Language Partitioning for Mission-time Linear Temporal Logic*

Alec E. Rosentrater[1 [0009−0007−8186−3631]**], Zili Wang[1 [0000−0003−1730−6180]**], Katherine Kosaian[2 [0000−0002−9336−6006]], Kristin Y. Rozier[1 [0000−0002−6718−2828]]

[1] Iowa State University, Ames, USA
{alecrose, ziliw1, kyrozier}@iastate.edu
[2] University of Iowa, Iowa City, USA
katherine-kosaian@uiowa.edu

Abstract. *Mission-time Linear Temporal Logic (MLTL)* is a popular variant of Linear Temporal Logic that introduces discrete, finite interval time bounds on the temporal operators. MLTL specifications reason about system behaviors encoded as finite *traces* that track values of key variables over time. The *language* of an MLTL specification is all the traces that satisfy the specification. A natural question is: Given an MLTL specification φ, can we find a set of related formulas $\psi_1, \ldots, \psi_n$ so that the language of φ is the disjoint union of the languages of $\psi_1, \ldots, \psi_n$ (i.e., the ψ's *partition* the language of φ)? Answering this is not only theoretically interesting, but could also facilitate verification, as language partitioning is useful for creating benchmark suites or optimizing model checking algorithms. We present an algorithm for MLTL language partitioning and prove it correct. Because the proofs are technically intricate, we formalize them in the theorem prover Isabelle/HOL to ensure correctness. We automatically obtain an implementation of our algorithms via code generation from Isabelle/HOL and conduct an experimental evaluation to demonstrate the practicality of using our algorithms.

Keywords: Mission-time Linear Temporal Logic · Language · Partition · Isabelle/HOL

1 Introduction

Partitioning is an essential capability for utility (ability to solve problems, e.g., via decomposition into subproblems), scalability (ability to solve complex, realistic problem instances), and parallelization (ability to utilize modern multi-core architectures). Partitioning of the transition relations in automata initially enabled BDD-based model checking to scale to large state spaces. Specifically, partitioning operations for LTL, e.g., the characteristic function of an LTL formula, made possible the first symbolic model checking algorithm for LTL [2]. Consequently, partitioning underlies other BDD-based verification algorithms such as

* Work supported in part by NSF GRFP - 2024364991 and NSF:CPS Award 2038903.
** These authors contributed equally to this work.

Boolean satisfiability [26], and Boolean synthesis [17,30]. Recent work focuses on partitioning techniques for LTL_f (LTL over finite traces) as a fundamental capability for synthesis [31,11].

Techniques for partitioning of properties described by temporal logics include tree-based classifiers to separate scenarios for Metric First-Order Temporal Logic (MFOTL) [29] and First Order Temporal Binding Logic (CMFTBL) [29]. Aimed at assessing coverage, [10] defines Inductive Validity Cores (IVCs) for safety properties to measure overlap of model elements and [9] defines a tree-based ontology of fault elements. Identifying logical dependencies in subformulas of conjunctions in LTL dramatically improved model-checking performance [7]. Affinity-based property partitioning relies on Hamming distance of support variable bitvectors to separate properties [1] and enables efficient strategy for parallel verification[5]. Strikingly, [6] highlights the impact of creating disjoint property subsets in industrial hardware verification tasks; here affinity-based partitioning minimizes redundant computation.

Driven by these studies, we seek efficient disjoint language partitioning for the logic Mission-time Linear Temporal Logic (MLTL). While efficient model checking algorithms, parallel verification, synthesis, and coverage are not yet defined for MLTL, partitioning algorithms could aid in their facilitation, as well as aid in optimizing existing techniques for MLTL satisfiability [22], MAX-SAT [13], benchmark generation [21], and requirements analysis [8]. The Formal Requirements Elicitation Tool (FRET) [25] accepts MLTL; a PVS library provides compositional proofs about MLTL language translation [3]. However, there are more extensive MLTL libraries in Isabelle/HOL [20,34]; we build on those to formalize a new partitioning algorithm. Our formalization derives from a new definition of *co-formulas*: for any MLTL formula φ, we can derive a set of co-formulas that disjointly capture the language accepted by φ.

Motivating Example In the context of benchmark generation, consider an example requirement "The spacecraft will deploy the solar array within 10 minutes, or the spacecraft will remain in low-power mode." This can be expressed as the MLTL formula $\mathcal{F}_{[0,10]}p \vee \mathcal{G}_{[0,10]}q$, where p becoming true indicates a successful solar array deployment, and q indicates the spacecraft is in low-power mode. Requirements of this structure pose a challenge when trying to create benchmarks for testing because the left clause is satisfied by many more traces than the right clause. Randomly sampling traces is unlikely to find a trace that satisfies $\mathcal{G}_{[0,10]}q$. However, defining co-formulas that capture the disjoint paths to formula satisfaction leads to better test coverage.

Contributions (1) We define, for the first time, co-formulas and language partitioning for MLTL. (2) We design algorithms for conducting language partitioning. (3) We formally prove the correctness of our definitions and algorithms in Isabelle/HOL; our formalization is approximately 5700 lines of code and is available online.[3] (4) From this library, we generate a verified implementation

[3] `https://drive.google.com/drive/folders/1tj5ulu2M5ksGHSD_V55K4f-UPwIGPSR9?usp=drive_link`

of the language partitioning algorithms. (5) An experimental evaluation of the implementation empirically demonstrates the utility of this tool.

The paper proceeds as follows: Section 2 reviews MLTL and the existing Isabelle/HOL library. Section 3 introduces language partitioning and presents the algorithms for performing language partitioning on an MLTL formula. Section 4 discusses the formalization in Isabelle/HOL. We conduct an evaluation of language partitioning as a sampling technique and compare it against existing techniques in Section 5 before concluding in Section 6.

2 Mission-time Linear Temporal Logic

Syntax and Semantics. MLTL [27] is a finite variant of LTL which places closed-interval integer bounds on the temporal operators. The intervals take the form $[a, b]$ where $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. MLTL formulas φ and ψ are syntactically defined over a finite set of atomic propositions $\mathcal{AP}$. With $p \in \mathcal{AP}$ as a propositional variable, the syntax of φ and ψ is given by the grammar:

$$\varphi, \psi := true \mid false \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathcal{F}_{[a,b]}\varphi \mid \mathcal{G}_{[a,b]}\varphi \mid \varphi \mathcal{U}_{[a,b]}\psi \mid \varphi \mathcal{R}_{[a,b]}\psi$$

Here, $\mathcal{F}, \mathcal{G}, \mathcal{U}, \mathcal{R}$ are the temporal operators Future, Globally, Until, and Release respectively. [4] MLTL formulas reason over finite *traces*, alternatively called signals or computations. Each trace π represents a discrete sequence of time steps and truth assignments to the propositional variables at each time step. Formally, a trace π of length $|\pi| = m$ is a sequence $\pi = \pi[0], \pi[1], \ldots, \pi[m-1]$, where each $\pi[i] \subseteq \mathcal{AP}$ is the atomic propositions true at time step i. We denote the suffix of a trace from time i as $\pi_i = \pi[i], \pi[i+1], \ldots, \pi[m-1]$.

Definition 1 (MLTL Semantics). *A trace π satisfies an MLTL formula φ, written $\pi \vDash \varphi$, defined recursively as follows:*

$\pi \models p$ *iff* $p \in \pi[0]$ $\qquad$ $\pi \models \neg\varphi$ *iff* $\pi \not\models \varphi$

$\pi \models \varphi \wedge \psi$ *iff* $\pi \models \varphi$ *and* $\pi \models \psi$ $\qquad$ $\pi \models \varphi \vee \psi$ *iff* $\pi \models \varphi$ *or* $\pi \models \psi$

$\pi \models \mathcal{F}_{[a,b]}\varphi$ *iff* $|\pi| > a$ *and* $\exists i \in [a, b].\ \pi_i \models \varphi$

$\pi \models \mathcal{G}_{[a,b]}\varphi$ *iff* $|\pi| \leq a$ *or* $\forall i \in [a, b].\ \pi_i \models \varphi$

$\pi \models \varphi\ \mathcal{U}_{[a,b]}\psi$ *iff* $|\pi| > a$ *and* $\exists i \in [a, b].\ (\pi_i \models \psi$ *and* $\forall j \in [a, i-1].\ \pi_j \models \varphi)$

$\pi \models \varphi\ \mathcal{R}_{[a,b]}\psi$ *iff* $|\pi| \leq a$ *or* $(\forall i \in [a, b].\ \pi_i \models \psi)$ *or* $\exists j \in [a, b-1].\ (\pi_j \models \varphi$ *and* $\forall k \in [a, j]\ \pi_k \models \psi)$

Definition 2 (Language). *The language of φ, denoted $\mathcal{L}(\varphi)$, is defined as the set of all traces that satisfy φ. That is, $\mathcal{L}(\varphi) = \{\pi \mid \pi \vDash \varphi\}$.*

All traces in MLTL are finite, but there is no fixed upper bound on the length of traces. $\mathcal{L}(\varphi)$ is then infinitely large, as there are infinitely many traces of arbitrary length; accordingly, we define the *restricted language* of an MLTL Formula.

[4] Notably, MLTL discards the Next ($\mathcal{X}$) operator, because it is equivalent to $\mathcal{F}_{[1,1]}$.

Definition 3 (Restricted Language). *The restricted language of an MLTL formula, $\mathcal{L}_r(\varphi)$, is defined as $\mathcal{L}_r(\varphi) = \{\pi \mid (|\pi| \geq r \wedge \pi \vDash \varphi)\}$. That is, $\mathcal{L}_r(\varphi)$ is the set of all traces of length at least r which satisfy φ.*

Often, it is useful to set the bound r to the *worst-case propagation delay* (wpd) of the formula φ [18,33]. Intuitively, the wpd of an MLTL formula φ is the number of time steps over which φ reasons; for instance, the formula $\mathcal{F}_{[0,10]}(\mathcal{G}_{[0,3]}\ p)$ reasons over $10 + 3 + 1 = 14$ time steps.

2.1 MLTL in Isabelle/HOL

Our formalization builds on a pre-existing Isabelle/HOL MLTL library [20]. This library formalizes MLTL and its various properties, including a formula progression algorithm for MLTL (but we only build upon the core MLTL library). We add to a growing body of work of formalized MLTL algorithms (see [32]), but this work is unique in that we develop theory directly in tandem with formalization.

MLTL formulas are defined as the custom datatype `'a mltl` [5] where `'a` denotes an arbitrary type in Isabelle/HOL, allowing flexibility for what types atomic propositions take on in various use cases. Traces are encoded with type `'a set list`, i.e., list of sets of some arbitrary type `'a`. The i-th state of a trace π is denoted π`!i`, where `!` indexes a list in Isabelle/HOL. The suffix of a trace π_i is denoted `drop i` π, which drops the first `i` elements of π. The function `semantics_mltl` encoding MLTL semantics has type `'a set list` $\Rightarrow$ `'a mltl` $\Rightarrow$ `bool`. We use this to formalize the restricted language of an MLTL formula:

```
definition language_mltl_r :: "['a mltl, nat]⇒'a set list set"
   where "language_mltl_r φ r = {π. semantics_mltl π φ ∧ length π ≥
r}"
```

`langauge_mltl_r` takes as input an MLTL formula φ and restriction length `r`, and defines the set of all traces π of length $\geq$ `r` that satisfy φ.

3 Language Partitioning for MLTL

Armed with the definition of the language of an MLTL formula, we return to the question posed in the abstract: "Can we find a set of related formulas to describe the language of an MLTL formula?"

A natural approach is to consider how the language of an MLTL formula can be decomposed into into related formulas, which we call co-formulas, that together capture the language of the original formula. That is, the union of the language of all the co-formulas is the language of the original formula, restricted up to the wpd of the original formula.

Definition 4 (Co-formula). *For MLTL formulas φ and ψ, ψ is a co-formula of φ if $\mathcal{L}_r(\psi) \subseteq \mathcal{L}_r(\varphi)$, where r is the wpd of φ.*

[5] We will use `this font` to display snippets of Isabelle/HOL code.

Definition 5 (Language decomposition and partition). *We define a language decomposition of an MLTL formula φ as a set of its co-formulas $\{\psi_1, ..., \psi_n\}$ such that $\bigcup_{i=1}^{n} \mathcal{L}_r(\psi_i) = \mathcal{L}_r(\varphi)$ for $r = wpd(\varphi)$.*

If additionally, $\forall i \neq j.\ \mathcal{L}_r(\psi_i) \cap \mathcal{L}_r(\psi_j) = \emptyset$, i.e., the languages of the ψ_i's are disjoint, then $\{\psi_1, ..., \psi_n\}$ is defined to be a language partition of φ. Consequently, any language partition of φ is a language decomposition of φ but not vice-versa.

Note that the notion of a co-formula is not the same as the standard notion of a *subformula*. A subformula is a clause that appears as a part of a larger formula, so the definition of is purely syntactic whereas our notion of a co-formula is *semantic* in nature. To exemplify this point, the language of a subformula is not necessarily a subset of the language of the original formula (in an extreme case, it may even be the complement of the language of the original formula: the language of $True$ is the complement of the language of $\neg True$).

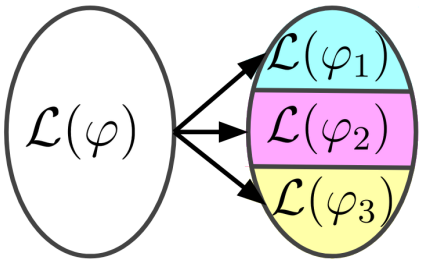

Fig. 1: Here we visualize the notion of language partition. The languages of the co-formulas $\varphi_1, \varphi_2, \varphi_3$ partition the co-domain of the language of the original formula φ.

Ideally, co-formulas in a language partition should capture meaningful fragments of the original formula's semantics. Consider the liveness property captured by the formula $\varphi = \mathcal{F}_{[3,10]}\psi$ that asserts ψ should happen between times 3 and 10. We can characterize satisfying traces of φ by the *first* time in which ψ holds, for instance in the beginning, middle, or end of the interval (i.e., say corresponding to $[3,4], [5,8], [9,10]$, respectively). We may similarly characterize the formula $\varphi\, \mathcal{U}_{[a,b]}\, \psi$ in terms of the *first* place in $[a,b]$ where ψ holds (similarly $\varphi\, \mathcal{R}_{[a,b]}\, \psi$ in terms of the *first* place in $[a,b]$ where φ holds). To formally capture this intuition, we introduce the combinatoric notion of an *integer composition*. A composition of an integer n is an ordered list of positive integers $n_1, ..., n_k$ such that $n = \sum_{i=1}^{k} n_i$. We formalize this in Isabelle/HOL as follows: [6]

```
definition is_composition :: "[nat, nat list] ⇒ bool"
  where "is_composition n L = ((∀i∈set L. i>0)∧(sum_list L=n))"
```

We then define a composition of an interval $[a,b]$ as a composition of $b - a + 1$, the number of time steps in the interval. Each summand in the composition of $[a,b]$ corresponds to a sub-interval of $[a,b]$ of that length. For example, $L = 2,4,2$ is a composition of the interval $[3,10]$ producing the sub-intervals $[3,4], [5,8], [9,10]$. We visualize this in Fig. 3.

We use this notion of integer composition to construct language decompositions and language partitions. For example, to construct a language de-

[6] Integer compositions has been formalized in Isabelle/HOL [14], but the previous formalization primarily focuses on combinatorial properties of integer compositions.

composition of our familiar example $\mathcal{F}_{[3,10]}\varphi$ with interval composition $2, 4, 2$, consider the formulas $\psi_1 = \mathcal{F}_{[3,4]}\varphi$, $\psi_2 = (\mathcal{G}_{[3,4]}\neg\varphi) \wedge (\mathcal{F}_{[5,8]}\varphi)$, and $\psi_3 = (\mathcal{G}_{[3,8]}\neg\varphi) \wedge (\mathcal{F}_{[9,10]}\varphi)$. ψ_1 requires φ to hold in the interval $[3, 4]$, ψ_2 requires that φ does *not* hold before time 5 and hence φ holds for the first time in the interval $[5, 8]$. ψ_3 requires that φ does *not* hold before time 8 and hence φ first holds in the interval $[9, 10]$. It is clear that any trace π satisfies φ iff π satisfies one of the ψ_i's. Furthermore, the ψ_i's are disjoint, and thus $\{\psi_1, \psi_2, \psi_3\}$ is a language partition of φ.

We define an algorithm LP to compute a language decomposition for an MLTL formula with arbitrary interval compositions. This algorithm also constructs a language partition for the MLTL formula, but only for a particular shape of interval compositions.

In Isabelle/HOL, we introduce the notion of interval compositions by extending the previous `mltl` datatype. We define the datatype `mltl_ext`, which augments the Future, Until, and Release temporal operators of the `mltl` datatype with an additional argument of type `nat list`, representing the desired integer composition of the operator's interval. For instance, the formula $\mathcal{F}_{[3,10]}\varphi$ with integer composition $[2, 4, 2]$ is represented as `Future_mltl_ext φ 3 10 [2, 4, 2]`. We do not augment the Global operator because the LP algorithm uses a fixed integer composition for the Global case (see Section 3.2 for technical details).

Now, we define the language partitioning function LP that computes the set of co-formulas which forms the language partition of an input formula φ of type `mltl_ext`. We formalize (see Sec. 4) that LP always produces a language decomposition of φ. However, in order to produce a language partition, LP requires compositions to be lists of all 1's.

3.1 Language Partitioning for MLTL

We provide a high level algorithmic overview of LP, the details of which are available in the formalization. LP recurses on the structure of the input MLTL formula, returning a set of co-formulas at each recursive call; thus we first define operations on sets of co-formulas. For MLTL formula φ, we use D_φ to denote a language decomposition of φ; later we set D_φ equal to the output of LP on φ and prove in our formalization that LP always computes language decompositions and, under certain interval compositions shapes, computes language partitions.

3.1.1 MLTL Set Operations We define the operations $\dot{\wedge}, \dot{\mathcal{F}}, \dot{\mathcal{G}}, \dot{\mathcal{U}}, \dot{\mathcal{R}}$, each operating on sets of co-formulas. We also provide intuition for how these operations use sets of co-formulas that are language decompositions of a subformula to build a language decomposition of a larger formula. However, we refer the interested reader to our formalization for how we made these intuitions rigorous.

$\dot{\wedge}$(Set And) Consider MLTL formulas φ and ψ, and their respective language decompositions D_φ and D_ψ. We seek to use D_φ and D_ψ to compute a language decomposition of $\varphi \wedge \psi$. We define that

$$D_\varphi \mathbin{\dot\wedge} D_\psi = \{x \wedge y \mid x \in D_\varphi \text{ and } y \in D_\psi\}$$

and illustrate an example of the $\dot\wedge$ operator in Fig. 2. Intuitively, each $x \in D_\varphi$ and $y \in D_\psi$ represents one possible way to satisfy φ and ψ, respectively, and hence $D_\varphi \mathbin{\dot\wedge} D_\psi$ computes all the ways to satisfy $\varphi \wedge \psi$. A trace π satisfies φ iff π satisfies some $x \in D_\varphi$, and π satisfies ψ if and only if π satisfies some $y \in D_\psi$. Thus, $D_\varphi \mathbin{\dot\wedge} D_\psi$ is a language decomposition of $\varphi \wedge \psi$.

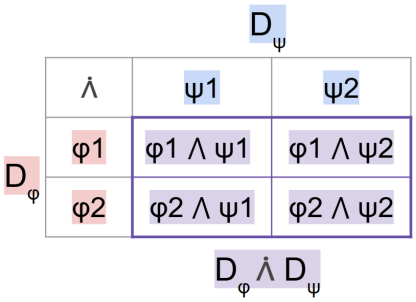

Fig. 2: For $D_\varphi = \{\varphi_1, \varphi_2\}$ and $D_\psi = \{\psi_1, \psi_2\}$, $D_\varphi \dot\wedge D_\psi = \{\varphi_1 \wedge \psi 1, \varphi_1 \wedge \psi 2, \varphi_2 \wedge \psi_1, \varphi_2 \wedge \psi_2\}$.

$\dot{\mathcal{F}}$(Set Future) For a language decomposition D_φ of MLTL formula φ, $\dot{\mathcal{F}}_{[a,b]} D_\varphi$ applies the Future operator to each element of D_φ. That is, we define $\dot{\mathcal{F}}_{[a,b]} D_\varphi = \{\mathcal{F}_{[a,b]} x \mid x \in D_\varphi\}$. Again $\dot{\mathcal{F}}_{[a,b]} D_\varphi$ is a language decomposition of $\mathcal{F}_{[a,b]}\varphi$; this is because a trace π satisfies $\mathcal{F}_{[a,b]}\varphi$ iff φ holds at some point in $[a, b]$, iff some $x \in D_\varphi$ holds at that point in $[a, b]$.

$\dot{\mathcal{G}}$(Set Globally) For language decomposition D_φ of MLTL formula φ, we actually *cannot* define $\dot{\mathcal{G}}_{[a,b]} D_\varphi$ in a similar manner as $\dot{\mathcal{F}}$. Suppose we were to try defining $\dot{\mathcal{G}}_{[a,b]} D_\varphi$ to apply $\mathcal{G}_{[a,b]}$ to each element of D_φ. Then $\dot{\mathcal{G}}_{[a,b]} D_\varphi$ does not capture the full language of $\mathcal{G}_{[a,b]}\varphi$. If $\varphi = p \vee q$ where $p, q \in \mathcal{AP}$, the set $D_\varphi = \{p, q\}$ is a language decomposition of φ. However if $a < b$, $\mathcal{G}_{[a,b]}(p \vee q)$ is not equivalent to $(\mathcal{G}_{[a,b]} p) \vee (\mathcal{G}_{[a,b]} q)$. A simple counterexample for $a = 0$ and $b = 1$ is the trace $\pi = \{p\}, \{q\}$, which satisfies $\mathcal{G}_{[0,1]}(p \vee q)$ but not $(\mathcal{G}_{[0,1]} p) \vee (\mathcal{G}_{[0,1]} q)$. The fact that equivalence only holds for $a = b$ motivates the following definition: $\dot{\mathcal{G}}_{[a,b]} D_\varphi = \{\bigwedge_{i=a}^{b} \mathcal{G}_{[i,i]} x_i \mid \forall i \in [a, b].x_i \in D_\varphi\}$. This enumerates all possible ways that D_φ can hold at every point in the interval $[a, b]$, essentially repeating the $\dot\wedge$ operator at each point in the interval. Hence we have that $\dot{\mathcal{G}}_{[a,b]} D_\varphi$ is a language decomposition for $\mathcal{G}_{[a,b]}\varphi$. This causes an exponential blow up in the number of co-formulas produced; specifically, $|\dot{\mathcal{G}}_{[a,b]} D_\varphi| = (|D_\varphi|^{b-a+1})$.

$\dot{\mathcal{U}}$(Set Until) Now, we define the $\dot{\mathcal{U}}$ operator to build a language decomposition for $\varphi \, \mathcal{U}_{[a,b]} \, \psi$. Recall that the semantics of Until requires φ to hold for all time steps until ψ holds; this implicit Globally φ in the semantics poses the same challenge as the previous case. As such, we choose not to consider a language decomposition of φ and define the operator asymmetrically. For D_ψ being a language decomposition of ψ, we define that $\varphi \, \dot{\mathcal{U}}_{[a,b]} \, D_\psi = \{\varphi \, \mathcal{U}_{[a,b]} \, y \mid y \in D_\psi\}$. This is a both syntactically and semantically similar definition to $\dot{\mathcal{F}}$ since we split the implicit Future operation on ψ. Thus using the same line of reasoning as $\dot{\mathcal{F}}$, we have that $\varphi \, \dot{\mathcal{U}}_{[a,b]} \, D_\psi$ is a language decomposition for $\varphi \, \mathcal{U}_{[a,b]} \, \psi$.

$\dot{\mathcal{R}}$(Set Release) Lastly, we define $\dot{\mathcal{R}}$ to be a language decomposition for $\varphi \, \mathcal{R}_{[a,b]} \, \psi$. Recall that in the semantics of Release, ψ is required to hold at all times steps up to and including when φ holds in the interval $[a, b]$, or if φ does not hold at all, that ψ always holds in $[a, b]$. We again see the same issue with an implicit Globally ψ, and thus only consider D_φ being a language decomposition of φ. We define that $D_\varphi \; \dot{\mathcal{R}}_{[a,b]} \; \psi = \{x \; \dot{\mathcal{R}}_{[a,b]} \; \psi \mid x \in D_\varphi\}$, noting its similarity to $\dot{\mathcal{U}}$. $\dot{\mathcal{R}}$ splits on the implicit Future operator on φ, and hence the same line of reasoning as $\dot{\mathcal{F}}$ and $\dot{\mathcal{U}}$ gives that $D_\varphi \; \dot{\mathcal{R}}_{[a,b]} \; \psi$ is a language decomposition for $\varphi \; \mathcal{R}_{[a,b]} \; \psi$.

3.1.2 $\dot{\wedge}, \dot{\mathcal{F}}, \dot{\mathcal{G}}, \dot{\mathcal{R}}, \dot{\mathcal{U}}$ in Isabelle/HOL In our Isabelle/HOL formalization, we encode sets of co-formulas as lists of formulas because it was more convenient to work with lists in a formal setting. The details of how we formalize each of the operators $\dot{\wedge}, \dot{\mathcal{F}}, \dot{\mathcal{G}}, \dot{\mathcal{R}}, \dot{\mathcal{U}}$ can be found in the functions `And_mltl_list`, `Future_mltl_list`, `Global_mltl_decomp`, `Until_mltl_list`, and `Release_mltl_list`, respectively. Each these functions is a simple `map` across input lists, except for `Global_mltl_decomp`, which is particularly interesting because it recurses on the length of the interval and repeatedly calls `And_mltl_list`.

3.2 *LP* Algorithm for Language Decomposition and Partitioning

The functions defined above achieve language decomposition; however, in order to reach language partitioning, the LP algorithm must combine these functions to guarantee disjointedness of co-formulas. LP takes as input an MLTL formula φ augmented with interval compositions (i.e., an input of type `mltl_ext`) and a desired decomposition depth d (of type `nat`). The decomposition depth allows flexibility for the user to control the granularity of the decomposition. As an example, we will see that $LP(\varphi \wedge \psi, d+1) = LP(\varphi, d) \; \dot{\wedge} \; LP(\psi, d)$ and for any φ, $LP(\varphi, 0) = \{\varphi\}$ as the base case. Then $LP(\varphi \wedge \psi, 1) = \{\varphi \wedge \psi\}$ since it hits the base case for $d = 0$, while $LP(\varphi \wedge \psi, 2)$ goes one layer deeper and computes $LP(\varphi \wedge \psi, 2) = LP(\varphi, 1) \; \dot{\wedge} \; LP(\psi, 1)$.

Essentially, LP recurses on the decomposition depth d and, within each depth, splits on the top-level structure of the input formula φ. We assume input formulas are in negation normal form (NNF) [4], and also transform outputs to be in NNF before each recursive call. In our formalization, we build upon the existing NNF transformation from prior work [20].

To give some idea of how this algorithm operates, we provide pseudocode and an intuitive explanation of LP for each formula structure; we refer the interested reader to our formalization for the full technical details, and also derive our implementation from our formalization via code generation. [7]

[7] Although Isabelle/HOL's code generation is not yet fully verified, it provides a significantly higher degree of confidence, especially for a development as technically intricate as this [15].

Base Cases As mentioned previously, we define for all shapes of the input formula φ that $LP(\varphi, 0) = \{\varphi\}$. Additionally for atomic formulas where φ is either an atomic proposition $p \in \mathcal{AP}$, $true$, $false$, or $\neg p$ (recall that φ is in NNF), we also define that $LP(\varphi, d) = \{\varphi\}$ for all natural numbers d. [8]

Non-Temporal Cases Let φ and ψ be MLTL formulas and $d > 0$. For the And case, we define that $LP(\varphi \wedge \psi, d) = LP(\varphi, d-1) \dot{\wedge} LP(\psi, d-1)$. In this case $\dot{\wedge}$ is sufficient to guarantee disjointedness. This is because the base cases produce singleton sets of co-formulas, which are trivial language partitions. Hence, we may assume, as the inductive hypothesis when inducting on d, that $LP(\varphi, d-1)$ and $LP(\psi, d-1)$ are language partitions. Then for two distinct formulas $x_1 \wedge y_1 \in LP(\varphi \wedge \psi, d)$ and $x_2 \wedge y_2 \in LP(\varphi \wedge \psi, d)$ where $x_1, x_2 \in LP(\varphi, d-1)$ and $y_1, y_2 \in LP(\psi, d-1)$, either $x_1 \neq x_2$ or $y_1 \neq y_2$. In both cases, the inductive hypothesis gives that either $\mathcal{L}_r(x_1) \cap \mathcal{L}_r(x_2) = \emptyset$ or $\mathcal{L}_r(y_1) \cap \mathcal{L}_r(y_2) = \emptyset$, where $r = wpd(\varphi \wedge \psi)$. This is sufficient to show that $\mathcal{L}_r(x_1 \wedge y_1) \cap \mathcal{L}_r(x_2 \wedge y_2) = \emptyset$.

For the Or case, we build on the intuition that $\varphi \vee \psi$ is satisfied in three disjoint ways: $\varphi \wedge \psi, \neg\varphi \wedge \psi$, and $\varphi \wedge \neg\psi$. Thus, applying $\dot{\wedge}$ in each case is sufficient to build a language partition of $\varphi \vee \psi$ from $LP(\varphi, d)$ and $LP(\psi, d)$. We refer to line 5 in Alg. 1 for the full definition.

Temporal Cases In the temporal cases of Future, Until, and Release, we consider a list $L = n_1, n_2, ..., n_k$ as composition of the associated interval $[a, b]$; thus recall that $\sum_{\ell=1}^{k} n_\ell = b - a + 1$. We introduce the notation $s_i = a + \sum_{\ell=1}^{i} n_\ell$ for all $i = 0, \ldots, k$ and observe that $s_0 = a$ and $s_k = b+1$. Intuitively, the sequence of s_i's marks out the sub-intervals that L defines for $[a, b]$; more precisely, we have that $s_j - s_{(j-1)} = n_j$ for all $j = 1, \ldots, k$. Then, we define the sub-intervals $\mathcal{I}_j = [s_{(j-1)}, (s_j) - 1]$. We illustrate this in Fig. 3 for the previous example. Now, we define LP for the temporal cases.

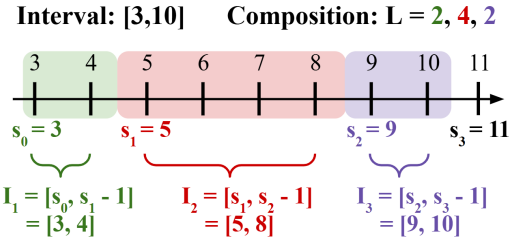

Fig. 3: We visualize the sub-intervals corresponding to the composition $L = 2, 4, 2$ for the interval $[3, 10]$. We have that $s_0 = 0$, $s_1 = 3 + 2 = 5$, $s_2 = 3 + 2 + 4 = 9$, and $s_3 = 3+2+4+2 = 11$, and intervals $\mathcal{I}_1, \mathcal{I}_2$, and $\mathcal{I}_3$, have lengths 2, 4, and 2, respectively.

For the Global case, let φ be an MLTL formula and $k \in \mathbb{N}$. We define $LP(\mathcal{G}_{[a,b]}\varphi, d) = \dot{\mathcal{G}}_{[a,b]} LP(\varphi, d-1)$. Since $\dot{\mathcal{G}}$ decomposes every time step, we may view the interval composition of $[a, b]$ as a list of all 1's. An analogous argument to the And case shows $LP(\mathcal{G}_{[a,b]}\varphi, d)$ is a language partition of $\mathcal{G}_{[a,b]}\varphi$.

Now for the Future case, let φ be an MLTL formula, $k \in \mathbb{N}$, and L be a list of length k that is a composition for the interval $[a, b]$ with corresponding sub-intervals $\mathcal{I}_j, = [s_{(j-1)}, (s_j) - 1]$ for $j = 1, \ldots, k$. Then, we want to capture the previous intuition for φ to define co-formulas that specify φ holds for the

[8] We define the naturals to be $\mathbb{N} = \{0, 1, 2, \ldots\}$

first time in interval $\mathcal{I}_j$. We define that

$$LP(\mathcal{F}_{[a,b]}\varphi, d) = \bigcup_{j=1}^{k}(\{\mathcal{G}_{[a,(s_{(j-1)})-1]}\ \neg\varphi\} \dot{\wedge}\ (\dot{\mathcal{F}}_{\mathcal{I}_j}\ LP(\varphi, d-1)))$$

Intuitively the Globally $\neg\varphi$ clause is what asserts some co-formula in $LP(\varphi, d-1)$ to hold for the first time on the interval $\mathcal{I}_j$. Note that for $j = 1$, the interval $[a, s_0 - 1] = [a, a-1]$ is malformed (recall that $s_0 = a$); we treat this more precisely in the formalization by splitting out the $j = 1$ case, but here we simply treat this formula as having only the right hand clause.

However, this is in fact *not sufficient* in general to guarantee that $LP(\varphi, d)$ is a language partition of $\mathcal{F}_{[a,b]}\varphi$. To see why disjointedness fails, consider the example $\mathcal{F}_{[0,5]}(p \vee q)$ for atomic propositions $p, q \in \mathcal{AP}$ with the interval composition $L = 3, 3$. Using that $LP(p \vee q, 1) = \{p \wedge q, \neg p \wedge q, p \wedge \neg q\}$, we have:

$$LP(\mathcal{F}_{[0,5]}(p \vee q), 2) = \left\{ \begin{matrix} \mathcal{F}_{[0,2]}(p \wedge q), \\ \mathcal{F}_{[0,2]}(\neg p \wedge q), \\ \mathcal{F}_{[0,2]}(p \wedge \neg q) \end{matrix} \right\} \bigcup \left\{ \begin{matrix} \mathcal{G}_{[0,2]}(\neg(p \vee q)) \wedge \mathcal{F}_{[3,5]}(p \wedge q), \\ \mathcal{G}_{[0,2]}(\neg(p \vee q)) \wedge \mathcal{F}_{[3,5]}(\neg p \wedge q), \\ \mathcal{G}_{[0,2]}(\neg(p \vee q)) \wedge \mathcal{F}_{[3,5]}(p \wedge \neg q) \end{matrix} \right\}$$

The length 3 trace $\{p, q\}, \{q\}, \{p\}$ satisfies all three formulas from the first set, while the length 6 trace $\{\}, \{\}, \{\}, \{p, q\}, \{q\}, \{p\}$ satisfies all three formulas from the second set. Although each formula in $LP(p \vee q, 1)$ is disjoint *at the same time step*, the sub-intervals on the Future operators are "wide enough" such that the traces can space out the times at which co-formulas of $LP(p \vee q, 1)$ hold. Thus in order to obtain disjointedness, it must be the case that either the interval composition L is a list of all 1's, or that the set $LP(p \vee q, k)$ has only size 1.

Lastly for the Until and Release case, let φ and ψ be MLTL formulas, $k \in \mathbb{N}$, and L be a list of length k that is a composition for the interval $[a, b]$ with corresponding sub-intervals $\mathcal{I}_j, = [s_{(j-1)}, (s_j) - 1]$ for $j = 1, \ldots, k$. For Until, we define the decomposition of $\varphi\ \mathcal{U}_{[a,b]}\psi$ based on the first time at which ψ holds. Refer to line 13 in Alg. 1 for the full definition of the Until case, noting the similarities with the Future case. Lastly for Release, we define the decomposition of $\varphi\ \mathcal{R}_{[a,b]}\psi$ based on the first time at which φ holds; refer to line 15 of Alg. 1. The difference here is that for Release, φ is not required to to hold at some point in the interval, and thus ψ being Globally true on the interval $[a, b]$ is permissible. Thus, we split out this special case in the semantics and use the Mighty Release to specify that φ must hold at some sub-interval $\mathcal{I}_j$. Syntactically, we define that $\varphi\ \mathcal{M}_{[a,b]}\psi = (\varphi\mathcal{R}_{[a,b]}\psi) \wedge (\mathcal{F}_{[a,b]}\varphi)$. For both Until and Release, we run into the same issue as Future in which disjointedness requires that the interval composition be all 1's.

4 Formalizing Language Partition in Isabelle/HOL

To ensure the correctness of our definitions and to leverage correct-by-construction generated code, we formalize the LP algorithm in Isabelle/HOL. We overview the top-level theorems, and then discuss insights from the unexpected complexity of our formalization.

Algorithm 1 The language partitioning algorithm

Input: MLTL Formula φ, Depth k.
Output: Finite set of MLTL co-formulas of φ.

function LP:
  case $k = 0$ or φ is an atomic formula **return** $\{\varphi\}$
  case $\varphi = \psi_1 \wedge \psi_2$ **return** $LP(\psi_1, k-1) \dot{\wedge}\ LP(\psi_2, k-1)$
  case $\varphi = \psi_1 \vee \psi_2$
    return $\big(LP(\psi_1, k-1) \dot{\wedge} \neg\psi_2\big), \big(\neg\psi_1 \dot{\wedge} LP(\psi_2, k-1)\big), \big(LP(\psi_1, k-1) \dot{\wedge} LP(\psi_2, k-1)\big)$
  case $\varphi = \mathcal{G}_{[a,b]}\psi$
    if $|LP(\psi, k-1)| \leq 1$ **then return** $LP(\psi, k-1)$
    else return $\dot{\mathcal{G}}_{[a,b]}(LP(\psi, k-1))$
    end if
  case $\varphi = \mathcal{F}_{[a,b]}\psi$
    return $\bigcup_{j=1}^{k}(\{\mathcal{G}_{[a,(s_{(j-1)})-1]}\ \neg\varphi\} \dot{\wedge} (\dot{\mathcal{F}}_{\mathcal{I}_j}\ LP(\varphi, k-1)))$
  case $\varphi = \varphi \mathcal{U}_{[a,b]} \psi$
    return $\bigcup_{j=1}^{k}(\{\mathcal{G}_{[a,s_{(j-1)}-1]}(\varphi \wedge \neg\psi)\} \dot{\wedge} (\varphi \dot{\mathcal{U}}_{\mathcal{I}_j} LP(\psi, k-1)))$
  case $\varphi = \varphi \mathcal{R}_{[a,b]} \psi$
    return $\{\mathcal{G}_{[a,b]}\psi\} \cup \bigcup_{j=1}^{k}(\{\mathcal{G}_{[a,s_{(j-1)}-1]}(\neg\varphi \wedge \psi)\} \dot{\wedge} (\varphi \dot{\mathcal{M}}_{\mathcal{I}_j} LP(\psi, k-1)))$
end function

Where $\varphi \mathcal{M}_{[a,b]} \psi$ *(Mighty Release)* is just syntatic sugar for $(\varphi \mathcal{R}_{[a,b]} \psi) \wedge (\mathcal{F}_{[a,b]} \varphi)$.

4.1 Top-Level Theorems

In Isabelle/HOL, we formalize LP as `LP_mltl`. This function takes as input an MLTL formula augmented with integer compositions (of type `mltl_ext`) and a natural number for the decomposition depth (of type `nat`). It outputs a list of co-formulas (of type `mltl`, because the output formulas no longer need to be associated with integer compositions). Note that we use lists because they are easier to work with than sets in Isabelle/HOL.

We first formalize that on well-defined inputs `LP_mltl` always computes a language decomposition of the input formula in the following theorem.

```
theorem LP_mltl_language_union:
  fixes φ::"'a mltl_ext" and k::"nat"
  assumes welldef: "intervals_welldef (mltl_ext_to_mltl φ)"
  assumes composition: "is_composition_MLTL φ"
  assumes D: "D = set (LP_mltl φ k)"
  assumes r: "r ≥ wpd_mltl (mltl_ext_to_mltl φ)"
  shows "language_mltl_r (mltl_ext_to_mltl φ) r
        = (⋃ψ∈D. language_mltl_r ψ r)"
```

The `welldef` assumption is an artifact of how MLTL is formalized [20], ensuring that all temporal intervals in the input formula φ are well-defined. The `mltl_ext_to_mltl` function converts from the extended datatype into the original

mltl datatype, which facilitates our proofs. [9] The *composition* assumption is also a well-definedness assumption, but on the extended datatype; it requires each *nat list* to be an integer composition of the associated interval. The assumptions *D* and *r* introduce abbreviations for the output set of co-formulas and for the wpd of the input formula, respectively. Given these assumptions, this theorem establishes that the restricted language of the input formula φ is equal to the union of the restricted languages in *D*. Note that the function *language_mltl_r*, which computes the restricted language, operates on formulas of type *mltl*, and thus we need to use our conversion function *mltl_ext_to_mltl* when appropriate.

In the next theorem, we formalize conditions necessary for *LP_mltl* to return a language partition.

```
theorem LP_mltl_language_disjoint:
   fixes φ::"'a mltl_ext" and ψ1 ψ2::"'a mltl" and k::"nat"
   assumes welldef: "intervals_welldef (mltl_ext_to_mltl φ)"
   assumes is_nnf: "∃φ_init. φ = convert_nnf_ext φ_init"
   assumes composition: "is_composition_MLTL_allones φ"
   assumes D: "D = set (LP_mltl φ k)"
   assumes diff_formulas: "(ψ1 ∈ D) ∧ (ψ2 ∈ D) ∧ ψ1 ≠ ψ2"
   assumes r: "r ≥ wpd_mltl (mltl_ext_to_mltl φ)"
   shows "(language_mltl_r ψ1 r) ∩ (language_mltl_r ψ2 r) = {}"
```

The assumption *welldef* is the same well-definedness condition as before, and again *D* and *r* introduces abbreviations for the output set of co-formulas and for the wpd of the input formula, respectively. However, now we assume in *composition* that *is_composition_allones* φ, which ensures that φ has well-defined interval compositions *and* that these compositions are lists of all 1's. *is_nnf* assumes that the input formula φ is in NNF, and *diff_formulas* assumes that $\psi 1$ and $\psi 2$ are different co-formulas from *D*. Under these assumptions, this theorem guarantees that the restricted languages of $\psi 1$ and $\psi 2$ are disjoint, meaning that the output co-formulas form a language partition.

4.2 Formalization Insights

Now we turn to a discussion of the challenges and insights from our formalization. First and most notably, this formalization is lengthy and involved. There are many low-level technical details that were difficult to get right.

Because the pencil-and-paper proofs were technically detailed, our formalization and our theoretical development worked symbiotically and hand-in-hand. Throughout the course of our formalization, we found a few tweaks to our initial draft of our theoretical results. For example, we identified the assumption that the languages of MLTL formulas need to be restricted to traces of length at least the wpd of the input formula during the formalization. Retrospectively, this is not a surprising assumption; prior work on formalizing the formula progression

[9] This function simply removes the integer compositions.

algorithm for MLTL [20] found a similar assumption was missing from the corresponding theoretical results [21]. Additionally, we had originally hoped that `LP_mltl` would produce a language partition with arbitrary interval compositions, but we found the subtle counterexample discussed earlier in Section 3 during our formalization. As our formalization ended up being particularly instrumental, we would recommend that anyone embarking on a similarly technically involved project using MLTL consider formalization.

The formalization challenges were primarily low-level. For example, although introducing the extended datatype was theoretically beneficial — the alternative we considered was to use the original datatype and a separate list of integer compositions. but detaching operators from their corresponding compositions would have been unfortunate. Our choice also introduced the need to develop formal infrastructure for the new datatype. In the inductive proofs for our top-level theorems, each assumption incurs an additional proof obligation in order to use the inductive hypothesis. As an example, we have to prove that the wpd of co-formulas returned from `LP_mltl` is at most the wpd of φ. Proofs involving `LP_mltl` are almost always lengthy, because of the numerous cases in each operator. An exemplary case is in the disjointedness proof for the Release case where we had to prove that ψ_1 and ψ_2 from the decomposition are disjoint; the formalization for the Release case in `LP_mltl` splits into 3 cases, thus requiring a total of 9 cases for every combination of ψ_1 and ψ_2.

We now turn to our experiments, in which we benefit from the efforts of our formalization by directly using the verified code export.

5 Experiments

To illustrate our experiments, we return to the example from the introduction, simplified here for clarity: $\left(\mathcal{F}_{[0,3]}p\right) \vee \left(\mathcal{G}_{[0,3]}q\right)$. As noted previously, there are numerically more traces that satisfy the first clause than the second. The language partitioning of this formula produces the nine co-formulas presented in Table 1.

We compare the language partitioning approach against three existing methods of trace generation: randomly sampling satisfying traces; generating satisfying traces with the FPROGG MLTL benchmark generator [28]; and enhancing the FPROGG tool with conflict-driving trace enumeration where formulas are repeatedly updated to exclude the previously generated traces. Each of these methods has its own limitations. Random sampling is susceptible to missing very small classes of satisfying traces. For example, out of 2048 possible traces of length 11 over the formula $\mathcal{G}_{[0,10]}p$, only one satisfies the formula. On the other hand, FPROGG leverages progressive SAT solving to always produce a satisfying trace, but only produces one trace per formula; we further extend the FPROGG tool with conflict-driving trace enumeration in order to produce more traces. As a fourth sampling method, we couple the conflict-driving trace enumeration of FPROGG with the LP algorithm to sample multiple traces from each co-formula.

Table 1: The co-formulas of $\mathcal{F}_{[0,3]}p \vee \mathcal{G}_{[0,3]}q$.

0	$((\mathcal{F}_{[0,0]}p) \wedge (\mathcal{G}_{[0,3]}q))$	1	$((\mathcal{G}_{[0,0]}(\neg p)) \wedge (\mathcal{F}_{[1,1]}q) \wedge (\mathcal{G}_{[0,3]}q))$
2	$((\mathcal{G}_{[0,1]}(\neg p)) \wedge (\mathcal{F}_{[2,2]}p) \wedge (\mathcal{G}_{[0,3]}q))$	3	$((\mathcal{G}_{[0,2]}(\neg p)) \wedge (\mathcal{F}_{[3,3]}p) \wedge (\mathcal{G}_{[0,3]}q))$
4	$((\mathcal{G}_{[0,3]}(\neg p)) \wedge (\mathcal{G}_{[0,3]}q))$	5	$((\mathcal{F}_{[0,0]}p) \wedge (\mathcal{F}_{[0,3]}(\neg q)))$
6	$((\mathcal{G}_{[0,0]}(\neg p)) \wedge (\mathcal{F}_{[1,1]}p) \wedge (\mathcal{F}_{[0,3]}(\neg q)))$	7	$((\mathcal{G}_{[0,1]}(\neg p)) \wedge (\mathcal{F}_{[2,2]}p) \wedge (\mathcal{F}_{[0,3]}(\neg q)))$
8	$((\mathcal{G}_{[0,2]}(\neg p)) \wedge (\mathcal{F}_{[3,3]}p) \wedge (\mathcal{F}_{[0,3]}(\neg q)))$		

We use coverage of co-formulas as a metric for how well a set of traces represents the original formula. This gives a simple and effective measure between sets of traces. By tallying the number of traces from each set that satisfy each co-formula, we compute the *Wasserstein Distance* between the distributions of traces over the co-formulas.

The Wasserstein distance[10] quantifies how close two distributions are based on the amount of "mass" (data points) that must be moved in order to transform one distribution into another [19]. We selected the Wasserstein Distance specifically as our target metric because it quantifies over distributions. Alternatively, the distances could have been computed using any number of distance measures between traces, such as the Hamming distance [12] or the weighted edit distance from [16] which was introduced for STL [23], a related temporal logic. However, because these metrics do not quantify over sets of traces, we did not investigate them for our initial experiments.

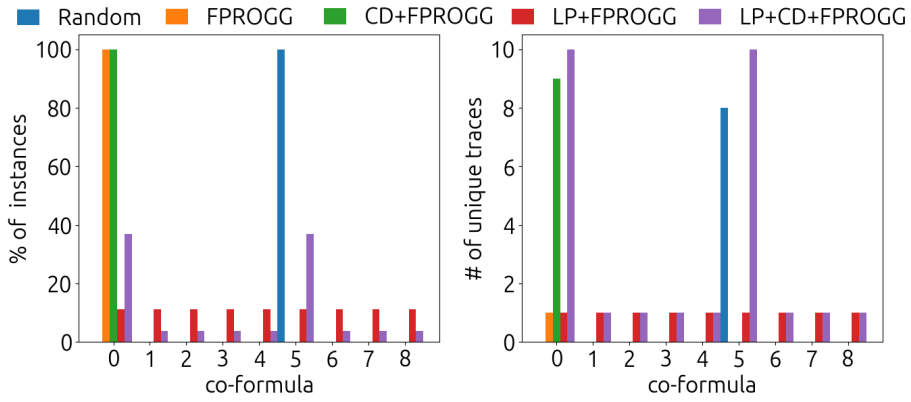

Fig. 4: Trace distributions over the coformulas of $\mathcal{F}_{[0,3]}p \vee \mathcal{G}_{[0,3]}q$.

We plot the coverage results of the different sampling techniques in Fig.4 with the percentage of the total number of traces generated by each method on the left and the coverage results by total count on the right.

Language partitioning produces a uniform distribution over the co-formulas, as expected. Language partitioning with conflict-driving samples at least 1 from each co-formula and produces additional traces from co-formulas 0 and 5. FPROGG produces only one trace satisfying the co-formula $(\mathcal{F}_{[0,0]}p) \wedge (\mathcal{G}_{[0,3]q})$. Adding

[10] Also known as the earth-mover's distance.

conflict-driving to FPROGG increases the number of traces it is able to produce, but they still all come from co-formula 0. Similarly, random sampling produces 8 traces, all from co-formula 5. It is important to note that while random sampling creates an equal number of traces as our baseline LP approach, it captures less co-formulas. The Wasserstein Distances between these distributions are presented in Table 2. The co-formulas do not have an innate ordering, so the spread of a distribution is only computed with the *number* of different co-formulas a set of traces satisfies. Because we compute the distances between the normalized distributions, all methods that only sample from one co-formula are considered identically representative. We see in Fig. 4 that random sampling, FPROGG, and FPROGG with conflict-driving all only produce traces satisfying one co-formula and thus the Wasserstein distance between them is zero. This highlights the difference in diversity of samples produced by each method, as opposed to the absolute number of traces produced for each co-formula. We see in the two LP-based approaches that the Wasserstein distance is nonzero, demonstrating a wider coverage of the co-formulas.

Table 2: Wasserstein distance results for $\mathcal{F}_{[0,3]}p \vee \mathcal{G}_{[0,3]}q$ over the normalized distributions.

	FPROGG	Conflict Driving with FPROGG	Language Partitioning	Language Partitioning with Conflict Driving
Random Sampling	0	0	19.8	14.0
FPROGG		0	19.8	14.0
Conflict Driving			19.8	14.0
Language Partitioning				11.5

Implementation To ensure correctness of our implementation, we use the exported LP algorithm code from the formalization in Haskell. We conducted the remainder of the experimentsin Python. We used some generative AI in the form of Github Copilot to increase development speed during the creation of the Python code.

6 Conclusion

We have defined the LP algorithm for effectively partitioning MLTL formulas, introducing and formalizing the notions of *co-formulas*, *language decomposition*, and *language partitioning* for MLTL. We fully verify this algorithm in Isabelle/HOL from which we export a high-assurance implementation. We provide an initial evaluation of it in the context of benchmark generation.

Furthermore, our formalization is extensible and has clear avenues for future work. For instance, we believe that it is possible to generalize the disjointedness proofs to include the case when the decomposition depth is 1 for arbitrary integer compositions. Additionally, it would be interesting to investigate the theoretical

maximum depth that a formula can be decomposed, as well the necessary depth of decomposition for practical applications. A loftier goal is to achieve fully verified MLTL benchmark generation by chaining our formalized algorithms with formally verified sat solvers (e.g., [24]). We envision our work to pave the way for future (formalized) investigations of improved MLTL model checking algorithms, parallel verification, and synthesis.

References

1. Cabodi, G., Camurati, P., Quer, S.: A graph-labeling approach for efficient cone-of-influence computation in model-checking problems with multiple properties. Software: Practice and Experience **46**(4), 493–511 (2016)
2. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at ltl model checking. Formal Methods in System Design **10**, 47–71 (1994), https://doi.org/10.1023/A:1008615614281
3. Conrad, E., Titolo, L., Giannakopoulou, D., Pressburger, T., Dutle, A.: A compositional proof framework for FRETish requirements. In: Popescu, A., Zdancewic, S. (eds.) CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022. pp. 68–81. ACM (2022). https://doi.org/10.1145/3497775.3503685, https://doi.org/10.1145/3497775.3503685
4. Degtyarev, A., Voronkov, A.: Chapter 4 - the inverse method. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 179–272. Handbook of Automated Reasoning, North-Holland, Amsterdam (2001). https://doi.org/https://doi.org/10.1016/B978-044450813-3/50006-0, https://www.sciencedirect.com/science/article/pii/B9780444508133500060
5. Dureja, R., Baumgartner, J., Ivrii, A., Kanzelman, R., Rozier, K.Y.: Boosting verification scalability via structural grouping and semantic partitioning of properties. In: 2019 Formal Methods in Computer Aided Design (FMCAD). pp. 1–9. IEEE (2019)
6. Dureja, R., Baumgartner, J., Kanzelman, R., Williams, M., Rozier, K.Y.: Accelerating parallel verification via complementary property partitioning and strategy exploration. In: # PLACEHOLDER_PARENT_METADATA_VALUE#. vol. 1, pp. 16–25. TU Wien Academic Press (2020)
7. Dureja, R., Rozier, K.Y.: More scalable ltl model checking via discovering design-space dependencies (D^3). 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. part I, 10805, pp. 309–327. Springer-Verlag (April 2018)
8. Elwing, J., Gamboa-Guzman, L., Sorkin, J., Travesset, C., Wang, Z., Rozier, K.Y.: Mission-Time LTL (MLTL) Formula Validation via Regular Expressions. In: Herber, P., Wijs, A. (eds.) Proceedings of the 18th International Conference on integrated Formal Methods (iFM). LNCS, Formal Methods subline, vol. 14300. Springer, Leiden, The Netherlands (November 2023)
9. Falco, G., Gilpin, L.H.: A stress testing framework for autonomous system verification and validation (v&v). In: 2021 IEEE International Conference on Autonomous Systems (ICAS). pp. 1–5 (2021). https://doi.org/10.1109/ICAS49788.2021.9551154
10. Ghassabani, E., Gacek, A., Whalen, M.W., Heimdahl, M.P.E., Wagner, L.: Proof-based coverage metrics for formal verification. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 194–199 (2017). https://doi.org/10.1109/ASE.2017.8115632
11. Hagemeier, C., De Giacomo, G., Vardi, M.Y.: Ltlf synthesis under unreliable input. arXiv preprint arXiv:2412.14728 (2024)
12. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal **29**(2), 147–160 (1950). https://doi.org/10.1002/j.1538-7305.1950.tb00463.x

13. Hariharan, G., Jones, P.H., Rozier, K.Y., Wongpiromsarn, T.: Maximum satisfiability of Mission-time Linear Temporal Logic. In: Petrucci, L., Sproston, J. (eds.) FORMATS. LNCS, vol. 14138, pp. 86–104. Springer (2023). https://doi.org/10.1007/978-3-031-42626-1_6, https://doi.org/10.1007/978-3-031-42626-1_6
14. Hofmeier, P., Karayel, E.: Combinatorial enumeration algorithms. Archive of Formal Proofs (November 2022), https://isa-afp.org/entries/Combinatorial_Enumeration_Algorithms.html, Formal proof development
15. Hupel, L., Nipkow, T.: A Verified Compiler from Isabelle/HOL to CakeML. In: Ahmed, A. (ed.) ESOP. LNCS, vol. 10801, pp. 999–1026. Springer (2018). https://doi.org/10.1007/978-3-319-89884-1_35, https://doi.org/10.1007/978-3-319-89884-1_35
16. Jakšić, S., Bartocci, E., Grosu, R., Ničković, D.: Quantitative monitoring of stl with edit distance. In: Falcone, Y., Sánchez, C. (eds.) Runtime Verification. pp. 201–218. Springer International Publishing, Cham (2016)
17. John, A.K., Shah, S., Chakraborty, S., Trivedi, A., Akshay, S.: Skolem functions for factored formulas. In: 2015 Formal Methods in Computer-Aided Design (FMCAD). pp. 73–80. IEEE (2015)
18. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2. In: FORMATS. pp. 196–214. LNCS, Springer, Vienna, Austria (September 2020), http://research.temporallogic.org/papers/KZJZR20.pdf
19. Kolouri, S., Park, S.R., Thorpe, M., Slepcev, D., Rohde, G.K.: Optimal mass transport: Signal processing and machine-learning applications. IEEE Signal Processing Magazine **34**(4), 43–59 (2017). https://doi.org/10.1109/MSP.2017.2695801
20. Kosaian, K., Wang, Z., Sloan, E., Rozier, K.: Formalizing MLTL formula progression in Isabelle/HOL (2024), https://arxiv.org/abs/2410.03465
21. Li, J., Rozier, K.Y.: MLTL benchmark generation via formula progression. In: Runtime Verification. pp. 426–433. Springer International Publishing (2018)
22. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability checking for mission-time LTL. Information and Computation p. 104923 (May 2022). https://doi.org/https://doi.org/10.1016/j.ic.2022.104923, https://www.sciencedirect.com/science/article/pii/S0890540122000761
23. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
24. Maric, F.: Formalization and implementation of modern SAT solvers. J. Autom. Reason. **43**(1), 81–119 (2009). https://doi.org/10.1007/S10817-009-9127-8, https://doi.org/10.1007/s10817-009-9127-8
25. NASA Technology Transfer Program: FRET : Formal Requirements Elicitation Tool (ARC-18066-1). Online: https://software.nasa.gov/software/ARC-18066-1 (2024)
26. Pan, G., Vardi, M.Y.: Symbolic techniques in satisfiability solving. In: SAT 2005: Satisfiability Research in the Year 2005. pp. 25–50. Springer (2006)
27. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science (LNCS), vol. 8413, pp. 357–372. Springer-Verlag (April 2014)
28. Rosentrater, A., Rozier, K.Y.: FPROGG: A Formula Progression-Based MLTL Benchmark Generator . To appear; emailed to authors (2025)

29. Schallau, T., Naujokat, S., Kullmann, F., Howar, F.: Tree-based scenario classification. In: Benz, N., Gopinath, D., Shi, N. (eds.) NASA Formal Methods. pp. 259–278. Springer Nature Switzerland, Cham (2024)
30. Tabajara, L.M., Vardi, M.Y.: Factored boolean functional synthesis. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 124–131. IEEE (2017)
31. Tabajara, L.M., Vardi, M.Y.: Partitioning techniques in ltlf synthesis. In: 2019 International Joint Conference on Artificial Intelligence (2019)
32. Wang, Z., Kosaian, K., Rozier, K.: Formally verifying a transformation from MLTL formulas to regular expressions (2024), accepted to TACAS 2025 (preprint forthcoming)
33. Zhang, P., Aurandt, A.A., Dureja, R., Jones, P.H., Rozier, K.Y.: Model predictive runtime verification for cyber-physical systems with real-time deadlines. In: Petrucci, L., Sproston, J. (eds.) Formal Modeling and Analysis of Timed Systems - 21st International Conference, FORMATS 2023, Antwerp, Belgium, September 19-21, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14138, pp. 158–180. Springer (2023). https://doi.org/10.1007/978-3-031-42626-1_10, https://doi.org/10.1007/978-3-031-42626-1_10
34. Zili Wang, Katherine Kosaian, and Kristin Yvonne Rozier: Formally Verifying a Transformation from MLTL Formulas to Regular Expressions. In: Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science (LNCS), vol. TBD, p. TBD. Springer-Verlag (May 2025)