computation length* [28] of MLTL formulas.

Definition 3 (Computation Length)**.** Let $p \in \mathcal{AP}$ be an atomic proposition and φ, ψ be MLTL formulas. Then the *computation length* of an MLTL formula is defined recursively as follows:

$$cplen(p) = 1 \qquad cplen(\neg\varphi) = cplen(\varphi)$$
$$cplen(\varphi \wedge \psi) = max(cplen(\varphi), cplen(\psi))$$
$$cplen(\Diamond_{[l,u]}\psi) = u + cplen(\psi)$$
$$cplen(\varphi \; \mathcal{U}_{[l,u]} \; \psi) = u + max(cplen(\varphi) - 1, cplen(\psi))$$

This definition captures the notion of the minimum length of trace such that no extension to that length trace could change the satisfaction of the formula. This is similar to the definition of *safety properties*, which states that no extension to any counterexample to the safety property will cause the extended counterexample to satisfy the property — once the property is violated it cannot be un-violated. Similarly, once a trace is of length $cplen(\varphi)$, no extension can change its satisfaction of φ. This is formalized in the following (Corollary `trace_append_iff` of [29]):

Corollary 1 (Computation Length Correctness)**.** *For an MLTL formula φ and trace π with $|\pi| \geq cplen(\varphi)$, appending any suffix π' to π will not affect whether φ holds over π starting at index 0, i.e.,*

$$\pi, 0 \models \varphi \textit{ iff } \forall \pi' . \left((\pi \cdot \pi'), 0 \models \varphi\right).$$

Two problems of interest for MLTL are the *satisfiability* and *monitoring* problems. Satisfiability checking of an MLTL formula φ asks whether there exists a trace π such that $\pi, 0 \models \varphi$. Monitoring an MLTL formula φ over trace π asks to compute $\pi, \tau \models \varphi$ for all τ. We consider *stream-based* monitoring, where the input is an unbounded sequence of states (i.e., a stream) so that the trace considered at time τ is $\pi[0], \pi[1], \ldots, \pi[\tau]$ and the output is a sequence of verdicts corresponding to $(\pi, 0 \models \varphi), (\pi, 1 \models \varphi), \ldots, (\pi, \tau' \models \varphi)$ where $\tau' \leq \tau$. In this context, we ignore the trace length constraints of the MLTL semantics, i.e., if $\pi, \tau \not\models \varphi \; \mathcal{U}_{[l,u]} \; \psi$

trivially holds because $\tau + l \geq |\pi|$, then we choose instead to not produce a verdict for τ since we always expect more states to populate π from the input stream.

B. Bit-vector Operations

We consider bit-vectors as a finite sequence of 1s and 0s with a fixed length. A bit-vector b has a length, denoted $|b|$. We consider standard bit-vector operations, including negation ($\sim$), and ($\&$), or ($|$), left-shift ($\ll$), and or-reduction (`redor`). We write constant bit-vectors in the form `0b[01]+` with the left-most bit being the least-significant bit (LSB). For example, `0b0101` is a bit-vector of length 4 and decimal value 10. We allow indexing of bit-vectors, so that $b[i]$ denotes the ith bit from the left of b (e.g., $b[0]$ is the LSB of b) and $b[i:j]$ denotes the extraction of bits i to j, inclusive, from b with $i \leq j$, producing a bit-vector of width $j - i + 1$. With slight abuse of notation, we also allow propositional operators on bit-vectors of length 1, for example, $b[i] \wedge b[j]$, and also let $\mathsf{T} \equiv 1 \equiv$ `0b1` and $\mathsf{F} \equiv 0 \equiv$ `0b0`. We also define the term $zeroes(w)$ with width $w \in \mathbb{N}_0$ to denote the bit-vector of width w that is 0 at every index, e.g., $zeroes(3) =$ `0b000`. The result of shifting a bit-vector by a value greater than or equal to the width of the bit-vector is the zero bit-vector of that width: $b \ll x = zeroes(|b|)$ if $x \geq |b|$.

III. MLTL Bit-vector Encoding

The intuition for the MLTL bit-vector encoding is to consider a trace as a set of bit-vectors, one for each proposition. Then, each MLTL operator can be encoded as a bit-vector operation e.g., $\neg$ as $\sim$, $\vee$ as $|$, and $\wedge$ as $\&$. We can encode each temporal operator as a sequence of shifts logarithmic in size, but only if the operator's interval's size is a power of 2. We introduce a fragment of MLTL that considers only these operators, then show that any MLTL formula has an equivalent formula in this fragment.

Definition 4. MLTL_2 is the fragment of MLTL formulas where all temporal operators have intervals whose sizes are powers of two, i.e., if $p \in \mathcal{AP}$ and φ and ψ are MLTL_2 formulas then so are

$$p \mid \neg\varphi \mid \varphi \wedge \psi \mid \Diamond_{[l,u]}\varphi \mid \varphi \; \mathcal{U}_{[l,u]} \; \psi$$

where $u - l + 1 = 2^k$ for some $k \in \mathbb{N}_0$. In terms of k, the $\Diamond$ and $\mathcal{U}$ operators must be of the form $\Diamond_{[l,l+2^k-1]}$ and $\mathcal{U}_{[l,l+2^k-1]}$.

Example MLTL_2 formulas include $\Diamond_{[0,7]}p$, $p \; \mathcal{U}_{[2,17]} \; q$, and $\Box_{[5,5]}p$. We now define a function $d()$ for converting MLTL formulas into equivalent MLTL_2 formulas by *decomposing* each temporal formula into a nested sequence of temporal operators, where each operator's interval length is expressed as a sum of powers of two. Given a temporal operator with interval $[l, u]$, let $k = max\{k \in \mathbb{N}_0 \mid l + (2^k - 1) \leq u\}$ be the maximum (one less) power of 2 that is less than or equal to u. Then we define d as follows:

$$d(p) = p \quad d(\neg\varphi) = \neg d(\varphi) \quad d(\varphi \wedge \psi) = d(\varphi) \wedge d(\psi)$$
$$d(\Diamond_{[l,u]}\psi) = \Diamond_{[l,l+(2^k-1)]} d(\Diamond_{[0,(u-l)-(2^k-1)]}\psi)$$
$$d(\psi \; \mathcal{U}_{[l,u]} \; \xi) = \psi \; \mathcal{U}_{[l,l+(2^k-1)]} \; (d(\psi \; \mathcal{U}_{[0,(u-l)-(2^k-1)]} \; \xi))$$

To complete the definition of d and ensure that d terminates, let $d(\Diamond_{[0,0]}\psi) = \psi$ and $d(\psi \; \mathcal{U}_{[0,0]} \; \xi) = \xi$. Intuitively, $d()$ creates a nesting of temporal operators whose sum of lower bounds and sum of upper bounds equals to the original operator's bounds. We also note that since d always uses the largest possible k to decompose each interval, the nested sequence of temporal operators is minimal in size. In fact, the number of introduced temporal operators is equal to the number of 1s in the original interval size's binary representation.

Example 1. Applying d to the following examples, we obtain equivalent MLTL_2 formulas:

$$\begin{aligned}
d(\Diamond_{[0,6]}p) &= \Diamond_{[0,3]} d(\Diamond_{[0,3]}p) \\
&= \Diamond_{[0,3]}(\Diamond_{[0,3]} d(\Diamond_{[0,0]}p)) \\
&= \Diamond_{[0,3]}(\Diamond_{[0,3]}p) \\
d(\Diamond_{[2,15]}p) &= \Diamond_{[2,9]} d(\Diamond_{[0,6]}p) = \Diamond_{[2,9]}(\Diamond_{[0,3]}(\Diamond_{[0,3]}p)) \\
d(p \; \mathcal{U}_{[2,15]} \; q) &= p \; \mathcal{U}_{[2,9]} \; d(p \; \mathcal{U}_{[0,6]} \; q) \\
&= p \; \mathcal{U}_{[2,9]} \; (p \; \mathcal{U}_{[0,3]} \; (d(p \; \mathcal{U}_{[0,3]} \; q))) \\
&= p \; \mathcal{U}_{[2,9]} \; (p \; \mathcal{U}_{[0,3]} \; (p \; \mathcal{U}_{[0,3]} \; d(p \; \mathcal{U}_{[0,0]} \; q))) \\
&= p \; \mathcal{U}_{[2,9]} \; (p \; \mathcal{U}_{[0,3]} \; (p \; \mathcal{U}_{[0,3]} \; q))
\end{aligned}$$

Lemma 1 (Future Equivalence)**.** *For any MLTL formula φ, $\Diamond_{[l_1,u_1]}\Diamond_{[l_2,u_2]}\varphi \equiv \Diamond_{[l_1+l_2,u_1+u_2]}\varphi$.*

Proof. Theorem 1 of [30]. □

Lemma 2 (Until Equivalence)**.** *For any MLTL formulas φ and ψ, $\varphi \; \mathcal{U}_{[l,u_1]} \; (\varphi \; \mathcal{U}_{[0,u_2]} \; \psi) \equiv \varphi \; \mathcal{U}_{[l,u_1+u_2]} \; \psi$.*

Proof. Theorem 3 of [28]. □

Before we proceed with proving the correctness and complexity of $d()$, we define the sequence $S(\varphi)$ for MLTL formula φ as the sequence of values $2^k - 1$ for each of the d-introduced operators given an MLTL formula φ (using $\cdot$ for concatenation and $k = max\{k \in \mathbb{N}_0 \mid l + (2^k - 1) \leq u\}$):

$$\begin{aligned}
S(p) = S(\neg\varphi) = S(\varphi \wedge \psi) &= [\,] \\
S(\Diamond_{[0,0]}\psi) = S(\psi \; \mathcal{U}_{[0,0]} \; \xi) &= [\,] \\
S(\Diamond_{[l,u]}\psi) &= [2^k - 1] \cdot S(\Diamond_{[0,(u-l)-(2^k-1)]}\psi) \\
S(\psi \; \mathcal{U}_{[l,u]} \; \xi) &= [2^k - 1] \cdot S(\psi \; \mathcal{U}_{[0,(u-l)-(2^k-1)]} \; \xi)
\end{aligned}$$

Using the formulas from Example 1, $S(\Diamond_{[0,6]}p) = [3, 3]$ and $S(\Diamond_{[2,15]}p) = S(p \; \mathcal{U}_{[2,15]} \; q) = [7, 3, 3]$.

Lemma 3 (Correctness and Complexity of $d()$)**.** *An MLTL formula φ with maximal temporal interval size K has an equivalent MLTL_2 formula ψ with $|cl(\psi)| \leq |cl(\varphi)| \cdot \log_2(K)$.*

Proof. We show that $d()$ produces such an MLTL_2 formula. The output of $d(\varphi)$ is an MLTL_2 formula by inspection since for each introduced temporal operator, its interval $[l, l+2^k-1]$ has size $(l + 2^k - 1) - l + 1 = 2^k$.

We now show $\varphi \equiv d(\varphi)$ for any MLTL formula φ via structural induction over φ. In the base case, $\varphi = p$ where $d(p) = p$ so that $d(p) \equiv p$. If $\varphi = \neg\psi$ or $\varphi = \psi \wedge \xi$, then $d(\neg\psi) \equiv \neg d(\psi)$ and $d(\psi \wedge \xi) \equiv d(\psi) \wedge d(\xi)$ by the inductive hypotheses $d(\psi) \equiv \psi$ and $d(\xi) \equiv \xi$.

Now, let $\varphi = \Diamond_{[l,u]}\psi$ and inductively assume $d(\psi) \equiv \psi$. Let s_i be the ith value in $S(\varphi)$. Applying $d()$ to φ and using Lemma 1:

$$\begin{aligned} d(\Diamond_{[l,u]}\psi) &= \Diamond_{[l,l+s_1]} d(\Diamond_{[0,(u-l)-s_1]}\psi) \\ &= \Diamond_{[l,l+s_1]}\Diamond_{[0,s_2]}\cdots\Diamond_{[0,s_n]} d(\psi) \\ &\equiv \Diamond_{[l,l+s_1+s_2]}\cdots\Diamond_{[0,s_n]} d(\psi) \\ &\equiv \Diamond_{[l,u]} d(\psi) \equiv \Diamond_{[l,u]}\psi \end{aligned}$$

where $u = l + s_1 + s_2 + \cdots + s_n$ by the definition of $d()$.

We apply a similar argument for $\varphi = \psi\ \mathcal{U}_{[l,u]}\ \xi$, where we inductively assume $d(\psi) \equiv \psi$ and $d(\xi) \equiv \xi$. Applying $d()$ to φ and using Lemma 2:

$$\begin{aligned} & d(\psi\ \mathcal{U}_{[l,u]}\ \xi) \\ &= d(\psi)\ \mathcal{U}_{[l,l+s_1]}\ d(\psi\ \mathcal{U}_{[0,(u-l)-s_1]}\ \xi) \\ &= d(\psi)\ \mathcal{U}_{[l,l+s_1]}\ (d(\psi)\ \mathcal{U}_{[0,s_2]}\ (\ldots d(\psi)\ \mathcal{U}_{[0,s_n]}\ d(\xi))) \\ &\equiv d(\psi)\ \mathcal{U}_{[l,l+s_1+s_2]}\ (\ldots d(\psi)\ \mathcal{U}_{[0,s_n]}\ d(\xi)) \\ &\equiv d(\psi)\ \mathcal{U}_{[l,u]}\ d(\xi) \equiv \psi\ \mathcal{U}_{[l,u]}\ \xi \end{aligned}$$

where $u = l + s_1 + s_2 + \cdots + s_n$ by the definition of $d()$.

We now bound the size of the closure of the formula produced by $d()$ in terms of the largest interval size in φ. Let ψ be the sub-formula of φ with this largest temporal interval size, i.e., the formula associated with $K = max\{|[l,u]| \mid \psi \in cl_T(\varphi)\}$, and let $[l,u]$ be this interval. Applying $d()$ to ψ introduces at most $|S(\psi)| \leq \log_2(K)$ new operators, and since K is maximal, we have $|cl(d(\varphi))| \leq |cl(\varphi)| \cdot \log_2(K)$. □

Note that although $d(\psi\ \mathcal{U}_{[l,u]}\ \xi)$ may introduce copies of ψ, the fact that the closure represents the sub-formulas in the DAG representation of the formula ensures that these copies do not contribute to the complexity of the encoding.

We now define an MLTL bit-vector encoding (MLTL-BV), inspired by the Temporal Exponential Propagation method proposed in [25].

Definition 5 (MLTL-BV Encoding)**.** Let φ, ψ be MLTL$_2$[2] formulas and π be a trace, then we define a bit-vector $bv(\varphi,\pi)$ of width $|\pi|$ as follows where $k \in \mathbb{N}_0$ and using $ite(b,x,y)$ as the if-then-else operator:

$$bv(p,\pi)[\tau] = ite(p \in \pi[\tau], 1, 0) \text{ for } 0 \leq \tau < |\pi|$$

$$bv(\neg\varphi,\pi) = \sim bv(\varphi,\pi)$$

$$bv(\varphi \wedge \psi,\pi) = bv(\varphi,\pi)\ \&\ bv(\psi,\pi)$$

$$\begin{aligned} bv(\Diamond_{[l,l+2^k-1]}\varphi,\pi) = \text{let } & b_0 = bv(\varphi,\pi) \ll l, \\ & b_1 = b_0 \mid (b_0 \ll 1), \\ & b_2 = b_1 \mid (b_1 \ll 2), \\ & b_3 = b_2 \mid (b_2 \ll 4), \\ & \quad \cdots \\ & b_k = b_{k-1} \mid (b_{k-1} \ll 2^{k-1}) \text{ in } b_k \end{aligned}$$

[2]The decomposition via d may not be necessary; in case the interval is a non-power-of-two the final shift may be by the "leftover" amount. Proofs and implementation of this approach is left to future work.

$$\begin{aligned} bv(\varphi\ \mathcal{U}_{[l,l+2^k-1]}\ \psi,\pi) = \text{let } & \\ & b_{\psi_0} = bv(\psi,\pi) \ll l,\ b_{\varphi_0} = bv(\varphi,\pi) \ll l \\ & b_{\psi_1} = b_{\psi_0} \mid (b_{\varphi_0}\&(b_{\psi_0} \ll 1)),\ b_{\varphi_1} = b_{\varphi_0}\&(b_{\varphi_0} \ll 1), \\ & b_{\psi_2} = b_{\psi_1} \mid (b_{\varphi_1}\&(b_{\psi_1} \ll 2)),\ b_{\varphi_2} = b_{\varphi_1}\&(b_{\varphi_1} \ll 2), \\ & b_{\psi_3} = b_{\psi_2} \mid (b_{\varphi_2}\&(b_{\psi_2} \ll 4)),\ b_{\varphi_3} = b_{\varphi_2}\&(b_{\varphi_2} \ll 4), \\ & \quad \cdots \\ & b_{\psi_k} = b_{\psi_{k-1}} \mid (b_{\varphi_{k-1}}\&(b_{\psi_{k-1}} \ll 2^{k-1})) \text{ in } b_{\psi_k} \end{aligned}$$

Example 2. Consider a trace π over $\mathcal{AP} = \{p,q\}$ as follows:

$$\pi = [\{\},\{\},\{p,q\},\{\},\{q\},\{q\},\{p,q\}].$$

Now consider the MLTL formulas p, q, $\neg p$, and $p \wedge q$. We can encode each as a bit-vector using Definition 5 as follows:

$$bv(p,\pi) = \texttt{0b0010001} \qquad bv(q,\pi) = \texttt{0b0010111}$$
$$bv(\neg p,\pi) = \texttt{0b1101110} \quad bv(p \wedge q,\pi) = \texttt{0b0010001}$$

Each bit of each bit-vector represents the truth value of its formula at each timestamp of the trace. For example, $bv(p \wedge q,\pi)[2] = 1$ which corresponds to $\pi, 2 \models p \wedge q$, where p and q both occur at $\pi[2]$.

Example 3. Consider the trace from Example 2 and the MLTL formula $\Diamond_{[1,3]}p$. After first applying $d(\Diamond_{[1,3]}p) = \Diamond_{[1,2]}\Diamond_{[0,1]}p$, we can encode the trace/formula as a bit-vector as follows:

$$bv(p,\pi) = \texttt{0b0010001}$$
$$bv(\Diamond_{[0,1]}p,\pi) = \texttt{0b0110011}$$
$$bv(\Diamond_{[1,2]}\Diamond_{[0,1]}p,\pi) = \texttt{0b1101110}$$

Similarly, consider the same trace and the MLTL formula $q\ \mathcal{U}_{[0,2]}\ p$ where $d(q\ \mathcal{U}_{[0,2]}\ p) = q\ \mathcal{U}_{[0,1]}\ (q\ \mathcal{U}_{[0,1]}\ p)$:

$$bv(p,\pi) = \texttt{0b0010001}$$
$$bv(q,\pi) = \texttt{0b0010111}$$
$$bv(q\ \mathcal{U}_{[0,1]}\ p,\pi) = \texttt{0b0010011}$$
$$bv(q\ \mathcal{U}_{[0,1]}\ (q\ \mathcal{U}_{[0,1]}\ p),\pi) = \texttt{0b0010111}$$

We now show that the encoding captures the MLTL semantics, where each bit of the resulting bit-vector for an input formula corresponds to whether the input trace satisfies the formula at the index of that bit.

Theorem 1 (MLTL-BV Correctness)**.** *Let $\mathcal{AP}$ be a finite set of atomic propositions, φ be an MLTL$_2$ formula over $\mathcal{AP}$, and π be a trace. Then for each $\tau < |\pi|$, the bit at index τ of $bv(\varphi,\pi)$ is 1 if and only if π at τ models φ, i.e.,*

$$bv(\varphi,\pi)[\tau] = 1 \textit{ iff } (\pi,\tau \models \varphi)\,.$$

Proof. We proceed with induction over the structure of φ. For the following, let π be a finite trace and τ be an index such that $\tau < |\pi|$.

In the base case of our structural induction, $\varphi = p$ where $p \in \mathcal{AP}$. By construction of $bv(p,\pi)$, we have that $bv(p,\pi)[\tau] = 1$ iff $p \in \pi[\tau]$, proving the property.

If $\varphi = \neg\psi$ or $\varphi = \psi \wedge \xi$, then the property holds from the correspondence between $\sim$, $\neg$ and $\&$, $\wedge$.

Now let $\varphi = \Diamond_{[l,u]}\psi$ where $u = l + 2^k - 1$. We inductively assume that the property holds for ψ. Fixing $l \geq 0$, we show that for all $k \geq 0$ it holds that

$$bv(\Diamond_{[l,u]}\psi, \pi)[\tau] = 1 \Leftrightarrow \left(\pi, \tau \models \Diamond_{[l,u]}\psi\right).$$

For the base case, $k = 0$ so that $u = l + 2^0 - 1 = l$ and $bv(\Diamond_{[l,l]}\psi, \pi) = bv(\psi, \pi) \ll l$. If $l \geq |\pi|$, then $\pi, \tau \not\models \Diamond_{[l,l]}\psi$ by the MLTL semantics since l is off the end of the trace. By the definition of $\ll$, we also have $bv(\Diamond_{[l,l]}\psi, \pi) = zeroes(|\pi|)$ so that $bv(\Diamond_{[l,l]}\psi, \pi)[\tau] = 0$, proving the case when $l \geq |\pi|$. Otherwise, we have

$$\begin{aligned} bv(\Diamond_{[l,l]}\psi, \pi)[\tau] &= (bv(\psi, \pi) \ll l)\,[\tau] \\ &= bv(\psi, \pi)[\tau + l] \end{aligned}$$

which is 1 only if $\pi, \tau + l \models \psi$ by the MLTL semantics and inductive hypothesis, proving the case when $l < |\pi|$.

For the inductive step, assume that the property holds for k, we show that it holds for $k+1$. Let $b_k = bv(\Diamond_{[l,l+2^k-1]}\psi, \pi)$. Then by Definition 5:

$$\begin{aligned} bv(\Diamond_{[l,l+2^{k+1}-1]}\psi, \pi)[\tau] &= \left(b_k \mid (b_k \ll 2^k)\right)[\tau] \\ &= b_k[\tau] \vee (b_k \ll 2^k)[\tau] \\ &= b_k[\tau] \vee b_k[\tau + 2^k]. \end{aligned}$$

Since the property holds for k, this value is 1 if and only if

$$\begin{aligned} &\exists i \in [l, l + 2^k - 1].\ \pi, (i + \tau) \models \psi \text{ or } \pi, (i + 2^k + \tau) \models \psi \\ \Leftrightarrow\ &\exists i \in [l, l + 2^k - 1 + 2^k].\ \pi, (i + \tau) \models \psi \text{ or } \pi, (i + \tau) \models \psi \\ \Leftrightarrow\ &\exists i \in [l, l + 2^{k+1} - 1].\ \pi, (i + \tau) \models \psi \\ \Leftrightarrow\ &\pi, \tau \models \Diamond_{[l,l+2^{k+1}-1]}\psi \end{aligned}$$

proving the property for $k+1$ and completing the case for $\varphi = \Diamond_{[l,u]}\psi$. The proof for the $\mathcal{U}$ case follows the same approach, see Appendix A for the full proof. □

Theorem 2 (MLTL-BV Complexity)**.** *Let $\mathcal{AP}$ be a finite set of atomic propositions, φ be an MLTL$_2$ formula over $\mathcal{AP}$, K be the size of the largest interval in φ, and π be a trace. Then $bv(\varphi, \pi)$ requires at most $5 \cdot |cl(\varphi)| \cdot \log_2(K)$ bitwise operations to compute.*

Proof. First, let π be a trace and φ be an MLTL formula. Let ψ be the sub-formula of φ with the largest temporal interval size, i.e., the formula associated with $K = max\{|[l, u]| \mid \psi \in cl_T(\varphi)\}$, and let $[l, u]$ be this interval.

By inspection, if $\varphi = p$, $\varphi = \neg\psi$ or $\varphi = \psi \wedge \xi$, then $bv(\varphi, \pi)$ requires one bitwise operation in addition to the operators of its sub-formulas. Otherwise, if $\varphi = \Diamond_{[l,u]}\psi$, then $bv(\varphi, \pi)$ requires an additional $\log_2(K)$ shifts and $\log_2(K-1)$ bitwise-or operations totaling $\log_2(K) + \log_2(K-1) < 2\log_2(K)$ bitwise operations. Alternatively, if $\varphi = \psi\ \mathcal{U}_{[l,u]}\ \xi$, then $bv(\varphi, \pi)$ requires $2\log_2(K)$ shifts, $\log_2(K-1)$ bitwise-or operations, and $2\log_2(K-1)$ bitwise ands totaling $2\log_2(K) + 3\log_2(K-1) < 5\log_2(K)$ bitwise operations. Therefore $bv(\varphi, \pi)$ requires no more than $|cl(\varphi)| \cdot 5\log_2(K)$ bitwise operations in total to compute. □

For an MLTL formula φ, computing $bv(d(\varphi), \pi)$ requires at most $\mathcal{O}(|cl(\varphi)| \cdot (\log_2(K))^2)$ operations since $|cl(d(\varphi))| \leq |cl(\varphi)| \cdot \log_2(K)$.

IV. Satisfiability Checking via MLTL-BV

A. Problem Encoding and Algorithm

We can use the MLTL-BV encoding to solve the satisfiability problem for MLTL, recalling that this poses whether there exists some trace π such that $\pi, 0 \models \varphi$. Using our bit-vector encoding, we can query whether there exists a bit-vector $bv(p)$ for each $p \in AP$ such that there is at least one bit set in $bv(\varphi)$ (i.e., $\texttt{redor}(bv(\varphi)) = 1$). We begin by proving a bound on the size of satisfying traces using Corollary 1.

Lemma 4. *An MLTL formula φ over $\mathcal{AP}$ is satisfiable iff there exists a trace π of length $|\pi| \leq cplen(\varphi)$ such that $\pi, 0 \models \varphi$.*

Proof. ($\Leftarrow$) Assume that there exists a trace π of length $|\pi| \leq cplen(\varphi)$ such that $\pi, 0 \models \varphi$. Then φ is satisfiable.
($\Rightarrow$) Assume φ is satisfiable and let π be a trace such that $\pi, 0 \models \varphi$. If $|\pi| \leq cplen(\varphi)$, then the property holds directly. Otherwise, $|\pi| > cplen(\varphi)$. Then taking the prefix of π with length $cplen(\varphi)$ defined as $\pi' = \pi[0], \pi[1], \ldots, \pi[cplen(\varphi) - 1]$, we have from Corollary 1 that $\pi', 0 \models \varphi$. Since $|\pi'| = cplen(\varphi) \leq cplen(\varphi)$, the property holds. □

Theorem 3. *An MLTL formula φ over $\mathcal{AP}$ is satisfiable iff there exists a trace π with $|\pi| \leq cplen(\varphi)$ such that $\texttt{redor}(bv(\varphi, \pi)) = 1$.*

Proof. ($\Rightarrow$) Assume that φ is satisfiable. From Lemma 4, there exists a finite trace π such that $|\pi| \leq cplen(\varphi)$ and $\pi, 0 \models \varphi$. From Theorem 1, it holds that $bv(\varphi, \pi)[0] = 1$ where the width of $bv(\varphi, \pi)$ is $|\pi| = cplen(\varphi)$, so $\texttt{redor}(bv(\varphi, \pi)) = 1$.
($\Leftarrow$) Assume that there exists a trace π with $|\pi| \leq cplen(\varphi)$ such that $\texttt{redor}(bv(\varphi, \pi)) = 1$. Let $\tau < |\pi|$ be an index of $bv(\varphi, \pi)$ such that $bv(\varphi, \pi)[\tau] = 1$, of which at least 1 such index exists from the definition of redor. Therefore from Theorem 1 it holds that $\pi, \tau \models \varphi$. □

We use this Theorem to encode the MLTL satisfiability problem into an SMTLIB2 [31] query using the theory of quantifier-free bit-vectors (QF_BV). Specifically, given an MLTL formula φ, we declare an uninterpreted bit-vector for each proposition of width equal to $cplen(\varphi)$ (representing a trace π), apply the MLTL-BV encoding to φ by defining a function for each operator in φ as in Definition 5, then assert that the corresponding function for φ is greater than zero (this is the same as redor, since redor is unsupported in the QF_BV logic). Figure 1 shows an example SMTLIB2 encoding for the formula $p \wedge \Diamond_{[1,3]}q$.

Incremental Checking We also contribute an incremental version of this encoding. Notice that the above encoding tries to find a trace of length $cplen(\varphi)$ or less in one shot; in principle we can query whether a trace of any arbitrary length or less satisfies φ by setting the bit-vector width accordingly. Many MLTL formulas are satisfied by much shorter traces than $cplen(\varphi)$. For example, according the to MLTL semantics (Definition 2), the formula $\neg(\Diamond_{2,5}p) \equiv \Box_{[2,5]}\neg p$ is satisfied by any trace of length 1, since p technically never holds in the

```
1 (declare-fun |p| () (_ BitVec 4)) (declare-fun |q| () (_ BitVec 4))
2 (define-fun b_0                   () (_ BitVec 4) (bvshl |q| (_ bv1 4)))
3 (define-fun |F[0,1] q|            () (_ BitVec 4) (bvor b_0 |q|))
4 (define-fun b_1                   () (_ BitVec 4) (bvshl |F[0,1] q| (_ bv3 4)))
5 (define-fun b_2                   () (_ BitVec 4) (bvshl b_1 (_ bv1 4)))
6 (define-fun |F[3,4] F[0,1] q|     () (_ BitVec 4) (bvor b_1 b_2))
7 (define-fun |p & F[3,4] F[0,1] q| () (_ BitVec 4) (bvand |p| |F[1,2] F[0,1] q|))
8 (assert (bvugt |p & F[3,5] q| (_ bv0 4)))
```

Fig. 1: SMTLIB2 encoding for the formula $p \wedge \Diamond_{[3,5]} q$. The encoding uses two uninterpreted bit-vectors for p and q (line 1) and then Definition 5 for the operators (lines 2-7). The assertion checks whether there exist bit-vectors for p and q such that the top-level operator of the formula's bit-vector value could be greater than zero (line 8).

range $[2, 5]$. To leverage this, we first check if bit-vectors of width 1 satisfy the formula φ, then of width equal to φ's best-case propagation delay [4] (intuitively, the minimal sum of the formula's lower bounds), then finally of width $cplen(\varphi)$. We denote this approach as QF_BV *.

B. Experimental Evaluation

We evaluate the encoding, comparing against existing state-of-the-art MLTL satisfiability-checking techniques, specifically the approaches presented in [9]. That work investigated reduction of MLTL satisfiability checking to LTL_f satisfiability checking, model checking, and SMT solving, where the SMT encoding outperformed the other approaches in general. Their primary SMT encoding uses the theory of uninterpreted functions and linear integer arithmetic (UFLIA). The authors also investigated a quantifier-free encoding (QF_UFLIA), but found it did not scale as well as interval size increased. We therefore focus our comparison to the UFLIA encoding.

Experimental Goals. Our experiments aim to answer the following research questions: (RQ1) Can the QF_BV encoding improve MLTL satisfiability checking performance compared to the UFLIA encoding? (RQ2) How does the performance of the QF_BV and QF_BV * approaches vary with the maximum interval size in MLTL formulas? (RQ3) Is there a performance advantage in using an incremental approach (QF_BV *) for large-interval formulas?

Tools and Environment. For the UFLIA encoding, we use the solvers Z3 [32], CVC5 [33], and for the QF_BV encodings YICES2 [34] and BITWUZLA [35], taking the smaller time from either solver for each encoding. Experimentation showed that BITWUZLA and YICES2 outperformed both CVC5 and Z3 on virtually all QF_BV instances, and BITWUZLA and YICES2 do not support the UFLIA theory. We ran experiments on a cluster with 2x 32-core AMD EPYC 7502 processors and 500GB of RAM, with a timeout of 20 minutes and memout of 16GB per instance. We report only solving time; encoding time for all instances was less than 5 seconds.

Benchmarks. We use pseudo-randomly-generated MLTL formulas via the same technique as in [9]: we generate 4 sets of 10,000 formulas with maximum temporal upper bounds in the range 10, 100, 1,000, and 10,000. Each set is composed of 100 formula subsets each with varying number of propositions (1 to 5), formula closure size (20, 40, 60, 80, and 100), and probability of each operator in the formula being temporal (33%, 50%, 75%, and 95%). We investigated other benchmarks, including formulas derived from case studies [9], [36], [37], and found similar results to the random formulas with upper bounds of 10,000. We also only employ the QF_BV * approach when the interval size is 1,000 or greater since the QF_BV approach solves all problems with sizes 10 and 100.

Max Bnd	Encoding	Solved	Sat	Unsat	Best	Unique
10	UFLIA	9,970	9,638	**332**	2	0
	QF_BV	**10,000**	**9,668**	**332**	**9,998**	**30**
100	UFLIA	9,774	9,473	301	**7,512**	0
	QF_BV	**10,000**	**9,696**	**304**	2,488	**226**
1,000	UFLIA	9,538	9,249	289	3,535	**3**
	QF_BV	**9,994**	**9,704**	**290**	600	0
	QF_BV *	**9,994**	**9,704**	**290**	**5,862**	0
10,000	UFLIA	**9,299**	8,981	**318**	3,885	**452**
	QF_BV	7,953	7,700	253	389	0
	QF_BV *	9,251	**8,986**	265	**5,430**	31

TABLE I: Evaluation results showing number of instances solved (Solved), satisfiable solved (Sat), unsatisfiable solved (Unsat), fastest time to solve across all encodings (Best), and instances solved by only that encoding (Unique). The highest value in each benchmark set for each statistic is bolded.

Experimental Results. Comparing performance across the approaches (RQ1), Table I summarizes results across all four sets of benchmarks and Figure 2 shows cactus plots for the 1,000 and 10,000 interval benchmarks. Across all benchmarks, the QF_BV-based approaches solved 743 instances that the UFLIA encoding could not and was faster than UFLIA on 62% of instances, demonstrating its effectiveness over the UFLIA in many instances.

To study how performance varies with interval size (RQ2), we observe that for smaller intervals (10 and 100), the QF_BV encoding consistently outperforms the UFLIA baseline. For larger intervals (1,000 and 10,000), the incremental QF_BV * encoding shows the best performance on the 1,000 benchmark set and remains highly competitive on the 10,000 set.

Regarding the benefit of incremental reasoning (RQ3), the QF_BV * variant clearly improves upon the non-incremental QF_BV encoding at large interval sizes. On average, the incremental approach required one query for 50.5% of formulas, two queries for 20.8% of formulas, and a full three queries for the remaining 28.7%, indicating that incremental strategies are particularly valuable when dealing with such formulas.

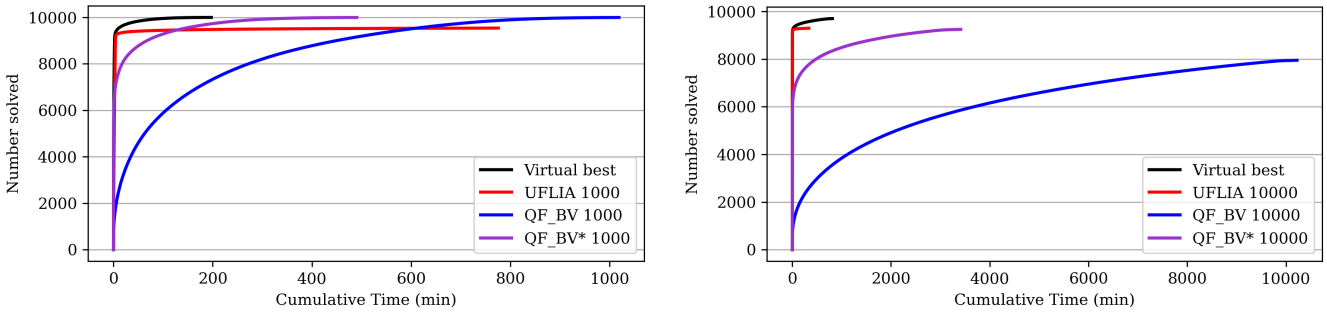

Fig. 2: Cactus plots showing how each encoding performs on 1,000 (left) and 10,000 (right) benchmarks. The incremental approach (purple) outperforms the standard approach (blue), and is complementary to the UFLIA encoding (red) as the virtual best (black) is better than any one encoding alone.

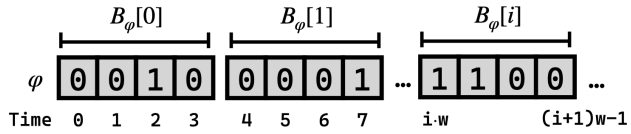

Fig. 3: Visualizing the contents of a buffer $\mathbf{B}_\varphi$ up to word i with a word size $w = 4$. At word $\mathbf{B}_\varphi[i]$, the first bit represents φ's value at $i * w$ and the final bit at $(i + 1) * w - 1$.

V. Monitoring

Given a finite trace π and an MLTL formula φ, the offline monitoring problem consists of computing whether $\pi, \tau \models \varphi$ holds for every time point τ. Conveniently, Definition 5 offers a direct method for solving this problem. In this section, we turn our attention to the online monitoring problem, which is similar in principle but differs in that π is treated as an unbounded stream of states.

In adapting our encoding for online monitoring, our algorithm aims to satisfy three key criteria: takes a stream of states as input and produces a stream of verdicts as output, operates exclusively on fixed-size bit-vectors, and avoids redundant computation by never re-evaluating parts of any bit-vector. Hardware considerations motivate the focus on fixed-size bit-vectors; most machines natively support operations on bit-vectors of size 8, 16, 32, or 64 bits. We refer to these fixed-size bit-vectors as *words* and assume a fixed word size w.

Given an input formula φ, the monitor consumes states from the input stream and stores them in a buffer, interpreting this buffer as a trace π. Then, for each sub-formula $\psi \in cl(\varphi)$, the monitor maintains a buffer $\mathbf{B}_\psi$ of words, i.e., $\mathbf{B}_\psi[i]$ is a word and $\mathbf{B}_\psi[i][j]$ (with $j < w$) is a bit representing the value of ψ at time $i * w + j$ (Figure 3). For now, we assume the size of these buffers is unbounded.

The monitor computes the words of these buffers one at a time via Definition 5. Intuitively, when the monitor performs the computation of $bv(\psi, \pi)$, it stores the relevant bits of the result in $\mathbf{B}_\psi[i]$ and uses $\mathbf{B}_\psi$ as a cache whenever referencing $bv(\psi, \pi)$ in subsequent computations. For instance, the monitor for $\psi = \neg\xi$ will compute $\mathbf{B}_\psi[i] = \sim \mathbf{B}_\xi[i]$, using $\mathbf{B}_\xi[i]$ in place of $bv(\xi, \pi)$.

For temporal operators, the monitor waits until all necessary information is available from its direct sub-formulas before computing via Definition 5. Consider that to compute the value

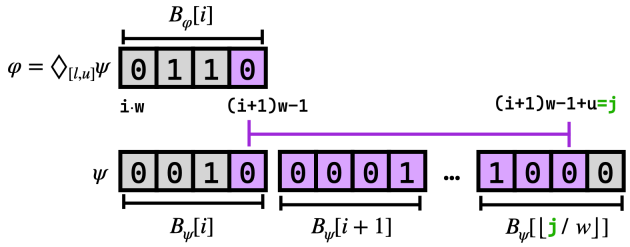

Fig. 4: The final bit of $\mathbf{B}_\varphi[i]$, representing φ's value at time $(i + 1) * w - 1$, relies on values of ψ through time $j = (i+1) * w - 1 + u$ (purple), which is stored in word $\mathbf{B}_\psi[\lfloor j/w \rfloor]$. The monitor therefore waits until $\mathbf{B}_\psi[\lfloor j/w \rfloor]$ is fully computed before computing $\mathbf{B}_\varphi[i]$.

for $\psi = \Diamond_{[l,u]}\xi$ at time τ requires the values of ξ through time $\tau + u$ (in the worst case). Now, computing the full word $\mathbf{B}_\psi[i]$ means computing ψ's values for times $i * w$ through $(i - 1) * w - 1$, therefore requiring values for ξ through time $j = ((i - 1) * w - 1) + u$ (see Figure 4). This value is stored in the word $\mathbf{B}_\xi[\lfloor j/w \rfloor]$, so the monitor waits until this word is computed before computing $\mathbf{B}_\psi[i]$. The same idea applies for the $\mathcal{U}_{[l,u]}$ operator.

Up to now we have assumed the buffers are unbounded, but we can drop values of the buffers once we no longer need them. For example, given an input formula φ, once the monitor computes $\mathbf{B}_\varphi[0]$, it no longer needs the values for $\mathbf{B}_p[0]$. We introduce a value $W(\psi, w)$ for an MLTL_2 formula ψ and word size $w > 0$ that represents the number of words ahead each sub-formula needs to look to compute the current word:

$$W(p, w) = 1, \quad W(\neg\varphi, w) = W(\varphi, w)$$
$$W(\varphi \wedge \psi, w) = max(W(\varphi, w), W(\psi, w))$$
$$W(\Diamond_{[l,u]}\psi, w) = \lfloor((w - 1) + u)/w\rfloor + W(\psi, w)$$
$$W(\varphi\ \mathcal{U}_{[l,u]}\ \psi, w) = \lfloor((w - 1) + u)/w\rfloor + max(W(\varphi), W(\psi))$$

Here, $(w-1)+u$ represents how many bits ahead the monitor needs to compute the final bit of the current word. Then, given an input formula φ, each buffer $\mathbf{B}_\psi$ for $\psi \in cl(\varphi)$ is assigned the size $|\mathbf{B}_\psi| = W(\varphi, w)$. Therefore the size of φ's monitor is bounded by $|cl(\varphi)| \cdot W(\varphi, w)$ words. As an example:

$$W(\Diamond_{[0,1]}\Diamond_{[0,1]}p, 4) = \lfloor(3 + 1)/4\rfloor + W(\Diamond_{[0,1]}p, 4)$$
$$= 1 + \lfloor(3 + 1)/4\rfloor + 1 = 3$$

We note this is not a tight bound (e.g., the input formula φ only strictly requires a buffer size of 1 to store the current

Spec Name	MLTL Formula
Future	$\Diamond_{[0,10]} p$
Until	$p \, \mathcal{U}_{[0,10]} \, q$
Minimum Duration	$p \vee ((\neg p \, \mathcal{U}_{[0,10]} \, p) \vee \Box_{[0,10]} \neg p)$
Eventually p Between q and r	$(q \wedge \Box_{[0,10]} \neg r \wedge \Diamond_{[10,30]} r) \rightarrow ((\neg r \, \mathcal{U}_{[10,20]} \, (p \wedge \neg r)) \vee \Box_{[10,20]} \neg p)$
Precedence Chain	$\Diamond_{[10,10]}(q_0 \wedge \Diamond_{[0,5]}(q_1 \wedge \Diamond_{[0,5]} q_2 \wedge (\Diamond_{[0,5]} q_3))) \rightarrow \Diamond_{[0,10]} p$

TABLE II: The instantiated pattern formulas [26] used for benchmarking provide some simple formulas (Future, Until), a larger formula over a single proposition (Min Duration), a complex formula capturing temporal ordering of 3 propositions (Between), and a formula featuring nesting of temporal operators (Precedence Chain).

word). Proving a tighter bound is left for future work.

Due to this waiting, the algorithm cannot compute the value of φ at time τ until consuming $W(\varphi, w) * w$ states beyond τ. This is a significant drawback as the monitor cannot report violations of the formula as soon as they occur. This is a major focus for future work as this is often a requirement for time-sensitive, safety-critical systems.

A. Experimental Evaluation

We implemented a prototype tool, SABRE (Stream Analysis via Bitwise Reasoning), that takes an MLTL formula as input and produces a C program that monitors the formula according to the presented algorithm. The C program takes as input a stream of states (in the form of a CSV file read from standard input) and outputs a stream of verdicts.

Experimental Goals. Our experiments aim to answer the following research questions: (RQ1) How does SABRE perform in terms of throughput and memory usage compared to existing tools when applied to realistic specifications? (RQ2) How does its performance vary with the shape of the input trace, particularly with respect to the frequency with which propositions occur? (RQ3) How does SABRE respond to different interval sizes, especially when those sizes are exact powers of two?

Tools and Environment. We compare SABRE to two other tools. The first tool we compare to is R2U2 [24], an online MLTL monitoring framework designed for resource-constrained real-time embedded systems. R2U2 offers a Rust [38] and C implementation. For our experiments, we use the C version, although both versions offered similar performance.

The second tool we compare against is HYDRA [17], an offline MTL monitoring tool. Since HYDRA monitors MTL formulas, we convert all MLTL formulas of the form $\varphi \, \mathcal{U}_{[l,u]} \, \psi$ to the equivalent MTL formula $\Diamond_{[l,l]}(\varphi \, \mathcal{U}_{[0,u-l]} \, \psi)$. HYDRA also takes as input a sequence of timestamped events instead of the traces as defined above. We convert each trace to a HYDRA-compatible trace by reporting an event for every timestamp in the trace regardless of whether the event is empty or not. In comparing SABRE to HYDRA we aim to gain insight as to how our method compares to an algorithm intended for offline usage. This also implicitly compares our method to other tools including MONPOLY [14], DEJAVU [15], AERIAL [39], and REELAY [40] as HYDRA outperformed those tools in experimentation [17], [41].

	Maximum % Throughput Difference			
	$\Diamond_{[0,1000]} p$	$\Diamond_{[0,1023]} p$	$\Diamond_{[0,5000]} p$	$\Diamond_{[0,10000]} p$
R2U2	37.20%	38.69%	38.19%	38.03%
HYDRA	11.58%	13.29%	11.59%	12.23%
SABRE	4.00%	8.33%	5.71%	4.00%

TABLE III: Percent differences between the highest and lowest throughput across all trace densities demonstrate that SABRE is the most robust to changes in trace density, offering relatively stable performance no matter the shape of the trace.

Each experiment involved running a tool on a specific formula/trace pair. We measured the throughput, defined as the number of verdicts reported divided by the total execution time, and the peak RAM usage, reported as the maximum resident set size (RSS) using the GNU `time` command. To ensure consistency, we sent all tool output to `/dev/null` and executed each formula/trace pair 10 times, reporting the median execution time and max RSS across those runs to reduce the impact of outliers, e.g., due to loading of shared libraries. Execution time does not include encoding or compilation time, though in all cases this time was less than 1 second and would not impact performance in an online setting. All experiments were conducted on a machine with an Apple M3 Pro processor and 18GM RAM.

Benchmarks. We use two sets of benchmarks to exercise the algorithm. P is a set of five formulas derived from a case study on common **P**atterns of MTL formulas [26], shown in Table II with interval sizes up to 20. F (for **F**uture) is the formula $\Diamond_{[0,u]} p$ with the values $u = 1000$, 1023, 5000 and 10000.

We run all experiments over pseudo-random traces of length $5,000,000$; long enough so that each tool's runtime is reliably measurable by the `time` command (i.e., on the order of seconds). We generated each trace with a specified *density*, defined as the relative proportion of 1s to 0s on average for each proposition in the trace. For P, all traces have a density of 0.5, so each proposition was present half as often as it was absent on average. For F, we vary the density to see its effect on each tool's performance.

Experimental Results. When applied to the P formulas (RQ1), SABRE consistently outperformed both R2U2 and HYDRA in terms of throughput (Figure 5, left), achieving an average of **10.52x** higher throughput than R2U2 on the P formulas and **4.96x** higher than HYDRA, despite HYDRA being an offline tool. On average, HYDRA's max RSS was **56.79MB** while R2U2's was **1.09MB** and SABRE's **1.05MB**, suggesting SABRE is not using more memory as a tradeoff to achieve its throughput.

Figure 5 (right) shows how each tool performs across a range of densities where it appears that HYDRA is the tool most insensitive to trace shape. However, since SABRE's throughput is higher than HYDRA's, we consider the percent difference between the maximum and minimum throughput for each tool on each formula (Table III). Taking the formula $\Diamond_{[0,1023]} p$, for which SABRE has the largest such percent

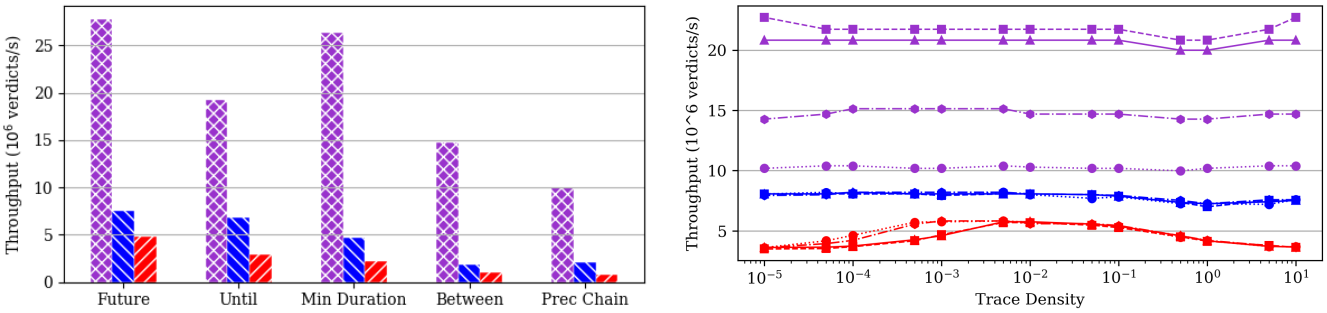

Fig. 5: Experiments compared SABRE (purple), HYDRA (blue), and R2U2 (red). (**Left**) Median throughput (in millions of verdicts per second) for each formula in Table II shows that SABRE performs best across all formulas. (**Right**) Varying trace density for $\Diamond_{[0,1000]}p$ (solid, ▲), $\Diamond_{[0,1023]}p$ (dashed, ■), $\Diamond_{[0,5000]}p$ (dash-dotted, ⬢), $\Diamond_{[0,10000]}p$ (dotted, ●). All HYDRA lines coincide.

difference, R2U2's maximum throughput is **37.20%** percent higher than its minimum across all densities, HYDRA's is **11.58%** higher, and SABRE's is only **8.33%** higher, demonstrating that SABRE is more robust with respect to trace shape than R2U2 or HYDRA. This is likely due to the algorithm not including any branching, i.e., always performing identical bit-vector operations at each execution no matter the values of the input.

SABRE's performance varied substantially with interval size (RQ3). It performed better on the formula $\Diamond_{[0,1023]}p$ than $\Diamond_{[0,1000]}p$, despite the former having a larger interval, highlighting the advantage of the encoding when interval sizes are powers of two. Performance tapered at larger intervals (e.g., 10000), approaching that of HYDRA, whereas the other tools do not suffer from such a performance degradation (Figure 5, right). This is likely due to the other tools' methods allowing early evaluation of temporal operators and so may only use some information in the interval, whereas SABRE uses all information in the temporal interval regardless.

VI. CONCLUSION

There are several promising directions for future work, especially in the context of monitoring. First, the algorithm's structure lends itself naturally to parallelization, as many sub-formulas are independent and can be evaluated concurrently. This is especially true in the offline monitoring setting, where the entire trace is available in advance, allowing evaluations to begin from multiple points in the trace simultaneously. Additionally, the encoding could possibly be adapted to use matrix operations, enabling the use of optimized GPU-based tools to further accelerate performance. Beyond parallelization, we also plan to extend the encoding framework to support other temporal logics, including both finite-trace variants and infinite-trace logics by encoding lasso-shaped traces. There is also potential for applying these techniques to non-boolean inputs.

Our findings suggest that reducing problems to bit-vector operations, as temporal exponentiation does so effectively, is a promising approach to problems related to temporal logic reasoning.

REFERENCES

[1] K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 2016, pp. 8–26.

[2] J. Schneider, D. Basin, F. Brix, S. Krstić, and D. Traytel, "Scalable online first-order monitoring," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 2, pp. 185–208, 2021.

[3] T. Reinbacher, K. Y. Rozier, and J. Schumann, "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)*, ser. Lecture Notes in Computer Science (LNCS), vol. 8413. Springer-Verlag, April 2014, pp. 357–372.

[4] B. Kempa, P. Zhang, P. H. Jones, J. Zambreno, and K. Y. Rozier, "Embedding online runtime verification for fault disambiguation on robonaut2," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2020, pp. 196–214.

[5] A. Aurandt, P. Jones, and K. Y. Rozier, "Runtime Verification Triggers Real-time, Autonomous Fault Recovery on the CySat-I," in *Proceedings of the 14th NASA Formal Methods Symposium (NFM 2022)*, ser. Lecture Notes in Computer Science (LNCS), vol. 13260. Caltech, California, USA: Springer, Cham, May 2022.

[6] A. Hammer, M. Cauwels, B. Hertz, P. Jones, and K. Y. Rozier, "Integrating runtime verification into an automated uas traffic management system," *Innovations in Systems and Software Engineering: A NASA Journal*, July 2021.

[7] C. Johannsen, M. Anderson, W. Burken, E. Diersen, J. Edgren, C. Glick, S. Jou, A. Kumar, J. Levandowski, E. Moyer, T. Roquet, A. VandeLoo, and K. Y. Rozier, *OpenUAS Version 1.0*. Athens, Greece (Virtual): IEEE, June 2021.

[8] J. B. Dabney, J. M. Badger, and P. Rajagopal, "Adding a verification view for an autonomous real-time system architecture," in *Proceedings of SciTech Forum*, ser. 2021-0566. AIAA, January 2021, p. Online.

[9] J. Li, M. Y. Vardi, and K. Y. Rozier, "Satisfiability checking for mission-time ltl (mltl)," *Information and Computation*, vol. 289, p. 104923, 2022.

[10] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[11] D. Basin, F. Klaedtke, and E. Zălinescu, "Algorithms for monitoring real-time properties," in *International Conference on Runtime Verification*. Springer, 2011, pp. 260–275.

[12] D. Basin, B. N. Bhatt, S. Krstić, and D. Traytel, "Almost event-rate independent monitoring," *Formal Methods in System Design*, vol. 54, pp. 449–478, 2019.

[13] D. Basin, F. Klaedtke, and S. Müller, "Monitoring security policies with metric first-order temporal logic," in *Proceedings of the 15th ACM symposium on Access control models and technologies*, 2010, pp. 23–34.

[14] D. Basin, F. Klaedtke, and E. Zalinescu, "The monpoly monitoring tool," in *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, vol. 3. EasyChair, 2017, pp. 19–28.

[15] K. Havelund, D. Peled, and D. Ulus, "Dejavu: A monitoring tool for first-order temporal logic," in *2018 IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS)*, 2018, pp. 12–13.

[16] D. A. Basin, S. Krstic, and D. Traytel, "Aerial: Almost event-rate independent algorithms for monitoring metric regular properties." in *RV-CuBES*, 2017, pp. 29–36.

[17] M. Raszyk, D. Basin, S. Krstić, and D. Traytel, "Multi-head monitoring of metric temporal logic," in *Automated Technology for Verification and Analysis: 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28–31, 2019, Proceedings 17*. Springer, 2019, pp. 151–170.

[18] B. d'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, "Lola: runtime monitoring of synchronous systems," in *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*. IEEE, 2005, pp. 166–174.

[19] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, M. Stenger, L. Tentrup, and H. Torfah, "Streamlab: stream-based monitoring of cyber-physical systems," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 421–431.

[20] I. Perez, F. Dedden, and A. Goodloe, "Copilot 3," Tech. Rep., 2020.

[21] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, and D. Thoma, "Tessla: temporal stream-based specification language," in *Formal Methods: Foundations and Applications: 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 26–30, 2018, Proceedings 21*. Springer, 2018, pp. 144–162.

[22] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "R2u2: monitoring and diagnosis of security threats for unmanned aerial systems," in *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*. Springer, 2015, pp. 233–249.

[23] K. Y. Rozier and J. Schumann, "R2u2: tool overview," in *International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, no. ARC-E-DAA-TN48849, 2017.

[24] C. Johannsen, P. Jones, B. Kempa, K. Y. Rozier, and P. Zhang, "R2u2 version 3.0: Re-imagining a toolchain for specification, resource estimation, and optimized observer generation for runtime verification in hardware and software," in *International Conference on Computer Aided Verification*. Springer, 2023, pp. 483–497.

[25] M. Valizadeh, N. Fijalkow, and M. Berger, "Ltl learning on gpus," in *International Conference on Computer Aided Verification*. Springer, 2024, pp. 209–231.

[26] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *IEEE Transactions on Software Engineering*, vol. 41, no. 7, pp. 620–638, 2015.

[27] C. Johannsen, P. Jones, K. Y. Rozier, and T. Wongpiromsarn, "MLTL Reasoning via Bit-vector Encoding, FMCAD 2025 Artifact," Jul. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.16312254

[28] J. Elwing, L. Gamboa-Guzman, J. Sorkin, C. Travesset, Z. Wang, and K. Y. Rozier, "Mission-time ltl (mltl) formula validation via regular expressions," in *International Conference on Integrated Formal Methods*. Springer, 2023, pp. 279–301.

[29] K. Kosaian, Z. Wang, E. Sloan, and K. Rozier, "Formalizing mltl formula progression in isabelle/hol," *arXiv preprint arXiv:2410.03465*, 2024.

[30] C. Johannsen, B. Kempa, P. H. Jones, K. Y. Rozier, and T. Wongpiromsarn, "Impossible made possible: Encoding intractable specifications via implied domain constraints," in *Formal Methods for Industrial Critical Systems*, A. Cimatti and L. Titolo, Eds. Cham: Springer Nature Switzerland, 2023, pp. 151–169.

[31] C. Barrett, A. Stump, C. Tinelli *et al.*, "The smt-lib standard: Version 2.0," in *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, vol. 13, 2010, p. 14.

[32] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[33] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli *et al.*, "cvc5: A versatile and industrial-strength smt solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2022, pp. 415–442.

[34] B. Dutertre, "Yices 2.2," in *International Conference on Computer Aided Verification*. Springer, 2014, pp. 737–744.

[35] A. Niemetz and M. Preiner, "Bitwuzla," in *International Conference on Computer Aided Verification*. Springer, 2023, pp. 3–17.

[36] M. Gario, A. Cimatti, C. Mattarei, S. Tonetta, and K. Y. Rozier, "Model checking at scale: Automated air traffic control design space exploration," in *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II 28*. Springer, 2016, pp. 3–22.

[37] M. Bozzano, A. Cimatti, A. Fernandes Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta, "Formal design and safety analysis of air6110 wheel brake system," in *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I 27*. Springer, 2015, pp. 518–535.

[38] A. Aurandt, P. H. Jones, and K. Y. Rozier, "Towards a safe, verified runtime monitor for embedded systems: R2u2 in embedded rust," in *Proceedings of the 17th NASA Formal Methods Symposium (NFM 2025)*, 2025.

[39] D. Basin, B. N. Bhatt, and D. Traytel, "Almost event-rate independent monitoring of metric temporal logic," in *Tools and Algorithms for the Construction and Analysis of Systems: 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II 23*. Springer, 2017, pp. 94–112.

[40] D. Ulus, "Reelay," https://github.com/doganulus/reelay, 2025.

[41] M. Raszyk, D. Basin, and D. Traytel, "Multi-head monitoring of metric dynamic logic," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020, pp. 233–250.

APPENDIX

Theorem 1 (MLTL-BV Correctness). *Let $\mathcal{AP}$ be a finite set of atomic propositions, φ be an MLTL$_2$ formula over $\mathcal{AP}$, and π be a trace. Then for each $\tau < |\pi|$, the bit at index τ of $bv(\varphi, \pi)$ is 1 if and only if π at τ models φ, i.e.,*

$$bv(\varphi, \pi)[\tau] = 1 \text{ iff } (\pi, \tau \models \varphi).$$

Proof. We proceed with induction over the structure of φ. For the following, let π be a finite trace and τ be an index such that $\tau < |\pi|$.

In the base case of our structural induction, $\varphi = p$ where $p \in \mathcal{AP}$. By construction of $bv(p, \pi)$, we have that $bv(p, \pi)[\tau] = 1$ iff $p \in \pi[\tau]$, proving the property.

If $\varphi = \neg\psi$ or $\varphi = \psi \wedge \xi$, then the property holds from the correspondence between $\sim$, $\neg$ and $\&$, $\wedge$.

Now let $\varphi = \Diamond_{[l,u]}\psi$ where $u = l + 2^k - 1$. We inductively assume that the property holds for ψ. Fixing $l \geq 0$, we show that for all $k \geq 0$ it holds that

$$bv(\Diamond_{[l,u]}\psi, \pi)[\tau] = 1 \Leftrightarrow \left(\pi, \tau \models \Diamond_{[l,u]}\psi\right).$$

For the base case, $k = 0$ so that $u = l + 2^0 - 1 = l$ and $bv(\Diamond_{[l,l]}\psi, \pi) = bv(\psi, \pi) \ll l$. If $l \geq |\pi|$, then $\pi, \tau \not\models \Diamond_{[l,l]}\psi$ by the MLTL semantics since a is off the end of the trace. By the definition of $\ll$, we also have $bv(\Diamond_{[l,l]}\psi, \pi) = zeroes(|\pi|)$ so that $bv(\Diamond_{[l,l]}\psi, \pi)[\tau] = 0$, proving the case when $l \geq |\pi|$. Otherwise, we have

$$\begin{aligned} bv(\Diamond_{[l,l]}\psi, \pi)[\tau] &= (bv(\psi, \pi) \ll l)[\tau] \\ &= bv(\psi, \pi)[\tau + l] \end{aligned}$$

which is 1 only if $\pi, \tau + l \models \psi$ by the MLTL semantics and inductive hypothesis, proving the case when $l < |\pi|$.

For the inductive step, assume that the property holds for k, we show that it holds for $k+1$. Let $b_k = bv(\Diamond_{[l,l+2^k-1]}\psi, \pi)$. Then by Definition 5:

$$\begin{aligned} bv(\Diamond_{[l,l+2^{k+1}-1]}\psi, \pi)[\tau] &= \left(b_k \mid (b_k \ll 2^k)\right)[\tau] \\ &= b_k[\tau] \vee (b_k \ll 2^k)[\tau] \\ &= b_k[\tau] \vee b_k[\tau + 2^k]. \end{aligned}$$

Since the property holds for k, this value is 1 if and only if

$$\begin{aligned} &\exists i \in [l, l + 2^k - 1].\ \pi, (i + \tau) \models \psi \text{ or } \pi, (i + 2^k + \tau) \models \psi \\ \Leftrightarrow\ &\exists i \in [l, l + 2^k - 1 + 2^k].\ \pi, (i + \tau) \models \psi \text{ or } \pi, (i + \tau) \models \psi \\ \Leftrightarrow\ &\exists i \in [l, l + 2^{k+1} - 1].\ \pi, (i + \tau) \models \psi \\ \Leftrightarrow\ &\pi, \tau \models \Diamond_{[l,l+2^{k+1}-1]}\psi \end{aligned}$$

proving the property for $k + 1$ and completing the case for $\varphi = \Diamond_{[l,u]}\psi$.

Let $\varphi = \psi\ \mathcal{U}_{[l,u]}\ \xi$ where $u = l + 2^k - 1$. We inductively assume that the property holds for ψ and ξ. Fixing $l \geq 0$, we show that for all $k \geq 0$ it holds that

$$bv(\psi\ \mathcal{U}_{[l,u]}\ \xi, \pi)[\tau] = 1 \Leftrightarrow \left(\pi, \tau \models \psi\ \mathcal{U}_{[l,u]}\ \xi\right).$$

For the base case, $k = 0$ so that $u = l + 2^0 - 1 = l$. Then $bv(\psi\ \mathcal{U}_{[l,l]}\ \xi, \pi) = bv(\Diamond_{[l,l]}\xi, \pi) = bv(\xi, \pi) \ll l$. If $l \geq |\pi|$, then $\pi, \tau \not\models \Diamond_{[l,l]}\xi$ by the MLTL semantics since a is off the end of the trace. By the definition of $\ll$, we also have $bv(\Diamond_{[l,l]}\xi, \pi) = zeroes(|\pi|)$ so that $bv(\Diamond_{[l,l]}\xi, \pi)[\tau] = bv(\psi\ \mathcal{U}_{[l,l]}\ \xi) = 0$, proving the case when $l \geq |\pi|$. Otherwise, we have

$$\begin{aligned} bv(\psi\ \mathcal{U}_{[l,l]}\ \xi, \pi)[\tau] &= bv(\Diamond_{[l,l]}\xi, \pi)[\tau] \\ &= (bv(\xi, \pi) \ll l)[\tau] \\ &= bv(\xi, \pi)[\tau + l] \end{aligned}$$

which is 1 only if $\pi, \tau + l \models \xi$ by the MLTL semantics and inductive hypothesis, proving the case when $l < |\pi|$.

For the inductive step, assume that the property holds for k, we show that it holds for $k + 1$. Let $b_{\psi_k} = bv(\Box_{[l,l+(2^k-1)]}\psi, \pi)$ and $b_{\xi_k} = bv(\psi\ \mathcal{U}_{[l,l+(2^k-1)]}\ \xi, \pi)$. Then by Definition 5:

$$\begin{aligned} bv(\psi\ \mathcal{U}_{[l,l+(2^{k+1}-1)]}\ \xi, \pi)[\tau] &= \left(b_{\xi_k} \mid \left(b_{\psi_k} \& (b_{\xi_k} \ll 2^k)\right)\right)[\tau] \\ &= b_{\xi_k}[\tau] \vee (b_{\psi_k}[\tau] \wedge b_{\xi_k}[\tau + 2^k]) \end{aligned}$$

by the MLTL-BV encoding. This value is 1 if and only if

$$\begin{aligned} &(\exists i \in [l, l + 2^k - 1].\ \pi, (i + \tau) \models \xi \text{ and} \\ &\quad \forall j \in [l, i - 1].\ \pi, (j + \tau) \models \psi) \text{ or} \\ &(\forall x \in [l, l + 2^k - 1].\ \pi, (x + \tau) \models \psi \text{ and} \\ &\quad (\exists y \in [l, l + 2^k - 1].\ \pi, (y + \tau + 2^k) \models \xi \text{ and} \\ &\quad\quad \forall z \in [l, y - 1].\ \pi, (z + \tau) \models \psi)) \\ \Leftrightarrow\ &\quad \text{(Simplifying } y \text{ interval)} \\ &(\exists i \in [l, l + 2^k - 1].\ \pi, (i + \tau) \models \xi \text{ and} \\ &\quad \forall j \in [l, i - 1].\ \pi, (j + \tau) \models \psi) \text{ or} \\ &(\forall x \in [l, l + 2^k - 1].\ \pi, (x + \tau) \models \psi \text{ and} \\ &\quad (\exists y \in [l + 2^k, l + 2^{k+1} - 1].\ \pi, (y + \tau) \models \xi \text{ and} \\ &\quad\quad \forall z \in [l, y - 1].\ \pi, (z + \tau) \models \psi)) \\ \Leftrightarrow\ &\quad \text{(Combining } x \text{ and } z \text{ intervals)} \\ &(\exists i \in [l, l + 2^k - 1].\ \pi, (i + \tau) \models \xi \text{ and} \\ &\quad \forall j \in [l, i - 1].\ \pi, (j + \tau) \models \psi) \text{ or} \\ &(\exists y \in [l + 2^k, l + 2^{k+1} - 1].\ \pi, (y + \tau) \models \xi \text{ and} \\ &\quad \forall z \in [l, y - 1].\ \pi, (z + \tau) \models \psi) \\ \Leftrightarrow\ &\quad \text{(Combining } i, y \text{ and } j, z \text{ intervals)} \\ &(\exists i \in [l, l + 2^{k+1} - 1].\ \pi, (i + \tau) \models \xi \text{ and} \\ &\quad \forall j \in [l, i - 1].\ \pi, (j + \tau) \models \psi) \\ \Leftrightarrow\ &\pi, \tau \models \psi\ \mathcal{U}_{[l,l+2^{k+1}-1]}\ \xi \end{aligned}$$

Proving the property for $k + 1$ and completing the proof. □