Dynamic set reasoning:

Specifying and optimizing monitor encodings

by

Christopher George Johannsen

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Kristin Y. Rozier, Major Professor
Phillip H. Jones
Tichakorn Wongpiromsarn

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2024

Copyright © Christopher George Johannsen, 2024. All rights reserved.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ABSTRACT

Specifying and monitoring temporal properties over sets on real-time embedded systems requires a logic that offers sufficient expressiveness and acceptable worst-case performance. If a system designer chooses to use a non-first-order logic, they sacrifice expressiveness; if they choose a first-order logic, they sacrifice performance. To mitigate this tradeoff, we present a first-order variant of Mission-time Linear Temporal Logic (MLTL) that can specify a wide range of behaviors and offers efficient monitoring of those behaviors. We also present a set of MLTL rewrite rules and use equality saturation to optimize MLTL monitor encodings automatically. After applying equality saturation to a set of human-authored MLTL formulas, our experimental evaluation found a $\sim 35\%$ average monitor size reduction.

CHAPTER 1. INTRODUCTION

Safety-critical cyber-physical systems (CPSs) such as air- and space-craft have grown increasingly complex, integrating many sensors and sub-systems into their designs. Although system designers can rigorously test or verify their systems using industrial-proven techniques such as model checking Clarke (1997); Baier and Katoen (2008); Jhala and Majumdar (2009); Gario et al. (2016); Bozzano et al. (2015), static analysis Gurfinkel et al. (2015); Brat et al. (2014), and theorem proving Davis et al. (1962), faults inevitably occur at runtime due to environmental factors, e.g., radiation causing a sensor failure in a satellite. If a fault occurs, a safety-critical CPS would ideally detect, diagnose, and mitigate the fault automatically.

One approach to fault detection is to deploy a *monitor* based on the system's English-level requirements that computes whether each requirement is satisfied or violated during runtime. Ad-hoc monitors, such as those written directly in C/C++, are error-prone: they are challenging to validate against the reference requirements, and adding new specifications to monitor is non-trivial. Contrastingly, runtime verification (RV) is a generalizable technique that considers whether a given system trace satisfies a formal specification. Designers need only translate their English requirements into some formalism and have a tool automatically generate monitors from the formalized representation.

In selecting an RV tool to use in the context of safety-critical CPSs, practitioners have the following constraints to consider:

Realizability The monitor must use bounded memory and computational resources. It must also be able to adapt to new specifications without re-compilation to allow different monitoring configurations for the evolving stages of a mission. The RV tool must also offer a sufficiently expressive and intuitive specification language for formalizing the English-level requirements.

Responsiveness The monitor must run continuously on the system and provide results (i.e., violations) as soon as they are available.

Unobtrusiveness The monitor must not impact the *certifiability* of the overall system. Flight certification is a lengthy, rigorous process for increasing confidence of software and hardware, so a monitor must not require instrumentation that involves editing an already certified sub-system.

As such, the **Realizable Response Unobtrusiveness Unit** (R2U2) Johannsen et al. (2023a) is an RV tool that satisfies each of these constraints and has been proven to detect faults on embedded, resource-constrained CPSs Kempa et al. (2020); Cauwels et al. (2020); Hertz et al. (2021); Aurandt et al. (2022).

Despite R2U2 being a great candidate for monitoring CPSs, system designers still face challenges in the formalization process. R2U2 uses Mission-time Linear Temporal Logic (MLTL) Reinbacher et al. (2014) as its formalism. This logic admits properties expressed using interval-bounded temporal operators. However, MLTL does not have first-order capabilities and thus cannot effectively capture many requirements, such as:

- *Each task in the scheduler shall execute within the next 3 seconds.*
- *Some request in the queue shall be granted or rejected within 10 seconds.*
- *Every active process shall make progress within 25 clock cycles.*

A first-order extension to MLTL would allow the expression of such commonly seen requirement patterns. Many challenges are associated with monitoring a first-order temporal logic, including quantification over unbounded domains and efficiently storing intermediate results of sub-formulas.

Barringer et al. (2012) presents a solution via a formalism for describing parametric specifications called Quantified Event Automata (QEA), where events carry data. An event has a name, such as *start*, and can be applied to parameters, such as $start(x)$ or $start(5)$. QEA

describe the runs of such events. While sufficiently expressive and efficient for our needs, QEA do not provide a sufficiently intuitive specification language for system designers.

The authors of Bauer et al. (2015) introduce a logic named $\mathsf{LTL}^{\mathsf{FO}}$, which they show is undecidable. The paper outlines a sound but incomplete procedure for monitoring such formulas using a specialized automata structure. The incompleteness and lack of time bounds on its operators make it unfit for our target systems.

Havelund et al. (2020) offers a solution using Quantified Temporal Logic (QTL), a first-order past-time temporal logic. In that work, the authors use binary decision diagrams Bryant (1992) to represent predicates over time efficiently. Owing to QTL's lack of future-tense and interval-bounded temporal operators, the presented technique does not offer a sufficient specification language for our needs.

The most immediately applicable approach to monitoring first-order temporal properties within our constraints is given in Baier and Katoen (2008); Basin et al. (2015), which defines Metric First Order Temporal Logic (MFOTL) and targets monitoring of large-scale databases. This logic is similar to MLTL, where the provided monitoring algorithms compute an intermediate relation for each temporal operator based on the techniques defined in Chomicki (1995). However, this technique relies on access to efficient relational algebra operations, often implemented in database management systems (DBMS) such as SQLite Hipp (2020). Further, their technique requires some syntactic restrictions due to the challenge of representing complements, i.e., the complement of a finite relation is infinite and cannot be directly represented using a finite relation.

Our solution addresses these issues by noting that many first-order specifications quantify over *bounded sets*, e.g., a set of tasks in a scheduler, open files, or requests in a queue. Using this knowledge, we can separate monitoring from the predicate evaluation, i.e., we leverage existing MLTL monitoring techniques to evaluate multiple versions of a specification over time. This way, we provide a future-tense, interval-bounded temporal logic as a specification language that we monitor efficiently using `R2U2`.

Aside from efficient encodings of temporal logic monitors, little work exists on the automatic optimization of tree-based monitors. Automata-based monitors may benefit from techniques used by SPOT Duret-Lutz et al. (2022), but techniques using a syntax tree-based encoding (such as R2U2 Kempa et al. (2020)) have no such literature. To the best of our knowledge, Kempa et al. (2020) is the only work that addresses this problem, using common sub-expression elimination (CSE), a common technique used in compiler optimization Cooper et al. (2008). CSE enables sub-expressions to share the result of syntactically identical sub-expressions, reusing results that do not need to be recomputed.

This work is organized as follows: Chapter 2 defines preliminaries, Chapter 3 presents a first-order extension to MLTL and algorithms for monitoring formulas in this new logic, Chapter 4 defines a set of rewrite rules for MLTL formulas and both correctness and memory-reduction proofs, as well as an automated technique for optimizing such monitors via equality saturation Tate et al. (2009), Chapter 5 presents the Configuration Compiler for Property Organization (C2PO), a tool for both encoding and optimizing MLTL monitors and Chapter 6 provides a discussion of this work's impact and future research directions.

CHAPTER 2. PRELIMINARIES

This chapter presents previous work and the notation we will use when referencing that work throughout. We start by defining the logic that `R2U2` uses: MLTL.

2.1 Mission-time Linear Temporal Logic (MLTL)

First, let $I = [l, u]$ be an (inclusive) interval where $l, u \in \mathbb{N}_0$ and $l \leq u$.

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

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

where $p \in \mathcal{AP}$ and $I = [l, u]$.

We evaluate MLTL formulas over finite traces. A trace π is a (finite) sequence of states, where each state defines a set of atomic propositions that are true in that state. A state at timestamp $\tau \in \mathbb{N}_0$ is given by $\pi[\tau] \subseteq \mathcal{AP}$ such that $|\pi|$ is the length of π where $|\pi| < +\infty$ and π_τ is the suffix of π starting at and including τ.

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

$\pi \models p$ iff $p \in \pi[0]$.

$\pi \models \neg\varphi$ iff $\pi \not\models \varphi$.

$\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$.

$\pi \models \varphi \vee \psi$ iff $\pi \models \varphi$ or $\pi \models \psi$.

$\pi \models \Diamond_{[l,u]} \varphi$ iff $|\pi| > l$ and $\exists i \in [l, u]$ such that $\pi_i \models \varphi$.

$\pi \models \Box_{[l,u]}\varphi$ iff $|\pi| \leq l$ or for all $i \in [l,u]$, $\pi_i \models \varphi$.

$\pi \models \varphi\ \mathcal{U}_{[l,u]}\ \psi$ iff $|\pi| > l$ and $\exists i \in [l,u]$ such that $\pi_i \models \psi$ and $\forall j \in [l, i-1]$ it holds that $\pi_j \models \varphi$.

$\pi \models \varphi\ \mathcal{R}_{[l,u]}\ \psi$ iff $|\pi| \leq l$ or $\forall i \in [l,u]$ it holds that $\pi_i \models \psi$ or $\exists j \in [l,u]$ such that $\pi_j \models \varphi$ and $\forall k \in [l,j]$ it holds that $\pi_k \models \psi$.

We say two MLTL formulas φ, ψ are *semantically equivalent* (denoted as $\varphi \equiv \psi$) if and only if $\pi \models \varphi \Leftrightarrow \pi \models \psi$ for all traces π over $\mathcal{AP}$. MLTL keeps the standard operator equivalences from LTL, including $\Diamond_I \varphi = (\mathsf{true}\ \mathcal{U}_I\ \varphi)$, $\Box_I \varphi = (\mathsf{false}\ \mathcal{R}_I\ \varphi)$. To complete the MLTL semantics, we define $\mathsf{false} = \neg\mathsf{true}$, $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, $\neg(\varphi\ \mathcal{U}_I\ \psi) \equiv (\neg\varphi\ \mathcal{R}_I\ \neg\psi)$ and $\neg\Diamond_I\varphi = \Box_I\neg\varphi$.

Notably, the duals for each temporal operator have different end-of-trace interpretations. For example, consider a single-state trace π where $|\pi| = 1$ and the formulas $\Diamond_{[1,1]}p$ and $\Box_{[1,1]}p$ that index a state off the end of the trace (i.e., $\pi[1]$). Interpreting the π over each formula we have that $\pi \not\models \Diamond_{[1,1]}p$ since $|\pi| \not> 1$ but $\pi \models \Box_{[1,1]}p$ since $|\pi| \leq 1$.

The notion of *when* a trace provides sufficient information to evaluate a formula is formalized in Kempa et al. (2020) as *propagation delay*.

Definition 3 (*Propagation Delay*)**.** The propagation delay of an MLTL formula φ is the difference between the time τ for which φ is evaluated and when it is possible to know if the trace starting at τ satisfies φ. A formula's *worst-case propagation delay* ($wpd(\varphi)$) is its maximum propagation delay, and the minimum value is its *best-case propagation delay* ($bpd(\varphi)$).

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

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

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

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

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

2.2 MLTL Monitoring

MLTL monitoring is the problem of computing the value of $\pi_\tau \models \varphi$ for each $\tau \in [0, |\pi| - 1]$ where π is interpreted as a *stream* of data i.e., we compute $\pi \models \varphi$ if π has sufficient information, otherwise we wait for an extension to π that offers sufficient information. To compactly reason about streams of results, we define a sequence of true/false verdicts and their respective timestamps, which we can then use to define the behavior of an MLTL Monitor.

Definition 5 (Execution Sequence)**.** An *execution sequence* $\langle T_\varphi \rangle$ is a sequence of verdict, timestamp pairs $T_\varphi = (b, \tau)$ such that $b \in \{\mathsf{true}, \mathsf{false}\}$ and $\tau \in \mathbb{N}_0$. We access a pair's values via $T_\varphi.b$ and $T_\varphi.\tau$ and the element at index i via $\langle T_\varphi \rangle[i]$. The timestamps in $\langle T_\varphi \rangle$ are monotonically increasing i.e., for each $i, j \in \mathbb{N}_0$ where $i < j$, it holds that $\langle T_\varphi \rangle[i].\tau < \langle T_\varphi \rangle[j].\tau$.

An execution sequence $\langle T_\varphi \rangle$ has length $|\langle T_\varphi \rangle|$ and represents a sequence of truth values for an MLTL formula φ with a length proportional to the last element's timestamp. We can also query an execution sequence for the verdict at a specific timestamp τ by finding the minimal timestamp in $\langle T_\varphi \rangle$ that is greater than τ, returning $\perp$ if no such value exists:

$$\langle T_\varphi \rangle.find(\tau) = \begin{cases} \perp & \text{if } \forall\ T_\varphi \in \langle T_\varphi \rangle : T_\varphi.\tau < \tau \\ T_\varphi.b & \text{otherwise, where } T_\varphi = min\{T \mid T \in \langle T_\varphi \rangle \wedge T.\tau \geq \tau\} \end{cases}$$

Definition 6 (MLTL Monitor)**.** Given an MLTL formula φ and trace π, an *MLTL Monitor* is an algorithm that computes an execution sequence $\langle T_\varphi \rangle$ for φ over π such that if there is sufficient information in π to evaluate $\pi_\tau \models \varphi$, then

$$\forall \tau : \langle T_\varphi \rangle.find(\tau) = (\pi_\tau \models \varphi).$$

A naive implementation of an MLTL Monitor could evaluate $\pi_\tau \models \varphi$ over every τ. An *efficient* MLTL Monitor maintains its state for every suffix of π. See Reinbacher et al. (2014) and Kempa et al. (2020) for presentations of efficient MLTL monitors. Reinbacher et al. (2014) defined both *synchronous* and *asynchronous* MLTL monitors. Kempa et al. (2020) provided updated *asynchronous* monitors, and since we build off of this work specifically, we also focus on asynchronous monitors.

2.3 Tree-based MLTL Monitoring Encoding

We focus on encodings of MLTL monitors that use an Abstract Syntax Tree-based (AST) representation. Each node in the AST of an MLTL formula φ computes and stores an execution sequence $\langle T_\varphi \rangle$ in a Shared Connection Queue (SCQ) Kempa et al. (2020) for the corresponding sub-formula with respect to an input trace. The following description is adapted from Kempa et al. (2020).

SCQs have both read and write operations. The read operation for an SCQ Q takes as input a timestamp τ and outputs true if a verdict for timestamp τ exists in Q and is true; otherwise, it outputs false. The write operation takes as input a timestamp τ and Boolean b and returns an updated SCQ such that

$$(Q.write(\tau, b)).read(\tau) = b.$$

For a complete description of SCQs and their operations, see Algorithms 1 and 2 of Kempa et al. (2020).

To compute the required SCQ size for tree-based MLTL monitoring, consider an AST node g and its set of sibling nodes $\mathcal{S}_g$ (not including g). A sibling of g is any node that shares a parent with g. The minimum required number of verdict/timestamp pairs for g is

$$mem_{node}(g) = max(max\{s.wpd \mid s \in \mathcal{S}_g\} - g.bpd, 0) + 1. \tag{2.1}$$

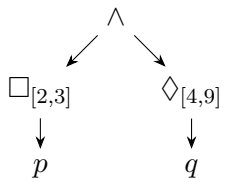


Figure 2.1 AST for the MLTL formula $(\square_{[2,3]}p) \wedge (\Diamond_{[4,9]}q)$ where $p, q \in \mathcal{AP}$.

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

$$mem_{AST}(g) = mem_{node}(g) + \sum\{mem_{AST}(c) \mid c \in \mathcal{C}_g\}. \tag{2.2}$$

Formula 2.1 accounts for the worst-case input with respect to evaluating the parent of g. Consider a trace π, time $0 \leq i < |\pi|$, and a node g such that g's value is known at index $i + g.bpd$ but the value of a sibling node s_{max} is known at index $i + s_{max}.wpd$ where

$$s_{max}.wpd = max\{s.wpd \mid s \in \mathcal{S}_g\}.$$

In order to evaluate g's parent at i, we must know the evaluations of both g and s_{max} at i and therefore buffer the values of g from indices $i + g.bpd$ to $i + s_{max}.wpd$. If $s_{max}.wpd - g.bpd < 0$ we do not need to buffer values for g, otherwise g requires a buffer of size

$$(i + s_{max}.wpd) - (i + g.bpd) = s_{max}.wpd - g.bpd.$$

As an example, consider the AST in Figure 2.1. We see $mem_{node}(\wedge) = 1$ since the $\wedge$ node has no siblings. Now, for each temporal node, we have

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

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

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

2.4 Equality Saturation

Term rewriting is an optimization technique standard in many programs, including compilers Joshi et al. (2002) and SMT solvers De Moura and Bjørner (2007). One problem with traditional term rewriting is that the original term is lost once rewritten, causing the optimized output to be order-sensitive to the rewrites applied. One technique to mitigate this is *equality saturation* Tate et al. (2009) (EqSat), which uses a data structure called an *e-graph* Nelson (1980) to represent the set of all terms seen throughout rewriting compactly, grouping equivalent terms. The main idea is to apply rewrites to the terms in the e-graph until a fixed point is reached. The resulting e-graph is deemed *saturated*, representing all derivable terms from the original term using the rewrites provided.

Recently, the authors of Willsey et al. (2021) improved the performance of traditional EqSat techniques by loosening the invariant requirements to be maintained after a certain number of operations, thereby amortizing the cost of maintaining the invariants. Zhang et al. (2023) use techniques from the database field to address the challenge of expression matching (i.e., finding which rules can be applied to which e-nodes).

Formally, an e-graph E is a set of e-classes, where each e-class C maintains a set of e-nodes. An e-node $f(c_1, \ldots, c_n)$ is a function symbol applied to a list of children e-classes. An e-class c represents a term t if any e-node in c represents t and an e-node $f(c_1, \ldots, c_n)$ represents $t = f(t_1, \ldots, t_n)$ if c_i represents t_i for each $i \in [1, n]$. Importantly, we say that any two terms in the same e-class are *equivalent*.

Once an e-graph is saturated, we must *extract* an optimal term from the e-graph. Extraction amounts to computing a cost for each e-node, denoting the minimum-cost e-node in an e-class as the *representative* of that e-class, then constructing a term starting from the e-class that represents the target expression using the representative from each e-class. For a detailed formalization of e-graphs, equality saturation, and extraction, see Willsey et al. (2021) and Zhang et al. (2023).

CHAPTER 3. FIRST-ORDER MLTL MONITORING

To fill the need by RV practitioners for a first-order variant of MLTL, we introduce First Order MLTL (FO-MLTL) and show how the resulting logic is insufficient for this purpose. We then define a bounded variant of FO-MLTL that allows us to leverage existing MLTL monitoring algorithms.

3.1 First-order MLTL

Let $S = (C, P)$ be a *signature* where C is a finite set of constant symbols, P is a finite set of predicates such that $C \cap P = \emptyset$, and $ar : P \Rightarrow \mathbb{N}_0$ is a function denoting the arity of the predicate symbols in P. Then, let V be a countably infinite set of variable symbols such that $V \cap (C \cup P) = \emptyset$.

Definition 7 (*FO-MLTL Syntax*)**.** A FO-MLTL formula Φ over a signature $S = (C, P)$ and set of variable symbols V is defined as:

$$\Phi ::= p(t_1, \ldots, t_{ar(p)}) \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \; \mathcal{U}_I \; \Phi \mid \forall x : \Phi$$

where $t_1, \ldots, t_{ar(p)} \in V \cup C$, $x \in V$, $p \in P$, and $I = [l, u]$.

Example 1. Consider a requirement that states that at least one task in the scheduler ($Sched$) must execute at least once between 0 and 10 time steps from now. We can express this in FO-MLTL using $V = \{t\}$, $P = \{Sched, Exec\}$, $C = \{\}$:

$$\forall t : Sched(t) \rightarrow \Diamond_{[0,3]} Exec(t)$$

where $Sched$ is a predicate used to denote set membership in the queue, and $Exec$ is a predicate that denotes whether its argument is executing in the current timestamp.

A finite sequence of (first-order) structures $\langle \mathcal{M} \rangle$ is called a *first-order trace*. We denote $|\langle \mathcal{M} \rangle| < \infty$ as the length of the sequence, $dom(\langle \mathcal{M} \rangle)$ as its (fixed) domain, $\langle \mathcal{M} \rangle[\tau]$ as the

first-order structure at index $\tau \in \mathbb{N}_0$, and $\langle \mathcal{M} \rangle_\tau$ be the suffix of $\langle \mathcal{M} \rangle$ starting at and including τ. A first-order structure $\mathcal{M}$ has a (potentially empty or infinite) set for each predicate symbol $p \in P$:

$$p^{\mathcal{M}} \subseteq dom(\mathcal{M})^{ar(p)}$$

such that $(o_1, \ldots, o_{ar(p)}) \in p^{\mathcal{M}}$ denotes that $p(o_1, \ldots, o_{ar(p)})$ is true with respect to $\mathcal{M}$ for $o_1, \ldots, o_{ar(p)} \in dom(\mathcal{M})$. Note that $|p^{\mathcal{M}}|$ is the number of tuples that satisfy p for $\mathcal{M}$. We abuse notation and use $p[\tau]$ to denote $p^{\langle \mathcal{M} \rangle[\tau]}$, i.e., the set of tuples in $dom(\mathcal{M})^{ar(p)}$ that satisfy p at τ.

Example 2. Consider the formula in Example 1 with the domain $\mathbb{N}_0$, where a task is identified by a task ID ($\mathbb{N}_0$). An example first-order trace of length 4 might be:

	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 3$
$Sched$	{1,2}	{1,2}	{1,2}	{1}
$Exec$	{}	{}	{2}	{1}

This trace describes two tasks in the queue with IDs 1 and 2, where task 2 executes at time step 2, then leaves the queue and task 1 executes at time step 3. To illustrate our notation, this trace defines $Sched^{\langle \mathcal{M} \rangle[0]} = \{1, 2\}$ so that the predicates $Sched(1)$ and $Sched(2)$ are true at $\tau = 0$ and $Exec^{\langle \mathcal{M} \rangle[2]} = \{2\}$ so that the predicate $Exec(2)$ is true at $\tau = 2$.

Example 3. Consider the English requirement: "At all times, each task in the scheduler shall have started after the task with ID 1" and the corresponding FO-MLTL formula

$$\forall t : Sched(t) \rightarrow StartedAfter(t, 1)$$

with same domain as in Example 2. The predicate $StartedAfter$ has an arity of 2 and is true if the first argument started after the second. An example first-order trace of length 3 might be:

	$\tau = 0$	$\tau = 1$	$\tau = 2$
$Sched$	{2,3}	{2,3}	{3}
$StartedAfter$	{(3,1),(2,1),(3,2)}	{(3,1),(2,1),(3,2)}	{(3,1)}

In this trace, both tasks 2 and 3 are in scheduler and started after task 1, where task 3 started after task 2 as well. At $\tau = 0$, this trace defines $StartedAfter^{\langle\mathcal{M}\rangle[0]} = \{(3,1),(2,1),(3,2)\}$ so that the predicates $StartedAfter(3,1)$, $StartedAfter(2,1)$ and $StartedAfter(3,2)$ are true at $\tau = 0$.

To interpret constant and variable symbols in formulas, we define a valuation function $v : V \cup C \Rightarrow dom(\mathcal{M})$ that maps constant and variable symbols to values in the relevant domain. We use $\Phi[x \mapsto y]$ to denote an FO-MLTL formula Φ where the variable x is renamed to y and similarly for the valuation function $v[x \mapsto y]$ which updates v such that $v[x \mapsto y](x) = y$.

FO-MLTL formulas also have propagation delays, a useful property for reasoning about the logic similar to MLTL.

Definition 8 (*Propagation Delay Semantics*)**.** Let Φ, Ψ be well-formed FO-MLTL formulas, then $bpd(\Phi)$ and $wpd(\Phi)$ are defined the same as in Definition 4 along with:

$$\text{if } \Phi = \forall x : \Psi \begin{cases} wpd(\Phi) = wpd(\Psi) \\ bpd(\Phi) = bpd(\Psi) \end{cases}$$

Definition 9 (*FO-MLTL Semantics*)**.** The satisfaction of a FO-MLTL formula by $\langle\mathcal{M}\rangle, v$ is defined recursively as:

$\langle\mathcal{M}\rangle, v \models p(t_1, \ldots, t_{ar(p)})$ iff $(v(t_1), \ldots, v(t_{ar(p)})) \in p[0]$

$\langle\mathcal{M}\rangle, v \models \neg\Phi$ iff $\langle\mathcal{M}\rangle, v \not\models \Phi$.

$\langle\mathcal{M}\rangle, v \models \Phi \wedge \Psi$ iff $\langle\mathcal{M}\rangle, v \models \Phi$ and $\langle\mathcal{M}\rangle, v \models \Psi$.

$\langle\mathcal{M}\rangle, v \models \Phi\ \mathcal{U}_{[l,u]}\ \Psi$ iff $|\langle\mathcal{M}\rangle| \geq l$ and there exists $j \in [l,u]$ such that $\langle\mathcal{M}\rangle_j, v \models \Psi$ and for all $k \in [l, j-1]$ it holds that $\langle\mathcal{M}\rangle_k, v \models \Phi$.

$\langle\mathcal{M}\rangle, v \models \forall x : \Phi$ iff for all $d \in dom(\mathcal{M})$ it holds that $\langle\mathcal{M}\rangle, v[x \mapsto d] \models \Phi$.

The following duals hold in FO-MLTL: $\Phi \vee \Psi \equiv \neg((\neg\Phi) \wedge (\neg\Psi))$, $\forall x : \Phi \equiv \neg\forall x : \neg\Phi$, $\Phi\ \mathcal{R}_I\ \Psi \equiv \neg((\neg\Phi)\ \mathcal{U}_I\ (\neg\Psi))$, $\Diamond_I\Phi \equiv \mathsf{true}\ \mathcal{U}_I\ \Phi$, $\Box_I\Phi \equiv \neg\Diamond_I\neg\Phi$. These semantics follow

Definition 2 except we interpret formulas over first-order traces and a valuation function, along with the definition of the $\forall$ quantifier.

Example 4. Consider the formula and trace from Example 2. We would say that

$$\langle \mathcal{M} \rangle, v \models \forall t : Sched(t) \wedge \Diamond_{[0,3]} Exec(t)$$

since if we update $v[t \mapsto 2]$, then $\langle \mathcal{M} \rangle, v \models Sched(t)$ since $v(t) = 2 \in Sched[0]$ and $\langle \mathcal{M} \rangle, v \models \Diamond_{[0,3]} Exec(t)$ since $v(t) = 2 \in Exec[2]$. Notice how we used v to provide meaning to the symbol t, i.e., we mapped the symbol t to the concrete value 2, which we then checked against each predicate.

Since FO-MLTL over a single time step can be interpreted as a standard first-order formula, its SAT problem is undecidable.

Theorem 1. *FO-MLTL -SAT is undecidable.*

Proof. We can interpret a first-order logic formula as an FO-MLTL formula with a single state (i.e., no temporal aspect). Since standard first-order logic satisfiability is undecidable Turing et al. (1936), FO-MLTL -SAT is undecidable. □

FO-MLTL is similar to MFOTL Basin et al. (2008), except for FO-MLTL's finite-trace and $\mathcal{U}$ semantics. In fact, MFOTL monitoring techniques could monitor FO-MLTL formulas Basin et al. (2015). However, this approach comes with all the advantages and drawbacks discussed previously (Chapter 1). The drawbacks make this logic insufficient for targeting embedded platforms with hard resource constraints.

To address this, we must restrict FO-MLTL quantifiers somehow, particularly in what they can quantify over. Once we do this, we can perform quantifier elimination and leverage existing MLTL algorithms to reason about this logic.

3.2 FO-MLTL Over Bounded Dynamic Sets

Many logics that restrict first-order logic do so by restricting the domain, i.e., only admitting finite or bounded domains Chen et al. (2022); Cerrito et al. (1999). In our case, we admit infinite

domains, but we restrict quantification to occur only over sets of bounded size that change over time. We place a syntactic restriction and cardinality constraint onto quantifiers. In particular, we require that all quantifiers be *guarded* Andréka et al. (1998) by a unary predicate as in

$$\exists x : D(x) \wedge \Phi \text{ or}$$

$$\forall x : D(x) \rightarrow \Phi.$$

where we require $|D|$ to be within some bound. We say that D represents a *bounded* ***D****ynamic set*: a set with bounded size that changes over time Johannsen et al. (2023b). The resulting logic is a variant of FO-MLTL, denoted as $\overline{\text{FO-MLTL}}$.

Definition 10 (*$\overline{\text{FO-MLTL}}$ Syntax*)**.** A $\overline{\text{FO-MLTL}}$ formula Φ over a signature $S = (C, P)$ and set of variable symbols V is defined as:

$$\Phi ::= p(t_1, \ldots, t_{ar(p)}) \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \; \mathcal{U}_I \; \Phi \mid \forall x \in D : \Phi$$

where $t_1, \ldots, t_{ar(p)} \in V \cup C$, $x \in V$, $p, D \in P$, $ar(D) = 1$, and $I = [l, u]$.

D is a unary predicate intended to represent membership for some set of bounded cardinality, i.e., we interpret $x \in D[\tau]$ as saying an object x is in the set D at time τ.

Example 5. The formula from Example 1 can be made into a corresponding $\overline{\text{FO-MLTL}}$ formula via

$$\forall t \in Sched : \Diamond_{[0,3]} Exec(t)$$

and defining that $max(Sched) = 2$.

The semantics for $\overline{\text{FO-MLTL}}$ are essentially the same as in FO-MLTL, except for the cardinality constraint in the quantifier semantics.

Definition 11 (*$\overline{\text{FO-MLTL}}$ Semantics*)**.** The satisfaction of a $\overline{\text{FO-MLTL}}$ formula by $\langle\mathcal{M}\rangle, v$ and function $max : P \rightarrow \mathbb{N}_0$ where $|D|$ is computable in $\mathcal{O}(1)$ is defined recursively as:

$\langle\mathcal{M}\rangle, v \models p(t_1, \ldots, t_{ar(p)})$ iff $|\langle\mathcal{M}\rangle| > 0$ and $(v(t_1), \ldots, v(t_{ar(p)})) \in p[0]$.

$\langle \mathcal{M} \rangle, v \models \neg\Phi$ iff $\langle \mathcal{M} \rangle, v \not\models \Phi$.

$\langle \mathcal{M} \rangle, v \models \Phi \wedge \Psi$ iff $\langle \mathcal{M} \rangle, v \models \Phi$ and $\langle \mathcal{M} \rangle, v \models \Psi$.

$\langle \mathcal{M} \rangle, v \models \Phi \; \mathcal{U}_{[l,u]} \; \Psi$ iff $|\langle \mathcal{M} \rangle| > i + l$ and there exists $j \in [i + l + u]$ such that $\langle \mathcal{M} \rangle_j, v \models \Psi$ and for all $k \in [i + l + j - 1]$ it holds that $\langle \mathcal{M} \rangle_k, v \models \Phi$.

$\langle \mathcal{M} \rangle, v \models \forall x \in D : \Phi$ iff $|\langle \mathcal{M} \rangle| > 0$, $|D[0]| \leq max(D)$, and for all $d \in D[0]$ it holds that $\langle \mathcal{M} \rangle, v[x \mapsto d] \models \Phi$.

Note that we require that computing the cardinality of D is $\mathcal{O}(1)$. Recall that the intention is for D to represent the contents of a real-time data structure, which, in practice, has an easily retrievable cardinality. Furthermore, if D were infinite, then it would necessarily always fail the cardinality constraint. $\overline{\mathsf{FO\text{-}MLTL}}$ admits all the duals from FO-MLTL, including the quantifier dual:

$$\forall x \in D : \Phi = \neg\exists x \in D : \neg\Phi.$$

For completeness, we define the semantics of the existential quantifier:

$\langle \mathcal{M} \rangle, v \models \exists x \in D : \Phi$ iff $|\langle \mathcal{M} \rangle| \leq 0$, $|D[0]| > max(D)$, or $(\langle \mathcal{M} \rangle, v[x \mapsto d]) \models \Phi$ for some $d \in D[0]$.

Unintuitively, a trace where the set predicate D violates its cardinality constraint will satisfy this expression due to the $|D[0]| > max(D)$ clause. We can add an implicit cardinality constraint to each $\exists$ operator instance.

Example 6. Consider the formula

$$\exists t \in Sched : \Diamond_{[0,3]} Exec(t),$$

which may be satisfied by a trace where $|Sched| > max(Sched)$. We can express that only traces where $|Sched| \leq max(Sched)$ shall satisfy the formula by adding the cardinality constraint on $Sched$ as a proposition:

$$(|Sched| \leq max(Sched)) \wedge \exists t \in Sched : \Diamond_{[0,3]} Exec(t).$$

3.3 $\overline{\text{FO-MLTL}}$ Monitoring

We now move on to defining efficient algorithms for monitoring $\overline{\text{FO-MLTL}}$ formulas. At a high level, the presented approach monitors each quantified expression via unrolling and uses existing MLTL monitors for the rest of the operators. For example, the monitor for a formula $\Phi = \forall x \in D : \Psi$ functionally considers the formula

$$|D| \leq max(D) \wedge \bigvee_{i \in [1, max(D)]} D(d_i) \rightarrow \Psi[x \mapsto d_i].$$

We call this an *unrolled formula* and the newly-introduced variables d_i *unrolled variables*. A set of unrolled variables (with associated set D) is denoted by U_D. To ensure an unrolled formula is well-formed, we add U_D to V.

3.3.1 Auxiliary Valuation Functions

Currently, the variables in U_D have no meaning. To provide such a meaning, we require a mapping of each $d_i \in U$ to an element in its associated set D at each timestamp.

Definition 12 (*Auxiliary Valuation Function*)**.** An *auxiliary valuation function* for a $\overline{\text{FO-MLTL}}$ formula Φ, set symbol D, unrolled variables U_D, and domain *dom* is a function mapping unrolled variable symbols to concrete values

$$u_D : U_D \Rightarrow dom \cup \perp$$

such that for all $d \in dom$:

1. $d \in D$ iff there is some symbol d_i for which u_D maps to d:

$$D(d) \Leftrightarrow \forall d_i : u_D(d_i) = d$$

2. Every symbol must map to a unique member of D, or $\perp$ otherwise:

$$\forall d_i, d_j : d_i \neq d_j \wedge u_D(d_i) \neq \perp \Rightarrow u_D(d_i) \neq u_D(d_j)$$

Intuitively, u_D is a function that maps indices of an array representing D to the members of D (i.e., reads the element d_i at index i), where the return value is $\perp$ if no element is at the given index. So, unrolled variables take on a value in D or the $\perp$ (empty) value. Importantly, any predicate that takes $\perp$ as an argument will evaluate to false.

However, since we are concerned with the behavior of D over time, we define a finite sequence of auxiliary valuation functions $\langle u_D \rangle$ where $|\langle u_D \rangle| = |\langle \mathcal{M} \rangle|$ for a given first-order trace $\langle \mathcal{M} \rangle$.

Example 7. Consider the set represented by the unary predicate $Sched$ where $max(Sched) = 3$. If we depict a single u_{Sched} function as an array, then the following trace shows a valid definition for $\langle u_{Sched} \rangle$

	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 3$
$Sched$	{2,3}	{2,3}	{3}	{4,5}
$\langle u_{Sched} \rangle$	[2,3⊥]	[2,3,⊥]	[⊥,3,⊥]	[⊥,5,4]

Now we can query $\langle u_{Sched} \rangle$ at a timestamp and index. For example, we have that $\langle u_{Sched} \rangle[0](d_1) = 2$, $\langle u_{Sched} \rangle[0](d_2) = 3$, and $\langle u_{Sched} \rangle[0](d_3) = \perp$ at $\tau = 0$.

3.3.2 $\overline{\text{FO-MLTL}}$ Monitoring Algorithms

Before we define the $\overline{\text{FO-MLTL}}$ monitor algorithms, we address the complexity raised by including variables whose valuations change over time for quantified formulas; we need to consider that we evaluate a different formula at every time step. Considering the formula from Example 5 and the trace in Example 7, at $\tau = 0$ we evaluate the formula

$$\big(Sched(2) \rightarrow \Diamond_{[0,3]} Exec(2)\big) \wedge \big(Sched(3) \rightarrow \Diamond_{[0,3]} Exec(3)\big) \wedge \big(Sched(\perp) \rightarrow \Diamond_{[0,3]} Exec(\perp)\big)$$

and at $\tau = 3$ we evaluate

$$\big(Sched(\perp) \rightarrow \Diamond_{[0,3]} Exec(\perp)\big) \wedge \big(Sched(5) \rightarrow \Diamond_{[0,3]} Exec(5)\big) \wedge \big(Sched(4) \rightarrow \Diamond_{[0,3]} Exec(4)\big)\,.$$

To address this, we instantiate a new monitor for every time step. Thankfully, we only need at most $wpd(\Phi)$ monitors at a given time. In our example, we instantiate a monitor for each time

Algorithm 1: $\overline{\text{FO-MLTL}}$ monitor initialization procedure. Assigns SCQ slots to sub-formulas according to Definitions 4, 8.

```
1  Init(Φ: FO-MLTL formula, W: map of WPDs for unrolled variables, O: map of offsets for
     unrolled variables): begin
2      if Φ = p(t) // predicate then
3          w_Φ ← W(t) if t is unrolled variable ;        // Store WPD of quantified formula
4          o_Φ ← O(t) if t is unrolled variable ;                // Store timestamp offset
5      else if Φ = ¬Ψ // unary operator then
6          Init(Ψ, w, o) ;
7          Q_Ψ ← SCQ(1) ;                                  // No siblings → no SCQ slots
8      else if Φ = ξ ∧ Ψ or Φ = ξ U_[l,u] Ψ // binary operator then
9          Init(ξ, w, o), Init(Ψ, w, o) ;
10         Q_ξ ← SCQ(max(wpd(Ψ) − bpd(ξ), 0) + 1) ;      // Size SCQ via Equation 2.1
11         Q_Ψ ← SCQ(max(wpd(ξ) − bpd(Ψ), 0) + 1) ;      // Size SCQ via Equation 2.1
12         τ_↓Ψ, τ_out ← 0 ;                                // Declare auxiliary vars
13     else if Φ = ∀x ∈ D : Ψ // quantifier then
14         C_Φ ← CircSCQBuffer(wpd(Ψ) + 1) ; // Create SCQ buffer of size wpd(Ψ) + 1
15         foreach i ∈ [0, wpd(Φ)] do
16             Q_Ψi ← Init(⋀_{j∈[1,max(D)]} D(d_{i,j}) → Ψ[x ↦ d_{i,j}], W[d_{i,j} ↦ wpd(Φ)], O[d_{i,j} ↦ j]) ;
17             C_Φ[i] ← Q_Ψi ;
18         end
19         cur_C ← 0 ;
20 end
```

from $\tau = 0$ to $\tau = 3 = wpd(\Phi)$, but at $\tau = 4$ we can re-use the monitor used at $\tau = 0$ since its verdict is guaranteed to be computed. We store these monitors in a circular buffer of SCQs, as shown in Algorithm 1 on line 14.

Additionally, we define the initialization procedure for each monitor in Algorithm 1. The procedure recurses down the AST of the target formula, allocating SCQs for each non-unary operator using Equation 2.1 and initializing necessary variables/buffers.

Next, we define the monitors for each of the $\overline{\text{FO-MLTL}}$ operators and show that each implements the $\overline{\text{FO-MLTL}}$ monitoring semantics. The monitor algorithm for an operator expression Φ with SCQ Q_Φ takes as input the SCQs of its child expressions and writes a verdict/timestamp pair to Q_Φ if sufficient information is available.

Algorithm 2: $\overline{\text{FO-MLTL}}$ $\Phi = p(t)$ monitor algorithm. This algorithm only presents the unary case for clarity.

Input: Timestamp τ, first-order trace $\langle \mathcal{M} \rangle$, valuation function v, set of auxiliary valuation function sequences $\mathfrak{D}_\Psi$

Var: SCQ Q_Φ, WPD w_Φ, timestamp offset o_Φ

1 **if** t ***is an unrolled variable from set symbol*** D **then**
2 $\quad \tau' \leftarrow (\tau - (\tau \ \% \ w_\Phi)) + o_\Phi$; `// Compute timestamp to index` $\langle u_D \rangle$ `with`
3 $\quad val \leftarrow \langle u_D \rangle[\tau'](t)$; `// Look up value of` t `using` $\langle u_D \rangle \in \mathfrak{D}_\Psi$
4 **else**
5 $\quad val \leftarrow v(t)$; `//` t `is free, look up value using` v
6 **end**
7 $res \leftarrow (val \in p[\tau], \tau)$; `// Compute` $p(t)$
8 $wr_{SCQ}(Q_\Phi, res)$;

Algorithm 3: $\overline{\text{FO-MLTL}}$ $\Phi = \forall x \in D : \Psi$ monitor algorithm.

Input: Timestamp τ

Var: SCQ Q_Φ, Set symbol D, Circular SCQ buffer C_Φ, Buffer index cur_C

1 `// Only check available values with monotonically increasing timestamps`
2 **while** $C_\Phi.get(cur_C).\tau \geq \tau$ **do**
3 $\quad verdict \leftarrow C_\Phi.get(cur_C).read(\tau) \wedge |D[\tau]| \leq max(D)$; `// Check result of` C_Φ `at` τ
4 $\quad T_\Phi \leftarrow (verdict, \tau)$; `// Construct result`
5 $\quad Q_\Phi.write(T_\Phi)$; `// Write result to SCQ`
6 $\quad cur_C \leftarrow cur_C.next()$; `// Move on to next value`
7 **end**

The monitor for a unary predicate expression $\Phi = p(t)$ (Algorithm 2) writes the result of $v(t) \in p[\tau]$ to Q_Φ at τ given a timestamp τ, first-order trace $\langle \mathcal{M} \rangle$, and valuation function v. If t is an unrolled variable, the monitor instead computes $\langle u_D \rangle[\tau'](t) \in p[\tau]$ using the mapping supplied via $\langle u_D \rangle$ at an offset timestamp τ'. The offset is determined by τ and the quantifier formula from which t is unrolled (line 2).

We then define the monitor algorithm for the $\forall$ operator in Algorithm 3. The monitor maintains an index in the buffer of SCQs (cur_C) for the SCQ with the minimal τ, representing the "oldest" relevant formula without a verdict. If this SCQ has a verdict for τ, the algorithm

checks the verdict value and if the cardinality constraint holds (line 3). The monitor writes the result of this check to its SCQ (lines 4-5) and moves on to the next SCQ in the buffer (line 6).

Kempa et al. (2020) defines monitors and proofs of correctness for the rest of the operators ($\neg$, $\wedge$, and $\mathcal{U}$). We must then show that Algorithm 3 correctly implements the $\forall$ $\overline{\text{FO-MLTL}}$ semantics.

For clarity, we abstract away the circular nature of the underlying data structures such that an SCQ Q_Φ defines an execution sequence $\langle T_\Phi \rangle$ and a circular buffer C_Φ defines a sequence of SCQs $\langle C_\Phi \rangle$.

Theorem 2 ($\forall$ Monitor Correctness)**.** *Let $\langle \mathcal{M} \rangle$ be a first-order trace, v be a valuation function, $\langle u_D \rangle$ be a sequence of auxiliary valuation functions, $\Phi = \forall x \in D : \Psi$ be an $\overline{\text{FO-MLTL}}$ formula, Q_Φ the SCQ for Φ, and $\langle T_\Phi \rangle$ be the execution sequence for Q_Φ. Then Algorithm 3 implements the $\forall$ $\overline{\text{FO-MLTL}}$ semantics, i.e., if $|\langle \mathcal{M} \rangle_\tau| \geq wpd(\Phi)$, then*

$$\forall \tau : \langle T_\Phi \rangle.find(\tau) = (\langle \mathcal{M} \rangle_\tau, v \models \forall x \in D : \Psi).$$

Proof Sketch. For an arbitrary timestamp τ, Algorithm 3 considers the formula

$$\Psi_\tau = \bigwedge_{j \in [1, max(D)]} D(d_{\tau,j}) \rightarrow \Psi[x \mapsto d_{\tau,j}]$$

where we mark each unrolled variable with the timestamp τ and an index j. We note that $(\langle \mathcal{M} \rangle_\tau, v \models \Psi_\tau) \Leftrightarrow (\langle \mathcal{M} \rangle_\tau, v \models \Phi)$ if v uses a well-defined auxiliary valuation function for $d_{\tau,j}$. Inductively assuming that the execution sequence $\langle T_{\Psi_\tau} \rangle$ for this formula corresponds to the $\overline{\text{FO-MLTL}}$ semantics, then

$$\langle T_{\Psi_\tau} \rangle.find(\tau) \Leftrightarrow (\langle \mathcal{M} \rangle_\tau, v \models \Psi_\tau).$$

Importantly, when monitoring a predicate sub-formula $p(d_{\tau,j})$ of Ψ_τ, Algorithm 2 will use $\langle u_D \rangle[\tau]$ to valuate $d_{\tau,j}$ instead of v. Therefore, using $\langle u_D \rangle$ over Ψ_τ, it holds that

$$\langle T_\Phi \rangle.find(\tau) \Leftrightarrow \langle T_{\Psi_\tau} \rangle.find(\tau) \wedge |D[\tau]| \leq max(D) \Leftrightarrow (\langle \mathcal{M} \rangle_\tau, v \models \Psi_\tau) \Leftrightarrow (\langle \mathcal{M} \rangle_\tau, v \models \Phi)$$

in other words, Algorithm 3 computes the $\forall$ operator using the unrolled formula Ψ_τ over $\langle \mathcal{M} \rangle, v$ and $\langle u_D \rangle$ and the necessary cardinality constraint. $\square$

3.3.3 $\overline{\text{FO-MLTL}}$ Monitor Space Bounds

While correctness is a necessary property for our algorithms, we must also provide bounds on their required memory, especially in the context of resource-constrained systems that we consider. We provide the memory bound via the number of required SCQ slots, recalling that one slot includes a Boolean value and timestamp.

Theorem 3 ($\overline{\text{FO-MLTL}}$ Monitor Size)**.** *The size of the monitor for a $\overline{\text{FO-MLTL}}$ formula Φ is bounded by*

$$|\Phi| \cdot (3 \cdot m \cdot (wpd(\Phi) + 1)^2)^d$$

SCQ slots where $|\Phi|$ is the number of connectives in Φ, m is the largest of all maximum set sizes, and d is the depth of Φ's AST.

Proof. A quantifier formula $\forall x \in D : \Psi$ creates at most $wpd(\Phi) + 1$ SCQs for buffering (line 15 of Algorithm 1). A buffered SCQ with index i represents the formula

$$\bigwedge_{j \in [1,m]} D(d_{i,j}) \rightarrow \Psi[x \mapsto d_{i,j}],$$

which using Equation 2.1 requires no more than $3 \cdot m$ SCQs (one for $\rightarrow$, $D(t)$, and $\Psi[x \mapsto t]$, then m copies due to unrolling), where each SCQ has no more than $wpd(\Phi)$ slots. Therefore, a quantifier formula requires no more than $3 \cdot m \cdot (wpd(\Phi)+1) \cdot (wpd(\Phi)) \leq 3 \cdot m \cdot (wpd(\Phi)+1)^2$ SCQ slots. No other node type requires more SCQ slots. Then, if Φ has a depth of d, in the worst-case we have d nested quantifiers, and so Φ requires at most $(3 \cdot m \cdot (wpd(\Phi) + 1)^2)^d$ SCQ slots. □

One important note is that the size of the $\overline{\text{FO-MLTL}}$ monitors does not include the size of the first-order structures $\mathcal{M}$. The monitors' size is not dependent on the first-order structures, but the predicate monitor in Algorithm 2 does rely on the existence of such structures to compute its result. Automata or tables can encode such structures as in Basin et al. (2015), or, in practice, arbitrary code that computes a Boolean result given a list of arguments.

CHAPTER 4. MLTL MONITOR ENCODING OPTIMIZATIONS

More than merely encoding specifications as MLTL monitors, we can automatically reduce a monitor's encoding size. We have already briefly mentioned one existing optimization, Common Sub-expression Elimination (Chapter 1). In this chapter, we present a set of rewrite rules for MLTL, prove that the rules maintain the MLTL semantics, prove that the rules either decrease or do not impact the memory requirements of the resulting monitor, and use an existing tool to perform equality saturation in order to compute an equivalent, maximally-memory-reduced formula with respect to the rewrite rules.

4.1 MLTL Rewrite Rules[1]

We present rewriting rules for reducing the AST encoding size of MLTL formulas that can be applied automatically during MLTL formula encoding to reduce its size. These rules are similar to SPOT's Duret-Lutz et al. (2022) optimizations for LTL, where SPOT minimizes an LTL formula's automata representation.

Before we present the rewrites, we recall that the MLTL semantics place some constraints on the length of a trace π (Section 2). When monitoring, we ignore the end-of-trace semantics since the monitors are stream-based, i.e., we assume that new information will eventually be provided. As such, a rewrite must maintain the MLTL *monitoring* semantics between the original and rewritten formula, where any end-of-trace behavior is ignored.

Figure 4.1 contains the MLTL rewrite rules. We first prove that both sides of each rewrite rule are equivalent in the MLTL monitoring semantics using trivially derived equivalences from LTL and the definitions of MLTL operators. We then show how each rewrite rule maintains or reduces the memory of a given MLTL formula.

[1] Adapted from Johannsen et al. (2023b)

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

Table 4.1 Table of MLTL rewrite rules where $\varphi, \psi, \varphi_1, \varphi_2, \varphi_3$ are well-formed MLTL formulas and $a, l, u, l_1, u_2, l_2, u_2, l_3, u_3 \in \mathbb{N}_0$ such that $l \leq u$, $l_1 \leq u_1$, $l_2 \leq u_2$, $l_3 \leq u_3$. Each group of rules has identical constraints on their interval bounds. An asterisk next to a rule (ex: (R3)) means that the rule is only valid in the MLTL monitoring semantics due to MLTL's end-of-trace semantics.

In addition to the rewrite rules, recall that MLTL does not include a Next-time operator ($\mathcal{X}$) as in LTL because it is equivalent to $\Box_{[1,1]}$. More generally, we can express $a \in \mathbb{N}_0$ nested $\mathcal{X}$ operations with a singleton interval such as in $\Box_{[a,a]}$. Therefore, we observe the following equivalences:

$$\Box_{[a,a]}\varphi \equiv \Diamond_{[a,a]}\varphi \equiv \psi\ \mathcal{U}_{[a,a]}\ \varphi. \tag{4.1}$$

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

$$\mathsf{false}\ \mathcal{U}_{[l,u]}\ \varphi \equiv \Box_{[l,l]}\varphi \qquad \mathsf{true}\ \mathcal{U}_{[l,u]}\ \varphi \equiv \Diamond_{[l,u]}\varphi \qquad \varphi\ \mathcal{U}_{[l,u]}\ \varphi \equiv \Box_{[l,l]}\varphi. \tag{4.2}$$

Theorem 4 (Equivalence of MLTL Rewrite Rules)**.** *Let $\varphi, \psi, \varphi_1, \varphi_2, \varphi_3$ be well-formed MLTL formulas and a, l, u, l_1, u_2, l_2, u_2, l_3, $u_3 \in \mathbb{N}_0$ such that $l \leq u$, $l_1 \leq u_1$, $l_2 \leq u_2$, $l_3 \leq u_3$. Then, each rewrite relation ($\mapsto$) in Fig 4.1 is also an equivalence relation according to the MLTL monitoring semantics.*

Proof Sketch. The proof shows that for a trace π and rewrite rule $\varphi \mapsto \psi$, $\pi \models \varphi \Leftrightarrow \pi \models \psi$. Most cases follow directly from the MLTL monitoring semantics along with some interval arithmetic. We do note that rules (R3), (R6), and (R8) do not hold in the MLTL semantics due to the differences in trace length constraints between $\Diamond, \mathcal{U}$ and $\Box, \mathcal{R}$, but do hold in the monitoring semantics. We show the proof for the hardest case (R2) here, with the rest in the Appendix.

First, let φ, ψ be MLTL formulas and $l_1 \leq u_1, l_2 \leq u_2$. We prove

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

for any $l_3 = min(l_1, l_2)$, $u_3 = l_3 + min(u_1 - l_1, u_2 - l_2)$, $l_3 < u_3$ using established equivalences and the MLTL semantics. Using (R1), we see that

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

This follows if both intervals $[l_1 - l_3, u_1 - l_3]$, $[l_2 - l_3, u_2 - l_3]$ are valid i.e., (a) $l_1 - l_3 \geq 0$, (b) $l_2 - l_3 \geq 0$, and (c) $l_1 - l_3 \leq u_1 - l_3$, (d) $l_2 - l_3 \leq u_2 - l_3$.

1. Recall that $l_3 = l_1$, then $l_3 \leq l_1$.
2. Recall that $l_3 = l_1 \leq l_2$, then $l_2 - l_3 \geq 0 \rightarrow l_2 \geq l_3 \rightarrow l_3 \leq l_2$ holds.
3. Since $l_1 \leq u_1$, we see that $l_1 - l_3 \leq u_1 - l_3 \rightarrow l_1 \leq u_1$ holds.
4. Since $l_2 \leq u_2$, we see that $l_2 - l_3 \leq u_2 - l_3 \rightarrow l_2 \leq u_2$ holds.

Now, let $u_3 = l_3 + min(u_1 - l_1, u_2 - l_2)$. Applying (R1) once more, we have

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

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

1. Consider the two cases of $u_1 - l_1 \leq u_2 - l_2$ and $u_2 - l_2 < u_1 - l_1$:

   (a) Assume $u_1 - l_1 \leq u_2 - l_2$, then $u_3 = u_1 - l_1 + l_3$. Replacing this in the target inequality, we have

   $$l_1 - l_3 \leq u_1 - (u_1 - l_1 + l_3) \rightarrow l_1 - l_3 \leq l_1 - l_3.$$

   (b) Otherwise, $u_2 - l_2 < u_1 - l_1$. Then $u_3 = u_2 - l_2 + l_3$, and replacing this in the target inequality, we have

   $$l_1 - l_3 \leq u_1 - (u_2 - l_2 + l_3) \rightarrow 0 \leq u_1 - l_3 - (u_2 - l_2) \rightarrow u_2 - l_2 \leq u_1 - l_3.$$

   Since $l_3 = l_1$, we have $u_2 - l_2 \leq u_1 - l_1$, which holds from our assumption.

2. Consider the two cases of $u_1 - l_1 \leq u_2 - l_2$ and $u_2 - l_2 < u_1 - l_1$:

   (a) Assume $u_1 - l_1 \leq u_2 - l_2$, then $u_3 = u_1 - l_1 + l_3$. Replacing this in the target inequality, we have

   $$l_2 - l_3 \leq u_2 - (u_1 - l_1 + l_3) \rightarrow l_2 - l_3 \leq u_2 - u_1 + l_1 - l_3 \rightarrow$$
   $$l_2 \leq u_2 - u_1 + l_1 \rightarrow u_1 - l_1 \leq u_2 - l_2,$$

   which is true from our assumption.

   (b) Otherwise, $u_2 - l_2 < u_1 - l_1$, then $u_3 = u_2 - l_2 + l_3$. Replacing this in the target inequality, we have

   $$l_2 - l_3 \leq u_2 - (u_2 - l_2 + l_3) \rightarrow l_2 - l_3 \leq u_2 - u_2 + l_2 - l_3 \rightarrow$$
   $$l_2 \leq u_2 - u_2 + l_2 \rightarrow l_2 \leq l_2.$$

Finally, let π be a trace. We prove that

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

1. ($\rightarrow$) Let π be defined such that $\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi)$. We show that $\pi \models \Box_{[l_3,u_3]}(\Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi \wedge \Box_{[l_2-l_3,u_2-l_3-u_3]}\psi)$ using the MLTL semantics. We apply the semantic definitions of $\wedge$ and $\Box_I$ to see that $\pi_i \models \Box_{[l_1,u_1]}\varphi$ and $\pi_i \models \Box_{[l_2,u_2]}\psi$ for all $i \in [l_3, u_3]$. Combining these relations using the semantics of $\wedge$ once more, we see that $\pi_i \models \Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi$ for all $i \in [l_3, u_3]$. Using the semantics of $\Box_I$ again, we see that $\pi \models \Box_{[l_3,u_3]}(\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi)$.

2. ($\leftarrow$) Conversely, let π be defined such that $\pi \models \Box_{[l_3,u_3]}(\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi)$. We show that $\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi)$ using the MLTL semantics. Then $\pi_i \models \Box_{[l_1,u_1]}\varphi$ and $\pi_i \models \Box_{[l_2,u_2]}\psi$ for all $i \in [l_3, u_3]$. Using the semantic definitions of $\wedge$ and $\Box_I$, we see that $\pi \models \Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi$ and $\pi \models \Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi$, so $\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi)$.

The proof for the $\Diamond$ version of (R2) is symmetric. □

4.2 Inapplicable LTL Equivalences[2]

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

Similarly, consider the LTL equivalence $\Diamond\Box\varphi \wedge \Diamond\Box\psi \equiv \Diamond\Box(\varphi \wedge \psi)$. Again, intuitively, the LTL operators $\Diamond$ and $\Box$ do not specify *when* their operands must hold, just that they both eventually always hold. When we add bounds to the left-hand side as in $\Diamond_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi \wedge \Diamond_{[l_3,u_3]}\Box_{[l_4,u_4]}\psi$, both φ and ψ have constraints on *when* they must hold in order for a trace to satisfy this formula,

[2] Adapted from Johannsen et al. (2023b)

and when this is exactly may differ for either φ or ψ. Speaking generally, MLTL places more constraints on the set of traces that satisfy a given formula than LTL.

4.3 Memory Effects of Rewrites on MLTL Monitor Encodings[3]

Applying these rewrite rules can reduce the overall memory requirements of the AST encoding of the MLTL formula. These rules reduce memory requirements in one of two ways: (1) by tightening propagation delays or (2) by reducing formula length.

From Equation 2.2, an AST node g's required memory is the difference between that $g.bpd$ and the maximum wpd of its siblings. Therefore, reducing $max\{s.wpd \mid s \in \mathcal{S}_g\}$ for a set of sibling nodes $\mathcal{S}_g$ can reduce the memory requirements of all other sibling nodes in $\mathcal{S}_g$. Furthermore, reducing g's wpd can reduce its ancestors' wpd, which in turn could reduce the memory requirements for the ancestors' set of siblings in the same manner. In the following, we use $\varphi(\psi_1 \mapsto \psi_2)$ to denote an MLTL formula that is identical to a formula φ where a sub-formula ψ_1 of φ is replaced with ψ_2.

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

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

Proof. We first note that ψ_1, ψ_2 have the same set of siblings i.e., $\mathcal{S}_{\psi_1} = \mathcal{S}_{\psi_2} = \mathcal{S}$. Then from Equation 2.1 we see that

$$\begin{aligned} mem_{node}(\psi_1) =& max(max\{s.wpd \mid s \in \mathcal{S}\} - \psi_1.bpd, 0) + 1 \\ \geq& max(max\{s.wpd \mid s \in \mathcal{S}\} - \psi_2.bpd, 0) + 1 \\ =& mem_{node}(\psi_2). \end{aligned}$$

□

[3] Adapted from Johannsen et al. (2023b)

Lemma 2 (*Memory Effect of Tighter WPD*)**.** *Let φ, ψ_1, ψ_2 be well-formed MLTL formulas where ψ_1 is a sub-formula of φ, ψ_2 is the sub-formula in $\varphi(\psi_1 \mapsto \psi_2)$, and $\psi_2.wpd \leq \psi_1.wpd$. Then $\varphi(\psi_1 \mapsto \psi_2)$ requires equal or lesser memory than φ when controlling for ψ_1 and ψ_2:*

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

Proof Sketch. The proof considers two cases: when the rewritten sub-formula ψ_1 has the maximum *wpd* of all its sibling nodes and when it does not. In the case that ψ_1 has the maximum *wpd*, then its sibling nodes' SCQ sizes will decrease, as shown by the formula for computing SCQ sizes (Equation 2.1), thereby decreasing the size of the overall formula φ. Otherwise, ψ_1 having a lower *wpd* will not impact the SCQ sizes of any nodes, so the memory requirements of φ remain the same. See the appendix for the full proof. □

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

Theorem 5 (Memory Reduction of Rewriting Rules)**.** *Let φ, ψ_1, ψ_2 be MLTL formulas where ψ_1 is a sub-formula of φ. Then applying a valid rewrite rule in Figure 4.1 to ψ_1 will result in a new formula $\varphi(\psi_1 \mapsto \psi_2)$ such that $\varphi \equiv \varphi(\psi_1 \mapsto \psi_2)$ and*

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

Proof Sketch. Similar to the proof for Theorem 4, we show that each rewrite rule $\varphi \mapsto \psi$ in Table 4.1 reduces or maintains the memory required to encode φ as an MLTL monitor using Lemmas 1 and 2 and removing a node from φ's AST will reduce its memory by at least the +1 constant of Equation 2.1. Similar to the proof for Theorem 4, we show the proof for (R2) with the rest being in the Appendix.

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

$$\psi_1.wpd = max(\varphi_1.wpd + u_1, \varphi_2.wpd + u_2)$$

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

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

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

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

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

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

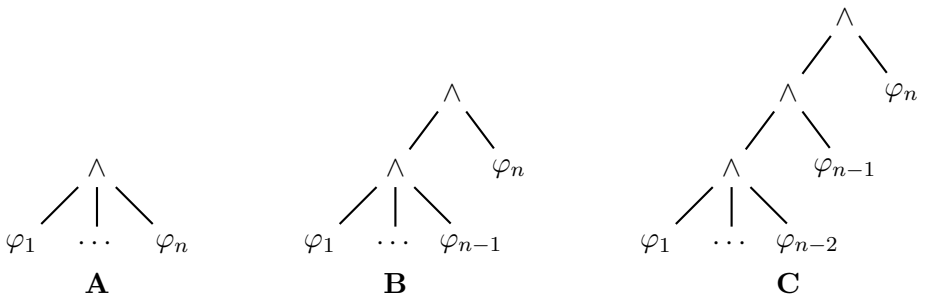


Figure 4.1 Example showing the progression of AST structures of a conjunction with n operands.

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

4.4 Topological Optimization

The memory requirements for associative, multi-arity operators such as conjunction and disjunction depend on the structure of their AST. Consider that we calculate memory requirements using the largest of all siblings' worst-case propagation delay. If there is a conjunction operator with n operands, then $n-1$ of those operands will depend on the maximal *wpd* of the operands. Where the operand with the maximal *wpd* is in the AST can drastically impact the memory requirements of the overall formula.

4.4.1 Heuristic-based Topological Optimization Algorithm

To explain a simple approach to optimizing such ASTs, refer to Figure 4.1 where w_i, b_i are the *wpd* and *bpd* of each φ_i respectively. Assume that the operands are ordered by *wpd*, i.e., $w_i \geq w_j$ for all $i, j \in [1, n]$ such that $i > j$. Using Equation 2.1, the memory requirement of structure **A** is

$$(w_n - b_1) + (w_n - b_2) + \cdots + (w_n - b_{n-1}) + (w_{n-1} - b_n)$$

and for structure **C** is

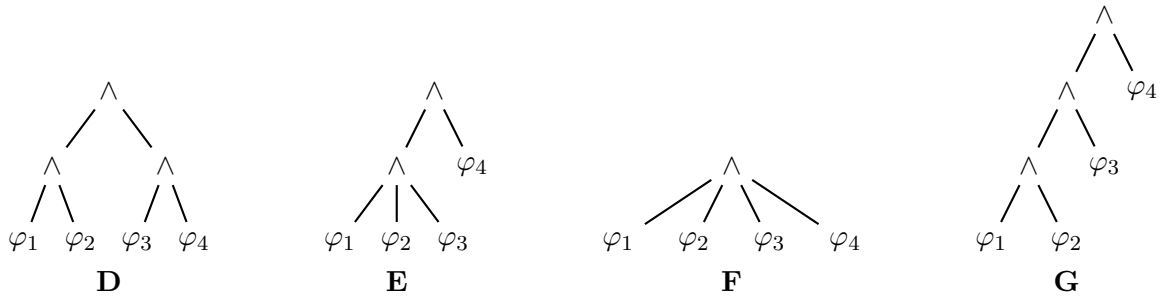


Figure 4.2 Given a formula $(\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4)$ where φ_1 to φ_4 are ordered by *wpd*, there are four relevant structures that may be optimal with respect to each formulas' *wpd*.

$$(w_{n-1} - b_1) + (w_{n-1} - b_2) + \cdots + (w_{n-1} - b_{n-2}) + (w_{n-2} - b_{n-1}) + (w_{n-1} - b_n) + (w_n - min(b_1, \ldots, b_{n-1})).$$

The memory difference between these two structures (i.e., $mem_{AST}(\mathbf{A}) - mem_{AST}(\mathbf{C})$) is thus

$$\Delta = (n-2)(w_n - w_{n-1}) - w_{n-2} + min(b_1, \ldots, b_{n-1}).$$

We can algorithmically lower the memory requirements for the encoding of an associative operator with n operands by first flattening the conjunction to obtain an **A** structure, then recursively restructuring the AST of the operator by:

$$(\varphi_1 \wedge \cdots \wedge \varphi_n) \mapsto ((\varphi_1 \wedge \cdots \wedge \varphi_{n-1}) \wedge \varphi_n) \mapsto \ldots$$

as in Figure 4.1, stopping when $\Delta \leq 0$ or $n = 2$. This heuristic-based technique does not guarantee the optimal formula encoding but may decrease its size nonetheless.

4.4.2 Single Expression Topology Analysis

In searching for an algorithm that finds the optimal topology for associative, multi-arity operators, we performed an exhaustive analysis of a subset of such operators. In particular, we consider a set of AST structures for the formula $(\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4)$ where each sub-formula φ_i has a constant $bpd = 0$ and, without loss of generality, is sorted by increasing *wpd*, i.e.,

$wpd(\varphi_1)$	$wpd(\varphi_2)$	$wpd(\varphi_3)$	$wpd(\varphi_4)$
1	1	1	37
1	1	2	36
...			
1	1	12	26
1	2	2	35
1	2	3	34
...			
1	11	14	14
2	2	2	34
2	2	3	33
...			
9	10	10	11
10	10	10	10

Table 4.2 The set of *wpd* assignments considered for each φ_i in $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ can be sorted in lexicographic order. We also consider sorting the *wdp* assignments by standard deviation, breaking ties via lexicographic ordering.

$wpd(\varphi_1) \leq \cdots \leq wpd(\varphi_4)$. With that restriction, Figure 4.2 depicts all four relevant structures, with labels ranging from **D**-**G**. No other structure will result in less required SCQ slots than these.

We explored all possible assignments of *wpd* for each node, respecting the nodes' ordering, where the *wpd*s of all nodes sum to 40, given in Table 4.2. We then computed the total SCQ costs for each configuration in Figure 4.2. Finally, in order to determine if there was some trend in those costs that we can use for computing/estimating the structure with minimal SCQ cost, we plotted each cost for each configuration and *wpd* assignment in both lexicographic order and in order of increasing standard deviation. The motivation behind using standard deviation is that assignments with similar standard deviations may cost the least with similar structures. For example, the first assignment in Table 4.2 may favor an **F** structure (which has a very high standard deviation), a middle assignment may favor a **D** assignment, and the last assignment a **G** structure (which has a very low standard deviation).

We found that of the 478 possible assignments, 9 (1.88%) were optimal with **D**, 185 (38.70%) with **E**, 107 (22.38%) with **F**, and 177 (37.03%) with **G**. Importantly, the 9 assignments that

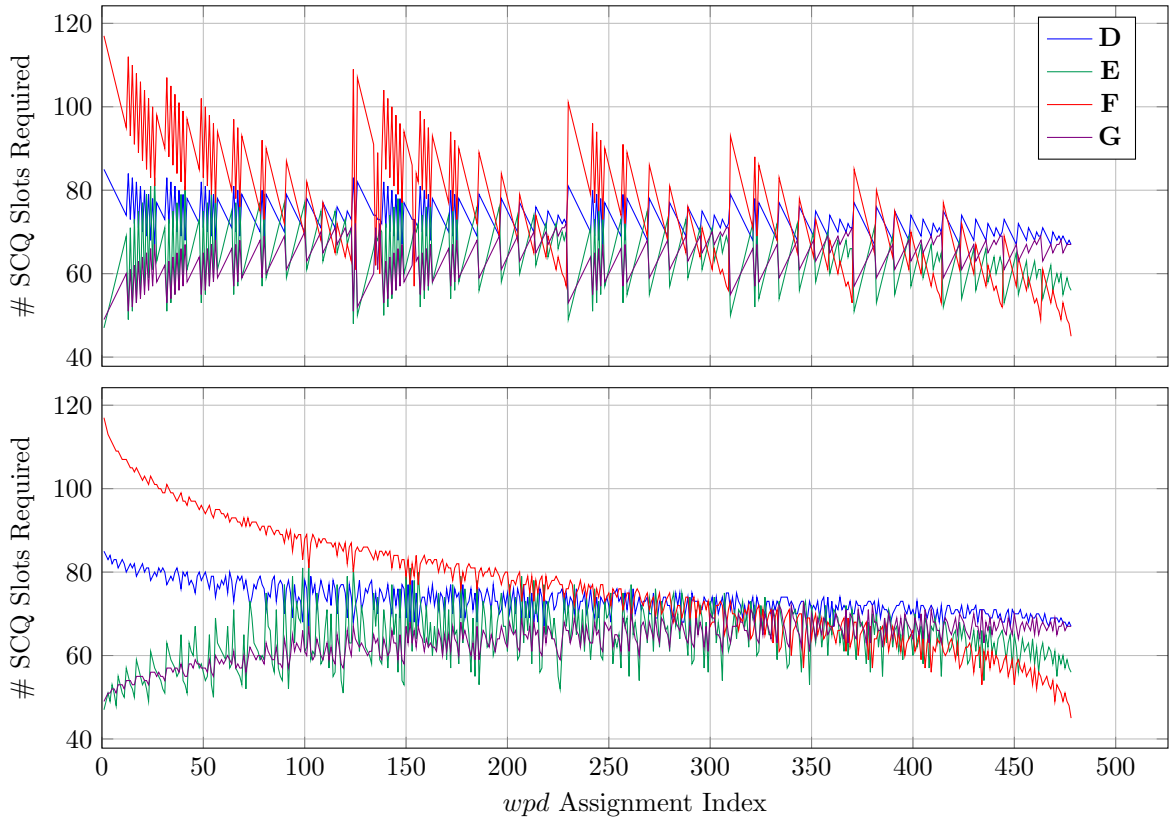


Figure 4.3 (Top) The SCQ cost of each structure in Figure 4.2 where the *wpd* assignment index is based off the ordering in Table 4.2. There appears to be some trend in finding the structure with minimal SCQ cost for any assignment but with some other non-standard lexicographic ordering. (Bottom) The SCQ cost of each structure in Figure 4.2 where the *wpd* assignment index is ordered based on the standard deviation of the value in Table 4.2. Standard deviation provides a better heuristic than the top figure but does guarantee a minimal cost structure. For example, in the 0-50 index range, both **E** and **G** are of minimal cost.

claimed optimality with **D** would also be optimal with **G**, implying that **D** is not a relevant structure for an operator with four operands.

As Figure 4.3 shows, the lexicographic ordering does not provide an effective metric for estimating the optimal structure – though the graph does have some structure, implying that another (non-standard) lexicographic ordering may better suit this task. The ordering based on standard deviation better estimates of the optimal structure. There are still outliers in the graph at many points. However, we can potentially use the standard deviation of the *wpd*s of all operands as a heuristic to estimate the minimal cost structure.

While such heuristics are a valuable exploration of the problem of optimizing the encoding for associative MLTL operators, we present an approach in the following section that computes the actual minimal structure in general.

4.5 Optimizing MLTL Monitor Encodings via Equality Saturation

We use equality saturation (Section 2.4) in the context of MLTL monitor optimization to algorithmically rewrite a formula and find its optimal equivalent representation provided the rewrite rules defined in Section 4.1.

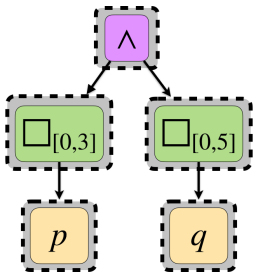
One important note is that EqSat subsumes the previously discussed optimization techniques of CSE and Topological Optimization. In other words, EqSat automatically performs CSE and Topological Optimizations. As opposed to the heuristic-based approach presented in Section 4.4.1, this technique computes the optimal encoding for associative, multi-arity operators.

4.5.1 MLTL Equality Saturation Example

To illustrate MLTL EqSat, we will walk through how to use e-graphs to represent MLTL formulas and apply rewrites to e-graphs. Consider the formula

$$\square_{[0,3]} p \wedge \square_{[0,5]} q.$$

The first step in MLTL EqSat is to construct the e-graph for this formula, which we can do using the formula's AST, where every node of the AST is an e-node (color boxes), and every e-node is assigned its own e-class (grey, dashed boxes):

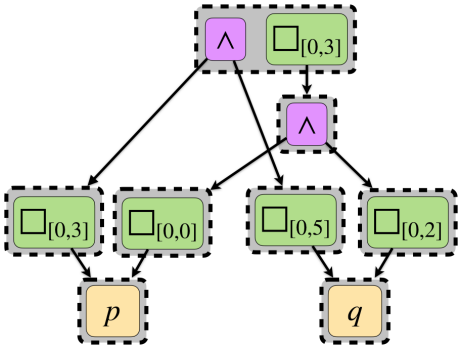


Then, we apply as many rewrites to the e-graph as possible until no more can be applied. Each rewrite will alter the e-graph by adding new e-nodes and e-classes as needed and merging e-classes

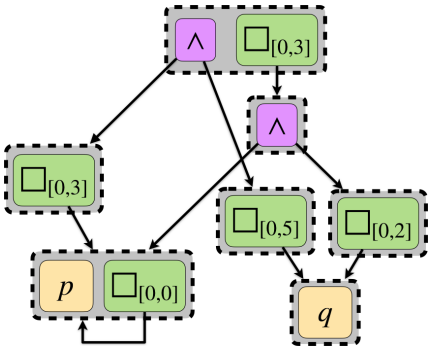
as new equivalence relations are found. Given the initial e-graph, we can apply the rewrite

$$\Box_{[0,3]}p \wedge \Box_{[0,5]}q \quad \mapsto \quad \Box_{0,3}(\Box_{[0,0]}p \wedge \Box_{[0,2]}q) \quad (R2)$$

since the root of the e-graph matches the pattern $\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi$. The resulting e-graph will require three new e-nodes, one for each new syntactically distinct sub-formula ($\Box_{0,3}(\Box_{[0,0]}p \wedge \Box_{[0,2]}q)$, $\Box_{[0,0]}p$, and $\Box_{[0,2]}q$). Because we know that the two formulas are equivalent, we place the root of the rewritten formula in the same e-class as the root of the original formula:



Next, we apply the rewrite $\Box_{[0,0]}p \mapsto p$ since the $\Box_{[0,0]}$ e-node matches the pattern $\Box_{[0,0]}\varphi$. The resulting e-graph requires no new e-nodes since the e-graph already contains all of the e-nodes from the original and rewritten formulas. Instead, we merely merge the e-classes of the $\Box_{[0,0]}$ and p e-nodes:



The self-loop from the $\Box_{[0,0]}$ encodes the infinite equivalence relation:

$$p \equiv \Box_{[0,0]}p \equiv \Box_{[0,0]}\Box_{[0,0]}p \equiv \ldots$$

The e-graph is now *saturated*; subsequent rewrites would not alter the e-graph. This e-graph, therefore, represents the set of all equivalent MLTL formulas derivable from the supplied rewrite rules.

4.5.2 MLTL Equality Saturation with `egglog`

The provided example does not address the most difficult aspects of EqSat, namely invariant maintenance and pattern matching. `egglog` Zhang et al. (2023) is a tool that performs EqSat and addresses both of these problems. We therefore encode the presented MLTL rewrite rules and formulas into `egglog` to perform EqSat automatically. The output of `egglog` is a saturated e-graph, from which we manually extract the formula with the lowest number of required SCQ slots.

`egglog` is a tool and language that combines Datalog Ceri et al. (1989), a logic-programming-style database query language, with e-graphs. To encode an MLTL formula into `egglog`, we define a language that supports each operator and then define each rewrite rule as in Table 4.1. Importantly, we support multi-arity conjunction and disjunction in our optimizations to address Section 4.4, but only up to an arity of 4. A snippet of this encoding can be found in Figure 4.4. Figure 4.5 depicts the result of EqSat using the `egglog` web demo[4]. All this is implemented in `C2PO` as an optimization pass, as depicted in Figure 4.6.

4.5.3 Optimal Encoding Extraction

`egglog` only supports extracting optimal expressions from a saturated e-graph if the e-nodes have a constant cost function, i.e., each operator has a constant cost. This feature makes sense in some applications like program optimization, where multiplication may be a constant factor more expensive than addition, for example.[5]

[4] https://egraphs-good.github.io/egglog/

[5] We note here that the tool that `egglog` builds off of, `egg` Willsey et al. (2021), provides a feature called *analyses* that enables non-constant cost assignment. However, `egg` is implemented in Rust and would have required deeper code integration with `C2PO` than merely encoding the problem into an easy-to-use input language.

```
(sort IntervalSort)
(function Interval (i64 i64)
                   IntervalSort)

(datatype MLTL
  (Bool bool)
  (Var String)
  (Not MLTL)
  (Implies MLTL MLTL)
  (Equiv MLTL MLTL)
  (And2 MLTL MLTL)
  (And3 MLTL MLTL MLTL)
  (And4 MLTL MLTL MLTL MLTL)
  (Or2 MLTL MLTL)
  (Or3 MLTL MLTL MLTL)
  (Or4 MLTL MLTL MLTL MLTL)
  (Global IntervalSort MLTL)
  (Future IntervalSort MLTL)
  (Until IntervalSort MLTL MLTL)
)
```

```
; Constant propagation
(rewrite
  (And2 (Bool true) a)
  a
)

; R1 Global
(rewrite
  (Global (Interval l1 u1)
      (Global (Interval l2 u2) a))
  (Global (Interval (+ l1 l2)
                    (+ u1 u2)) a)
)

; R5 Global
(rewrite
  (Or2 (Global (Interval l1 u1) a)
       (Global (Interval l2 u2) a))
  (Global (Interval l2 u2) a)
  :when ((>= l2 l1) (<= u2 u1))
)
```

Figure 4.4 (Left) The presented `egglog` encoding defines a datatype `MLTL` to describe the language of MLTL. (Right) Each rewrite rules requires one expression to pattern match against, another to rewrite a matched expression to, and an optional condition that must be met in order to apply the rule.

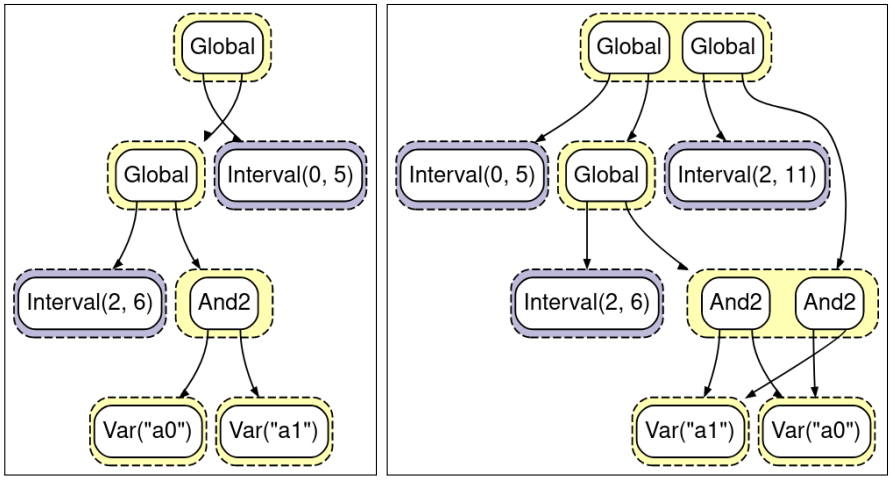


Figure 4.5 The web demo for `egglog` provides a visualization for how the tool performs EqSat. (Left) The initial e-graph for the MLTL formula $\square_{[0,5]}\square_{[2,6]}(a_0 \wedge a_1)$ has e-nodes for each node in the AST as well as different sorted e-nodes for each interval. (Right) After performing EqSat by applying (R5) and flipping the operands of the $\wedge$, the e-graph includes new e-nodes for the MLTL formulas $\square_{[2,11]}(a_0 \wedge a_1)$ and $a_1 \wedge a_0$.

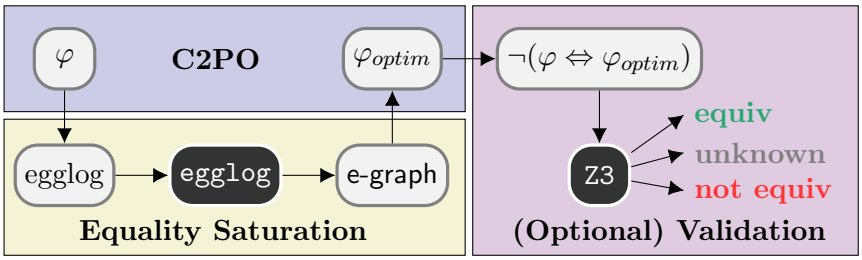


Figure 4.6 Grey boxes denote data and black boxes denote tools. Starting with an MLTL formula φ, `C2PO` encodes this into an `egglog` query à la Figure 4.5. `egglog` then produces a JSON representation of the saturated e-graph, from which `C2PO` extracts an optimized formula φ_{optim}. `C2PO` then optionally encodes this into an `SMTLIB2` query and uses `Z3` to check if $\neg(\varphi \Leftrightarrow \varphi_{optim})$ is satisfiable and therefore equivalent or not.

In our case, the cost of e-nodes is the total number of required SCQ slots for an e-node and its children, which is not constant. To perform the extraction, we find the root of the e-graph, compute the propagation delays of each e-node and remove those with non-tight delays, then the cost of each e-node, then find the minimal-cost e-node in each e-class, then finally extract the optimized expression by choosing the minimal cost e-node from each e-class starting from the root of the e-graph.

Root e-node: One oddity of using `egglog` is that the resulting representation of the saturated e-graph does not include information on the root e-class, i.e., the e-class that includes the root node of the original input. Therefore, to find this e-class, we manually pattern match all e-nodes in the e-graph against the original AST to find the e-node representing the root.

Propagation Delays: Due to rule (R5), not all e-nodes in an e-class have equal propagation delays for their representative formulas. In other words, not all equivalent MLTL formulas have the same propagation delays. Therefore, we compute the propagation delay of each e-node and remove any e-node with a *bpd* lower than the minimal *bpd* in the e-class or with a *wpd* higher than the maximal *wpd* in the e-class. The resulting e-graph is the same as before, except with only the e-nodes with the tightest propagation delays in their respective equivalence classes. The removed nodes will not be of minimal cost due to Lemmas 1 and 2.

e-node Costs: We compute the cost of each e-node by traversing the e-graph (leaf-to-root) and use each e-class' propagation delays to compute how many SCQ slots each of a node's children require. Therefore, each node's cost is not how many SCQ slots it requires but how many its children require. This representation is sufficient since the root e-node does not have any siblings.

e-class Representatives: The representative for an e-class is the e-node with minimal cost, which we compute in a single traversal of the e-graph.

Final Extraction: To extract the final, optimized formula, we recurse down the e-graph from the root node, constructing an MLTL formula from the representative of each e-class and its children.

4.6 Experimental Evaluation

For evaluating the presented technique, we collected a set of benchmarks of MLTL formula, compiled from various sources. These benchmarks are useful as MLTL benchmarks for problems including monitoring or satisfiability. The benchmark sets are:

- **Random-0 Li et al. (2019):** 500 randomly-generated MLTL formulas with 0-rooted temporal operators.
- **Random Li et al. (2019):** 10,000 randomly-generated MLTL formulas.
- **Boeing WBS Bozzano et al. (2015)Li et al. (2019)**: 147 MLTL formulas derived from a set of real-world LTL formulas for verifying a Boeing Wheel Brake System (WBS) design with random interval bounds placed on each temporal operator.
- **NASA ATC Gario et al. (2016)Li et al. (2019):** 36 MLTL formulas derived from a set of real-world LTL formulas for verifying a NASA Air-Traffic Control design with random interval bounds placed on each temporal operator.
- **UTM Cauwels et al. (2020):** 124 real-world MLTL formulas from a set of MLTL specifications for an automated UAS Traffic Manager. Most formulas are either non-temporal or follow a simple structure, such as $\Box_{[0,3]}(p \wedge q)$.

Benchmark	# Formulas	# Proven Equiv	Avg % Reduction	Avg Optim Time
Random-0	500	177	8.54%	0.02s
Random	10,000	3077	8.64%	0.02s
Boeing WBS	147	68	34.55%	148.25s
NASA ATC	36	26	23.90%	0.28s
UTM	124	124	2.90%	0.02s
FMSD17	7	6	13.95%	0.02s
RV14	6	6	5.72%	0.02s

Table 4.3 Experimental results show a noticeable reduction in SCQ slots for each benchmark and reasonable average time to perform EqSat. EqSat resulted in no timeouts. Not all optimized formulas were proven equivalent, but all other formulas were deemed "unknown" by Z3. More human-authored and semi-synthetic formulas were proven equivalent than randomly-generated.

Benchmark	# Formulas with % SCQ Reduction										
	<1	**<10**	**<20**	**<30**	**<40**	**<50**	**<60**	**<70**	**<80**	**<90**	**<100**
Random-0	59	298	96	33	5	4	3	0	0	2	0
Random	1604	4884	2680	651	120	29	14	8	2	0	5
Boeing WBS	12	5	0	2	79	40	6	3	0	0	0
NASA ATC	6	1	5	9	14	0	1	0	0	0	0
UTM	107	3	6	6	1	1	0	0	0	0	0
FMSD17	3	1	1	1	0	0	0	0	0	0	0
RV14	4	1	0	0	1	0	0	0	0	0	0

Table 4.4 Each column represents a bucket of formulas from each benchmark that was reduced by that amount. For example, the column " <20" denotes the number of formulas that were reduced by more than 10 but less than 20 percent. The highlighted cells showcase that some formulas were reduced by as much as 50-99%.

- **FMSD17 Moosbrugger et al. (2017):** 7 real-world MLTL formulas specifying behavior to catch dangerous MAVLink Protocol commands and a Denial of Service hijack attack.
- **RV14 Geist et al. (2014):** 6 real-world MLTL formulas designed to monitor for a fluxgate magnetometer buffer overflow error on an in-flight UAS.

When running the optimizations on the benchmarks, we computed the required SCQ slots of the original and optimized formulas and performed validation by checking that both the original and optimized formulas were equivalent. We checked equivalence using the first-order logic encoding of Li et al. (2019) and checking that the formula $\neg(\varphi \Leftrightarrow \mathit{optim}(\varphi))$ was unsatisfiable

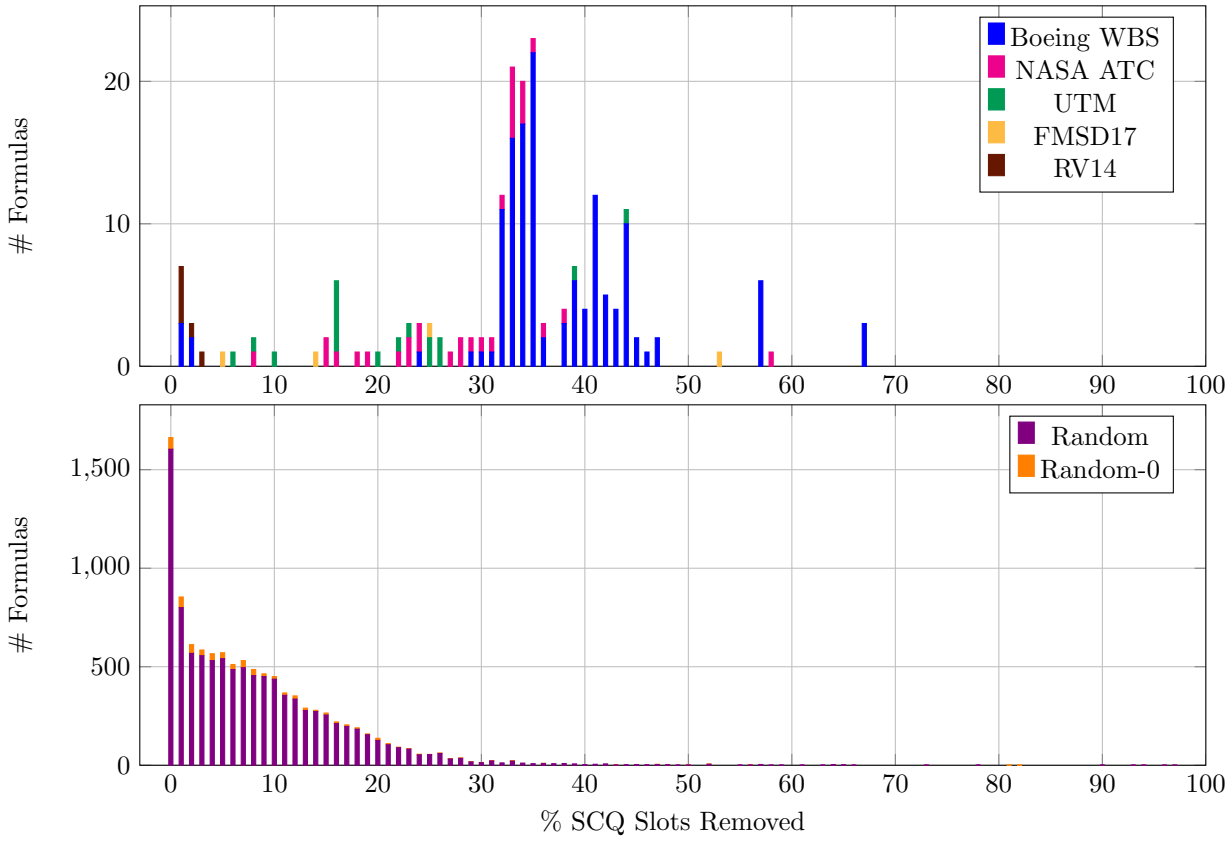


Figure 4.7 A visualization of Table 4.4 shows how much was saved in more detail, with the x-axis representing percentage of SCQ slots reduced (rounded to nearest percentage) and y-axis representing the number of formulas with that percentage. (Top) The graph semi-synthetic and human-authored formulas **does not include formulas with <1% reduction for clarity**. (Bottom) Both 0-rooted and fully randomly-generated formulas follow the same distribution as one another with the majority of formulas seeing at least some savings.

using the SMT solver Z3 De Moura and Bjørner (2008) with a timeout of one hour (3600s). This equivalence check corresponds to the "(Optional) validation" box of Figure 4.6. In all problem instances, we found a reduction in required SCQ slots and either found that the formulas were equivalent or unknown due to timeout or Z3 terminating early. As a result of using the encoding from Li et al. (2019), we did not enable the rewrites (R3), (R6), or (R8) since they do not hold in the standard MLTL semantics.

The results show that both randomly generated benchmark sets see an average percent reduction in SCQ slots of 8.5%. The Boeing WBS and NASA ATC benchmarks see a much

higher average savings of 34.5% and 23.90%, respectively. For the human-authored sets, note that the sample sizes are relatively small (FMSD17, RV14) or contain many copies of the same formula pattern (UTM).

These results are promising as we expect real-world specifications to adhere to patterns that equality saturation can leverage. In an embedded, resource-constrained environment, any amount of memory savings is critical and can be the difference between some specifications fitting on-board or not. On real-world specifications, this automated technique reduces memory requirements for encoding MLTL monitors.

CHAPTER 5. CONFIGURATION COMPILER FOR PROPERTY ORGANIZATION[1]

The Configuration Compiler for Property Organization (C2PO) is the standard formula compiler for R2U2 Johannsen et al. (2023a). C2PO encodes a set of MLTL formulas specified in a custom input language into a R2U2-compatible binary format. Much like other compilers, C2PO takes in a file, performs input validation, parses the file contents, generates a corresponding AST, type checks the AST, performs a series of passes over the AST, then assembles the final AST into an R2U2-compatible binary. Figure 5.1 depicts this pipeline.

5.1 Input Language

C2PO's input language composes several sections for defining structures, declaring input signals and their types, macro-style definitions, defining atomic checker components, and defining future- and past-time specifications. Figure 5.2 contains an example file that uses most of these capabilities. We highlight features of the language here.

[1] Adapted from Johannsen et al. (2023a)

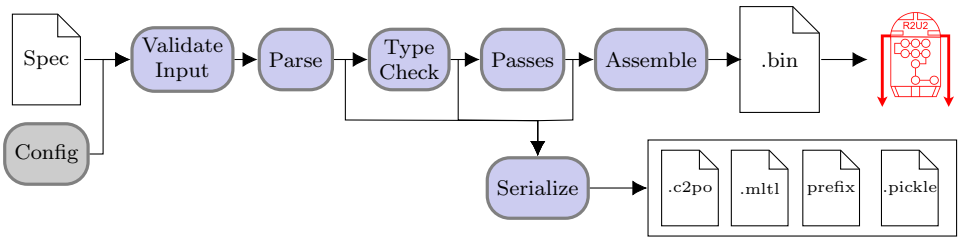


Figure 5.1 Given a specification (in a .c2po or .mltl file) and a configuration, C2PO validates the configuration, parses, and type checks the specification, applies the passes in Figure 5.1, and assembles the final R2U2-compatible binary. C2PO can exit and/or serialize its input after parsing, type checking, or passes.

```
1  STRUCT
2    Request: { state: int; time_active: float; };
3    Arbiter: { ReqSet: set<Request>; };
4
5  INPUT
6    state_0, state_1, state_2, state_3: int;
7    time_active_0, time_active_1, time_active_2, time_active_3: float;
8
9  DEFINE
10   Wt := 0; Gr := 1; Rj := 2; -- wait, grant, reject
11
12   req_0 := Request(state_0, time_active_0);
13   req_1 := Request(state_1, time_active_1);
14   req_2 := Request(state_2, time_active_2);
15   req_3 := Request(state_3, time_active_3);
16
17   Arb0 := Arbiter({req_0, req_1});
18   Arb1 := Arbiter({req_2, req_3});
19   ArbSet := {Arb0, Arb1};
20
21 FTSPEC
22   (req_0.time_active - req_1.time_active) < 10.0 &&
23   (req_1.time_active - req_0.time_active) < 10.0;
24
25   foreach(arb: ArbSet)(
26    foreach(req: arb.ReqSet)(
27     (req.state == Wt) U[0,5] (req.state == Gr || req.state == Rj)
28    )
29   );
```

Figure 5.2 An example C2PO specification file uses structs (lines 2 – 3, 12 – 13), sets (lines 3, 15 – 16), and set aggregation operators (lines 22 – 23) to formalize the English requirements, "The active times for req_0 and req_1 shall differ by no more than 10.0 seconds," (lines 19 – 20) and, "For each request req of each arbiter in $ArbSet$, req's status shall be Grant or Reject within the next 5 seconds and, until then, shall be Waiting" (lines 25-29)

Types: C2PO supports Boolean, contract, user-definable struct, parametric set, and integer and floating point types with configurable widths.

Atomic Checkers: Atomic Checkers are a construct for generating Boolean values from sensor data by applying a filter to some signal and comparing the filter output to some constant. C2PO support the definition of such construct via the ATOMICS section, for example:

```
ATOMIC
    a0 := int(temp) <= 5;
    a1 := rate(speed) < 100;
```

Bitwise	Arithmetic	Relational	Logical	Temporal
`&`, `|`, `^`, `~`, `<<`, `>>`	`+`, `-`, `*`, `/`, `%`	`==`, `!=`, `>`, `<`, `>=`, `<=`	`&&`, `||`, `xor`, `->`, `<->`	`F`, `G`, `U`, `R`

Figure 5.3 C2PO supports the operators above, where the Booleanizer computes the operators in red cells and the Temporal Logic engine computes those in blue cells.

Atomic Checkers are particularly useful in hardware implementations of R2U2 but are also supported by the C implementation.

Booleanizer Expressions: C2PO also supports a more powerful R2U2 engine for computing Boolean values called the Booleanizer. This engine allows for arbitrary expressions over integer and floating point types with any of the operations listed in Figure 5.3.

Assume-Guarantee Contracts (AGCs): Assume Guarantee Contracts (AGCs) provide a template for structuring and validating complex requirements in aerospace operational concepts Badger et al. (2019). AGCs feature a guard or trigger clause called the "assumption" and a system invariant called the "guarantee;" they have been used to structure both English and formal (e.g., temporal logic) requirements by projects, including the NASA Lunar Gateway Vehicle System Manager Dabney et al. (2022).

C2PO supports an input syntax for expressing AGCs, as presented in Kempa et al. (2022). The input syntax for declaring an AGC is `'assumption' => 'guarantee'` where the semantics provides three distinct cases: the AGC is "inactive" if the assumption is false, "true" if both the assumption and guarantee are "true", and "false" otherwise.

Set Aggregation: A typical pattern in real-world specifications applies an identical formula to various input signals, such as testing all temperature sensors for an overheating condition. A naive encoding of these specifications in MLTL can be large to the point of obscuring intent while providing ample opportunity for copy-paste errors, typos, or incomplete updates to variables – which are difficult for humans to spot during validation. C2PO mitigates this issue by supporting set aggregation operators that compactly encode these expressions as sets of streams with a predicate applied to each element Hammer et al. (2021).

To illustrate, consider the specification in Fig. 5.2. The direct encoding of this specification without the "foreach" operator is

```
(rq0.status == W) U[0,5] (rq0.status == G || rq0.status == R) &&
(rq1.status == W) U[0,5] (rq1.status == G || rq1.status == R) &&
(rq2.status == W) U[0,5] (rq2.status == G || rq2.status == R) &&
(rq3.status == W) U[0,5] (rq3.status == G || rq3.status == R)
```

Contrast this with the more compact encoding using the `foreach` operator on lines $22 - 26$ in Fig. 5.2. The latter retains the intent of the English-level requirement while being semantically equivalent to the direct encoding. This concise representation eases validation by improving readability and reduces the potential for errors by avoiding replicated values that require simultaneous updates.

5.2 Passes

C2PO performs its core capabilities via *passes* of its AST, just like many other compilers. Figure 5.1 provides a list of all supported passes, which passes each is incompatible with, and a description of each pass.

5.3 Serialization

C2PO supports many formats for serializing specifications. These formats support easier debugging and usage of outside tools. Supported formats include SMT (for SAT queries) Li et al. (2019), `egglog` (Section 4.1), the MLTL Standard, C2PO input, prefix-notation C2PO, and Python's `pickle` format.

The MLTL Standard format is used by other tools, including WEST Elwing et al. (2023) and previous versions of MLTL-based tools Kempa et al. (2020); Li et al. (2019). This format is relatively restrictive; users define MLTL formulas one per line. An example specification set in this format could look like:

```
G[0,10] a0
```

ID	Pass Name	Incomp. Passes	Description
1	Expand Definitions		Perform macro-expansion of definition symbols.
2	Convert Function Calls		Convert function calls to struct instantiations.
3	Resolve Contracts		Replaces contracts with MLTL formulas for each output value.
4	Unroll Set Aggregation		Replaces set aggregation operators with equivalent MLTL expression.
5	Resolve Struct Accesses		Resolves struct access operations to underlying member expression.
6	Compute Atomics		Compute atomic proposition expressions for AST.
7	Apply Rewrite Rules	9, 10	Perform single-pass rewrite rule optimization.
8	Equality Saturation	9, 10	Perform equality saturation.
9	To NNF	7, 8, 10	Convert output to negation normal form.
10	To BNF	7, 8, 10	Convert output to Boolean normal form.
11	Remove Extended Operators	9, 10	Reduces to minimum set of MLTL operators (p, $\neg$, $\vee$, $\mathcal{U}$)
12	Multi-Arity to Binary		Converts all multi-arity operators ($\wedge$, $\vee$) to binary variants.
13	CSE		Perform Common Sub-expression Elimination.
14	Check Satisfiable		Check if formulas are satisfiable using SMT encoding.
15	Compute SCQs		Compute SCQ size for each AST node.

Table 5.1 C2PO's compilation pipeline contains the 15 passes above, presented in order of execution. Each row defines for each pass its name, which other passes it is incompatible with, and a high-level description. Red rows denote required passes, green rows denote optional passes, and blue rows denote optimizations. Unlike rewrite-based optimizations, CSE does not alter the syntax of the encoded MLTL, so passes that rely on a syntactic structure like NNF can also undergo CSE.

```
a1 U[0,15] (a0 & a1)

(F[0,3] a2) R[0,5] a3
```

As compared to C2PO's input language, this format lacks support for non-Boolean types and other convenience features discussed previously but are useful for purely machine-readable benchmarks.

C2PO can also output its input language, which helps translate between the MLTL Standard and C2PO, or for debugging between compiler passes. Similarly, prefix-notation C2PO is useful when debugging an arity-based optimization like that presented in Section 4.4.

Python's `pickle` format serializes the internal data structures that represent an MLTL AST and its current state during compilation, which enables AST property comparison across multiple runs through the compiler, where each run's data must be persistent. For example, a developer

small.c2po

```
INPUT
  b0: bool;
  i0: int;

FTSPEC
  G[1,2] (i0 > 5) &&
    F[0,5] b0;
```

small.map

```
b0:0
i0:1
```

small.asm

```
BZ b0  iload  1
BZ b1  iconst 5
BZ b2  igt    0 1 a0
BZ b3  iload  0 a1
TL n0  load   a0
TL n1  global n0
TL n2  load   a1
TL n3  until  True n2
TL n4  and    n1 n3
TL n5  return n4 0
CG TL SCQ n0  (0, 3)
CG TL SCQ n1  (3, 10)
CG TL LB  n1  1
CG TL UB  n1  2
CG TL SCQ n2  (10, 13)
CG TL SCQ n3  (13, 18)
CG TL LB  n3  0
CG TL UB  n3  5
CG TL SCQ n4  (18, 21)
CG TL SCQ n5  (21, 22)
```

Figure 5.4 `C2PO` generates `small.asm` given the command `python c2po.py --map small.map small.c2po`. The map file is necessary for `R2U2` to load the correct values from the signal buffer during runtime. Without the `--extops` option enabled, the `F` operator is converted to a `U` operator as node `n3` in the assembly.

can run `C2PO` on a specification with a new optimization turned on and off, pickling their ASTs, then compare the ASTs' properties, such as their sizes, required SCQ slots, or worst-case computation time.

5.4 Assembly

Finally, `C2PO` assembles the final AST into an `R2U2`-compatible binary. `R2U2`'s binary format largely follows the structure of the AST already, so `C2PO` merely performs a topological sort of the AST and generates an instruction for each node. Note that due to CSE we cannot perform a tree traversal since the AST is not a tree, i.e., CSE may combine nodes and, therefore, a node may have multiple parents.

Each AST node type has an instruction type, grouped into one of four categories: an Atomic Checker, Booleanizer, Temporal Logic, or Configuration. We have already discussed the Atomic

Checker, Booleanizer, Temporal Logic types. Configuration instructions define the memory layout for SCQs and temporal operators, i.e., how much memory each node in the AST requires and where that memory should be. Figure 5.4 shows a small example, where the assembly on the right includes Booleanizer (BZ), Temporal Logic (TL), and Configuration (CG) instructions. A `.map` file defines where each input signal will be in the signal buffer for `R2U2` during runtime.

CHAPTER 6. CONCLUSION

The techniques presented offer a novel way of expressing, optimizing, and monitoring MLTL formulas that reason over bounded dynamic sets. In doing so, specification authors can be more confident that the formalizations of their requirements are correct. Further, by using the MLTL monitor optimizations, practitioners can automatically reduce the size of their MLTL monitors in R2U2. This automated technique may allow users to fit more specifications on board without manually tuning their formulas' syntax. The interface and automation provided by C2PO make these techniques more approachable than without C2PO, including by allowing users to specify their requirements in an intuitive, high-level language.

Soon, we plan to implement the $\overline{\text{FO-MLTL}}$ monitoring algorithms into R2U2 and provide an easy-to-specify interface via C2PO. We will also explore the possibility of encoding $\overline{\text{FO-MLTL}}$ formulas to MLTL, which would allow the usage of MLTL-based techniques more broadly, such as MLTL-SAT Li et al. (2019) and benchmark generation Li and Rozier (2018). Such an encoding could place a syntactic restriction on $\overline{\text{FO-MLTL}}$ formulas, where objects in dynamic sets remain in their set until each relevant temporal sub-formula is finished evaluating, similar to the encoding provided in Johannsen et al. (2023b). This restriction may relax the requirement to store auxiliary results for the quantifier monitor.

There is also an opportunity to define more rewrite rules, especially using $\mathcal{U}$. More rules could inspire rewrites into other metric temporal logics. The impact of the rewrites on other properties of MLTL formulas, like satisfiability checking, bounded model checking, and an automata-based encoding size would also be of interest.

Further MLTL optimizations may be had from combining the presented approach with assumption-based runtime verification Cimatti et al. (2022). Assumptions could allow us to perform vacuity checking and reduce sub-formulas that are never exercised.

The equality saturation optimizations also only work with single-formula specifications, and we plan to extend this capability to multi-formula specifications soon. This ability would make MLTL EqSat faster, where an e-graph can reuse rewritten expressions from other formulas instead of performing a separate equality saturation computation for both formulas separately.

We will also explore connecting C2PO to WEST Elwing et al. (2023), a tool for exploring satisfying trace sof MLTL formulas using regular expressions, and FPROGG, a tool implementing the MLTL benchmark-generating algorithms in Li and Rozier (2018). These connections will create opportunities for new workflows in the design process for specification authors, allowing users to explore the traces that satisfy a set of specifications and generate particular traces of interest.

BIBLIOGRAPHY

Andréka, H., Németi, I., and Van Benthem, J. (1998). Modal languages and bounded fragments of predicate logic. *Journal of philosophical logic*, 27:217–274.

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

Badger, J. M., Strawser, P., and Claunch, C. (2019). A distributed hierarchical framework for autonomous spacecraft control. In *2019 IEEE Aerospace Conference*, pages 1–8. IEEE.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Barringer, H., Falcone, Y., Havelund, K., Reger, G., and Rydeheard, D. (2012). Quantified event automata: Towards expressive and efficient runtime monitors. In *FM 2012: Formal Methods: 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings 18*, pages 68–84. Springer.

Basin, D., Klaedtke, F., Müller, S., and Pfitzmann, B. (2008). Runtime monitoring of metric first-order temporal properties. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Basin, D., Klaedtke, F., Müller, S., and Zălinescu, E. (2015). Monitoring metric first-order temporal properties. *Journal of the ACM (JACM)*, 62(2):1–45.

Bauer, A., Küster, J.-C., and Vegliach, G. (2015). The ins and outs of first-order runtime verification. *Formal Methods in System Design*, 46(3):286–316.

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

Brat, G., Navas, J. A., Shi, N., and Venet, A. (2014). Ikos: A framework for static analysis based on abstract interpretation. In *Software Engineering and Formal Methods: 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings 12*, pages 271–277. Springer.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318.

Cauwels, M., Hammer, A., Hertz, B., Jones, P., and Rozier, K. Y. (2020). Integrating Runtime Verification into an Automated UAS Traffic Management System. In *Proceedings of DETECT: international workshop on moDeling, vErification and Testing of dEpendable CriTical systems*, Communications in Computer and Information Science (CCIS), L'Aquila, Italy. Springer.

Ceri, S., Gottlob, G., Tanca, L., et al. (1989). What you always wanted to know about datalog(and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166.

Cerrito, S., Mayer, M. C., and Praud, S. (1999). First order linear temporal logic over finite time structures. In *Logic for Programming and Automated Reasoning: 6th International Conference, LPAR'99 Tbilisi, Georgia, September 6–10, 1999 Proceedings 6*, pages 62–76. Springer.

Chen, Y., Zhang, X., and Li, J. (2022). Finite quantified linear temporal logic and its satisfiability checking. In *Artificial Intelligence Logic and Applications: The 2nd International Conference, AILA 2022, Shanghai, China, August 26–28, 2022, Proceedings*, pages 3–18. Springer.

Chomicki, J. (1995). Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Transactions on Database Systems (TODS)*, 20(2):149–186.

Cimatti, A., Tian, C., and Tonetta, S. (2022). Assumption-based runtime verification. *Formal Methods in System Design*, 60(2):277–324.

Clarke, E. M. (1997). Model checking. In *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*, pages 54–56. Springer.

Cooper, K., Eckhardt, J., and Kennedy, K. (2008). Redundancy elimination revisited. In *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 12–21.

Dabney, J. B., Rajagopal, P., and Badger, J. M. (2022). Using assume-guarantee contracts for developmental verification of autonomous spacecraft. Flight Software Workshop (FSW) Online: https://www.youtube.com/watch?v=HFnn6TzblPg.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

De Moura, L. and Bjørner, N. (2007). Efficient e-matching for smt solvers. In *Automated Deduction–CADE-21: 21st International Conference on Automated Deduction Bremen, Germany, July 17-20, 2007 Proceedings 21*, pages 183–198. Springer.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Gbaguidi Aisse, A., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., et al. (2022). From spot 2.0 to spot 2.10: What's new? In *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part II*, pages 174–187. Springer.

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

Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., and Rozier, K. Y. (2016). Model checking at scale: Automated air traffic control design space exploration. In *Proceedings of 28th International Conference on Computer Aided Verification (CAV 2016)*, volume 9780 of *LNCS*, pages 3–22, Toronto, ON, Canada. Springer.

Geist, J., Rozier, K. Y., and Schumann, J. (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)*, volume 8734, pages 215–230. Springer-Verlag.

Gurfinkel, A., Kahsai, T., Komuravelli, A., and Navas, J. A. (2015). The seahorn verification framework. In *International Conference on Computer Aided Verification*, pages 343–361. Springer.

Hammer, A., Cauwels, M., Hertz, B., Jones, P., and Rozier, K. Y. (2021). Integrating runtime verification into an automated uas traffic management system.

Havelund, K., Peled, D., and Ulus, D. (2020). First-order temporal logic monitoring with bdds. *Formal Methods in System Design*, 56(1):1–21.

Hertz, B., Luppen, Z., and Rozier, K. Y. (2021). Integrating runtime verification into a sounding rocket control system. In *Proceedings of the 13th NASA Formal Methods Symposium (NFM 2021)*. Available online at http://temporallogic.org/research/NFM21/.

Hipp, R. D. (2020). SQLite.

Jhala, R. and Majumdar, R. (2009). Software model checking. *ACM Computing Surveys (CSUR)*, 41(4):1–54.

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

Johannsen, C., Kempa, B., Jones, P. H., Rozier, K. Y., and Wongpiromsarn, T. (2023b). Impossible made possible: Encoding intractable specifications via implied domain constraints. In *International Conference on Formal Methods for Industrial Critical Systems*, pages 151–169. Springer.

Joshi, R., Nelson, G., and Randall, K. (2002). Denali: A goal-directed superoptimizer. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, PLDI '02, page 304–314, New York, NY, USA. Association for Computing Machinery.

Kempa, B., Johannsen, C., and Rozier, K. Y. (2022). Improving Usability and Trust in Real-Time Verification of a Large-Scale Complex Safety-Critical System. *Ada User Journal*, September.

Kempa, B., Zhang, P., Jones, P. H., Zambreno, J., and Rozier, K. Y. (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), pages 196–214, Vienna, Austria. Springer.

Li, J. and Rozier, K. Y. (2018). Mltl benchmark generation via formula progression. In *International Conference on Runtime Verification*, pages 426–433. Springer.

Li, J., Vardi, M. Y., and Rozier, K. Y. (2019). Satisfiability checking for mission-time ltl. In *Proceedings of 31st International Conference on Computer Aided Verification (CAV 2019)*, LNCS, New York, NY, USA. Springer.

Moosbrugger, P., Rozier, K. Y., and Schumann, J. (2017). R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems. pages 1–31.

Nelson, C. G. (1980). *Techniques for Program Verification*. PhD thesis, Stanford, CA, USA. AAI8011683.

Reinbacher, T., Rozier, K. Y., and Schumann, J. (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)*, volume 8413 of *Lecture Notes in Computer Science (LNCS)*, pages 357–372. Springer-Verlag.

Tate, R., Stepp, M., Tatlock, Z., and Lerner, S. (2009). Equality saturation: a new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 264–276.

Turing, A. M. et al. (1936). On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5.

Willsey, M., Nandi, C., Wang, Y. R., Flatt, O., Tatlock, Z., and Panchekha, P. (2021). Egg: Fast and extensible equality saturation. volume 5, pages 1–29. ACM New York, NY, USA.

Zhang, Y., Wang, Y. R., Flatt, O., Cao, D., Zucker, P., Rosenthal, E., Tatlock, Z., and Willsey, M. (2023). Better together: Unifying datalog and equality saturation. volume 7, New York, NY, USA. Association for Computing Machinery.

APPENDIX. MLTL REWRITE RULE PROOFS

Lemma 2 (*Memory Effect of Tighter WPD*)**.** *Let φ, ψ_1, ψ_2 be well-formed MLTL formulas where ψ_1 is a sub-formula of φ, ψ_2 is the sub-formula in $\varphi(\psi_1 \mapsto \psi_2)$, and $\psi_2.wpd \leq \psi_1.wpd$. Then $\varphi(\psi_1 \mapsto \psi_2)$ requires equal or lesser memory than φ when controlling for ψ_1 and ψ_2:*

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

Proof. As in the proof for Lemma 1, ψ_1, ψ_2 have the same set of siblings i.e., $\mathcal{S}_{\psi_1} = \mathcal{S}_{\psi_2} = \mathcal{S}$. First, assume $\psi_1.wpd \leq max\{s_{\psi_1}.wpd \mid s_{\psi_1} \in \mathcal{S}_{\psi_1}\}$ i.e., ψ_1 does not have the maximum *wpd* of all of its sibling nodes. Then

$$max\{s_{\psi_1}.wpd \mid s_{\psi_1} \in \mathcal{S}_{\psi_1}\} = max\{s_{\psi_2}.wpd \mid s_{\psi_2} \in \mathcal{S}_{\psi_2}\}$$

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

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

$$\psi_1.wpd > max\{s_{\psi_1}.wpd \mid s_{\psi_1} \in \mathcal{S}_{\psi_1}\}$$

Then, the amount of memory required for each node $s_{\psi_1} \in \mathcal{S}_{\psi_1}$ is

$$mem_{node}(s_{\psi_1}) = max(\psi_1.wpd - s_{\psi_1}.bpd, 0).$$

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

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

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

Then, the sibling nodes of each ancestor of ψ_2 will have a lower or equal memory requirement than the sibling nodes of each ancestor of ψ_1. The proof follows. □

Theorem 4 (Equivalence of MLTL Rewrite Rules)**.** *Let $\varphi, \psi, \varphi_1, \varphi_2, \varphi_3$ be well-formed MLTL formulas and a, l, u, l_1, u_2, l_2, u_2, l_3, $u_3 \in \mathbb{N}_0$ such that $l \leq u$, $l_1 \leq u_1$, $l_2 \leq u_2$, $l_3 \leq u_3$. Then, each rewrite relation ($\mapsto$) in Fig 4.1 is also an equivalence relation according to the MLTL monitoring semantics.*

Proof. We prove each rule in Figure 4.1 is semantics-preserving, i.e., the right- and left-hand sides of the $\mapsto$ operator are equivalent with respect to the MLTL monitoring semantics. Recall that two MLTL formulas φ, ψ are equivalent only if $\pi \models \varphi \Leftrightarrow \pi \models \psi$ for all π.

(R1) Let φ be an MLTL formula, π be a trace, and $l_1 \leq u_1, l_2 \leq u_2$. We prove

$$\pi \models \Box_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi \Leftrightarrow \pi \models \Box_{[l_1+l_2,u_1+u_2]}\varphi.$$

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

($\leftarrow$) Let π be defined such that $\pi \models \Box_{[l_1+l_2,u_1+u_2]}\varphi$. We show that $\pi \models \Box_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi$. Then $\pi_k \models \varphi$ for all $k \in [l_1+l_2, u_1+u_2]$. Splitting this interval into two intervals, we have that $\pi_{i+j} \models \varphi$ for each $i \in [l_1, u_1]$, $j \in [l_2, u_2]$, as in the converse proof. Then $\pi \models \Box_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi$ by the semantics of $\Box_I$.

The proof for the $\Diamond$ version of (R1) is symmetric.

(R2): Let φ, ψ be MLTL formulas and $l_1 \leq u_1, l_2 \leq u_2$. We prove

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

for any $l_3 = min(l_1, l_2)$, $u_3 = l_3 + min(u_1 - l_1, u_2 - l_2)$, $l_3 < u_3$ using established equivalences and the MLTL semantics. Using (R1), we see that

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

This follows if both intervals $[l_1 - l_3, u_1 - l_3]$, $[l_2 - l_3, u_2 - l_3]$ are valid i.e., (a) $l_1 - l_3 \geq 0$, (b) $l_2 - l_3 \geq 0$, and (c) $l_1 - l_3 \leq u_1 - l_3$, (d) $l_2 - l_3 \leq u_2 - l_3$.

(a) Recall that $l_3 = l_1$, then $l_3 \leq l_1$.

(b) Recall that $l_3 = l_1 \leq l_2$, then $l_2 - l_3 \geq 0 \rightarrow l_2 \geq l_3 \rightarrow l_3 \leq l_2$ holds.

(c) Since $l_1 \leq u_1$, we see that $l_1 - l_3 \leq u_1 - l_3 \rightarrow l_1 \leq u_1$ holds.

(d) Since $l_2 \leq u_2$, we see that $l_2 - l_3 \leq u_2 - l_3 \rightarrow l_2 \leq u_2$ holds.

Now, let $u_3 = l_3 + min(u_1 - l_1, u_2 - l_2)$. Applying (R1) once more, we have

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

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

(a) Consider the two cases of $u_1 - l_1 \leq u_2 - l_2$ and $u_2 - l_2 < u_1 - l_1$:

i. Assume $u_1 - l_1 \leq u_2 - l_2$, then $u_3 = u_1 - l_1 + l_3$. Replacing this in the target inequality, we have

$$l_1 - l_3 \leq u_1 - (u_1 - l_1 + l_3) \rightarrow l_1 - l_3 \leq l_1 - l_3.$$

ii. Otherwise, $u_2 - l_2 < u_1 - l_1$. Then $u_3 = u_2 - l_2 + l_3$, and replacing this in the target inequality, we have

$$l_1 - l_3 \leq u_1 - (u_2 - l_2 + l_3) \rightarrow 0 \leq u_1 - l_3 - (u_2 - l_2) \rightarrow u_2 - l_2 \leq u_1 - l_3.$$

Now, since $l_3 = l_1$, we have $u_2 - l_2 \leq u_1 - l_1$ which is true from our assumption.

(b) Consider the two cases of $u_1 - l_1 \leq u_2 - l_2$ and $u_2 - l_2 < u_1 - l_1$:

i. Assume $u_1 - l_1 \leq u_2 - l_2$, then $u_3 = u_1 - l_1 + l_3$. Replacing this in the target inequality, we have

$$\begin{aligned} l_2 - l_3 \leq u_2 - (u_1 - l_1 + l_3) \rightarrow l_2 - l_3 \leq u_2 - u_1 + l_1 - l_3 \rightarrow \\ l_2 \leq u_2 - u_1 + l_1 \rightarrow u_1 - l_1 \leq u_2 - l_2, \end{aligned}$$

which is true from our assumption.

ii. Otherwise, $u_2 - l_2 < u_1 - l_1$, then $u_3 = u_2 - l_2 + l_3$. Replacing this in the target inequality, we have

$$\begin{aligned} l_2 - l_3 \leq u_2 - (u_2 - l_2 + l_3) \rightarrow l_2 - l_3 \leq u_2 - u_2 + l_2 - l_3 \rightarrow \\ l_2 \leq u_2 - u_2 + l_2 \rightarrow l_2 \leq l_2. \end{aligned}$$

Finally, let π be a trace. We prove that

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

(a) ($\rightarrow$) Let π be defined such that $\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi)$. We show that $\pi \models \Box_{[l_3,u_3]}(\Box_{[l_1-l_3,u_1-l_3-u_3]}\varphi \wedge \Box_{[l_2-l_3,u_2-l_3-u_3]}\psi)$ using the MLTL semantics. We

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

(b) ($\leftarrow$) Conversely, let π be defined such that $\pi \models \Box_{[l_3,u_3]}(\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\psi)$. We show that $\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi)$ using the MLTL semantics. Then $\pi_i \models \Box_{[l_1,u_1]}\varphi$ and $\pi_i \models \Box_{[l_2,u_2]}\psi$ for all $i \in [l_3, u_3]$. Using the semantic definitions of $\wedge$ and $\Box_I$, we see that $\pi \models \Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi$ and $\pi \models \Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi$, so $\pi \models (\Box_{[l_3,u_3]}\Box_{[l_1,u_1]}\varphi) \wedge (\Box_{[l_3,u_3]}\Box_{[l_2,u_2]}\psi)$.

The proof for the $\Diamond$ version of (R2) is symmetric.

(R3): Let φ be a MLTL formula and $l \leq u$. We prove

$$\Box_{[a,a]}\Diamond_{[l,u]}\varphi \equiv \Diamond_{[l,u]}\Box_{[a,a]}\varphi$$

using established MLTL equivalences. Using Equation 4.1 and (R1) to expand the expression until we have a of the $\Box_{[1,1]}$ operators in lines 3 and 9 of the following proof we can show:

$$\begin{aligned}
\Box_{[a,a]}\Diamond_{[l,u]}\varphi &\equiv \Box_{[1,1]}\Box_{[a-1,a-1]}\Diamond_{[l,u]}\varphi \\
&\equiv \cdots \text{Applying } a \text{ times} \\
&\equiv \Box_{[1,1]} \cdots \Box_{[1,1]}\Diamond_{[l,u]}\varphi \\
&\equiv \Diamond_{[1,1]} \cdots \Diamond_{[1,1]}\Diamond_{[l,u]}\varphi \\
&\equiv \Diamond_{[l+a,u+a]}\varphi \\
&\equiv \Diamond_{[l+a-1,u+a-1]}\Diamond_{[1,1]}\varphi \\
&\equiv \cdots \text{Applying } a \text{ times} \\
&\equiv \Diamond_{[l,u]}\Diamond_{[1,1]} \cdots \Diamond_{[1,1]}\varphi \\
&\equiv \Diamond_{[l,u]}\Box_{[1,1]} \cdots \Box_{[1,1]}\varphi \\
&\equiv \Diamond_{[l,u]}\Box_{[a,a]}\varphi.
\end{aligned}$$

The proof for the $\Diamond_{[a,a]}$ version of (R3) is symmetric.

(R4): Let φ be a MLTL formula and $l_1 \leq u_1$, $l_2 \leq u_2$, $l_1 \leq l_2 \leq u_1 + 1$, $u_3 = max(u_1, u_2)$. We prove that

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

We first show that $\pi \models \Box_{[l,l]}\varphi \wedge \Box_{[l+1,u]}\varphi \Leftrightarrow \pi \models \Box_{[l,u]}\varphi$ for any trace pi and $l < u$.

(a) ($\rightarrow$) Let π be a trace such that $\pi \models \Box_{[l,l]}\varphi \wedge \Box_{[l+1,u]}\varphi$. We show that $\pi \models \Box_{[l,u]}\varphi$. From the semantics of $\Box$ and (R1), we see that $\pi_i \models \varphi$ for all $i \in [l, l]$ and $\pi_j \models \varphi$ for all $j \in [l + 1, u]$. Now, since $[l, l] \cup [l + 1, u] = [l, u]$, it follows that $\pi_k \models \varphi$ for all $k \in [l, u]$. This matches the semantic definition of $\Box$ and therefore $\pi \models \Box_{[l,u]}\varphi$.

(b) ($\leftarrow$) Conversely, if π is a trace such that $\pi \models \Box_{[l,u]}\varphi$, then $\pi \models \Box_{[l,l]}\varphi \wedge \Box_{[l+1,u]}\varphi$ because $\pi_i \models \varphi$ for all $i \in [l, u]$ where $[l, l] \cup [l + 1, u] = [l, u]$ as before.

From above, we can expand each $\Box_I$ operator to a conjunction of singleton intervals, remove repeated conjunctive clauses, then use the above equivalence again to simplify:

$$\begin{aligned}
\Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\varphi \equiv & (\Box_{[l_1,l_1]}\varphi \wedge \Box_{[l_1+1,u_1]}) \wedge \Box_{[l_2,u_2]}\varphi \\
\equiv & \cdots \text{Applying } u_1 - l_1 \text{ times} \\
\equiv & (\Box_{[l_1,l_1]}\varphi \wedge \cdots \wedge \Box_{[u_1,u_1]}\varphi) \wedge \Box_{[l_2,u_2]}\varphi \\
\equiv & (\Box_{[l_1,l_1]}\varphi \wedge \cdots \wedge \Box_{[u_1,u_1]}\varphi) \wedge (\Box_{[l_2,l_2]}\varphi \wedge \Box_{[l_2+1,u_2]}\varphi) \\
\equiv & \cdots \text{Applying } u_2 - l_2 \text{ times} \\
\equiv & \Box_{[l_1,l_1]}\varphi \wedge \cdots \wedge \Box_{[u_1,u_1]}\varphi \wedge (\Box_{[l_2,l_2]}\varphi \wedge \cdots \wedge \Box_{[u_2,u_2]}\varphi) \\
\equiv & \Box_{[l_1,l_1]}\varphi \wedge \cdots \wedge \Box_{[l_2-1,l_2-1]}\varphi \wedge \Box_{[l_2,l_2]}\varphi \wedge \cdots \wedge \Box_{[u_2,u_2]}\varphi \\
\equiv & \Box_{[l_1,l_1+1]}\varphi \wedge \cdots \wedge \Box_{[l_2-1,l_2-1]}\varphi \wedge \Box_{[l_2,l_2]}\varphi \wedge \cdots \wedge \Box_{[u_2,u_2]}\varphi \\
\equiv & \cdots \text{Applying } l_2 - l_1 - 1 \text{ times} \\
\equiv & \Box_{[l_1,l_2-1]}\varphi \wedge \Box_{[l_2,l_2]}\varphi \wedge \cdots \wedge \Box_{[u_2,u_2]}\varphi \\
\equiv & \cdots \text{Applying } u_2 - l_2 + 1 \text{ times}
\end{aligned}$$

$$\equiv \Box_{[l_1,u_2]}\varphi$$

Note that from lines 6 to 7 we remove repeated clauses e.g., $l_2 \leq u_1 \leq u_2$ so there must be two instances of the expression $\Box_{[u_1,u_1]}\varphi$ in the formula.

The proof for the $\Diamond$ version of (R4) is symmetric.

(R5): Let φ be a MLTL formula and $l_1 \leq l_2 \leq u_2 \leq u_1$. We prove that

$$\Box_{[l_1,u_1]}\varphi \vee \Box_{[l_2,u_2]}\varphi \equiv \Box_{[l_2,u_2]}\varphi.$$

We first show that

$$\pi \models \Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi \Leftrightarrow \pi \models \Box_{[l,u_1]}\varphi$$

for any trace π.

(a) ($\rightarrow$) Assume for the purposes of contraction that π is a trace such that $\pi \models \Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi$ but $\pi \not\models \Box_{[l,u_1]}\varphi$. Therefore there is some $i \in [l, u_1]$ such that $\pi_i \not\models \varphi$. By (R1), we see that

$$\Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi \equiv \Box_{[l,u_1]}\varphi \vee \Box_{[0,u_2-u_1]}\Box_{[l,u_1]}\varphi$$

But $\pi \not\models \Box_{[l,u_1]}\varphi \vee \Box_{0,u_2-u_1}\Box_{[l,u_1]}\varphi$ since $\pi \not\models \Box_{[l,u_1]}\varphi$. Therefore if $\pi \models \Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi$, then $\pi \models \Box_{[l,u_1]}\varphi$.

(b) ($\leftarrow$) Conversely, assume that π is a trace such that $\pi \models \Box_{[l,u_1]}\varphi$ but $\pi \not\models \Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi$. But this is a contradiction, since by the definition of disjunction, if $\pi \models \Box_{[l,u_1]}\varphi$, then $\pi \models \Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi$ since the left-hand disjunctive clause models π. Therefore if $\pi \models \Box_{[l,u_1]}\varphi$, then $\pi \models \Box_{[l,u_1]}\varphi \vee \Box_{[l,u_2]}\varphi$.

Next we consider the cases for when $l_1 < l_2$ and $l_1 = l_2$.

(a) ($l_1 < l_2$) Starting from the left-hand side of the equivalence, we use (R3), the above equivalence, and the Absorption Law of Propositional Logic to show that:

$$\Box_{[l_1,u_1]}\varphi \vee \Box_{[l_2,u_2]}\varphi \equiv (\Box_{[l_1,l_2-1]}\varphi \wedge \Box_{[l_2,u_1]}\varphi) \vee \Box_{[l_2,u_2]}\varphi$$

$$\equiv (\Box_{[l_1,l_2-1]}\varphi \wedge \Box_{[l_2,u_1]}\varphi) \vee \Box_{[l_2,u_2]}\varphi$$
$$\equiv (\Box_{[l_1,l_2-1]}\varphi \vee \Box_{[l_2,u_2]}\varphi) \wedge (\Box_{[l_2,u_1]}\varphi \vee \Box_{[l_2,u_2]}\varphi)$$
$$\equiv (\Box_{[l_1,l_2-1]}\varphi \vee \Box_{[l_2,u_2]}\varphi) \wedge \Box_{[l_2,u_2]}\varphi$$
$$\equiv \Box_{[l_2,u_2]}\varphi$$

(b) ($l_1 = l_2$) Then starting with the left-hand side, replacing l_2 with l_1 and using the above equivalence we show:

$$\Box_{[l_1,u_1]}\varphi \vee \Box_{[l_2,u_2]}\varphi \equiv \Box_{[l_1,u_1]}\varphi \vee \Box_{[l_1,u_2]}\varphi$$
$$\equiv \Box_{[l_2,u_2]}\varphi$$

(R6): Let φ, ψ be MLTL formulas, π be a trace, and $l \leq u$. We prove that

$$\pi \models \Box_{[a,a]}(\varphi \, \mathcal{U}_{[l,u]} \psi) \Leftrightarrow \pi \models \varphi \, \mathcal{U}_{[l+a,u+a]} \psi$$

(a) ($\rightarrow$) Let π be a trace such that $\pi \models \Box_{[a,a]}(\varphi \, \mathcal{U}_{[l,u]} \psi)$. We show that $\pi \models \varphi \, \mathcal{U}_{[l+a,u+a]} \psi$. By the semantics of $\Box_{[a,a]}$, we see that $\pi_a \models \varphi \, \mathcal{U}_{[l,u]} \psi$. Similar to the proof for (R1), we say that $\pi_a \models \varphi \, \mathcal{U}_{[l,u]} \psi$ relative to the bounds of $[l, u]$. For instance, if $\pi_{a+l} \models \psi$, then $\pi_a \models \varphi \, \mathcal{U}_{[l,u]} \psi$. We then say that there is some $i \in [l, u]$ such that $\pi_{a+i} \models \psi$ and for all $j \in [l, u]$ such that $j < i$, $\pi_{a+j} \models \varphi$. This directly corresponds to the semantics of $\mathcal{U}_{[l+a,u+a]}$, so therefore $\pi \models \varphi \, \mathcal{U}_{[l+a,u+a]} \psi$.

(b) ($\leftarrow$) Conversely, let π be a trace such that $\pi \models \varphi \, \mathcal{U}_{[l+a,u+a]} \psi$. Then $\pi_i \models \psi$ for some $i \in [l + a, u + a]$ and $\pi_j \models \varphi$ for all $j \in [l + a, u + a]$ such that $j < i$. Therefore it follows that $\pi \models \Box_{[a,a]}(\varphi \, \mathcal{U}_{[l,u]} \psi)$ from the MLTL semantics.

The proof for the $\Diamond_{[a,a]}$ version of (R6) is symmetric.

(R7): Let $\varphi_1, \varphi_2, \varphi_3$ be MLTL formulas, π be a trace, and $l \leq u_1$, $l \leq u_2$, $u_1 \leq u_2$. We prove that

$$\pi \models (\varphi_1 \, \mathcal{U}_{[l,u_1]} \varphi_2) \wedge (\varphi_3 \, \mathcal{U}_{[l,u_2]} \varphi_2) \Leftrightarrow \pi \models (\varphi_1 \wedge \varphi_3) \, \mathcal{U}_{[l,u_1]} \varphi_2.$$

(a) ($\rightarrow$) Let π be such that $\pi \models (\varphi_1 \; \mathcal{U}_{[l,u_1]} \varphi_2) \wedge (\varphi_3 \; \mathcal{U}_{[l,u_2]} \varphi_2)$. We show that $\pi \models (\varphi_1 \wedge \varphi_3) \; \mathcal{U}_{[l,u_1]} \varphi_2$. From the semantics of $\mathcal{U}_I$, there must be some $i \in [l, u_1]$ such that $\pi_i \models \varphi_2$ in order to satisfy the clause $\varphi_1 \; \mathcal{U}_{[l,u_1]} \varphi_2$. Further, using the relation $u_1 \leq u_2$, we know that $\pi_j \models \varphi_1$ and $\pi_j \models \varphi_3$ for all $j \in [l, u_1] \subseteq [l, u_2]$ such that $j < i$. Putting this all together using the semantic definition of $\mathcal{U}_I$, $\pi \models (\varphi_1 \wedge \varphi_3) \; \mathcal{U}_{[l,u_1]} \varphi_2$.

(b) ($\leftarrow$) Conversely, let π be such that $\pi \models (\varphi_1 \wedge \varphi_3) \; \mathcal{U}_{[l,u_1]} \varphi_2$. Then $\pi \models (\varphi_1 \; \mathcal{U}_{[l,u_1]} \varphi_2) \wedge (\varphi_3 \; \mathcal{U}_{[l,u_2]} \varphi_2)$ because $\pi_i \models \varphi_2$ for some $i \in [l, u_1] \subseteq [l, u_2]$ and $\pi_j \models \varphi_1$ and $\pi_j \models \varphi_3$ for all $j \in [l, u_1]$ such that $j < i$.

(R8): Let φ be a MLTL formula, and $l_1 \leq u_1$. We prove that

$$\varphi \; \mathcal{U}_{[l_1,u_1]} \Box_{[0,u_2]} \varphi \equiv \Box_{[l_1, l_1+u_2]} \varphi.$$

We first show that

$$\Box_{[l,l]} \psi \vee (\Box_{[l,l]} \varphi \wedge (\varphi \; \mathcal{U}_{[l+1,u]} \psi)) \equiv \varphi \; \mathcal{U}_{[l,u]} \psi$$

for any trace π.

(a) Let π be such that $\pi \models \Box_{[l,l]} \psi \vee (\Box_{[l,l]} \varphi \wedge (\varphi \; \mathcal{U}_{[l+1,u]} \psi))$. We show that $\pi \models \varphi \; \mathcal{U}_{[l,u]} \psi$. Distributing the $\Box_{[l,l]} \psi$ in the left-hand expression, we obtain the formula $(\Box_{[l,l]} \psi \vee \Box_{[l,l]} \varphi) \wedge (\Box_{[l,l]} \psi \vee \varphi \; \mathcal{U}_{[l+1,u]} \psi)$. Consider the case where $\pi_l \models \psi$, then $\pi \models \varphi \; \mathcal{U}_{[l,u]} \psi$ by the semantics of $\mathcal{U}_I$. Otherwise, $\pi_l \not\models \psi$, then $\pi_l \models \varphi$ and $\varphi \; \mathcal{U}_{[l+1,u]} \psi$. It follows that $\pi \models \varphi \; \mathcal{U}_{[l,u]} \psi$ by the semantics of $\mathcal{U}_I$.

(b) Conversely, let π be a trace such that $\pi \not\models \Box_{[l,l]} \psi \vee (\Box_{[l,l]} \varphi \wedge (\varphi \; \mathcal{U}_{[l+1,u]} \psi))$. Then $\pi \not\models \varphi \; \mathcal{U}_{[l,u]} \psi$ because either $\pi_l \not\models \psi$ and $\pi_l \not\models \varphi$ or $\pi \not\models \varphi \; \mathcal{U}_{[l+1,u]} \psi$.

Now using the above equivalence, (R4), (R5), and Equation 4.1:

$$\begin{aligned}
\varphi \; \mathcal{U}_{[l,u_1]} (\Box_{[0,u_2]} \varphi) \equiv & \Box_{[l,l]} \Box_{[0,u_2]} \varphi \vee (\Box_{[l,l]} \varphi \wedge (\varphi \; \mathcal{U}_{[l+1,u_1]} \Box_{[0,u_2]} \varphi)) \\
\equiv & \Box_{[l,l+u_2]} \varphi \vee (\Box_{[l,l]} \varphi \wedge (\varphi \; \mathcal{U}_{[l+1,u_1]} \Box_{[0,u_2]} \varphi)) \\
\equiv & \Box_{[l,l+u_2]} \varphi \vee (\Box_{[l,l]} \varphi \wedge (\Box_{[l+1,l+1+u_2]} \varphi
\end{aligned}$$

$$\vee (\Box_{[l+1,l+1]}\varphi \wedge (\varphi \,\mathcal{U}_{[l+2,u_1]}\Box_{[0,u_2]}\varphi)))$$

$$\equiv \cdots \text{(Applying } u_1 \text{ times)}$$

$$\equiv \Box_{[l,l+u_2]}\varphi \vee (\Box_{[l,l]}\varphi \wedge (\cdots \wedge (\Box_{[u_1-1,u_1+u_2-1]}\varphi \vee (\Box_{[u_1-1,u_1-1]}\varphi \wedge \varphi \,\mathcal{U}_{[u_1,u_1]}\Box_{[0,u_2]}\varphi))))$$

$$\equiv \Box_{[l,l+u_2]}\varphi \vee (\Box_{[l,l]}\varphi \wedge (\cdots \wedge (\Box_{[u_1-1,u_1+u_2-1]}\varphi \vee (\Box_{[u_1-1,u_1-1]}\varphi \wedge \Box_{[u_1,u_1]}\Box_{[0,u_2]}\varphi))))$$

$$\equiv \Box_{[l,l+u_2]}\varphi \vee (\Box_{[l,l]}\varphi \wedge (\cdots \wedge (\Box_{[u_1-1,u_1+u_2-1]}\varphi \vee (\Box_{[u_1-1,u_1-1]}\varphi \wedge \Box_{[u_1,u_1+u_2]}\varphi))))$$

$$\equiv \Box_{[l,l+u_2]}\varphi \vee (\Box_{[l,l]}\varphi \wedge (\cdots \wedge (\Box_{[u_1-1,u_1+u_2-1]}\varphi \vee \Box_{[u_1-1,u_1+u_2]}\varphi)))$$

$$\equiv \Box_{[l,l+u_2]}\varphi \vee (\Box_{[l,l]}\varphi \wedge (\cdots \wedge \Box_{[u_1-1,u_1+u_2]}\varphi))$$

$$\equiv \cdots \text{(Applying } u_1 \text{ times)}$$

$$\equiv \Box_{[l,l+u_2]}\varphi \vee (\Box_{[l,u_1+u_2]}\varphi)$$

$$\equiv \Box_{[l,l+u_2]}\varphi$$

□

Theorem 5 (Memory Reduction of Rewriting Rules)**.** *Let φ, ψ_1, ψ_2 be MLTL formulas where ψ_1 is a sub-formula of φ. Then applying a valid rewrite rule in Figure 4.1 to ψ_1 will result in a new formula $\varphi(\psi_1 \mapsto \psi_2)$ such that $\varphi \equiv \varphi(\psi_1 \mapsto \psi_2)$ and*

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

Proof. (R1): Then $\psi_1 = \Box_{[l_1,u_1]}\Box_{[l_2,u_2]}\varphi$ and $\psi_2 = \Box_{[l_1+l_2,u_1+u_2]}\varphi$ where $l_1 \leq u_1$ and $l_2 \leq u_2$.

Therefore, because $\psi_1.wpd = \varphi.wpd + u_1 + u_2 = \psi_2.wpd$ and $\psi_1 \equiv \psi_2$ by Theorem 4, we have $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$.

Next, using Lemma 1 we see that $mem_{node}(\Box_{[l_1,u_1]}) = mem_{node}(\Box_{[l_1+l_2,u_1+u_2]})$ since $\psi_1.bpd = \varphi.bpd + l_1 + l_2 = \psi_2.bpd$. Then

$$\begin{aligned}
mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) =& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2) \\
=& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + \\
& mem_{node}(\Box_{[l_1+l_2,u_1+u_2]}) + mem_{AST}(\varphi) \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& mem_{node}(\Box_{[l_1+l_2,u_1+u_2]}) + mem_{AST}(\varphi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& mem_{node}(\Box_{[l_1,u_1]}) + mem_{AST}(\varphi) \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\Box_{[l_1,u_1]}) + \\
& mem_{node}(\Box_{[l_2,u_2]}) + mem_{AST}(\varphi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1) \\
=& mem_{AST}(\varphi)
\end{aligned}$$

The proof for the rule $\Diamond_{[l_1,u_1]}\Diamond_{[l_2,u_2]}\varphi \to \Diamond_{[l_1+l_2,u_1+u_2]}\varphi$ follows.

(R2): First, let $\psi_1 = \Box_{[l_1,u_1]}\varphi_1 \wedge \Box_{[l_2,u_2]}\varphi_2$ and $\psi_2 = \Box_{[l_3,u_3]}(\Box_{[l_1-l_3,u_1-u_3]}\varphi_1 \wedge \Box_{[l_2-l_3,u_2-u_3]}\varphi_2)$ where $l_1 \leq u_1$, $l_2 \leq u_2$, $l_3 = min(l_1, l_2)$, $u_3 = l_3 + min(u_1 - l_1, u_2 - l_2)$, and $l_3 < u_3$. We show that

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

Therefore we have $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2. A similar derivation is used to show that $\psi_1.bpd = \psi_2.bpd$, so

$mem_{node}(\wedge_1) \geq mem_{node}(\Box_{[l_3,u_3]})$ by Lemma 1 where $\wedge_1, \wedge_2$ denote the $\wedge$-nodes in ψ_1, ψ_2 respectively.

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

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

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

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

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

(R3): Then $\psi_1 = \Box_{[a,a]}\Diamond_{[l,u]}\varphi$ and $\psi_2 = \Diamond_{[l+a,u+a]}\varphi$ where $l \leq u$. Therefore, because $\psi_1.wpd = \varphi.wpd + u + a = \psi_2.wpd$ and $\psi_1 \equiv \psi_2$ by Theorem 4, we have $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2.

Next, using Lemma 1 we see that $mem_{node}(\Box_{[a,a]}) = mem_{node}(\Diamond_{[l+a,u+a]})$ since $\psi_1.bpd = \varphi.bpd + l + a = \psi_2.bpd$. Then

$$\begin{aligned}
mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) =& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2) \\
=& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + \\
& \quad mem_{node}(\Diamond_{[l+a,u+a]}) + mem_{AST}(\varphi) \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& \quad mem_{node}(\Diamond_{[l+a,u+a]}) + mem_{AST}(\varphi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& \quad mem_{node}(\Box_{[a,a]}) + mem_{AST}(\varphi) \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\Box_{[a,a]}) + \\
& \quad mem_{node}(\Diamond_{[l,u]}) + mem_{AST}(\varphi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1) \\
=& mem_{AST}(\varphi)
\end{aligned}$$

The proof follows for $\Diamond_{[l,u]}\Box_{[a,a]}\varphi \to \Diamond_{[l+a,u+a]}\varphi$, $\Diamond_{[a,a]}\Box_{[l,u]}\varphi \to \Box_{[l+a,u+a]}\varphi$, $\Box_{[l,u]}\Diamond_{[a,a]}\varphi \to \Box_{[l+a,u+a]}\varphi$ follows.

(R4): Then $\psi_1 = \Box_{[l_1,u_1]}\varphi \wedge \Box_{[l_2,u_2]}\varphi$ and $\psi_2 = \Box_{[l_1,u_3]}\varphi$ where $l_1 \leq u_1, l_2 \leq u_2$, $l_1 \leq l_2 \leq u_1 + 1$, and $u_3 = max(u_1, u_2)$. We show that

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

Therefore we have $\psi_1 \equiv \psi_2$ by Theorem 4 so that $mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2.

Next we show that

$$\begin{aligned}\psi_1.bpd =& min(\varphi.bpd + l_1, \varphi.wpd + l_2)\\ =& \varphi.bpd + min(l_1, l_2)\\ =& \varphi.bpd + l_1\\ =& \psi_2.bpd\end{aligned}$$

Using Lemma 1 we see that $mem_{node}(\wedge) = mem_{node}(\Box_{[l_1,u_3]})$ since $\psi_1.bpd = \psi_2.bpd$. Then

$$\begin{aligned}mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) =& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2)\\ =& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) +\\ & mem_{node}(\Box_{[l_1,u_3]}) + mem_{AST}(\varphi)\\ \leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) +\\ & mem_{node}(\Box_{[l_3,u_3]}) + mem_{AST}(\varphi)\\ =& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) +\\ & mem_{node}(\wedge) + mem_{AST}(\varphi)\\ \leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\wedge) +\\ & mem_{node}(\Box_{[l_1,u_1]}) + mem_{node}(\Box_{[l_2,u_2]}) + 2 \cdot mem_{AST}(\varphi)\\ =& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1)\\ =& mem_{AST}(\varphi)\end{aligned}$$

The proof follows for $\Diamond_{[l_1,u_1]}\varphi \vee \Diamond_{[l_2,u_2]}\varphi \rightarrow \Diamond_{[l_1,u_3]}\varphi$.

(R5): Let $\psi_1 = \Box_{[l_1,u_1]}\varphi \vee \Box_{[l_2,u_2]}\varphi$ and $\psi_2 = \Box_{[l_3,u_3]}\varphi$ where $l_1 \leq l_2 \leq u_2 \leq u_1$ and $l_2 = l_3$, $u_2 = u_3$. We show that

$$\begin{aligned}\psi_1.wpd =& max(\varphi.wpd + u_1, \varphi.wpd + u_2)\\ =& \varphi.wpd + max(u_1, u_2)\\ =& \varphi.wpd + u_1\end{aligned}$$

$$\begin{aligned}
&\geq \varphi.wpd + u_2 \\
&= \varphi.wpd + u_3 \\
&= \psi_2.wpd
\end{aligned}$$

Therefore we have $\psi_1 \equiv \psi_2$ by Theorem 4 so that

$mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2.

Next we show that

$$\begin{aligned}
\psi_1.bpd &= min(\varphi.bpd + l_1, \varphi.wpd + l_2) \\
&= \varphi.bpd + min(l_1, l_2) \\
&= \varphi.bpd + l_1 \\
&\leq \varphi.bpd + l_2 \\
&= \varphi.bpd + l_3 \\
&= \psi_2.bpd
\end{aligned}$$

Using Lemma 1 we see that $mem_{node}(\vee) = mem_{node}(\square_{[l_3,u_3]})$ since $\psi_1.bpd \leq \psi_2.bpd$. Then

$$\begin{aligned}
mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) &= (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2) \\
&= (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + \\
&\qquad mem_{node}(\square_{[l_3,u_3],\psi_2}) + mem_{AST}(\varphi) \\
&\leq (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
&\qquad mem_{node}(\square_{[l_3,u_3],\psi_2}) + mem_{AST}(\varphi) \\
&= (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
&\qquad mem_{node}(\vee) + mem_{AST}(\varphi) \\
&\leq (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\vee) + \\
&\qquad mem_{node}(\square_{[l_1,u_1]}) + mem_{node}(\square_{[l_2,u_2]}) + 2 \cdot mem_{AST}(\varphi) \\
&= (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1)
\end{aligned}$$

$$=mem_{AST}(\varphi)$$

(R6): Let $\psi_1 = \Box_{[a,a]}(\varphi \mathcal{U}_{[l,u]} \psi)$ and $\psi_2 = \varphi \mathcal{U}_{[l+a,u+a]} \psi$ where $l \leq u$. Therefore, because $\psi_1.wpd = max(\varphi.wpd, \psi.wpd) + u + a = max(\varphi.wpd + u + a, \psi.wpd + u + a) = \psi_2.wpd$ and $\psi_1 \equiv \psi_2$ by Theorem 4, we have

$mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2.

Next, using Lemma 1 we see that $mem_{node}(\Box_{[a,a]}) = mem_{node}(\mathcal{U}_{[l+a,u+a]})$ since $\psi_1.bpd = min(\varphi.bpd, \psi.bpd) + l + a = min(\varphi.bpd + l + a, \psi.bpd + l + a) = \psi_2.bpd$. Then

$$\begin{aligned}
mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) =& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2) \\
=& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + \\
& mem_{node}(\mathcal{U}_{[l+a,u+a]}) + mem_{AST}(\varphi) + mem_{AST}(\psi) \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& mem_{node}(\mathcal{U}_{[l+a,u+a]}) + mem_{AST}(\varphi) + mem_{AST}(\psi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& mem_{node}(\Box_{[a,a]}) + mem_{AST}(\varphi) + mem_{AST}(\psi) \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\Box_{[a,a]}) + \\
& mem_{node}(\mathcal{U}_{[l,u]}) + mem_{AST}(\varphi) + mem_{AST}(\psi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1) \\
=& mem_{AST}(\varphi)
\end{aligned}$$

The proof follows for $(\Box_{[a,a]}\varphi)\mathcal{U}_{[l,u]}(\Box_{[a,a]}\psi) \rightarrow \varphi \mathcal{U}_{[l+a,u+a]} \psi$.

(R7): Let $\psi_1 = (\varphi_1 \mathcal{U}_{[l,u_1]} \varphi_2) \wedge (\varphi_3 \mathcal{U}_{[l,u_2]} \varphi_2)$ and $\psi_2 = (\varphi_1 \wedge \varphi_3)\mathcal{U}_{[l,u_3]}\varphi_2$ where $l \leq u_1$, $l \leq u_2$, $u_1 \leq u_2$, and $u_3 = u_1$. We show that

$$\begin{aligned}
\psi_1.wpd =& max(max(\varphi_1.wpd, \varphi_2.wpd) + u_1, max(\varphi_3.wpd, \varphi_2.wpd) + u_2) \\
\geq& max(max(\varphi_1.wpd, \varphi_2.wpd) + u_1, max(\varphi_3.wpd, \varphi_2.wpd) + u_1)
\end{aligned}$$

$$
\begin{aligned}
&= max(max(\varphi_1.wpd, \varphi_2.wpd), max(\varphi_3.wpd, \varphi_2.wpd)) + u_1 \\
&= max(max(\varphi_1.wpd, \varphi_3.wpd), \varphi_2.wpd) + u_1 \\
&= max(max(\varphi_1.wpd, \varphi_3.wpd), \varphi_2.wpd) + u_3 \\
&= \psi_2.wpd
\end{aligned}
$$

Therefore we have $\psi_1 \equiv \psi_2$ by Theorem 4 so that
$mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2) \leq mem_{AST}(\varphi) - mem_{AST}(\psi_1)$ by Lemma 2.

Next we show that

$$
\begin{aligned}
\psi_1.bpd &= min(min(\varphi_1.bpd, \varphi_2.bpd) + l, min(\varphi_3.bpd, \varphi_2.bpd) + l) \\
&= min(min(\varphi_1.bpd, \varphi_2.bpd), min(\varphi_3.bpd, \varphi_2.bpd)) + l \\
&= min(min(\varphi_1.bpd, \varphi_3.bpd), \varphi_2.bpd) + l \\
&= \psi_2.bpd
\end{aligned}
$$

Using Lemma 1 we see that $mem_{node}(\wedge_{\psi_1}) = mem_{node}(\mathcal{U}_{[l,u_3]})$ since $\psi_1.bpd \leq \psi_2.bpd$. Then

$$
\begin{aligned}
mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) &= (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2) \\
&= (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + \\
&\qquad mem_{node}(\mathcal{U}_{[l,u_3]}) + mem_{node}(\wedge_{\psi_2}) + \\
&\qquad mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2) + mem_{AST}(\varphi_3) \\
&\leq (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
&\qquad mem_{node}(\mathcal{U}_{[l,u_3]}) + mem_{node}(\wedge_{\psi_2}) + \\
&\qquad mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2) + mem_{AST}(\varphi_3) \\
&= (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
&\qquad mem_{node}(\wedge_{\psi_1}) + mem_{node}(\wedge_{\psi_2}) + \\
&\qquad mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2) + mem_{AST}(\varphi_3) \\
&\leq (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\wedge_{\psi_1}) +
\end{aligned}
$$

$$
\begin{aligned}
& mem_{node}(\mathcal{U}_{[l,u_1]}) + mem_{node}(\mathcal{U}_{[l,u_2]}) \\
& mem_{AST}(\varphi_1) + mem_{AST}(\varphi_2) + mem_{AST}(\varphi_3) \\
=&(mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1) \\
=&mem_{AST}(\varphi)
\end{aligned}
$$

The proof follows for $(\varphi_1 \mathcal{U}_{[l,u_1]} \varphi_2) \vee (\varphi_1 \mathcal{U}_{[l,u_2]} \varphi_3) \rightarrow \varphi_1 \mathcal{U}_{[l,u_1]} (\varphi_2 \vee \varphi_3)$.

(R8): Let $\psi_1 = \varphi \mathcal{U}_{[l,u_1]} \Box_{[0,u_2]} \varphi$, $\psi_2 = \Box_{[l,l+u_2]} \varphi$ where $l \leq u_1$. We show that

$$
\begin{aligned}
\psi_1.wpd =& max(\varphi.wpd, \varphi.wpd + u_1) + u_2 \\
=& \varphi.wpd + u_1 + u_2 \\
\geq& \varphi.wpd + l + u_2 \\
=& \psi_2.wpd
\end{aligned}
$$

Therefore we have $\psi_1 \equiv \psi_2$ by Theorem 4 so that

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

by Lemma 2. Next we show that

$$
\begin{aligned}
\psi_1.bpd =& min(\varphi.bpd, \varphi.bpd + 0) + l \\
=& \varphi.bpd + l \\
=& \psi_2.bpd
\end{aligned}
$$

Using Lemma 1 we see that $mem_{node}(\mathcal{U}_{[l,u_1]}) = mem_{node}(\Box_{[l,l+u_2]})$ since $\psi_1.bpd \leq \psi_2.bpd$. Then

$$
\begin{aligned}
mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) =& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + mem_{AST}(\psi_2) \\
=& (mem_{AST}(\varphi(\psi_1 \mapsto \psi_2)) - mem_{AST}(\psi_2)) + \\
& mem_{node}(\Box_{[l,l+u_2]}) + mem_{node}(\varphi) + \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) +
\end{aligned}
$$

$$
\begin{aligned}
& mem_{node}(\Box_{[l,l+u_2]}) + mem_{node}(\varphi) + \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + \\
& mem_{node}(\mathcal{U}_{[l,u_1]}) + mem_{node}(\varphi) + \\
\leq& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{node}(\mathcal{U}_{[l,u_1]}) + \\
& mem_{node}(\Box_{[0,u_2]}) + 2 \cdot mem_{node}(\varphi) \\
=& (mem_{AST}(\varphi) - mem_{AST}(\psi_1)) + mem_{AST}(\psi_1) \\
=& mem_{AST}(\varphi)
\end{aligned}
$$

The proof follows for $\varphi \mathcal{U}_{[l,u_1]} \Diamond_{[0,u_2]} \varphi \rightarrow \Diamond_{[l,l+u_2]} \varphi$.

□