MLTL Multi-type: A Typed Logic for Cyber-Physical Systems

GOKUL HARIHARAN*, Iowa State University, Ames, United States
BRIAN KEMPA*, Iowa State University, Ames, United States
TICHAKORN WONGPIROMSARN, Iowa State University, Ames, United States
PHILLIP JONES, Electrical and Computer Engineering, Iowa State University, Ames, United States
KRISTIN ROZIER, Iowa State University, Ames, United States

Modern cyber-physical systems-of-systems (CPSoS) operate in complex systems-of-systems that must seamlessly work together to control safety- or mission-critical functions. Linear Temporal Logic (LTL) and Mission-time Linear Temporal logic (MLTL) intuitively express CPSoS requirements for automated system verification and validation. However, both LTL and MLTL presume that all signals populating the variables in a formula are sampled over the same rate and type (e.g., time or distance), and agree on a standard "time" step. Formal verification of cyber-physical systems-of-systems needs validate-able requirements expressed over (sub-)system signals of different types, such as signals sampled at different timescales, distances, or levels of abstraction, expressed in the same formula. Previous works developed more expressive logics to account for types (e.g., timescales) by sacrificing the intuitive simplicity of LTL. However, a legible direct one-to-one correspondence between a verbal and formal specification will ease validation, reduce bugs, increase productivity, and linearize the workflow from a project's conception to actualization. Validation includes both transparency for human interpretation, and tractability for automated reasoning, as CPSoS often run on resource-limited embedded systems. To address these challenges, we introduced Mission-time Linear Temporal Logic Multi-type (Hariharan et al., Numerical Software Verification Workshop, 2022), a logic building on MLTL. MLTLM enables writing formal requirements over finite input signals (e.g., sensor signals and local computations) of different types, while maintaining the same simplicity as LTL and MLTL. Furthermore, MLTLM maintains a direct correspondence between a verbal requirement and its corresponding formal specification. Additionally, reasoning a formal specification in the intended type (e.g., hourly for an hourly rate, and per second for a seconds rate) will use significantly less memory in resource-constrained hardware. This article extends the previous work with (1) many illustrated examples on types (e.g., time and space) expressed in the same specification, (2) proofs omitted for space in the workshop version, (3) proofs of succinctness of MLTLM compared to MLTL, and (4) a minimal translation to MLTL of optimal length.

Additional Key Words and Phrases: Linear Temporal Logic, Mission-time Linear Temporal Logic, Runtime Verification, Cyber-Physical Systems, Formula Succinctness

1 Introduction

Safety-critical systems, such as aircraft, spacecraft, robots, and autonomous vehicles, require precise, unambiguous specifications for automated reasoning such as model checking, synthesis, requirements debugging, runtime verification (RV), and checking for satisfiability, reachability, realizability, vacuity, and other important properties of system requirements. Modern cyber-physical systems-of-systems present a unique challenge for specification,

*Both authors contributed equally to this research.

Authors' Contact Information: Gokul Hariharan, Iowa State University, Ames, Iowa, United States; e-mail: gokul@iastate.edu; Brian Kempa, Iowa State University, Ames, Iowa, United States; e-mail: bckempa@iastate.edu; Tichakorn Wongpiromsarn, Iowa State University, Ames, Iowa, United States; e-mail: nok@iastate.edu; Phillip Jones, Electrical and Computer Engineering, Iowa State University, Ames, Iowa, United States; e-mail: phjones@iastate.edu; Kristin Rozier, Iowa State University, Ames, Iowa, United States; e-mail: kyrozier@iastate.edu.

ACM 1558-3465/2024/11-ART
https://doi.org/10.1145/3704809

and consequently for scalable verification and validation, due to their distributed and hierarchical nature. Automated reasoning for CPSoS needs seamless construction of global properties combining local phenomena and coordinating requirements for numerical computations like intelligent sensor fusion and signal processing over data and variables of different types (e.g., sampling frequencies).

LTL provides an intuitive way to precisely specify system requirements for timelines in operational concepts of CPSoS [40]. The relative computational efficiency of automated reasoning (e.g., model checking, satisfiability checking) adds to the appeal of LTL as a specification logic [36, 37]. Since CPSoS specifications most often need to describe finite missions with referenceable time steps, variations of LTL over finite signals emerged with intervals on the temporal operators. Variations of Metric Temporal Logic (MTL)[34], such as Signal Temporal Logic (STL)[19] and Mission-time Linear Temporal Logic (MLTL)[30, 35] vary in the types of finite bounds they introduce on LTL's temporal operators, and the complexity of automated reasoning over these logics. MLTL is a variant that has finite, closed, integer bounds on LTL's temporal operators that provides a good balance between expressivity and computational efficiency. MLTL has emerged as a popular specification logic for complex CPSoS such as the NASA Lunar Gateway Vehicle System Manager [16], and a JAXA autonomous satellite mission [33]; see [31] for a collection of MLTL patterns over a weather balloon, automated air traffic management system, sounding rocket, and satellite. Furthermore, recent work has contributed very efficient, flight-certifiable, encodings of MLTL for runtime verification in resource-limited embedded hardware [27].

Nonetheless, realistic requirements for CPSoS need to reason over signals of different types in the same requirement; a requirement specified as an LTL/MLTL formula implicitly presumes that all input signals populating its variables are of the same type (e.g., share a common notion of a time step). Therefore, describing a global property of a system where sub-systems operate at different timescales, (or, more generally, over different types) becomes tedious. For example, consider specifying global safety properties of a deep-space exploration craft. One subsystem of the spacecraft may regulate monthly cycles to wake from hibernation and execute course corrections, whereas another subsystem may operate on the nanosecond frequency to make hyper-sensitive adjustments; it is not obvious how to correctly reason about these in the same formula. Expressing specifications by assuming signals of the same type yields indiscernible specifications with poor correlation between the verbal and logical specifications, and an unreasonably huge memory overhead (e.g., 3600 units of second-to-second information is stored in excess, instead of only keeping the hourly data). Moreover, automated reasoning with signals of different types is ubiquitous to CPSoS, and yet, to the best of our knowledge, there was no realizable logic that accommodated multiple signal types in a single specification, until MLTLM [24].

Previous works provide some options for special cases of this problem, with significant complexity drawbacks. These largely center on two philosophies: higher-order logics reasoning over sets of formulas (instead of one formula combining different types), and annotations to deal with multiple time granularities across formula variables, though not necessarily other combinations of different types. Examples of distributed sets of specifications count on locally evaluating sub-system-level synchronous [7] or asynchronous [6, 32] signals; this set can coordinate through a global formula evaluated over the local formulas [7]. For example, HyperLTL focuses on specifications over sets of formulas over signals of the same type [12], oppositely from this work where we focus on constructing single formulas that seamlessly reason over signals of different types. Table 1 collects related work.

The particular instance of different types in the form of input signals over different time granularities that comprise parts of the same, single temporal logic specification arises frequently in CPSoS; see [20] for a survey. Most previous works focus on developing well expressible languages to define temporally distributed specifications precisely. Again, this often comes with higher-order reasoning (see for example [21]) and complexity penalties; e.g., [13] introduces the notion of temporal universes and uses a set-theory representation of different timescales to abstract notions of time granularities. Propositional Interval Temporal Logic (PITL) adds chop (";") and project operators to LTL to increase expressivity for time granularities over infinite signals; another variation adds

Time-granular logic	Syntax elements	Ref.
PITL	empty, proj, ";", $\Box$, $\Diamond$	[11]
Non standard FOT	$\forall$, $<_e$, $<_w$, $<_1$, $\exists$	[4]
ITL	$<$, m, O, s, f etc.	[2]
Euzenate's extension	6×6 table of operators	[14]
Automata representation	Automata	[28, 29]
Spider diagrams	Spider diagrams	[10]
2D MTL internal eternal	L_i, L_e etc.	[5]
Monodic SOL	Layered representation of FOT	[21]
Multiclock Esterel	nothing, pause, exit, emit, etc.	[9]

Table 1. Various time-granular specification languages and their syntax elements.

temporal relations like "just before" [14]. First-Order Theory (FOT) enables writing time-granular specifications to account for continuous-in-time events and relate them to discretized-in-time representations [4]. Other methods include using automata to represent time-granularity [28, 29] and using spider-diagram representations for time-granular specifications [10], and a two-dimensional metric temporal logic that can be potentially used to represent time granularities [5]. Multiclock Esterel, a language for hardware verification, also introduces many syntactic elements that can potentially pose a higher learning curve for a transition from LTL to a multi-type language for CPSoS. [9, 23]. More recently, the SCADE tool integrates different time-granular languages into a single framework [8, 15], and has found good acceptance in the industrial community. SCADE is targetted to developping correct software for embedded systems, however, it is also important to reason over properties on a higher level regarding the functioning of the embedded system over signals of different types, for example, consider the property, "Within 4 miles from the origin, the drone shouldn't exceed a velocity of 12 mph." Moreover, while SCADE is suited for verification in the design phase of a system, realtime verification during functioning of CPSoS calls for a completely different set of capabilities, like compacted (as little as possible) memory for verification, non-invasive embedment of a verification engine, and a suitable language to reason over expressible properties that is also accomodative in a low memory space and with a quick response time.

The existing solutions therefore precisely express time granularity, but at the cost of sacrificing the intuitive simplicity of LTL and MLTL. A realizable logic with time granularities (or, more generally, types) for CPSoS needs to maintain the simplicity of LTL/MLTL to easily write specifications that CPSoS designers can readily validate, and more importantly, to have tractable computational complexity for automated reasoning (e.g., model checking and runtime verification).

Therefore, we build upon the popular logic MLTL to create MLTLM [24]. The syntax of MLTLM matches that of MLTL except for the addition of a single signal-type label on each temporal operator to signify the output type of that operator. MLTLM has these advantages: (a) maintains the same specification simplicity as LTL and MLTL, (b) reasons in the inherent type of the verbal specification, thereby providing a direct correspondence to its respective formal specification, and (c) readily implementable in embedded systems.

Consider the following MLTLM specifications across types that illustrate these advantages[1] (Section 4 elaborates these examples in more detail):

(1) "Within 4 miles from the origin, the drone shouldn't exceed a velocity of 12 mph."

$$\Box_{[0,4,miles]}(v < 12)$$

[1]These examples assume some knowledge of MLTL (see Section 2 for details); $\Diamond_{[lb,ub]}$ and $\Box_{[lb,ub]}$ denote the future and global operators respectively, that a property holds at some and all instances inside the interval respectively.

(2) "The car drives at a velocity less than 60mph for a distance of 10 miles, and then maintains 70mph for 60 minutes."

$$\square_{[0,10,miles]}(v < 60) \wedge \square_{[10,10,miles]}(\square_{[0,60,minutes]}(v == 70))$$

(3) "The drone's surveillance camera needs to be in on-state for 10 continuous minutes, every hour." (Assuming a mission-time of 3 hours)

$$\square_{[0,3,hours]}\Diamond_{[0,50,minutes]}\square_{[0,10,minutes]}\text{camera-on}$$

(4) "The spacecraft maintenance cycle runs at least once a month over the five-year mission. "

$$\square_{[0,5,years]}\Diamond_{[0,30,days]}\text{maintenance}$$

(5) "Verify monthly that the thrusters did not burn more than 3 seconds at a time."

$$\square_{[0,12,month]}(\neg\square_{[0,3,seconds]}\text{burn-thrusters})$$

Notice from the examples a close correspondence between the verbal and respective formal specification. Formal specifications in LTL or MLTL for these requirements would be hard to interpret, debug, and synthesize due to the lack of types. Furthermore, it is more relaxing, less prone to bugs, and easier to maintain a set of specifications that does not encode the information of types into the specification (e.g., logically expressing months in terms of years), thus simplifying the workflow. Fig. 1 contrasts the workflow when using MLTLM versus MLTL: In MLTL, the project manager gives a verbal specification, which is implemented by a specification designer in coordination with the system engineer, however, the concrete specification may not conform with the project management's requirement, as verbal and logical specifications are often subject to misinterpretations. This would trigger a cycle of iterations all the way from the management to the system designer. In contrast, in MLTL, the project management provides a verbal specification that has a direct correspondence to an MLTLM specification. The iterations are then confined to the concrete specification and the system engineer (note that we assume that the management does not have fundamentally contradictory requirements, in which case both approaches would have to cycle throughout).

This paper is organized as follows. Section 2 gives a prelude to the conventional single type temporal logic, MLTL, and gives a background on R2U2 – an industry-used runtime verification engine for CPSoS that we build upon to monitor MLTLM specifications. We contribute:

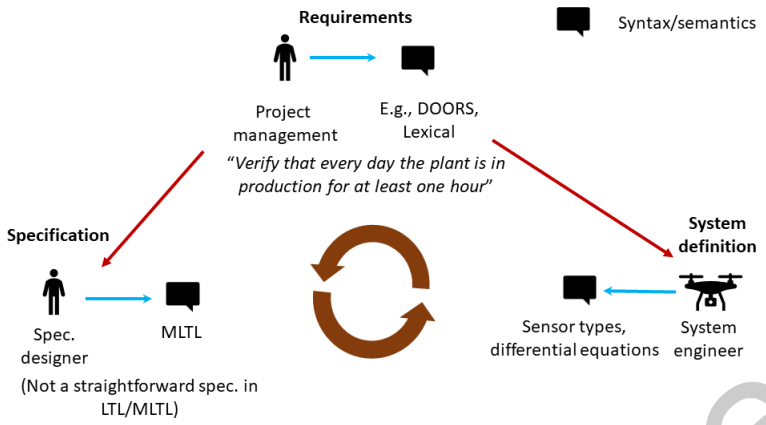
(1) (Section 3) the formal definition for the logic MLTLM (Mission-time Linear Temporal Logic Multi-type), including syntax and semantics (carried forward from [24]);
(2) (Section 4) illustrated example specifications expressed in MLTLM with signals of different types (new in this paper);
(3) (Section 5) Detailed translations from MLTLM to MLTL with proofs of succinctness between MLTLM and MLTL (new in this paper);
(4) (Section 6) an open-source implementation of a runtime verification tool for MLTLM, built on top of R2U2 [26, 27, 38], compared to translated MLTL (this includes a translator from MLTLM to MLTL that produces provably optimally short MLTL formulas absent in [24]).

Section 7 discusses conclusions and future directions.

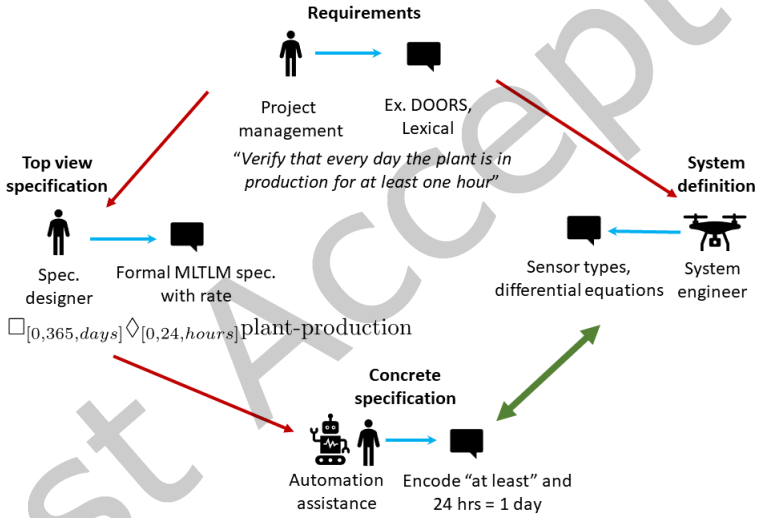
2 Preliminaries

2.1 Signals and Trajectories

Definition 2.1. (Signal) A signal σ over an atomic proposition p is defined as a finite sequence $\sigma = a_0, a_1, \ldots, a_N$ where $\sigma[i] = a_i \in \{\text{true}, \text{false}\}$ indicates whether p holds at the discrete time instance i. A subsequence of a signal from position i is denoted by $\sigma[i..]$, i.e., $\sigma[i..] = a_i, a_{i+1}, \ldots, a_N$.



(a) Workflow when using LTL/MLTL



(b) Envisioned workflow using MLTLM

Fig. 1. Iteration workflow for CPSoS runtime verification of project requirements describing the system and specification in simple lexical language. (a) Traditionally, modifications to the system or specification at any level restarts the cycle. (b) We propose that project management first verify a top view specification in a simple syntax while iteration of the detailed specification is contained to system engineering. The automated assistant may suggest suitable projections between types

Definition 2.2. (Signal Type) All signals have an associated type. We use superscript to denote the type of a signal, i.e., $\sigma^{\mathbb{A}}$ denotes a signal of type $\mathbb{A}$.

Definition 2.3. (Trajectory) A trajectory π over atomic propositions $p_0, ..., p_n$ is a set of signals, i.e., $\pi = \{\sigma_0^{\mathbb{A}_0}, \sigma_1^{\mathbb{A}_1}, ..., \sigma_n^{\mathbb{A}_n}\}$ where $\sigma_i^{\mathbb{A}_i}$ is a signal over p_i. $\pi_{p_i}^{\mathbb{A}_i}[j]$ refers to the jth value of the signal of type $\mathbb{A}_i$ over atomic proposition p_i in π.

Related work in linear temporal logic use "traces" or "computations" [3, 12], which is typically described as a sequence of sets of atomic propositions. In contrast, we generalize "traces" by allowing member signals to be of different types and call them collectively as a trajectory.

Definition 2.4 (Projection). The projection function $T_{\mathbb{A}}^{\mathbb{B}}$ takes a signal σ of type $\mathbb{A}$ and returns a signal of type $\mathbb{B}$.

2.2 MLTL

MLTL is a variant of LTL [3] on finite signals with closed temporal bounds [35, 38] on natural numbers.

Definition 2.5. (MLTL Syntax [35]) The syntax of an MLTL formula φ over a set of atomic propositions $\mathcal{AP}$ is recursively defined as:

$$\varphi \;:=\; \mathsf{true} \mid p \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where $p \in \mathcal{AP}$, φ_1 and φ_2 are MLTL formulas, $I := [lb, ub]$ is a closed interval bound, such that lb and ub are natural numbers such that $lb \leq ub$.

Abstract Syntax Tree (AST) The AST representation of an MLTL formula has nodes of logical operators and leaves of atomic propositions connected to represent the recursive structure of the expression from Def. 2.5.

Definition 2.6. (MLTL Semantics [35]) The evaluation of an MLTL formula φ on a trajectory π where all signals have uniform type produces a signal σ, denoted by $((\pi, \varphi) \mapsto \sigma)$, defined recursively on the signals σ_1 and σ_2 representing the evaluation of its children subformulas φ_1 and φ_2 respectively as

$$\sigma[i] := \begin{cases} \pi_p[i], & \text{if } \varphi = p,\ p \in \mathcal{AP}, \\ \neg\sigma_1[i], & \text{if } \varphi = \neg\varphi_1, \\ \sigma_1[i] \wedge \sigma_2[i], & \text{if } \varphi = \varphi_1 \wedge \varphi_2, \\ \mathsf{true},\ \text{iff } |\sigma_1|, |\sigma_2| > (i + lb),\ \text{and } \exists j \in [i + lb, i + ub] & \\ \text{such that } \sigma_2[j] = \mathsf{true},\ \text{and } \forall k < j \text{ where} & \text{if } \varphi = \varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2. \\ k \in [i + lb, i + ub],\ \sigma_1[k] = \mathsf{true}, & \end{cases}$$

Other common operators are defined via equivalences, i.e., false $\Leftrightarrow \neg$true, future $\Diamond_I \varphi \Leftrightarrow$ true $\mathcal{U}_I\ \varphi$, globally $\Box_I \varphi \Leftrightarrow \neg(\Diamond_I \neg\varphi)$, and next $\bigcirc\varphi \Leftrightarrow \Box_{[1,1]}\varphi$.

2.3 R2U2

The *Realizable, Responsive, Unobtrusive Unit*[2] (R2U2) is a runtime verification (RV) engine to monitor MLTL specifications for flight mission systems [26, 35] used in robotics [27], NASA drone aircraft [22, 39], and is being evaluated for use on the Lunar Gateway space station [16, 17]. R2U2 is *Realizable*: implemented on real hardware, *Responsive*: reports specification violation immediately, and *Unobtrusive:* uses existing data sources instead of modifying the system to add instrumentation. R2U2 features specification reconfiguration and real-time performance with guaranteed memory bounds to better support the needs of flight systems. R2U2 is an open-source RV engine with well-documented industrial use. We build an MLTLM RV engine upon R2U2 to provide existing users a seamless transition to a multi-type logic. Our R2U2-based MLTLM RV engine upholds all the existing guarantees of R2U2.

3 Mission-time Linear Temporal Logic Multi-type (MLTLM)

MLTLM is a lightweight extension to MLTL that enables temporal reasoning over system trajectories composed of signals of different types.

[2] r2u2.temporallogic.org

Definition 3.1. (MLTLM Syntax) The syntax of an MLTLM formula φ over a set of atomic propositions $\mathcal{AP}$, and the set of types $\mathcal{T}$ is recursively defined as:

$$\varphi \;:=\; \mathsf{true} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_J \varphi_2$$

where $p \in \mathcal{AP}$, $\mathbb{A} \in \mathcal{T}$, φ_1 and φ_2 are MLTLM formulas, and $J := [lb, ub, \mathbb{A}]$ is a finite interval bound such that lb and ub are natural numbers, $lb \leq ub < \infty$.

Notably, MLTLM syntax is MLTL syntax with signal types associated with temporal operators.

Definition 3.2. (MLTLM Semi- Semantics) Given an arbitrary type $\mathbb{A} \in \mathcal{T}$, the evaluation of an MLTLM formula φ on a trajectory π produces a signal $\sigma^{\mathbb{A}}$, denoted by $(\pi, \varphi) \mapsto \sigma^{\mathbb{A}}$, defined recursively for any $\mathbb{B}, \mathbb{C}, \mathbb{D} \in \mathcal{T}$, where $(\pi, \varphi_1) \mapsto \sigma_1^{\mathbb{B}}$, $(\pi, \varphi_2) \mapsto \sigma_2^{\mathbb{C}}$, as follows.

$$\sigma^{\mathbb{A}}[i] := \begin{cases} T^{\mathbb{A}}_{\mathbb{A}_j}(\pi^{\mathbb{A}_j}_{p_j})[i], & \text{if } \varphi = p_j,\ p_j \in \mathcal{AP},\ \text{and } \sigma_j^{\mathbb{A}_j} \in \pi, \\ T^{\mathbb{A}}_{\mathbb{B}}(\neg\sigma_1^{\mathbb{B}}[i]), & \text{if } \varphi = \neg\varphi_1, \\ T^{\mathbb{A}}_{\mathbb{B}}(\sigma_1^{\mathbb{B}}[i]) \wedge T^{\mathbb{A}}_{\mathbb{C}}(\sigma_2^{\mathbb{C}}[i]), & \text{if } \varphi = \varphi_1 \wedge \varphi_2, \\ T^{\mathbb{A}}_{\mathbb{D}}(T^{\mathbb{D}}_{\mathbb{B}}(\sigma_1^{\mathbb{B}})[i..]\ \mathcal{U}_{[lb,ub]} T^{\mathbb{D}}_{\mathbb{C}}(\sigma_2^{\mathbb{C}})[i..]), & \text{if } \varphi = \varphi_1 \mathcal{U}_{[lb,ub,\mathbb{D}]} \varphi_2, \end{cases}$$

where $\sigma[i..]$ is the subsequence of signal σ starting from discrete point i and all operators are evaluated according to Def. 2.6.

Definition 3.2 imposes that binary logical operators operate on signals of the same type. Note that when evaluating the fourth case in Def. 3.2, the signal types produced by subformulas φ_1 and φ_2 must be projected into signals of the type associated with the temporal operator. Additional common operators like implication, disjunction, and globally are constructed by standard equivalence relations as in MLTL, with all derived temporal operators inheriting the type specifier on their interval bounds.

Atomic propositions do not have a type (and is not needed) in our approach. Similarly formulas do not have a type. Only signals have a type. For example, consider the atomic proposition $p :=$ (velocity $>$ 10 mph). This atomic proposition does not have a type. But one can sample verdicts for this proposition at every 10 s, every minute or every hour, or every kilometer, each giving a different "type" of signal.

We will examine several projection functions, however, writing MLTLM formulas requires only assurance of their existence, not their definition; this provides a separation of concerns we leverage to ease specification writing and linearizing the verification workflow. Nevertheless, a formula can only be evaluated after projection between types are specified (and therefore, *Semi-* Semantics). For example, consider a formula φ specifying that φ_1 should hold every hour for 10 hours, and φ_2 should hold every second for 100 seconds. In MLTLM, φ could be written as $\square_{[0,10,\text{hours}]}\varphi_1 \wedge \square_{[0,100,\text{seconds}]}\varphi_2$. In MLTL, φ would need to be written assuming a monitor rate, say seconds, as $\square_{[0,0]}\varphi_1 \wedge \square_{[3600,3600]}\varphi_1 \wedge \square_{[7200,7200]}\varphi_1 \wedge \cdots$ and $\square_{[0,100]}\varphi_2$. The formula is longer and embeds the relation between hours and seconds. If the specification must be evaluated at a monitor rate of minutes instead, the canonical encoding must be updated by the specification author as discussed in more detail in Section 1 (Fig. 1). In contrast, in MLTLM, the top view specification remains the same even in the face of implementation details like evaluation rate.

3.1 Evaluation of an MLTLM Formula

Evaluation of an MLTLM formula on a trajectory requires signals for all atomic propositions. Evaluating an MLTLM formula naming at most one type over a trajectory is equivalent to evaluating an MLTL formula over a computation containing signals of that type.

With projection, a new signal of a different type can be derived from an existing signal in the trajectory. For example, the return of a high-rate sensor can be down-sampled to match the type of a low rate sensor. This

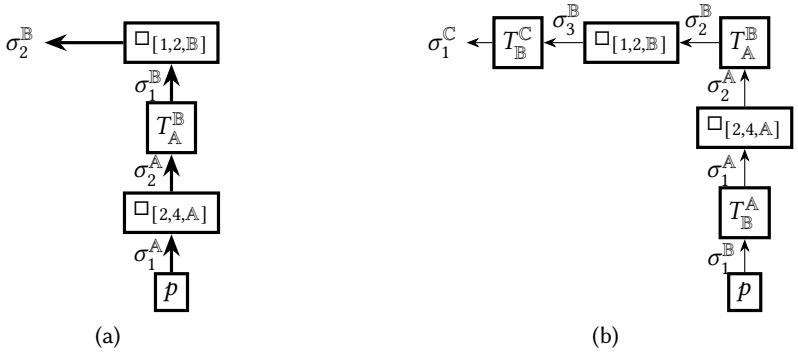


Fig. 2. Illustration of two possible evaluations of a formula $\square_{[1,2,\mathbb{B}]}(\square_{[2,4,\mathbb{A}]}p)$

.

"derived signal" evaluation is where all signals are first projected to a common type before evaluation. Using signals, types, and projection, we can evaluate a formula with mixed types by considering each subformula to represent the signal of its own evaluation and projecting where necessary as explained further in the next section.

Critically, operator semantics are defined for any type, but only when the input(s) and output types match. The inputs to the temporal logic operator must be projected to the written type in the operator's bound if needed. Fundamentally, MLTLM formulas represent a directed dataflow graph between domains of MLTL connected by projections.

Tutorial Example of an Application of the MLTLM Semantics (Def. 3.2). To clarify the semantics in Def. 3.2, consider the formula $\square_{[1,2,\mathbb{B}]}(\square_{[2,4,\mathbb{A}]}p)$. The global ($\square$) operator is a common unary temporal operator derived from the equivalence relation $\square_{[lb,ub,\mathbb{A}]}\varphi \Leftrightarrow \neg(\text{true}\ \mathcal{U}_{[lb,ub,\mathbb{A}]}\ \neg\varphi)$. This is the same as adding the following case to the MLTLM semantics:

$$\sigma^{\mathbb{A}}[i] := \text{true iff } \sigma_1^{\mathbb{A}}[j] = \text{true } \forall j \in [i+lb, i+ub],$$
$$\text{if } \varphi = \square_{[lb,ub,\mathbb{A}]}\varphi_1.$$

Applying Def. 3.2, the evaluation of formula $\square_{[1,2,\mathbb{B}]}(\square_{[2,4,\mathbb{A}]}p)$ depends on the type of any known signals for p and the desired output type. Consider generating a signal of type $\mathbb{B}$ from the above formula, and that $\pi_p^{\mathbb{A}}$ is known for p. In Fig. 2a, the known signal for p, $\sigma_1^{\mathbb{A}}$, is input to $\square_{[2,4,\mathbb{A}]}$ whose satisfaction signal, $\sigma_2^{\mathbb{A}}$, is first projected to $T_{\mathbb{A}}^{\mathbb{B}}(\sigma_2^{\mathbb{A}}) = \sigma_1^{\mathbb{B}}$ to match types with $\square_{[1,2,\mathbb{B}]}$, finally generating $\sigma_2^{\mathbb{B}}$ which meets the required output of type $\mathbb{B}$.

Fig. 2b considers another case with the same formula where we need an output signal of type $\mathbb{C}$, and know $\pi_p^{\mathbb{B}}$. In Fig. 2b, evaluating the subformula $\square_{[2,4,\mathbb{A}]}p$ requires a signal for p in type $\mathbb{A}$, as per the semantics, but we only know p in type $\mathbb{B}$. This implies a projection $T_{\mathbb{B}}^{\mathbb{A}}(\sigma_1^{\mathbb{B}}) = \sigma_1^{\mathbb{A}}$ before the result is input to $\square_{[2,4,\mathbb{A}]}$, generating $\sigma_2^{\mathbb{A}}$. Another type incompatibility arises between $\sigma_2^{\mathbb{A}}$ and $\square_{[1,2,\mathbb{B}]}$, so it is again (implicitly) projected to a type $\mathbb{B}$ through $T_{\mathbb{A}}^{\mathbb{B}}(\sigma_2^{\mathbb{A}}) = \sigma_2^{\mathbb{B}}$. Since the desired output type is $\mathbb{C}$, there is one last projection $T_{\mathbb{B}}^{\mathbb{C}}(\sigma_3^{\mathbb{B}}) = \sigma_1^{\mathbb{C}}$.

4 Example Specifications with Projections over Different Types

E1 Type space

(a) *The verbal specification*: "In the region 4 miles from the origin, the drone shouldn't exceed a velocity of 12 mph."

(b) *The formal MLTLM specification*: $\square_{[0,4,miles]}(v < 12)$

The formal specification says that the drone's speed should not exceed 12 mph 4 miles from the origin. The atomic proposition $(v < 12)$ may be sampled by a sensor say every mile, every feet, or every second. Correspondingly, the signal is projected to the type *miles* and then the temporal operator is evaluated as per the MLTL semantics. To illustrate different sampling frequencies, consider these scenarios:

Case 1 (Fig. 3) Assume that the proposition $(v < 12)$ is sampled every mile (Recall that atomic propositions and formulas do not have a type, only signals have a type, see Section 3). In this case, a projection is not needed as the atomic proposition is evaluated in the same type as its parent operator.

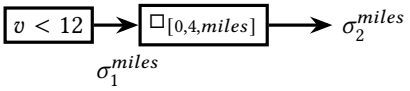


Fig. 3. Example **E1**, **Case 1**, when $(v < 12)$ is sampled every mile

Case 2 (Fig. 4) Assume that $(v < 12)$ is sampled every feet. In this case we project feet to miles. 1 mile is

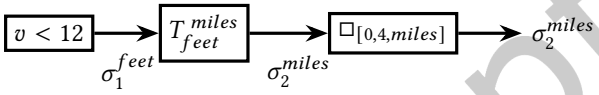


Fig. 4. Example **E1**, **Case 2**, when $(v < 12)$ is sampled every feet

5280 feet. Thus we need to convert every 5280 signal values into one signal value using a projection. Consider the following projection candidates:

Definition 4.1. (Modulo-Reduction Function) Let $\sigma^{\mathcal{A}}$ and $\sigma^{\mathcal{B}}$ denote the set of all signals of types $\mathbb{A}$ and $\mathbb{B}$ respectively. The function $f_s : \sigma^{\mathcal{A}} \to \sigma^{\mathcal{B}}$ implements the projection $T_{\mathbb{A}}^{\mathbb{B}}(\sigma^{\mathbb{A}})$ by modulo-reduction with positive integer stride s when:

$$f_s(\sigma^{\mathbb{A}}) = \sigma^{\mathbb{B}} \text{ such that } \sigma^{\mathbb{B}}[i] = \sigma^{\mathbb{A}}[i \cdot s] \tag{1}$$

The modulo-reduction function outputs every sth value from the input signal.

Definition 4.2. (Majority-Reduction Function) The function $g_s : \sigma^{\mathcal{A}} \to \sigma^{\mathcal{B}}$, implements the projection $T_{\mathbb{A}}^{\mathbb{B}}$ by majority-reduction with positive integer stride s when:

$$g_s(\sigma^{\mathbb{A}}) = \sigma^{\mathbb{B}} \text{ such that}$$

$$\sigma^{\mathbb{B}}[i] = \begin{cases} \text{true}, & \text{if } \mathcal{N}_0(\{j \in [i \cdot s, (i+1) \cdot s) : (\sigma^{\mathbb{A}}[j] = \text{true})\}) \geq \lfloor s/2 \rfloor, \\ \text{false}, & \text{otherwise}, \end{cases}$$

where $\mathcal{N}_0(\cdot)$ is the set cardinality.

The majority-reduction function outputs the majority value of every s values of the input signal.

Definition 4.3. (Anyone Function) The function $\exists_s : \sigma^{\mathcal{A}} \to \sigma^{\mathcal{B}}$, implements the Anyone projection $T_{\mathbb{A}}^{\mathbb{B}}$ with positive integer stride s when:

$$\exists_s(\sigma^{\mathbb{A}}) = \sigma^{\mathbb{B}} \text{such that}$$

$$\sigma^{\mathbb{B}}[i] = \begin{cases} \text{true}, & \text{if } \exists j \in [i \cdot s, (i+1) \cdot s), \sigma^{\mathbb{A}}[j] = \text{true}, \\ \text{false}, & \text{otherwise}. \end{cases}$$

Definition 4.4. (All Function) The function $\forall_s : \sigma^{\mathcal{A}} \rightarrow \sigma^{\mathcal{B}}$, implements the All projection $T^{\mathbb{B}}_{\mathbb{A}}$ with positive integer stride s when:

$$\forall_s(\sigma^{\mathbb{A}}) = \sigma^{\mathbb{B}} \text{such that}$$

$$\sigma^{\mathbb{B}}[i] = \begin{cases} \text{true}, & \text{if } \forall j \in [i \cdot s, (i+1) \cdot s), \sigma^{\mathbb{A}}[j] = \text{true}, \\ \text{false}, & \text{otherwise.} \end{cases}$$

The Anyone function projects to a true value from a stride of length s if at least one element in the stride is true. Similarly, the All function projects to true when all values in the stride is true.
We may use any of these projections in Defs. 4.1-4.4 with $s = 5280$, or some other projection that converts every 5280 signal values to a single value in the resultant signal. The choice of the projection determines the output signal, i.e., different projections may result in different signal outputs for the same formula.

Case 3 (Fig. 5) Assume that $(v < 12)$ is sampled every minute. We need to project a minute-type signal to a mile-type signal.

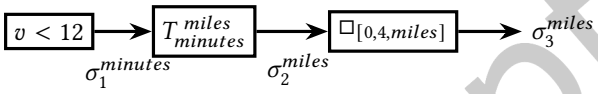


Fig. 5. Example **E1**, **Case 3**, when $(v < 12)$ is sampled every minute

As an example projection, consider the following:

Definition 4.5. (Time-to-Space projection) Let $v : \mathbb{Z} \rightarrow \mathbb{R}^+$ be the speed time series, let Δt be the sampling rate of the signal σ^{time}, and let $\bar{v}(i) = \sum_{j=0}^{i-1} v(j)/i$ be the average speed up to the ith instant. Consider a function $f : \mathbb{N} \rightarrow \mathbb{N}$, such that $f(i) = (\lfloor \bar{v}(i)\ \Delta t\ i \rfloor)$. The Time-to-Space projection is such that for each $i \geq 0$, the kth value of σ^{space} is assigned as

$$\sigma^{space}[k] \ = \ \sigma^{time}[i], \qquad \forall k \in [f(i), f(i+1)).$$

The Time-to-Space projection models the distance as the average speed times the time converted into an integer using the floor function. The truth value at a distance x not covered by the floor function is assigned the value at $(x - 1)$. Depending on the use-case, one may consider more complex, creative, and realistic relationships, e.g., include acceleration, use a differential equation or a machine learning model to relate signals sampled at a time rate to a signal sampled at a spatial rate.

E2 Types space and time

(a) *The verbal specification*: "The car drives at a velocity less than 60mph for a distance of 10 miles, and then maintains 70mph for 60 minutes."

(b) *The formal MLTLM specification*:

$$\square_{[0,10,miles]}(v < 60) \wedge \square_{[10,10,miles]}(\square_{[0,60,minutes]}(v == 70)) \tag{2}$$

The left-hand side of the conjunction in Eq. (2) expresses that the car drives at a velocity less than 60 mph for 10 miles, and the right-hand side expresses that the velocity is maintained at 70 mph for the next 60 minutes. Consider different scenarios based on how velocity is sampled:

Case 1 (Fig. 6a) The velocity is sampled every minute. For this, follow the same steps as in Case 3 of the previous example.

Case 2 (Fig. 6b) The velocity v is sampled every mile (e.g., by a GPS tracking device). Follow the same steps as Case 3 of the previous example.

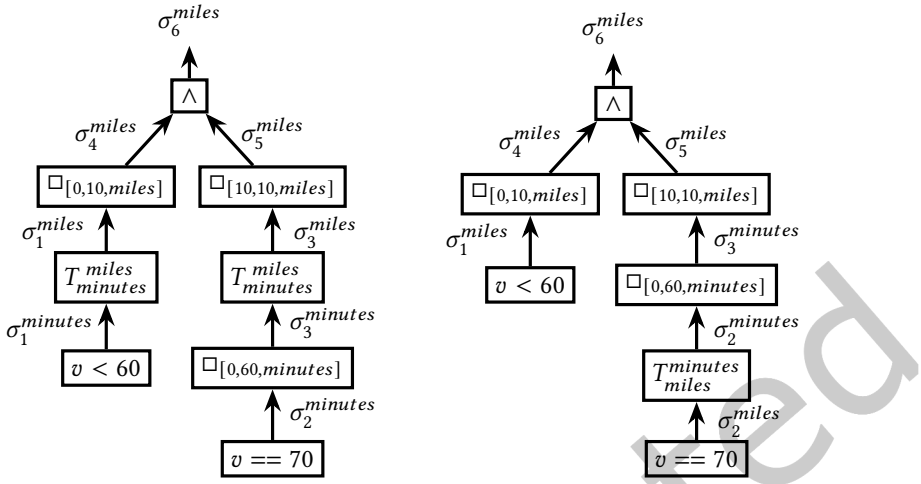


Fig. 6. Example **E2**, (a) **Case 1** and (b) **Case 2** where velocity v is sampled every minute and mile respectively

E3 Types of different time scales I

(a) *The verbal specification:* "The drone's surveillance camera needs to be in on state for 10 continuous minutes, every hour." (Assuming a mission-time of 3 hours)

(b) *The formal MLTLM specification:*

$$\square_{[0,3,hours]} \Diamond_{[0,50,minutes]} \square_{[0,10,minutes]} \text{camera-on}$$

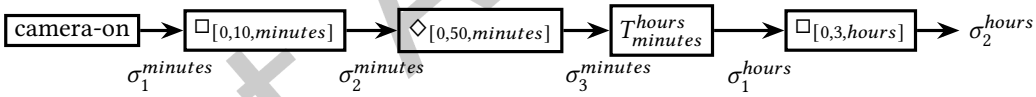


Fig. 7. Example **E3**, where camera-on is sampled every minute

Consider a signal of length 300 for "camera-on" in Fig. 7. A sample evaluation is as follows (we represent the signal as a (time-stamp,verdict) tuple for illustration purposes in this example; a signal is a boolean vector in the rest of this paper)

$$\begin{aligned}
\sigma_1^{minutes} &= [(0, \text{true}), (1, \text{true}), \ldots, (299, \text{true})],\\
\sigma_2^{minutes} &= [(0, \text{true}), (1, \text{true}), \ldots, (290, \text{true})],\\
\sigma_3^{minutes} &= [(0, \text{true}), (1, \text{true}), \ldots, (240, \text{true})],\\
\sigma_1^{hours} &= [(0, \text{true}), (1, \text{true}), (2, \text{true}), (3, \text{true})],\\
\sigma_2^{hours} &= [(0, \text{true})].
\end{aligned}$$

Here, $\sigma_1^{minutes}$ is generated by a sensor at a minute rate. This passes through the $\square_{[0,10,minutes]}$ operator, and generates a verdict for every 10 minutes, i.e., 0-10, 1-11, 2-12, ..., 290-300. Thus $\sigma_2^{minutes}$ has 291

verdicts. Then, $\sigma_2^{minutes}$ passes through $\Diamond_{[0,50,minutes]}$, and generates a verdict for every 50 minutes, i.e., 0-50, 1-51, 2-52, . . ., 240-290. Hence, $\sigma_3^{minutes}$ has 241 verdicts. Next, the minute-wise signal is projected into an hour-wise signal by going over $T_{minutes}^{hours}$. This projection can be anyone of the reduction functions (Defs. 4.1-4.4) with a stride of $s = 60$. As there are only 4 strides ($s = 60$) in a signal of length 240, the length of σ_1^{hours} is 4. Next, the $\Box_{[0,3,hours]}$ evaluates σ_1^{hours} to produce σ_2^{hours}, yielding a signal of length 1.

E4 Types of different time scales II

(a) *The verbal specification*: "The spacecraft maintenance cycle runs at least once a month over the five-year mission."

(b) *The formal MLTLM specification*:

$$\Box_{[0,5,years]}\Diamond_{[0,30,days]}\text{maintenance}$$

The evaluation of this specification is similar to Example **E3**.

E5 Types of different time scales III

(a) *The verbal specification*: "Verify monthly that the thrusters did not burn more than 3 seconds at a time"

(b) *The formal MLTLM specification*:

$$\Box_{[0,12,month]}(\neg\Box_{[0,3,seconds]}\text{burn-thrusters})$$

The evaluation of this specification is similar to Example **E3**.

5 Equivalent Formula in MLTL and Succinctness of MLTLM

We develop a theory to derive equivalent MLTL formula for an MLTLM formula with a class of logical projections. Next, we develop translators based on it with the modulo-reduction projection (Def. 4.1). We then prove succinctness of MLTLM formulas compared to translated equivalent MLTL formulas with the modulo-reduction projection. We only focus on the modulo-reduction projection as the theory can be readily extended to other projections.

5.1 Equivalent MLTLM Formula for Every MLTL Formula

For a formula naming at most one type, all properties that hold in MLTL hold in MLTLM, i.e., $\Diamond_{[lb,ub,\mathbb{A}]}\varphi \Leftrightarrow \text{true}\,\mathcal{U}_{[lb,ub,\mathbb{A}]}\,\varphi$, $\Box_{[lb,ub,\mathbb{A}]}\varphi \Leftrightarrow \neg(\Diamond_{[lb,ub,\mathbb{A}]}\neg\varphi)$ and so on. The following lemma expresses that formulas expressible in MLTL form a subset of formulas expressible in MLTLM. The lemma attests that there is no loss in using MLTLM compared to MLTL. The transformation is simple, and the formula is, at worst, the same length.

Lemma 5.1. *An equivalent MLTLM formula of the same length exists for every MLTL formula, and this translation is possible in constant time.*

Proof. We can represent any MLTL formula as an MLTLM formula by appending a signal type to the interval bound of every temporal operator. This follows from the definition of MLTLM. The formula length, being the total number of operators plus atomic propositions, is not affected by appending a type name to the temporal operators. Hence the resultant MLTLM formula is of the same length as the MLTL formula. □

5.2 Equivalent MLTL Formulas for MLTLM Formulas with Logical Projections

Lemma 5.2 (Adequate set of MLTL). *The operators ¬, ∧ and □ form an adequate set, i.e., any MLTL formula can be equivalently expressed exclusively with these operators.*

Proof. In propositional logic the adequate set is {¬,∧}, and this extends directly to temporal logic as well. Therefore, it suffices to show that any formula $\varphi_1\mathcal{U}_{[lb,ub]}\varphi_2$ can be equivalently expressed in MLTL using only

the $\Box$ operator. The definition of the until operator can be equivalently expanded as,

$$\begin{aligned}
\varphi_1 \mathcal{U}_{[lb,ub]} \varphi_2 \; &= \Box_{[lb,lb]} \varphi_2 \\
&\quad \vee (\Box_{[lb,lb]} \varphi_1 \wedge \Box_{[lb+1,lb+1]} \varphi_2) \\
&\quad \vee (\Box_{[lb,lb+1]} \varphi_1 \wedge \Box_{[lb+2,lb+2]} \varphi_2) \\
&\quad \vee (\Box_{[lb,lb+2]} \varphi_1 \wedge \Box_{[lb+3,lb+3]} \varphi_2) \\
&\quad \vdots \\
&\quad \vee (\Box_{[lb,ub-1]} \varphi_1 \wedge \Box_{[ub,ub]} \varphi_2), \\
&= \neg(\neg \Box_{[lb,lb]} \varphi_2 \\
&\quad \wedge \neg(\Box_{[lb,lb]} \varphi_1 \wedge \Box_{[lb+1,lb+1]} \varphi_2) \\
&\quad \wedge \neg(\Box_{[lb,lb+1]} \varphi_1 \wedge \Box_{[lb+2,lb+2]} \varphi_2) \\
&\quad \wedge \neg(\Box_{[lb,lb+2]} \varphi_1 \wedge \Box_{[lb+3,lb+3]} \varphi_2) \\
&\quad \vdots \\
&\quad \wedge \neg(\Box_{[lb,ub-1]} \varphi_1 \wedge \Box_{[ub,ub]} \varphi_2))
\end{aligned}$$

□

Definition 5.3 (Logical Projection). Let Φ be the set of MLTLM formulas. Consider a formula $\varphi = \Box_{[lb,ub,\mathbb{D}]} \varphi_1$ where $(\pi, \varphi_1) \mapsto \sigma^t$, where $t \in \{\mathbb{A}, \mathbb{B}, \dots\}$ and $t \neq \mathbb{D}$. Let $T_t^{\mathbb{D}}$ project σ^t to a signal of type $\mathbb{D}$. The projection is called a logical projection if there exists a function $g : \Phi \to \Phi$ such that $((\pi, g(\varphi)) \mapsto \sigma^t) \Leftrightarrow ((\pi, \varphi) \mapsto \sigma^{\mathbb{D}})$.

Theorem 5.4 (Expressive Equivalence of MLTL and MLTLM with Logical Projections). *Let all the atomic propositions of an MLTLM formula φ generate signals of the same type, $\mathbb{A}$. Furthermore, for every type t in φ, let there exist a chain of logical projections from t to $\mathbb{A}$. Then, there exists an MLTLM formula ψ, such that the signal generated by φ is equivalent to a signal of type $\mathbb{A}$ generated by ψ.*

Proof. Recollect that the translation is from a multi-type logic, MLTLM, to a single type logic, MLTL. Because we translate to a single-type logic, it is pre-requisite that the class of MLTLM formulas that have a translation to MLTL, have atomic propositions that generate signals of the same type. MLTLM formulas that don't satisfy this assumption cannot be translated to MLTL. We prove that

$$((\pi, \varphi) \mapsto \sigma^{\mathbb{D}}) \Leftrightarrow ((\pi, \psi) \mapsto \sigma^{\mathbb{A}}).$$

The proof can be succinctly expressed as a recursive function h,

$$h(\varphi) := \begin{cases} \varphi, & \text{if } \varphi \text{ has only one type, } \mathbb{A}, \text{ in the entire formula,} \\ g(\, \Box_{[lb,ub,t]} h(\varphi_1)\,), & \text{if } \varphi = \Box_{[lb,ub,t]} \varphi_1 \text{ and } t \neq \mathbb{A}, \\ \Box_{[lb,ub,\mathbb{A}]} h(\varphi_1), & \text{if } \varphi = \Box_{[lb,ub,\mathbb{A}]} \varphi_1, \\ \neg h(\varphi_1), & \text{if } \varphi = \neg \varphi_1, \\ h(\varphi_1) \wedge h(\varphi_2), & \text{if } \varphi = \varphi_1 \wedge \varphi_2, \end{cases} \tag{3}$$

where we use an abuse of notation for the function g to represent any of the different, but respective functions of corresponding logical projections. □

For clarity, we derive the logical projection function for the modulo-reduction projection (Def. 4.1) as an example. Let, $\varphi = \Box_{[lb,ub,\mathbb{B}]}\varphi_1$ and $(\pi, \varphi) \mapsto \sigma^{\mathbb{B}}$ and let $(\pi, \varphi_1) \mapsto \sigma^{\mathbb{A}}$, we have

$$
\begin{aligned}
& \sigma^{\mathbb{B}}[i] = \Box_{[lb,ub,\mathbb{B}]}\sigma_1^{\mathbb{B}}[i..], \\
\Leftrightarrow\ & \sigma^{\mathbb{B}}[i] = \text{true iff } \forall j \in [i+lb, i+ub],\ \sigma_1^{\mathbb{B}}[j] = \text{true}, \\
\Leftrightarrow\ & \sigma^{\mathbb{B}}[i] = \text{true iff } \forall j \in [i+lb, i+ub],\ \sigma^{\mathbb{A}}[j\,s] = \text{true}, \\
\Leftrightarrow\ & \sigma^{\mathbb{B}}[i] = \text{true iff } \forall j' \in [(i+lb)\,s, (i+ub)\,s,\ s],\ \sigma^{\mathbb{A}}[j'] = \text{true}, (\text{ where } j' = j\,s) \\
\Leftrightarrow\ & \sigma^{\mathbb{A}}[i'] = \Box_{[lb\,s,lb\,s]}(\sigma^{\mathbb{A}}[i'] \wedge \Box_{[s,s]}\sigma^{\mathbb{A}}[i'] \wedge \Box_{[2s,2s]}\sigma^{\mathbb{A}}[i'] \wedge \cdots \wedge \Box_{[(ub-lb)s,(ub-lb)s]}\sigma^{\mathbb{A}}[i']), (\text{ where } i' = i\,s)
\end{aligned}
$$

where $[lb, ub, s]$ represents a set starting from lb, ending at ub, with a stride of length s. Using the formula notation, we have that $((\pi, g(\varphi)) \mapsto \sigma^{\mathbb{A}}) \Leftrightarrow ((\pi, \varphi) \mapsto \sigma^{\mathbb{B}})$ where

$$
\begin{aligned}
g(\varphi)\ = \Box_{[lb\,s,lb\,s,\mathbb{A}]}(&\varphi_1 \wedge \Box_{[s,s,\mathbb{A}]}\varphi_1 \wedge \Box_{[2s,2s,\mathbb{A}]}\varphi_1 \wedge \cdots \\
&\wedge \Box_{[(ub-lb)s,(ub-lb)s,\mathbb{A}]}\varphi_1).
\end{aligned}
$$

5.3 Translations from MLTLM to MLTL for Formulas with the Modulo-Reduction Projection

We developed four translators from MLTLM to MLTL based on the recursive formula Eq. 3. The four translators vary in their expansions of the Until operator as follows. For a formula $p\ \mathcal{U}_{[1,4,\mathbb{B}]}q$, where p and q generate signals of type $\mathbb{A}$ the four translators expand as the following equivalent MLTL formulas assuming that $s = 2$ in the modulo-reduction projection (Eq. (1)):

(1) Translator 1:

$$
\begin{aligned}
&((\Box_{[2,2]}(q)) \\
&\vee (\Box_{[2,2]}(p) \wedge \Box_{[4,4]}(q)) \\
&\vee (\Box_{[2,2]}(p) \wedge \Box_{[4,4]}(p) \wedge \Box_{[6,6]}(q)) \\
&\vee (\Box_{[2,2]}(p) \wedge \Box_{[4,4]}(p) \wedge \Box_{[6,6]}(p) \wedge \Box_{[8,8]}(q)))
\end{aligned}
$$

This translation is the most expanded form. This version is a raw expansion of the until operator (Def. 3.2).

(2) Translator 2:

$$
\begin{aligned}
&\Box_{[2,2]}(q \\
&\vee (p \wedge \Box_{[2,2]}(q)) \\
&\vee (p \wedge \Box_{[2,2]}(p) \wedge \Box_{[4,4]}(q)) \\
&\vee (p \wedge \Box_{[2,2]}(p) \wedge \Box_{[4,4]}(p) \wedge \Box_{[6,6]}(q)));
\end{aligned}
$$

This translation factors out a global operator from the rest of the formula.

(3) Translator 3: The same as Translator 2, but with the addition that common global operators over conjunctions and disjunctions are taken out, e.g., the formula $\Box_{[2,2]}p \wedge \Box_{[4,6]}q$ is translated to $\Box_{[2,2]}(p \wedge \Box_{[2,4]}q)$.

(4) Translator 4:

$$
\Box_{[2,2]}(q \vee \Box_{[2,2]}(p \wedge (q \vee \Box_{[2,2]}(p \wedge (q \vee \Box_{[2,2]}(p \wedge \Box_{[2,2]}(q)))))));
$$

Translator 4 expands from the next normal form of an Until expression. We prove that this is the shortest expansion of the until operator in MLTLM to an equivalent MLTL formula when using the modulo-reduction projection, yielding succinctness results of MLTL vs MLTLM. We recall that we proved that every MLTL formula can be expressed in MLTLM with the same formula length (Lemma 5.1). We now prove that an equivalent formula

to $p\ \mathcal{U}_{[lb,ub,\mathbb{B}]}q$ in MLTL is at least $(ub - lb)$ long. We first consider succinctness of LTL and MLTL. We derive results based on the Θ operator as has been the standard in other succinctness proofs in literature [1, 25]

Lemma 5.5. *(Succinctness of MLTL vs LTL) The property expressed as $p\,\mathcal{U}_{[lb,ub]}q$ where $p, q \in \mathcal{AP}$, cannot be expressed in an LTL formula of size less than ub.*

Proof. The adequate set of operators for an LTL formula is $\{\neg,\ \wedge,\ \mathcal{U},\ \bigcirc\}$ [3]. The formula $p\ \mathcal{U}_{[lb,ub]}q$ says that the property p holds until there exists a state such that q holds inside the finite interval $[lb, ub]$. Therefore, an equivalent LTL formula must equivalently express that the property q holds in at least one state in the range $[lb, ub]$. However, the until operator in LTL reasons over infinite states, and cannot by itself reason over a finite set of states in an interval. Propositional operators $(\neg, \wedge)$ cannot reason across states. Therefore, expressing that q holds in at least one of the states inside $[lb, ub]$ needs the next operator in LTL $(\bigcirc)$. The number of $\bigcirc$ in the equivalent LTL formula is at least ub to reason up to the last state in $p\ \mathcal{U}_{[lb,ub]}q$. Therefore, the size of the equivalent LTL formula is at least ub. □

We note that the next normal form of an Until expression $p\ \mathcal{U}_{[lb,ub]}q$ is expanded as:

$$\begin{aligned}
&\Box_{[lb,lb]}(p\ \mathcal{U}_{[0,ub-lb]}q)\\
=&\Box_{[lb,lb]}(q \vee (p \wedge \Box_{[1,1]}(p\ \mathcal{U}_{[0,ub-lb-1]}q)))\\
=&\Box_{[lb,lb]}(q \vee (p \wedge \Box_{[1,1]}(q \vee (p \wedge \Box_{[1,1]}(p\ \mathcal{U}_{[0,ub-lb-2]}q)))))
\end{aligned}$$

Looking from the first equation in the above set of equivalent formulas, this can be expanded n times until $ub - lb - n$ becomes zero. The equivalent formula for the above in LTL is

$$\bigcirc^{lb}(q \vee (p \wedge \bigcirc(q \vee (p \wedge \bigcirc(q \vee \ldots))))),$$

where $\bigcirc^s$ is a shorthand notation for $\bigcirc$ repeated s times. There are lb next operators outside, and $ub - lb$ next operators inside the outermost brackets, thus there are $\Theta(ub)$ operators in this expansion, and therefore this is an optimal expansion of the Until operator from MLTL to LTL.

Lemma 5.6. *(Succinctness of MLTLM vs MLTL with the Modulo-Reduction Function) Let $p, q \in \mathcal{AP}$ generate signals of type $\mathbb{A}$. Let $\mathbb{B}$ be a type such that the projection from type $\mathbb{A}$ to $\mathbb{B}$ is a modulo-reduction function with a stride of length $s > 1$. Then, the property expressed as $p\ \mathcal{U}_{[lb,ub,\mathbb{B}]}q$ cannot be expressed as an MLTL formula of length less than $(ub - lb)$.*

Proof. According to the until semantics, the property q needs to hold at least in one state between $[lb\ s, ub\ s, s]$, i.e., between $lb\ s$ and $ub\ s$ with a stride of length $s > 1$. Therefore the equivalent MLTL formula must express that q holds in one of every sth state in the interval $[lb\ s, ub\ s]$. The until operator of MLTL reasons in the full interval, and not for every sth state in the interval. The sth state is shortest referred by $\Box_{[s,s]}$ (this uses a single operator). Referring to the next $2s$th state from the current state may be done using $\Box_{[2s,2s]}$ from the current state, or $\Box_{[s,s]}$ from the sth state, in either case using a single operator. This caries on till the end of the interval. Thus, referring to every sth state needs at least $(ub\ s - lb\ s)/s$ operators. This is reminiscent of the argument for the expressive limitation of LTL [41, Corollary 4.2] □

Translator 4 uses an expansion based on the next normal form, and thus for each until operator $\mathcal{U}_{[lb,ub,\mathbb{B}]}$, it produces an MLTL formula of length $\Theta(ub - lb)$, and is optimally short. The four translators were used to translate 70 randomly generated MLTLM formulas, and the resultant MLTL formulas were set as specifications to the R2U2 RV engine [26, 27]. All four translators, and our MLTLM monitor extension produced consistent output signals for 53 different input signals, each of length 4000, confirming the theory presented in this section.

6 An Implementation of an MLTLM Monitor with the Modulo-Reduction Projection

6.1 Implementation Details

We now illustrate space and time optimization possibilities by implementing an MLTLM RV engine. The generic syntax and semantics of MLTLM separates the specification from the signal type, i.e., the specification remains the same irrespective of the signal type. It is apparent from the semantics (Def. 3.2) that the output signal type is determined only in the fourth case with the temporal operator. For example, the formula $p \wedge q$ represents multiple output signal types depending on the trajectory types used for p and q, whereas the formula $\Box_{[0,0,\mathbb{A}]}(p \wedge q)$ has a single output type $\mathbb{A}$ irrespective of the trajectory types used for p and q. Our implementation needs a single output type, and hence considers a subset of MLTLM formulas that have a temporal operator at the root of their ASTs, and assumes that the type on the root temporal operator is the desired output type (we call this **the root node type imposition**). Formulas without a temporal operator as their ASTs' root are assigned a default output type, $\mathbb{D}$.

Furthermore, to make the evaluation of an MLTLM formula complete, two more ingredients are essential, (a) the placement of projections in the AST of an MLTLM formula and (b) defined projections between type signals. Consider the MLTLM formula, $\Box_{[0,0,\mathbb{A}]}(p \wedge q)$. Let us assume that only a signal of type $\mathbb{B}$ is available from p and a signal of type $\mathbb{C}$ from q, as denoted in Fig. 8a. From the semantics Def. 3.2, it is clear that a conjunction is allowed only between signals of the same type, which implies that there are implicit projections to match signal types in the conjunction as shown in Fig. 8b.

We have two (out of many) options here to match types, (a) to project to a common signal type $\mathbb{D}$ at the conjunction, and then to a type $\mathbb{A}$ to match type in $\Box_{[0,0,\mathbb{A}]}$ (Fig. 8b), and (b) place a projection to type $\mathbb{A}$ at the conjunction, then a second projection is not needed to match types in $\Box_{[0,0,\mathbb{A}]}$ (Fig. 8c). While the former option is of interest in the broader scope of applications with MLTLM like signal processing, the latter is the situation with the minimal number of projections. *The generalization for this minimal projection placement is to impose that signals are projected to the type of the closest ancestor node with a type* (**the closest ancestor type imposition**). All nodes in the unique path connecting a node to the root of the AST are ancestor nodes of the node (the node inclusive). In this example, the closest ancestor of the conjunction is $\Box_{[0,0,\mathbb{A}]}$ whose type is $\mathbb{A}$. We further assume that all such projections exist to evaluate a formula. We emphasize that the root-node imposition is only used to showcase a preliminary MLTLM monitor so that projection to types can become automated.

Our MLTLM RV engine currently only implements the modulo-reduction projection (see Def. 4.1). The MLTLM engine has added projection operators (see Def. 2.4) at appropriate places according to the semantics of MLTLM (Def. 3.2) respecting the closest ancestor type imposition. The modulo-reduction projection operator drops the appropriate signal values not needed in evaluating a formula and reports the output signal type corresponding to the type in the root of the AST of the formula.

6.2 Optimization Results with Random Formulas

We randomly draw MLTLM formulas using the procedure in [18] and plot the length of MLTL translations. The randomly drawn formulas are parametrized by the probability of drawing a temporal operator (P), the maximum difference between the lower and upper bounds (M), and the maximum signal length (T). We fix $M = T = 6$ in our study here. Furthermore, the memory and time also depend on stride, s of the modulo-reduction function (see Eq. (1)). In real systems, specifications may reason over say, seconds, minutes, hours and days, which correspond to $s = 60$, and 24. We conservatively consider four signal types, which we call $\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$ and $\mathbb{D}$, where (see Eq. (1) for $f_s(\sigma)$), with

$$f_2(\sigma^{\mathbb{A}}) = \sigma^{\mathbb{B}}, \quad f_3(\sigma^{\mathbb{B}}) = \sigma^{\mathbb{C}}, \quad f_4(\sigma^{\mathbb{C}}) = \sigma^{\mathbb{D}},$$

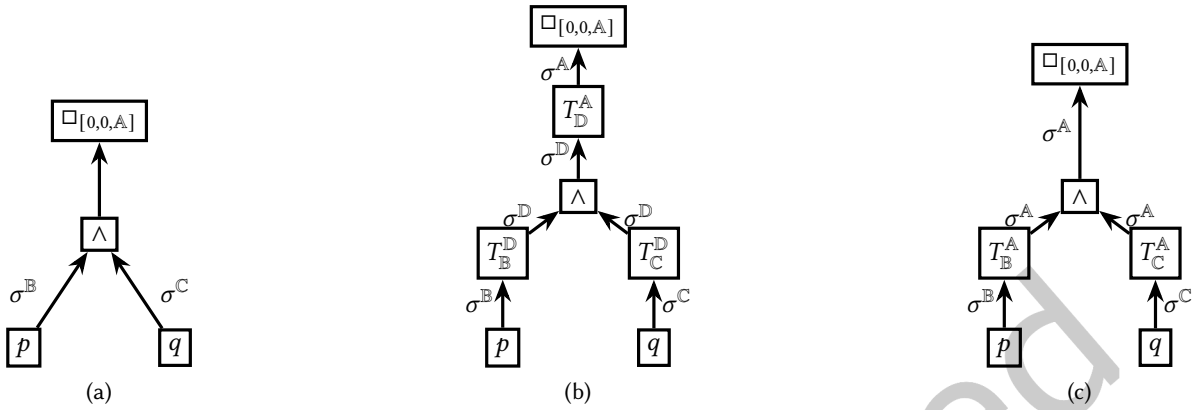


Fig. 8. The evaluation of an MLTLM formula depends on the placement of projections to match types in binary operators

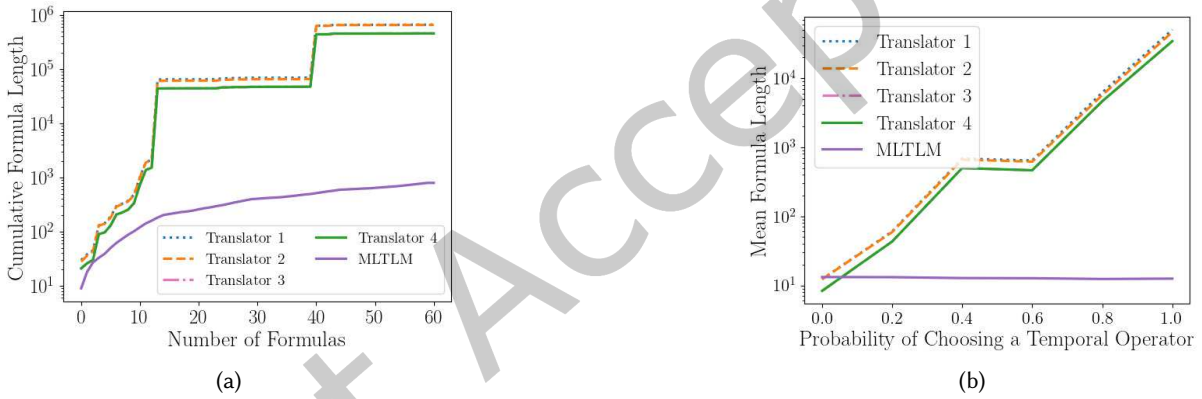


Fig. 9. (a) Cumulative formula length with the number of randomly drawn formulas with $P = 0.5$, and (b) mean formula length against the probability of choosing a temporal operator

with stride lengths $s = 2, 3, 4$. Note that the memory savings will be much larger with a stride like $s = 60$ (e.g., from second to minute).

Fig. 9a shows the cumulative formula length with randomly drawn formulas. At $P = 0.5$, the four translators produce MLTL formulas of nearly the same length. Among the translators, Translator 4 performs better due to being optimally short (Lemma 5.6). In contrast, the formula lengths of the MLTLM formulas are substantially smaller. Note that the plots are on log-scale; the difference between Translator 4 and the rest is about 200,000 units. Hence, there is no loss in using MLTLM in comparison to MLTL (see Section 3), but using MLTLM will result in much smaller and more intuitive formulas depending on the projection function.

Fig. 9b shows the mean formula length (averaged over 60 random formulas) by varying the probability of choosing a temporal operator. $P = 0$ corresponds to no temporal operators, and in that case, the translators and the MLTLM formula perform nearly equally well. This is expected – if a formula contains mere propositional

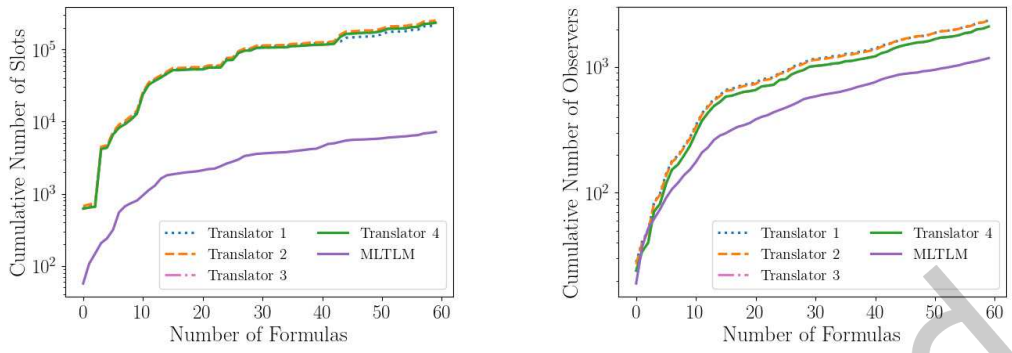


Fig. 10. Cumulative (a) memory and (b) time needed to verify random MLTLM formulas vs translated equivalent MLTL formulas

logic, its length should be independent of the temporal specification language. However, on close observation, the MLTLM formula at $P = 0$ is slightly longer. This is because in MLTL there is only one signal type, hence there is no need for a output signal type specifier, whereas in MLTLM, a proposition (say, $p \wedge q$, $p, q \in \mathcal{AP}$) represents a family of outputs of different types. As per the root node imposition (Section 6.1), we always use a temporal operator at the start of any formula (as in $\Box_{[0,0,\mathbb{B}]}(p \wedge q)$ in the place of $p \wedge q$), and this adds to excess length of an MLTLM formula compared to an MLTL formula with propositions. However, propositions like $p \wedge q$ are valid MLTLM formulas, but our implementation needs an output-type identifier.

On increasing the probability of choosing a temporal operator, the equivalent formulas in MLTL become significantly longer owing to the expansion to the base type as discussed in Section 5. Fig. 10 shows the estimated resource and time requirements on hardware. The memory to evaluate a formula is statically assigned in R2U2 [27] as dynamic memory is often not permitted in flight software. Hence, we compare the amount of static memory that needs to be assigned for equivalent formulas in MLTLM and (translated) MLTL (Fig. 10a). Similarly, the time taken for formula evaluation is directly proportional to the number of nodes created in the AST. We call the nodes in the AST as observers (as seen in the Y axis labels of Fig. 10b). We see that equivalent formula require much less memory in MLTLM than MLTL (Fig. 10a). Similarly, the evaluation time is faster for MLTLM as it needs many fewer observers (Fig. 10b).

6.3 Optimization Results with Real Formulas

We now consider the real formula examples, E3, E4 and E5 from Section 4. We use the modulo-reduction function as the projection between types. Recall that the formulas are

E1 $\Box_{[0,3,hours]}\Diamond_{[0,50,minutes]}\Box_{[0,10,minutes]}$camera-on
E2 $\Box_{[0,5,years]}\Diamond_{[0,30,days]}$maintenance
E3 $\Box_{[0,12,month]}(\neg\Box_{[0,3,seconds]}$burn-thrusters)

For E1, the stride length is 60 for an hour to minute conversion. For E2, the stride length assumed 365, for a conversion from day to year. For E3, the stride length is 2592000, assuming that a month is 30 days. The corresponding MLTL formulas are obtained using the most succinct translator, Translator 4 in Section 5.3.

Fig. 11a shows that in all cases, MLTLM (dark) uses much less memory compared to MLTL. In specific, for large stride lengths, the savings is much greater, as in E5. This is expected as MLTLM compacts data using the

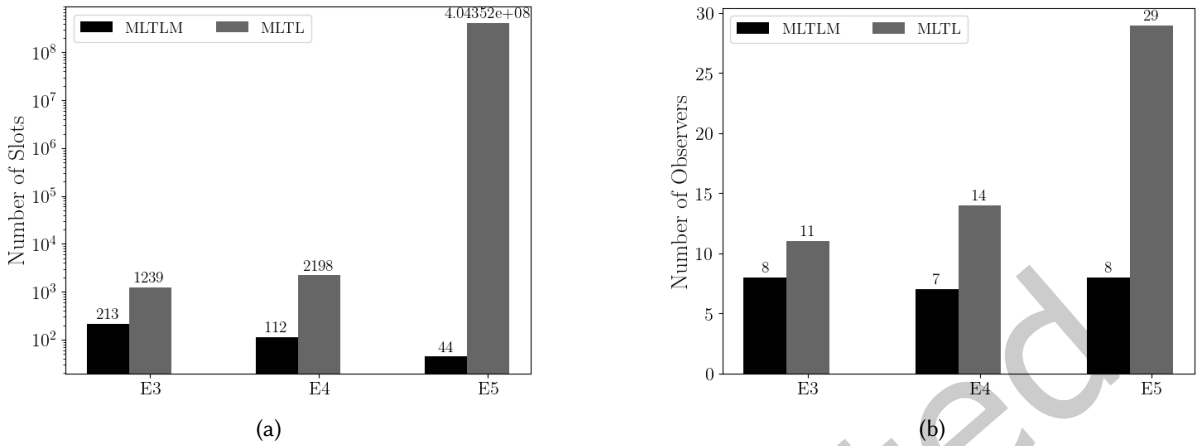


Fig. 11. (a) Memory and (b) time needed to verify real MLTLM formulas from Section 4, E3, E4, and E5, vs translated equivalent (most succinct) MLTL formulas

projection, whereas the MLTL monitor would store that same data in memory until a verdict can be ascertained. This demonstrates that MLTLM is potentially useful in resource constrained hardware, allowing monitoring a broader spectrum of specifications that would be nearly impossible using MLTL. Fig. 11b shows the length of instructions needed in hardware (denotes the speed of evaluating a specification) using MLTLM and MLTL. In all cases, MLTLM has a smaller instruction length, i.e., a greater speed of evaluation.

We end this section with a few remarks. Results show that there is great opportunity to have short intuitive formulas that encode timescales directly in the formula to simplify the workflow (Fig. 1), and in addition, an optimally configured RV engine for MLTLM is likely to have profound memory savings making it more suitable for resource constrained hardware.

7 Conclusion

Writing specifications naturally needs reasoning across multiple signal types, be it signals coming from different sensors at different rates, or belonging to observers in parallel universes (distributed systems). We developed a multi-type logic to express such specifications, and then explored succinctness and memory savings when considering the modulo-reduction projection. As discussed, this serves multiple purposes: 1) for the user, specifications are easy to write, 2) the theoretical satisfaction in different types is defined unambiguously, and 3) implementations can better utilize resources when compared with a single signal-type logic. Moreover, we expect that MLTLM will simplify the workflow by keeping the syntax simple and accessible, and postponing the nuances into the projection function. More importantly, MLTLM separates the specification from signal type. For example, let us suppose that a pressure sensor is changed in the Lunar Gateway, and it generates data in a different rate than the old sensor, or perhaps in a different unit like Pascals in the place of atmospheric pressure. Specifications for a single type logic would have to be changed to account for the signal type. MLTLM side-steps this process: The signal type will not affect the specification in any manner. In the future, we plan to have an automated assistant, that will allow a user to choose different projections in different contexts of specifications, (like “at least”, “at most”, “only once” etc.), and will also inform the user about the amount of memory he will need to dedicate/save on the hardware (the memory needed may vary based on the type of projection). This will allow the industrial verification community to seamlessly move to a multi-type logic. We will also consider more human

authored MLTLM specifications on real systems to get a better perspective on optimization opportunities. Lastly, the MLTLM monitor built upon R2U2 was validated across a regression suite of specifications and trajectories, but the current implementation can be improved to have tighter bounds on memory usage, which needs further investigation.

References

[1] Micah Adler and Neil Immerman. 2003. An n! lower bound on formula size. *ACM Transactions on Computational Logic (TOCL)* 4, 3 (2003), 296–314.

[2] James F. Allen and Patrick J. Hayes. 1985. A Common-Sense Theory of Time. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1* (Los Angeles, California) *(IJCAI'85)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 528–531.

[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.

[4] Philippe Balbiani. 2008. Time Representation and Temporal Reasoning from the Perspective of Non-Standard Analysis. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning* (Sydney, Australia) *(KR'08)*. AAAI Press, 695–704.

[5] Stefano Baratella and Andrea Masini. 2020. A two-dimensional metric temporal logic. *Mathematical Logic Quarterly* 66, 1 (2020), 7–19. https://doi.org/10.1002/malq.201700036

[6] Omar Bataineh, David S. Rosenblum, and Mark Reynolds. 2019. Efficient Decentralized LTL Monitoring Framework Using Tableau Technique. 18, 5s, Article 87 (2019), 21 pages.

[7] Andreas Bauer and Yliès Falcone. 2012. Decentralised LTL Monitoring. In *FM 2012: Formal Methods*, Dimitra Giannakopoulou and Dominique Méry (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 85–100.

[8] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. 2003. The synchronous languages 12 years later. *Proc. IEEE* 91, 1 (2003), 64–83. https://doi.org/10.1109/JPROC.2002.805826

[9] Gérard Berry and Ellen Sentovich. 2001. Multiclock esterel. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. Springer, 110–125.

[10] Paolo Bottoni and Andrew Fish. 2011. Policy specifications with Timed Spider Diagrams. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 95–98. https://doi.org/10.1109/VLHCC.2011.6070385

[11] Howard Bowman and Simon Thompson. 2003. A Decision Procedure and Complete Axiomatization of Finite Interval Temporal Logic with Projection. *Journal of Logic and Computation* 13, 2 (2003), 195–239. https://doi.org/10.1093/logcom/13.2.195

[12] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties. In *Principles of Security and Trust*, Martín Abadi and Steve Kremer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284.

[13] James Clifford and Ahobala Rao. 1986. A simple, general structure for temporal domains. (1986).

[14] Quentin Cohen-Solal, Maroua Bouzid, and Alexandre Niveau. 2015. An Algebra of Granular Temporal Relations for Qualitative Reasoning. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) *(IJCAI'15)*. AAAI Press, 2869–2875.

[15] Jean-Louis Colaço, Bruno Pagano, Cédric Pasteur, and Marc Pouzet. 2018. Scade 6: From a kahn semantics to a kahn implementation for multicore. In *2018 Forum on Specification & Design Languages (FDL)*. IEEE, New York, 5–16.

[16] James B Dabney, Julia M Badger, and Pavan Rajagopal. 2021. Adding a Verification View for an Autonomous Real-Time System Architecture. In *AIAA Scitech 2021 Forum*. 0566.

[17] James B. Dabney, Julia M. Badger, and Pavan Rajagopal. 2023. Trustworthy Autonomy for Gateway Vehicle System Manager. In *2023 IEEE Space Computing Conference (SCC)*. 57–62. https://doi.org/10.1109/SCC57168.2023.00018

[18] Marco Daniele, Fausto Giunchiglia, and Moshe Y Vardi. 1999. Improved automata generation for linear temporal logic. In *International Conference on Computer Aided Verification*. Springer, 249–260.

[19] Alexandre Donzé. 2013. On Signal Temporal Logic. In *Runtime Verification*, Axel Legay and Saddek Bensalem (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 382–383.

[20] Jerome Euzenat and Angelo Montanari. 2005. Time granularity. *Handbook of Temporal Reasoning in Artificial Intelligence* (January 2005).

[21] Massimo Franceschet, Angelo Montanari, Adriano Peron, and Guido Sciavicco. 2006. Definability and decidability of binary predicates for time granularity. *Journal of Applied Logic* 4, 2 (June 2006), 168–191. https://doi.org/10.1016/j.jal.2005.06.004

[22] Johannes Geist, Kristin Yvonne Rozier, and Johann Schumann. 2014. Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems. In *Proceedings of the 14th International Conference on Runtime Verification (RV14)*, Vol. 8734. Springer-Verlag, 215–230.

[23] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. 1991. The synchronous data flow programming language LUSTRE. *Proc. IEEE* 79, 9 (1991), 1305–1320.

[24] Gokul Hariharan, Brian Kempa, Tichakorn Wongpiromsarn, Phillip H. Jones, and Kristin Y. Rozier. 2022. MLTL Multi-type (MLTLM): A Logic for Reasoning About Signals of Different Types. In *Software Verification and Formal Methods for ML-Enabled Autonomous Systems*, Omri Isac, Radoslav Ivanov, Guy Katz, Nina Narodytska, and Laura Nenzi (Eds.). Springer International Publishing, 187–204.
[25] Neil Immerman. 2012. *Descriptive complexity*. Springer Science & Business Media.
[26] Chris Johannsen, Phillip Jones, Brian Kempa, Kristin Yvonne Rozier, and Pei Zhang. 2023. R2U2 Version 3.0: Re-Imagining a Toolchain for Specification, Resource Estimation, and Optimized Observer Generation for Runtime Verification in Hardware and Software. In *Computer Aided Verification*, Constantin Enea and Akash Lal (Eds.). Springer Nature Switzerland, 483–497.
[27] Brian Kempa, Pei Zhang, Phillip H. Jones, Joseph Zambreno, and Kristin Yvonne Rozier. 2020. Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2. In *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS) (Lecture Notes in Computer Science (LNCS), Vol. 12288)*. Springer, Vienna, Austria, 196–214.
[28] Ugo Dal Lago, Angelo Montanari, and Gabriele Puppis. 2007. Compact and tractable automaton-based representations of time granularities. *Theoretical Computer Science* 373, 1 (2007), 115–141. https://doi.org/10.1016/j.tcs.2006.12.014
[29] Ugo Dal Lago, Angelo Montanari, and Gabriele Puppis. 2007. On the Equivalence of Automaton-Based Representations of Time Granularities. In *14th International Symposium on Temporal Representation and Reasoning (TIME'07)*. 82–93. https://doi.org/10.1109/TIME.2007.56
[30] Jianwen Li, Moshe Y. Vardi, and Kristin Y. Rozier. 2019. Satisfiability Checking for Mission-Time LTL. In *Proceedings of 31st International Conference on Computer Aided Verification (CAV) (LNCS, Vol. 11562)*. Springer, New York, NY, USA, 3–22.
[31] Zachary Luppen, Michael Jacks, Nathan Baughman, Benjamin Hertz, James Cutler, Dae Young Lee, and Kristin Yvonne Rozier. 2022. 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.
[32] Menna Mostafa and Borzoo Bonakdarpour. 2015. Decentralized Runtime Verification of LTL Specifications in Distributed Systems. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 494–503.
[33] Naoko Okubo. 2020. Using R2U2 in JAXA program. Electronic correspondence. 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.
[34] J. Ouaknine and J. Worrell. 2008. Some Recent Results in Metric Temporal Logic. In *Formal Modeling and Analysis of Timed Systems*, Franck Cassez and Claude Jard (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–13.
[35] Thomas Reinbacher, Kristin Y. Rozier, and Johann Schumann. 2014. 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)*. Springer-Verlag, 357–372.
[36] K.Y. Rozier and M.Y. Vardi. 2010. LTL Satisfiability Checking. *International Journal on Software Tools for Technology Transfer (STTT)* 12, 2 (March 2010), 123 – 137. https://doi.org/DOI10.1007/s10009-010-0140-3
[37] Kristin Y. Rozier. 2011. Linear Temporal Logic Symbolic Model Checking. *Computer Science Review* 5, 2 (2011), 163–203. https://doi.org/10.1016/j.cosrev.2010.06.002
[38] Kristin Yvonne Rozier and Johann Schumann. 2017. R2U2: Tool Overview. In *Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES)*, Vol. 3. Kalpa Publications, Seattle, WA, USA, 138–156.
[39] Johann Schumann, Patrick Moosbrugger, and Kristin Y. Rozier. 2016. Runtime Analysis with R2U2: A Tool Exhibition Report. In *Proceedings of the 16th International Conference on Runtime Verification (RV15)*. Springer-Verlag, Madrid, Spain.
[40] Moshe Y. Vardi. 2001. Branching vs. Linear Time: Final Showdown. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tiziana Margaria and Wang Yi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–22.
[41] Pierre Wolper. 1983. Temporal logic can be more expressive. *Information and Control* 56, 1 (1983), 72–99.

Received 28 December 2023; revised 11 August 2024; accepted 18 October 2024