Shrinking Mission-time LTL Runtime Monitors with Equality Saturation

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

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

***Abstract*—Runtime Verification (RV) verifies safety-critical systems in real time, often under hard time and space constraints that only become more restrictive as systems grow in complexity. Equality Saturation (EqSat) is an algorithm that efficiently optimizes programs via rewriting thus reducing their cost in terms of time and memory. We propose applying EqSat to RV to automatically reduce the resource usage of RV monitors, enabling more efficient deployment in resource-constrained environments. In particular, we target the flight- and space-certified RV framework R2U2 and adapt EqSat for its specification logic, Mission-time LTL (MLTL). We then extract a resource-reduced MLTL formula from the output of EqSat using an Integer Linear Programming (ILP) encoding and offer an alternative, more-efficient greedy extraction method. We implement this approach in a fully-automated toolchain that applies EqSat to an input formula using the off-the-shelf tool EggLog, performs ILP-based extraction via Gurobi, and outputs an MLTL formula optimized for R2U2 resource usage. Experimental evaluation demonstrates memory reduction for 61% of 1,177 real-world RV benchmarks by an average of 22%. On balance, we find that applying EqSat using the greedy extraction method and associative/commutative rewrites disabled provides the best approach that effectively reduces R2U2 resource usage while maintaining low optimization overhead.**

I. Introduction

Safety-critical systems such as spacecraft have restrictive hardware, often with only kilobytes of memory to spare for on-board verification. Real-time verification of other cyber-physical systems such as autonomous aircraft is often similarly limited by hardware constraints. Every byte and clock cycle matters in these extremely resource-constrained environments and could mean the difference between verifying such systems successfully or not at all. Runtime Verification (RV) monitors formal specifications in real time on systems with hard time and memory constraints, but only if the RV monitor respects those constraints.

R2U2 is an open-source, flight-certified, stream-based RV framework targeting embedded systems, especially in the aerospace domain, and has successfully contributed real-time monitoring in such environments, including NASA's Robonaut 2 [29], the CySat-I CubeSat [2], a sounding rocket control system [22], ESA's OP-SAT satellite [36], and others [20], [25], [12]. R2U2 uses the RV specification language Mission-time LTL (MLTL) [40], [31], a logic that balances expressiveness with computational efficiency. Many tools support MLTL elicitation and analysis [44], [30], [18], [45]. Other RV frameworks monitor similar stream-based formalisms, including Hydra [39], CoPilot [38], Tessla [10], and LOLA [13]. Only three RV frameworks are flight-certified (CoPilot, LOLA, R2U2) and one is space-certified (R2U2). However, these systems become increasingly complex as technology advances, generate more data in real time, and create more verification tasks for domain engineers, presenting new challenges for deploying RV within system constraints. Developing intelligent methods that automatically configure constraint-respecting RV monitors for such systems is essential for meeting these challenges.

In parallel to RV development, Equality Saturation (EqSat) has been growing in impact for decades, starting as a technique for compiler super-optimization [28] then defined as an algorithm for generic term languages [42], resulting in a wide range of applications [9], [11], [43], [48], [37], [47]. EqSat uses an e-graph [34], a broadly-applicable data structure used in theorem proving [14], to compactly represent equivalent versions of the input program. Given a set of rewrites, EqSat then applies the rewrites to the e-graph until *saturation*, i.e., when no more rewrites are applicable, resulting in an e-graph that represents all programs equivalent to the input program derivable via the rewrites. Finally, extraction produces an optimized program equivalent to the input from the saturated e-graph. Integer Linear Programming (ILP) is an effective and complete way to perform extraction [43], [48] and there exist efficient tools for solving ILP problems such as Gurobi [19].

Egg [46] improved EqSat and unified previous approaches by offering a generic platform for user-defined term languages and rewrites, addressing the ad-hoc nature of previous work. EggLog [49] again improved EqSat and took unification a step further by defining an input language for defining term languages, rewrites, and cost functions. EqSat with Egg/EggLog excels at program optimization, including in hardware logic synthesis [9], datapath optimization [11], linear algebra [43], tensor graphs [48], and floating point expression accuracy [37].

The time has come to investigate how EqSat can address the challenges of deploying RV on modern, complex systems. Focusing on R2U2, as it is the only space-certified RV monitoring framework, we propose leveraging the optimization capabilities of EqSat and its mature tool ecosystem for automatically reducing (i.e., shrinking) R2U2 monitor resource

https://doi.org/

This article is licensed under a Creative Commons Attribution 4.0 International License

usage by applying EqSat to MLTL. Current resource-reducing algorithms for RV either modify the underlying monitoring algorithm, not the specification itself, using control-theoretic approaches [32], [23] or exploit algorithmic constructs that impose complex relationships between temporal sub-formulas, making it hard to add additional optimization algorithms like EqSat [29]. The end result is a fully-automated toolchain that uses verified rewrites to perform EqSat on an input formula and extracts an optimized formula equivalent to the input, offering an effective method for addressing the challenge of configuring RV for resource-constrained systems.

Our contributions connect EqSat to real-time resource-constrained RV, targeting R2U2, including

1) an adapted set of MLTL rewrites for EqSat;
2) an ILP-based extraction method;
3) a cheaper, greedy extraction method that provides an efficient alternative while achieving similar memory reductions;
4) an open-source toolchain implementing EqSat and the extraction methods using EGGLOG and GUROBI;
5) an experimental evaluation demonstrating the usefulness of EqSat on real-world RV problems.

After providing preliminary definitions of MLTL and EqSat in Section II, we adapt EqSat to MLTL by defining the rewrite set and some optimizations in Section III. Section IV defines two methods for extracting an optimized formula from the output of EqSat: an ILP encoding and a greedy alternative. We implement our approach in Section V, including automatically proving the correctness of each set of rewrites [1], and experimentally evaluate it in Section VI over a set of 1,177 real-world RV monitoring problems demonstrating memory reduction on 62% of benchmarks by an average of 22%. We conclude with a discussion of the myriad of new research avenues for EqSat and RV in Section VII.

II. PRELIMINARIES

A. Mission-time Linear Temporal Logic

Definition 1 (*MLTL Syntax*)**.** The syntax of an MLTL formula over a set of atomic propositions $\mathcal{AP}$ where $p \in \mathcal{AP}$, $i_0, i_1 \in \mathbb{N}_0$ with $i_0 \leq i_1$, and φ, ψ are MLTL formulas is defined as:

$$\mathsf{true} \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \, \mathcal{U}_{[i_0,i_1]} \, \psi$$

The *closure* [31] of an MLTL formula φ, denoted $cl(\varphi)$, captures the set of syntactic elements of φ, i.e., elements in the Directed Acyclic Graph (DAG) representation of φ. Let $\circ \in \{\wedge, \mathcal{U}_{[i_0,i_1]}\}$, then

$$cl(\mathsf{true}) = \{\mathsf{true}\} \quad cl(p) = \{p\} \quad cl(\neg\varphi) = \{\neg\varphi\} \cup cl(\varphi)$$
$$cl(\varphi \circ \psi) = \{\varphi \circ \psi\} \cup cl(\varphi) \cup cl(\psi).$$

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

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

- $\pi \models \mathsf{true}$ • $\pi \models p$ iff $p \in \pi[0]$ • $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$
- $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$
- $\pi \models \varphi \, \mathcal{U}_{[i_0,i_1]} \, \psi$ iff $|\pi| > i_0$ and $\exists \tau_0 . \pi_{\tau_0} \models \psi$ such that $i_0 \leq \tau_0 \leq i_1$ and $\tau_0 < |\pi|$ and $\forall \tau_1 . \pi_{\tau_1} \models \varphi$ such that $i_0 \leq \tau_1 < \tau_0$.

We say two MLTL formulas φ, ψ are *semantically equivalent* iff $\pi \models \varphi \leftrightarrow \pi \models \psi$ for all finite traces π over $\mathcal{AP}$. MLTL keeps the standard operator equivalences from LTL, including $\Diamond_{[i_0,i_1]}\varphi = (\mathsf{true} \, \mathcal{U}_{[i_0,i_1]} \, \varphi)$, $\varphi \, \mathcal{R}_{[i_0,i_1]} \, \psi = \neg(\neg\varphi \, \mathcal{U}_{[i_0,i_1]} \, \neg\psi)$, and $\Box_{[i_0,i_1]}\varphi = (\mathsf{false} \, \mathcal{R}_{[i_0,i_1]} \, \varphi)$. To complete the MLTL semantics, we define $\mathsf{false} = \neg\mathsf{true}$, $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, and $\neg\Diamond_{[i_0,i_1]}\varphi = \Box_{[i_0,i_1]}\neg\varphi$.

Given a top-level MLTL formula φ and a sub-formula $\psi \in cl(\varphi)$, ψ's *children* $\mathcal{C}(\psi)$ are the set of immediate sub-formulas of ψ and its *siblings* $\mathcal{S}(\psi, \varphi)$ are the set of formulas that share a parent formula with ψ in φ. For example, in the formula $\varphi = \psi \wedge \xi$, we have $\mathcal{C}(\varphi) = \{\psi, \xi\}$, $\mathcal{S}(\psi, \varphi) = \{\xi\}$, and $\mathcal{S}(\xi, \varphi) = \{\psi\}$. In the following, we abuse notation and assume that the top-level operator is given when querying a formula's siblings, i.e., we use simply $\mathcal{S}(\psi)$. Let $lb(\varphi) = i_0$ and $ub(\varphi) = i_1$ if $\varphi = \psi \, \mathcal{U}_{[i_0,i_1]} \, \xi$, otherwise $lb(\varphi) = ub(\varphi) = 0$.

The *propagation delay* of an MLTL formula defines the necessary number of states to evaluate the formula in the best and worst cases.

Definition 3 (*Propagation Delay*)**.** [29] Let φ, ψ be MLTL formulas. Let $min(\emptyset) = max(\emptyset) = 0$. Then the best- and worst-case propagation delays of φ respectively are:

$$bpd(\varphi) = min(bpd(\psi) \mid \psi \in \mathcal{C}(\varphi)) + lb(\varphi)$$
$$wpd(\varphi) = max(wpd(\psi) \mid \psi \in \mathcal{C}(\varphi)) + ub(\varphi).$$

R2U2 requires storing verdict-timestamp pairs $\langle v, \tau \rangle$ with $v \in \{\mathsf{true}, \mathsf{false}\}$ and $\tau \in \mathbb{N}_0$ in case it evaluates one sibling formula before another. To illustrate this case, consider the formula $\varphi \wedge \psi$. If R2U2 determines the verdict of φ for time i before it knows the verdict of ψ for that same time then R2U2 will need to store that verdict of φ until it evaluates ψ for time i. In the worst case, this requires waiting at most $wpd(\psi) - bpd(\varphi)$ timesteps. Generally, if φ has more siblings than just ψ, we consider φ's sibling with the largest wpd. The minimum required memory (in number of verdict/timestamp pairs) for monitoring a formula φ with R2U2 (as defined in [29], [3]) is:

$$\mathsf{Mem}(\varphi) = max(max\{wpd(\psi) \mid \psi \in \mathcal{S}(\varphi)\} - bpd(\varphi), 0) + 1. \quad (1)$$

Each verdict-timestamp pair is represented using a machine integer (e.g., 1 bit for verdict, 31 bits for timestamp).

In practice R2U2 monitors use node sharing to avoid repeated computation [29], so the representation of φ is a directed acyclic graph (DAG). To compute $\mathsf{MemDAG}(\varphi)$, we sum the memory requirements of all syntactically distinct sub-formulas of φ, i.e., the elements in $cl(\varphi)$:

$$\mathsf{MemDAG}(\varphi) = \sum_{\psi \in cl(\varphi)} \mathsf{Mem}(\psi) \quad (2)$$

Rewrite	Constraints
$\square_{[i_0,i_1]}\square_{[j_0,j_1]}\varphi \mapsto \square_{[i_0+j_0,i_1+j_1]}\varphi$	none
$\square_{[i_0,i_1]}\varphi \wedge \square_{[j_0,j_1]}\psi \mapsto \square_{[k_0,k_1]}(\square_{[i_0-k_0,i_1-k_1]}\varphi \wedge \square_{[j_0-k_0,j_1-k_1]}\psi)$	$k_0 = min(i_0, j_0); k_1 = k_0 + min(i_1 - i_0, j_1 - j_0)$
$(\Diamond_{[i_0,i_1]}\varphi) \rightarrow ((\psi\, \mathcal{U}_{[j_0,j_1]}\, \varphi) \vee \square_{[k_0,k_1]}\psi) \mapsto (\Diamond_{[i_0,i_1]}\varphi) \rightarrow (\psi\, \mathcal{U}_{[i_0,i_1]}\, \varphi)$	$j_0 = i_0;\ j_1 \geq i_1;\ k_0 \leq i_0;\ k_1 \geq i_1$
$(\square_{[i_0,i_1]}\neg\varphi) \wedge (\Diamond_{[j_0,j_1]}\varphi) \mapsto (\square_{[i_0,i_1]}\neg\varphi) \wedge (\Diamond_{[i_1+1,j_1]}\varphi)$	$i_0 \leq j_0;\ j_0 < i_1;\ i_1 < j_1$
$(\square_{[i_0,i_1]}\varphi) \wedge (\Diamond_{[j_0,j_1]}\neg\varphi) \mapsto \mathsf{false}$	$i_0 \leq j_0;\ i_1 \geq j_1$
$(\varphi\, \mathcal{U}_{[i_0,i_1]}\, (\varphi \wedge \psi)) \vee \square_{[i_0,i_1]}\varphi \mapsto \psi\, \mathcal{R}_{[i_0,i_1]}\, \varphi$	none

Fig. 1. A sample of the rewrite set chosen for EqSat. Gray rows denote rewrites defined in previous work [24].

B. Equality Saturation

An e-graph $E = \langle C, N, c_r \rangle$ [35] contains a set C of equivalence classes (e-classes) where each e-class $c \in C$ represents a set of e-nodes that are in N and $c_r \in C$ is the *root* e-class. We denote the elements of c as $nodes(c) \subseteq N$. An e-node $n = f(c_1, \ldots, c_{|f|})$ is a function symbol f applied to a list of $|f|$ child e-classes. We denote the set of child e-classes of n as $children(n) = c_1, \ldots, c_{|f|}$. An e-class c represents a term t if any e-node $n \in nodes(c)$ represents t and an e-node $f(c_1, \ldots, c_{|f|})$ represents $t = f(t_1, \ldots, t_{|f|})$ if c_i represents t_i for each $i \in [1, n]$. Importantly, any two terms in the same e-class are *equivalent*. The e-class that an e-node n belongs to is denoted $class(n) \in C$.

Given a set of rewrites $\mathfrak{R}$ with $(t_1 \mapsto t_2) \in \mathfrak{R}$, equality saturation (EqSat) is the technique of applying all valid rewrites to the terms of E until no more rewrites can be applied. The e-graph is then considered *saturated*. Once an e-graph is saturated, one can *extract* from the e-graph an optimal term that is equivalent to the input by constructing a term represented by the root e-class c_r that minimizes a language-specific cost function. For a detailed formalization of e-graphs, EqSat, and extraction, see [46], [49].

We note here that one advantage of EqSat over traditional term rewriting systems for optimization is that it sidesteps confluence: the e-graph maintains a set of all previously-seen terms regardless of rewrite order. Termination is still a desirable property in both contexts [41].

III. Adapting MLTL Rewrites for EqSat

While previous work has explored MLTL rewrites for reducing minimum memory requirements for R2U2 monitors, the rewrites were not designed for EqSat. One desirable property of a rewrite set $\mathfrak{R}$ for EqSat is enabling exploration of as many relevant $\mathfrak{R}$-derivable formulas as possible without blowing up the search space. This might include rewrites that actually *decrease* the target metric to unlock the application of otherwise unapplicable rewrites. Another desirable property is compactness, i.e., fewer, more general rewrites ensure the pattern-matching behavior of EqSat runs efficiently. For example, the rewrite $\Diamond_{[1,1]}\Diamond_{[1,1]}\varphi \mapsto \Diamond_{[2,2]}\varphi$ can be captured more generally with $\Diamond_{[i_0,i_1]}\Diamond_{[j_0,j_1]}\varphi \mapsto \Diamond_{[i_0+j_0,i_1+j_1]}\varphi$.

A. MLTL Rewrites for EqSat

Previous work introduced 15 rewrites for MLTL formulas (denoted *general rewrites*) and then proved each rewrite is semantics-preserving and reduces or maintains the R2U2 resource requirements, i.e., $\mathsf{MemDAG}(\psi) \leq \mathsf{MemDAG}(\varphi)$ for each rewrite $\varphi \mapsto \psi$ [24]. These proofs were hand-written and did not offer an intelligent way to apply the rewrites.

Recent work defined a set of 157 rewrites (denoted *FRET rewrites*) tailored to formulas derived from the structured natural language supported by the NASA-developed Formal Requirements Elicitation Tool (FRET) [1]. Many of these are instances of the 15 general rewrites, such as in the example demonstrating compactness above, or are derivable via some combination of the general rewrites. 101 of the 157 are "scoped" rewrites that are specific to the structure of FRET requirements and are not easily generalizable.

We therefore combine and extend both sets to obtain 69 MLTL rewrites, including 17 propositional and 52 temporal rewrites. This set contains the general rewrites, generalizes the rewrites in the FRET set (except for the "scoped" rewrites), and adds completely novel rewrites. Figure 1 provides a sample of this set. The appendix contains the full set. While we do not show the set to be *terminating*, experiments show that EqSat always terminates with this rewrite set (without associative/commutative rewrites).

B. Associativity and Commutativity

Associative and commutative (AC) operators (such as $\wedge/\vee$) can be problematic when considering rewrites (in fact, the AC matching problem [15]). For example, we cannot apply the simplifying rewrite

$$(\varphi\, \mathcal{U}_{[i_0,i_1]}\, (\varphi \wedge \psi)) \vee \square_{[i_0,i_1]}\varphi \mapsto \psi\, \mathcal{R}_{[i_0,i_1]}\, \varphi$$

to the formula $\square_{[0,10]}p \vee (p\, \mathcal{U}_{[0,10]}\, (p \wedge q))$ because it does not match the left-hand side of the rewrite exactly. We can address this in EqSat by introducing a copy of the simplifying rewrite with the operands of $\vee$ swapped. Alternatively, to avoid having to introduce duplicate commutative versions of all rules in our rewrite set, we include the commutative rewrite $\varphi \vee \psi \mapsto \psi \vee \varphi$. This flexibility comes at a cost: the number of formulas derivable (and therefore number of e-nodes introduced during EqSat) via AC rules is exponential

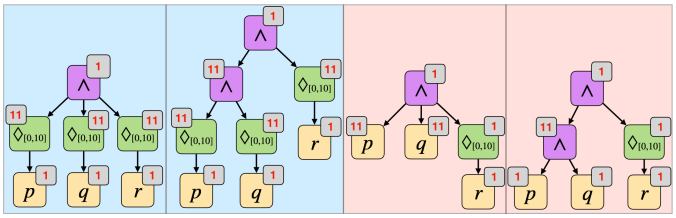

Fig. 2. (**Left, blue**) The first formula uses a 3-arity $\wedge$ to represent $\Diamond_{[0,10]}p \wedge \Diamond_{[0,10]}q \wedge \Diamond_{[0,10]}r$ where the second uses only a 2-arity $\wedge$. The red text aside each node denotes $\mathsf{Mem}(\varphi)$. The first formula has a total Mem of 37 and the second 48. (**Right, red**) Similarly, the first formula uses a 3-arity $\wedge$ to represent $p \wedge q \wedge \Diamond_{[0,10]}r$ where the second uses only a 2-arity $\wedge$. This time the first formula has a total Mem of 25 and the second only 16.

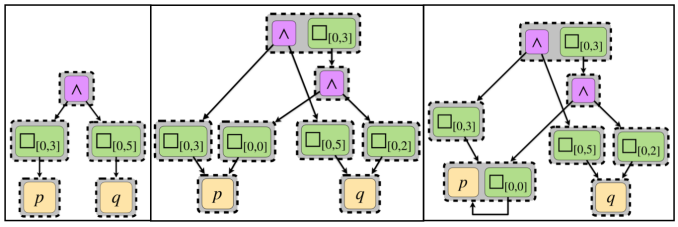

Fig. 3. The e-graph representations of $\Box_{[0,3]}p \wedge \Box_{[0,5]}q$ include e-classes (grey boxes, dashed outline) and e-nodes (colored boxes). **(Left)** The initial e-graph representation of $\Box_{[0,3]}p \wedge \Box_{[0,5]}q$ constructs one e-class for each e-node. **(Middle)** The e-graph after applying $\Box_{[0,3]}p \wedge \Box_{[0,5]}q \mapsto \Box_{0,3}(\Box_{[0,0]}p \wedge \Box_{[0,2]}q)$ creates new nodes for the new operators and places the top-level operators for the original and rewritten formulas in the same e-class. **(Right)** The e-graph after applying $\Box_{[0,0]}p \mapsto p$ merges two e-classes since the rewrite implies the terms' equivalence. The self-loop on the e-class represents the unbounded application of $\Box_{[0,0]}$ to p as in $\Box_{[0,0]}\Box_{[0,0]} \ldots \Box_{[0,0]}p$. This e-graph is *saturated* and ready for extraction.

in the number of arguments to the AC operators. We offer the ability to enable/disable AC rewrites; experiments (Section VI) show that while enabling AC rewrites may result in memory reduction, the computational cost is very high.

C. Multi-Arity Operators

Conjunction and disjunction admit definitions with variable numbers of operands (i.e., arities). For example, the formula $p \wedge q \wedge r$ can have an AST representation using two $\wedge$ operators with two children each (i.e., $\text{AND}(p, \text{AND}(q, r))$) or a single $\wedge$ operator with three children (i.e., $\text{AND}(p, q, r)$). Supporting multi-arity $\wedge/\vee$ can lead to memory savings, although not universally, as Figure 2 demonstrates [27]. We do not consider higher arities as this significantly degrades performance of EqSat, similar to AC rules, though in theory this could be arbitrarily high.

D. Cycle Removal

A cycle in an e-graph represents an unbounded application of the operators in the cycle. A cyclic term is invalid either because terms must be finitely representable or because it would be non-optimal because our aim is to minimize the extracted formula's cost. Because any e-node generated by a rewrite that introduces a cycle will be of lower cost than the original e-node, we mark such rewrites as *subsuming*. A subsuming rewrite removes the original e-node from the e-graph during EqSat (a feature supported by EGGLOG). We detect such rewrites by checking if the rewritten term is in the closure of the original formula, for example, $\Box_{[0,0]}\varphi \mapsto \varphi$ is such that $\varphi \in cl(\Box_{[0,0]}\varphi)$. Cycles may also appear during EqSat due to other e-class merges, however, so while this method may not remove all cycles from the final e-graph, it will reduce its size.

E. Example Computation of MLTL EqSat

We now show an example of how EqSat applies to MLTL. Consider the MLTL formula $\xi = \Box_{[0,3]}p \wedge \Box_{[0,5]}q$ and the pair of rewrites $\mathfrak{R} = \{\Box_{[0,0]}\varphi \mapsto \varphi, \Box_{[i_0,i_1]}\varphi \wedge \Box_{[j_0,j_1]}\psi \mapsto \Box_{[k_0,k_1]}(\Box_{[i_0-k_0,i_1-k_1]}\psi \wedge \Box_{[j_0-k_0,j_1-k_1]}\psi)\}$ with $k_0 = min(i_0, j_0)$ and $k_1 = k_0 + min(i_1 - i_0, j_1 - j_1)$. Figure 3 shows the progression of the e-graph for ξ when applying $\mathfrak{R}$. The right-most e-graph is *saturated* (i.e., no rewrite in $\mathfrak{R}$ is applicable); the top e-class therefore represents all MLTL formulas equivalent to ξ derivable via $\mathfrak{R}$.

IV. EXTRACTION

After performing EqSat, we must then extract from the saturated e-graph an optimized MLTL formula equivalent to the input. Others have explored extraction via reductions to satisfiability checking [28], multi-objective optimization [47], and Integer Linear Programming (ILP) [43], [48]. We present both an ILP method as well as a cheaper, greedy extraction method.

A. ILP-based Extraction

Let $E = \langle C, N, c_r \rangle$ be an e-graph. We introduce sets of Boolean variables to denote whether an e-node or e-class is active for each e-node and e-class of E:

$$B_N = \{b_n \mid n \in N\} \qquad B_C = \{b_c \mid c \in C\}$$

where $b_n, b_c \in \{0, 1\}$ for each $b_n \in \mathbb{B}_N$, $b_c \in \mathbb{B}_C$. Next we introduce sets of integer variables for wpd, bpd, total cost (Mem), and a topological ordering of e-classes:

$$I_N^{wpd} = \{i_n^w \mid n \in N\}, \qquad I_N^{bpd} = \{i_n^b \mid n \in N\},$$
$$I_C^{wpd} = \{i_c^w \mid c \in C\}, \qquad I_C^{bpd} = \{i_c^b \mid c \in C\},$$
$$I_N^{\mathsf{Mem}} = \{i_n^m \mid n \in N\}, \qquad I_C^T = \{i_c^t \mid c \in C\}$$

The complete set of variables we declare for our ILP problem is then

$$B_N \cup B_C \cup I_N^{wpd} \cup I_N^{bpd} \cup I_C^{wpd} \cup I_C^{bpd} \cup I_N^{\mathsf{Mem}} \cup I_C^T.$$

We now define some helper functions to make defining our constraints simpler. Let $n = f(c_1, \ldots, c_{|f|}) \in N$ and $c \in C$. We define $parents(n)$ as the set of e-nodes whose children include $class(n)$ and $sibs(n)$ as the set of all e-nodes that share a parent e-class p with n, but are not in the same e-class as n:

$$parents(n) = \{g(c'_1, \ldots, c'_{|g|}) \mid class(n) \in \{c'_1, \ldots, c'_{|g|}\}\}$$
$$sibs(n) = \{(s, p) \mid s \in nodes(c), c \in children(p), p \in parents(n), c \neq class(n)\}.$$

$$\textbf{Root} := (b_{c_r} = 1) \quad \textbf{Active}_N := \bigwedge_{n \in N} \left(b_n \rightarrow \bigwedge_{c \in c_1, \ldots, c_{|f|}} b_c \right) \quad \textbf{Active}_C := \bigwedge_{c \in C} \left(b_c = \sum_{n \in nodes(c)} b_n \right)$$

$$\textbf{Topo} := \bigwedge_{(c,n,c_n) \in T} i^t_c > b_{c_n} \cdot i^t_{c_n} \text{ where } T = \{(c, n, c_n) \mid c \in C, n \in nodes(c), c_n \in children(n)\}$$

$$\textbf{BPD}_N := \bigwedge_{n \in N} \left(i^b_n = min(i^b_c \mid c \in children(n)) + lb(n) \right) \quad \textbf{WPD}_N := \bigwedge_{n \in N} (i^w_n = max(i^w_c \mid c \in children(n)) + ub(n))$$

$$\textbf{BPD}_C := \bigwedge_{c \in C} \left(i^b_c = \sum_{n \in nodes(c)} b_n \cdot i^b_n \right) \qquad \textbf{WPD}_C := \bigwedge_{c \in C} \left(i^w_c = \sum_{n \in nodes(c)} b_n \cdot i^w_n \right)$$

$$\textbf{Mem}_N := \bigwedge_{n \in N} \left(i^m_n = max(max\{b_s \cdot b_p \cdot i^w_s \mid (s, p) \in sibs(n)\} - i^b_n, 0) + 1 \right)$$

$$\textbf{Valid} := \textbf{Root} \wedge \textbf{Active}_N \wedge \textbf{Active}_C \wedge \textbf{Topo} \qquad \textbf{Minimal} := \textbf{BPD}_N \wedge \textbf{WPD}_N \wedge \textbf{BPD}_C \wedge \textbf{WPD}_C \wedge \textbf{Mem}_N$$

Fig. 4. The constraints for the ILP encoding include validity constraints (first two rows) and minimality constraints (next three rows). For $\textbf{BPD}_N$/$\textbf{WPD}_N$ with $n = f(c_1, \ldots, c_{|f|})$, we define $lb(n) = i_0$ and $ub(n) = i_1$ if $f \in \{\Diamond_{[i_0,i_1]}, \Box_{[i_0,i_1]}, \mathcal{U}_{[i_0,i_1]}, \mathcal{R}[i_0, i_1]\}$, otherwise $lb(n) = ub(n) = 0$.

Next, we present a set of named constraints for our ILP encoding in Figure 4 and explain each of their meanings here:

- **Root:** The root e-class must be active.
- **Active**$_N$, **Active**$_C$**:** If an e-node $n = f(c_1, \ldots, c_{|f|})$ is active then each of $c_1, \ldots, c_{|f|}$ must be active and if c is active then exactly one of $n \in nodes(c)$ must be active, otherwise none shall be active.
- **Topo:** To ensure the extracted term is valid we must ensure there are no cycles (e.g., disallow infinite applications of $\Box_{[0,0]}$ or $\neg$). We therefore add a constraint that asserts a topological ordering on the active e-classes.
- **BPD**$_N$, **WPD**$_N$**:** The propagation delays for each e-node n and e-class c are such that i^b_n is the minimum bpd and i^w_n is the maximum wpd of its child e-classes, corresponding to Equation 3.
- **BPD**$_C$, **WPD**$_C$**:** The bpd and wpd for an e-class c is the bpd and wpd of the active e-node in c where we can take the sum of all active e-node propagation delays since at most one e-node will be active.
- **Mem**$_N$**:** We constrain the possible values for the total memory of each e-node, which reflects Equation 1. When computing the sibling of n with the maximum wpd, the optimizer will only consider active sibling nodes whose shared parent with n is also active.
- **Valid**, **Minimal:** We categorize the constraints as either *validity* (ensuring we can extract a valid MLTL formula) or *minimality* (defining our objective function).

The *objective function* then minimizes the total cost of all active e-nodes. We define the complete ILP problem as

$$\text{minimize} \sum_{n \in N} b_n \cdot i^m_n \text{ s.t. } \textbf{Valid} \wedge \textbf{Minimal}.$$

After solving the ILP problem, we have an assignment to each Boolean variable defined for each e-node and e-class that satisfies the constraints and is minimal with respect to the objective function. Let $repr(c) = n \in N$ be the *representative* of e-class $c \in C$ where $n \in nodes(c)$ and $b_n = 1$. Only one such n will exist due to **Active**$_C$. We define the recursive extraction function $extr(c)$ that returns an MLTL formula for an e-class c. Given an e-class c, let $repr(c) = f(c_1, \ldots, c_{|f|})$, then:

$$extr(c) = f(extr(c_1), \ldots, extr(c_{|f|})).$$

Then the final, output formula of the extraction is $extr(c_r)$.

For maximum transparency, we focus on the practical aspects of the work and experimentally validate the minimality of the approach in lieu of a proof that the encoding is minimal. For each formula φ of the 1,177 formulas we evaluated, the ILP-extracted formula ψ had either equal or lower values compared to MemDAG(φ). Proving that $extr(c_r)$ is minimal is challenging due to the structure of the constraints on the e-graph. Simply inducting on the structure of $extr(c_r)$ or the e-graph is not sufficient since the memory for an arbitrary subformula $\varphi \in cl(extr(c_r))$ relies on all nodes it shares parents with while also impacting those nodes' memories as well. A potential proof strategy may be to assume that some other formula φ represented by c_r has a smaller MemDAG, then inducting on both φ and $extr(c_r)$ and showing that each subformula must have equal MemDAG. We leave the theoretical proof of the general-case minimality of the ILP encoding to future work.

B. Greedy Extraction

Section VI will demonstrate that extraction via ILP is often harder than EqSat itself. We therefore define an alternative greedy extraction method. The key to this method is to define a *local* cost function.

Definition 4 (Local e-node Functions)**.** Let $E = \langle C, N, c_r \rangle$ be an e-graph. A function $cost(f(c_1, \ldots, c_{|f|}))$ is *local* if its definition depends only on f and $c_1, \ldots, c_{|f|}$. A function $cost(c)$ is *local* if its definition depends only on c.

We define a local cost function for each e-node, compute the e-node in each e-class with the minimal cost, then perform a greedy search of the e-graph starting from the root e-class. This is the default extraction method natively supported by EGGLOG. The drawback to this method is that it does not consider node sharing; the cost of each e-node is counted as many times as it appears in the output term.

Equation 1 is not local, since it refers to nodes' siblings, not just children. To address this, we first define *approximate* values for wpd and bpd of e-nodes and e-classes, similar to Definition 3. Let $lb(f) = a$ and $ub(f) = b$ if f is a temporal operator with interval $[a, b]$, otherwise $lb(f) = ub(f) = 0$. The approximate bpd' and wpd' values for an e-node $n = f(c_1, \ldots, c_{|f|})$ are

$$bpd'(n) = min(bpd'(c_i) \mid c_i \in c_1, \ldots, c_{|f|}) + lb(f),$$
$$wpd'(n) = max(wpd'(c_i) \mid c_i \in c_1, \ldots, c_{|f|}) + ub(f).$$

The approximate bpd' and wpd' of an e-class $c \in C$ are

$$bpd'(c) = max(bpd'(n) \mid n \in nodes(c)),$$
$$wpd'(c) = min(wpd'(n) \mid n \in nodes(c)).$$

Essentially, we take the *tightest* propagation delays of all nodes in an e-class (highest bpd, lowest wpd). This may not correspond to an actual e-node in the e-graph, however. For example, if the e-nodes that represent formulas $\Diamond_{[2,4]}$true and $\Diamond_{[3,5]}$true are in the same e-class c then $bpd(c) = 3$ and $wpd(c) = 4$, despite no such formula existing. Crucially, all four of these functions are *local*.

We then define an auxiliary cost function $cost'$ for e-nodes and e-classes. The cost of an e-class c is the cost of the e-node in $nodes(c)$ with minimal cost:

$$cost'(c) = min\{cost'(n) \mid n \in nodes(c)\}.$$

We then define $cost'(n)$ with $n = f(c_1, \ldots, c_{|f|})$ as follows:

$$\sum_{c \in c_1, \ldots, c_{|f|}} max\{wpd'(c_i) \mid c_i \in \{c_1, \ldots, c_{|f|}\} \setminus \{c\}\} - bpd'(c) + cost'(c).$$

Again, crucially, both these functions are local.

The cost function for an e-node in the root e-class $n \in nodes(c_r)$ is $cost(n) = cost'(n) + 1$. This approximates the minimum value of Mem(φ) for formulas represented by the e-class $class(n)$. Intuitively, $cost'$ operates the same as Equation 1, except that it adds (an approximation of) Mem(ψ) to ψ's parent instead of assigning it to ψ.

Theorem 1 ($cost$ Locality)**.** *$cost(f(c_1, \ldots, c_{|f|}))$ is a local function, i.e., its definition depends only on f and $c_1, \ldots, c_{|f|}$.*

The proof follows since all supporting functions are local. The $cost$-extracted formula may have a Mem that is *higher* than the input formula φ, since we assume ψ is a DAG. Essentially, it's possible that the extracted formula rewrites some shared sub-formula $\psi \in cl(\varphi)$ to a representation with equal MemDAG but keeps ψ un-rewritten in some other part of φ. This blocks the output formula from leveraging node sharing of ψ, resulting in a formula with higher MemDAG(φ) overall. Experiments show that this infrequently occurs in practice (16 out of 1,177 formulas), but one can simply discard the optimization in these cases.

```
(sort IntvlSort)
(function Intvl (i64 i64) IntvlSort)
(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 IntvlSort MLTL)
  (Future IntvlSort MLTL)
  (Until IntvlSort MLTL MLTL)
  (Release IntvlSort MLTL MLTL)
)

(rewrite ; Constant folding
  (And2 (Bool true) p) p
)

; G[i_0,i_1] G[j_0,j_1] p |-> G[i_0+j_0,i_1+j_1] p
(rewrite
 (Global (Intvl a b) (Global (Intvl c d) p))
 (Global (Intvl (+ a c) (+ b d)) p)
)

; G[i_0,i_1] p | G[j_0,j_1] p |-> G[i_0,i_1] p
; if a <= c and b >= d
(rewrite
 (Or2 (Global (Intvl a b) p) (Global (Intvl c d) p))
 (Global (Intvl a b) a)
 :when ((<= a c) (>= b d))
 :subsume
)
```

Fig. 5. The EGGLOG encoding defines a datatype `MLTL` and rewrites for that datatype. Rewrite rules include a term to pattern match against, a term to rewrite to, and an optional condition that must hold to apply the rule. The final rewrite is *subsuming* since $\Box_{[i_0,i_1]}\varphi \in cl(\Box_{[i_0,i_1]}\varphi \vee \Box_{[j_0,j_1]}\varphi)$.

V. R2U2 RESOURCE-OPTIMIZING TOOLCHAIN

We implemented our methods in a new toolchain within the formula compiler for R2U2, the Configuration Compiler for Property Organization (C2PO) [24] [1] [2]. Given a set of rewrites and an input formula, the toolchain proves that all rewrites are valid (i.e., semantics-preserving), then performs EqSat and either ILP or greedy extraction.

The toolchain uses EGGLOG for performing EqSat. Figure 5 gives our encoding of MLTL and a selection of rewrites in EGGLOG. When using the greedy extraction method, the toolchain also defines *analyses* for computing bpd, wpd, and $cost$ in the EGGLOG encoding and has EGGLOG perform extraction via $cost$. Analyses are a feature of EGGLOG that allow for computing local cost functions based on simple, syntactic properties of the term language.

For the ILP extraction method, the toolchain has EGGLOG output the saturated e-graph in JSON, then encodes that e-graph into an ILP problem. The toolchain uses GUROBI [19] for ILP solving.

[1] Updated code available at https://github.com/R2U2/r2u2/.

[2] Artifact available at https://doi.org/10.5281/zenodo.21521555.

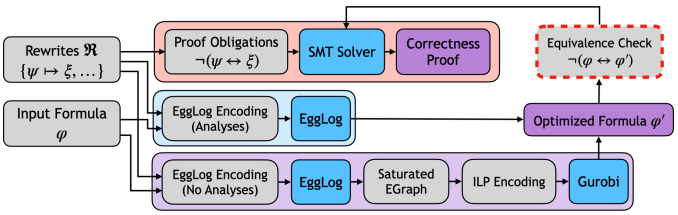

Fig. 6. The implemented toolchain starts with a set of rewrites $\mathfrak{R}$ and an MLTL formula φ (left) and outputs a set of correctness proofs for $\mathfrak{R}$ and an optimized formula (purple boxes). Blue boxes denote external tools. Upon a change to $\mathfrak{R}$, it proves each rewrite in $\mathfrak{R}$ correct (top, shaded red background). For EqSat and extraction, the greedy approach encodes the entire problem into EGGLOG, which produces an optimized formula (middle, shaded blue). The ILP approach encodes just the EqSat problem into EGGLOG to obtain a saturated e-graph, then extracts a formula by solving the ILP encoding using GUROBI (bottom, shaded purple). Optionally, the toolchain checks that the output formula is equivalent to the input via a satisfiability query (dashed red box).

***Proving Correctness of* $\mathfrak{R}$.** Prior work presented an automated technique for proving the correctness of a rewrite $\varphi \mapsto \psi$ [1]. A rewrite is correct if $\varphi \equiv \psi$, which can be proven by showing that the MLTL formula $\neg(\varphi \leftrightarrow \psi)$ is unsatisfiable (along with constraints on any symbolic interval values). Let $\mathsf{SAT}(\varphi)$ represent the First Order Logic (FOL) encoding for checking the satisfiability of φ [31]. Then, for example, the rewrite $\Diamond_{[i_0,i_1]}\Diamond_{[j_0,j_1]}\varphi \mapsto \Diamond_{[i_0+j_0,i_1+j_1]}\varphi$ is correct if the FOL formula

$$\exists i_0, i_1, j_0, j_1.\mathsf{SAT}\left(\neg\left(\Diamond_{[i_0,i_1]}\Diamond_{[j_0,j_1]}\varphi \leftrightarrow \Diamond_{[i_0+j_0,i_1+j_1]}\varphi\right)\right)$$

is unsatisfiable. Since this method proves $\varphi \equiv \psi$, we can soundly apply any rewrite in either direction (i.e., either $\varphi \mapsto \psi$ or $\psi \mapsto \varphi$). The toolchain uses CVC5 [5] and Z3 [14].

Of the 69 rewrites from Section III, the toolchain automatically proved 62 correct within a 10 minute timeout per instance. The following 7 rewrites timed out:

1) $\Box_{[i_0,i_1]}\Box_{[j_0,j_1]}\varphi \mapsto \Box_{[i_0+j_0,i_1+j_1]}\varphi$
2) $\Diamond_{[i_0,i_1]}\Diamond_{[j_0,j_1]}\varphi \mapsto \Diamond_{[i_0+j_0,i_1+j_1]}\varphi$
3) $\Diamond_{[i_0,i_1]}\varphi \vee \Diamond_{[j_0,j_1]}\psi \mapsto \Diamond_{[k_0,k_1]}(\Diamond_{[i_0-k_0,i_1-k_1]}\varphi \vee \Diamond_{[j_0-k_0,j_1-k_1]}\psi)$ with $k_0 = min(i_0, j_0)$ and $k_1 = k_0 + min(i_1 - i_0, j_1 - j_0)$
4) $\Box_{[i,i]}\Diamond_{j_0,j_1}\varphi \mapsto \Diamond_{[i+j_0,i+j_1]}\varphi$
5) $(\Diamond_{[i,i]}\varphi)\ \mathcal{U}_{[j_0,j_1]}\ (\Diamond_{[i,i]}\psi) \mapsto \varphi\ \mathcal{U}_{[i+j_0,i+j_1]}\ \psi$
6) $(\Diamond_{[i,i]}\varphi)\ \mathcal{R}_{[j_0,j_1]}\ (\Diamond_{[i,i]}\psi) \mapsto \varphi\ \mathcal{R}_{[i+j_0,i+j_1]}\ \psi$
7) $(\varphi\ \mathcal{U}_{[i_0,i_1]}\ (\psi \wedge \psi)) \vee \Box_{[i_0,i_1]}\psi \mapsto \varphi\ \mathcal{R}_{[i_0,i_1]}\ \psi$

We suspect the FOL encoding of these formulas' encodings have alternating quantifier structures that are particularly challenging for CVC5 and Z3. It is also of note that CVC5 and Z3 proved other rules similar to those above, such as the rewrite $\Diamond_{[i,i]}\Box_{[j_0,j_1]}\varphi \mapsto \Box_{[i+j_0,i+j_1]}\varphi$ (see unproven rewrite 4). We emphasize these 7 are not *incorrect*, they merely timed out during the automated equivalence check.

VI. Experimental Evaluation of Optimization Strategies

We ran experiments on the Nova HPC cluster at Iowa State University. Each node used AMD EPYC 9655 processors with a minimum of 64 cores and 1.5TB per node running RHEL 9.6. For each benchmark, we ran with exclusive access to a node and a maximum time of 1 hour and memory of 64GB. That is, both EGGLOG and GUROBI could each use up to 1 hour and 64GB before a time/memory-out.

A. Benchmarks

We compiled benchmark sets of previously-developed MLTL formulas from various sources for evaluation.

- **Boeing WBS [7]**: 360 MLTL formulas derived from a set of real-world LTL formulas for verifying a Boeing Wheel Brake System (WBS) design with interval bounds placed on each temporal operator.
- **NASA ATC [16]:** 60 MLTL formulas derived from a set of real-world LTL formulas for verifying a NASA Air-Traffic Control (ATC) design with interval bounds placed on each temporal operator.
- **UTM [8]:** 124 MLTL formulas from a set of MLTL specifications for an automated UAS Traffic Manager (UTM).
- **MAVLink [33]:** 7 MLTL formulas specifying behavior to catch dangerous MAVLink Protocol commands and a Denial of Service hijack attack.
- **Fluxgate [17]:** 6 MLTL formulas that monitor for a fluxgate magnetometer buffer overflow error on an in-flight UAS.

We also contribute a new benchmark set of 620 MLTL formulas derived from 68 MTL specification patterns [4]. Every unbounded MTL operator (e.g., $p\ \mathcal{U}\ q$) had $[0, M]$ bounds added. Each pattern then had up to three symbolic interval values (i_0, i_1, and M), from which we generated 5 formulas with $(i_0, i_1, M) \in \{(0, 10, 10), (0, 100, 100), (0, 1000, 1000), (10, 100, 100), (100, 1000, 1000)\}$. 14 patterns included *precedence chains*, a sequence of propositions must hold within some response time from each other. For example, the formula for a chain of length 3 is $p \wedge \Diamond_{[0,i_1]}(q \wedge \Diamond_{[0,i_1]}r)$. We generated 5 variants of each of these formulas, with chains from length 1 to 5. In total, we collected 1,177 MLTL formulas.

B. Experimental Setup

To evaluate our proposed methods, we pose the following research questions:

RQ1 What method is most effective for reducing memory for R2U2 monitors while balancing computational cost?
RQ2 How do AC rewrites affect EqSat and extraction performance?
RQ3 How does each method affect the worst-case execution time (WCET) of R2U2 monitors?
RQ4 How does each method affect the memory requirements for other MLTL monitoring techniques?
RQ5 Is there a relationship between e-graph size and performance for ILP extraction?

To support answering these questions, we ran our toolchain over each benchmark in four different modes: ILP extraction with AC (I-AC), ILP extraction without AC (I-No-AC), greedy

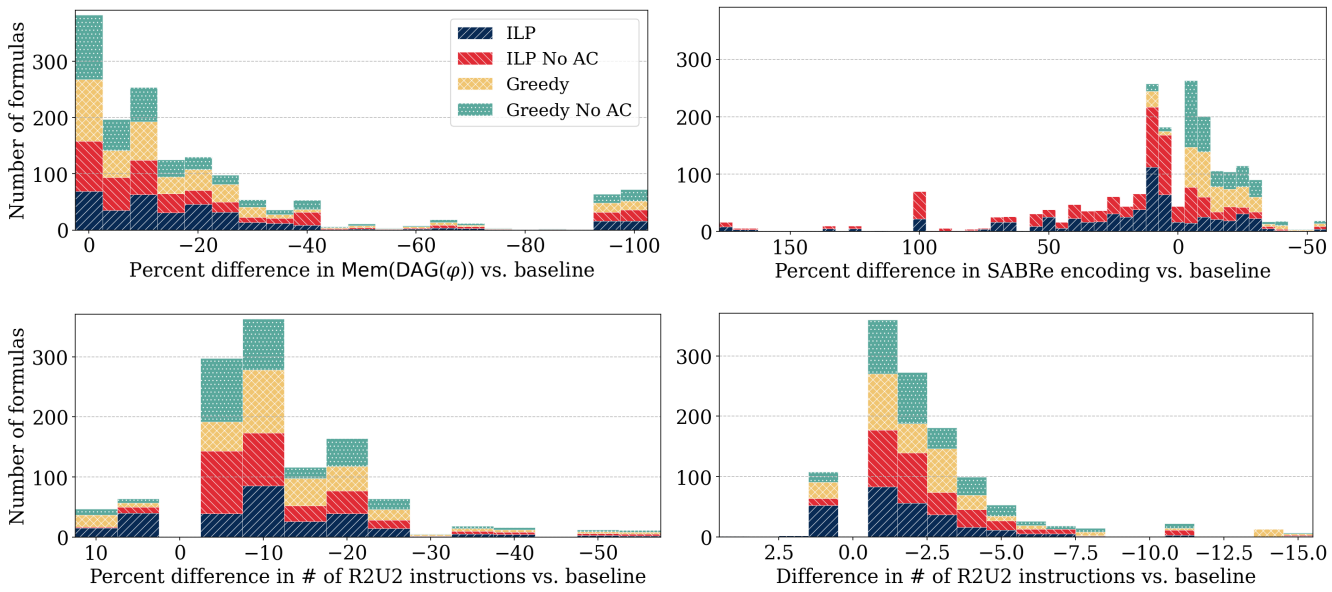

Fig. 7. Histograms show the distribution of formulas across multiple metrics for all four modes compared to the baseline. ***For clarity, the plots omit all formulas where a metric was equal to the baseline after optimization, i.e., when their difference was 0.*** All percentages were rounded to the nearest 5%, after omitting zero-change formulas. More negative values (towards the right) are favorable compared to the baseline for all metrics. (**Top left**) All four modes have roughly the same distribution in terms of percent difference in MemDAG versus the baseline. While most see small changes below 20%, some formulas see drops in MemDAG as high as 90+%. (**Top right**) The ILP methods struggled compared to the greedy methods for reducing the size of SABRE monitor encodings. (**Bottom**) All modes reduce the number of R2U2 instructions for a given formula most of the time, but occasionally see an increase by a modest amount.

extraction with AC (G-AC), and greedy extraction without AC (G-No-AC). We measure the total time to encode and run EGGLOG (which includes both EqSat and extraction for the greedy method) and GUROBI (only for ILP method). We also measure the percent reduction between the baseline formula φ and the extracted formula ψ for each benchmark (more negative is better):

$$\frac{\mathsf{MemDAG}(\psi) - \mathsf{MemDAG}(\varphi)}{\mathsf{MemDAG}(\varphi)} \cdot 100.$$

In practice, C2PO emits an R2U2-specific assembly language that R2U2 executes to monitor a given MLTL formula. To compare WCET between formulas, we therefore measure the number of R2U2 instructions generated for the original and optimized formulas. The lower the number of instructions, the less time needed to evaluate each timestep, and the lower the WCET. We do not compute concrete WCETs (e.g., in terms of clock cycles) as each would be platform-specific.

To see how the technique applies to other monitors beyond R2U2, we consider the MLTL monitoring algorithm/tool SABRE [26]. C2PO can generate SABRE monitors from MLTL formulas in the form of executable C code. SABRE monitors allocate the same amount of memory for all nodes in the DAG of the input formula φ, essentially 1 bit per value in $wpd(\varphi)$. We also report the actual total allocated size in bytes for each SABRE monitor.

Finally, we also report the size of the final e-graphs after EqSat for each formula as reported by EGGLOG (in number of e-nodes and e-classes).

C. Results and Discussion

RQ1: On balance, the greedy heuristic extraction method without AC rewrites (G-No-AC) is the most effective mode, as Figure 8 (left) and the MemDAG column of Figure 9 show: G-No-AC is very cheap to run (0.029s average runtime) with no time/memory-out while still providing similar memory reduction to the other modes. We do note that 16 (G-No-AC) formulas saw an *increase* in MemDAG (see end of Section IV), but in those cases, the user can simply disregard the resulting formula. All modes contribute to the virtual best (Min. MemDAG column of Figure 9), demonstrating that each mode has advantages over the others in some situations. While on average the expensive I-AC mode saw lower reductions in MemDAG, the formulas that did see a reduction were reduced the most (Figure 9, column Δ MemDAG vs Baseline avg $\% > 0\%$). Figure 7 illustrates the distribution of memory reductions compared to the baseline, where most formulas saw a modest reduction $\leq 20\%$, but some saw reductions as high as $90 + \%$ (these contained large conjunctions of $\square_{[i_0,i_1]}\varphi$).

RQ2: Enabling AC rewrites increases the optimization runtime, frequently causing time/memory-outs (Figure 8, right), and did not drastically improve memory reduction. The No-AC modes see no memory-outs from EGGLOG, demonstrating the cost of enabling AC for EqSat, and the ILP-No-AC mode saw 186 fewer GUROBI timeouts than ILP-AC (Figure 9). However, enabling AC rewrites can still be useful as it can often lead to some reductions in MemDAG compared to the No-AC modes.

RQ3: All modes successfully reduce the worst-case execution

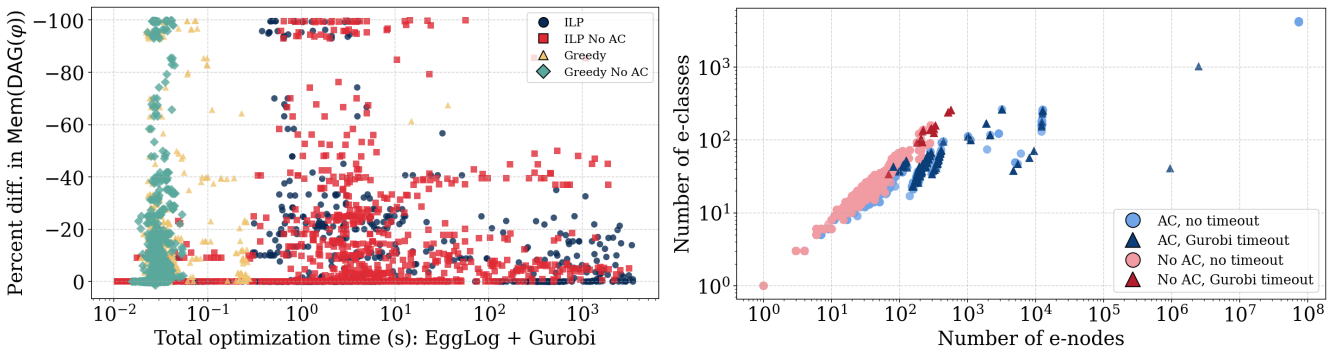

Fig. 8. (**Left**) The percent difference in MemDAG (higher is better) versus optimization time (left is better) shows that the greedy methods achieve similar decreases in MemDAG to the ILP methods while incurring a fraction of the computational cost. (**Right**) For the ILP methods, the size of the e-graph correlates with whether GUROBI timed out on instances, where the AC instances tended to have larger e-graphs. $\sim 95\%$ of the instances with e-node and e-class counts less than 100 did not timeout.

Method	MemDAG			Δ MemDAG vs Baseline					# R2U2 Instr.		# SABRE Bytes	
	avg	median	# min	avg %	avg % $< 0\%$	# $< 0\%$	# $= 0\%$	# $> 0\%$	avg	median	avg	median
Baseline	3999.55	308	0	N/A	N/A	N/A	N/A	N/A	21.23	16	6250.91	966
I-AC	3875.74	265	44	6.92%	22.73%	357	815	**0**	20.81	15	6099.95	1138
I-No-AC	**3490.01**	**233**	80	**11.37%**	**22.98%**	580	592	**0**	20.27	15	6325.03	1108
G-AC	3727.20	238.5	**114**	9.49%	17.60%	**632**	540	**0**	20.02	**14**	**5156.24**	965.5
G-No-AC	3758.52	**233**	40	10.14%	19.23%	618	**538**	16	**19.74**	15	5258.63	**915.5**
VB	3409.28	231	N/A	13.72%	22.34%	720	452	0	19.38	14	5053.35	915.5

Fig. 9. Comparison of various metrics across the four modes compared to the baseline. The VB row reports the virtual best *for that column*, e.g., computes the average MemDAG when considering only the method with the lowest MemDAG and separately for number of R2U2 instructions and SABRE bytes. The bolded value in each column is the *best* for that metric (non VB). The first group of three columns (MemDAG) shows the average, median and number of formulas that were strictly minimal compared to all other methods (e.g., for 35 formulas I-AC resulted in a strictly lower MemDAG than any other method). The second group of five columns (Δ MemDAG vs Baseline) shows various metrics for the percent difference between that mode and the baseline. The avg % column gives the overall average percent reduction, the avg % $< 0\%$ shows the average only including formulas with some reduction. The next columns show the number of formulas with some reduction ($< 0\%$), no reduction ($= 0\%$), or increase ($> 0\%$) in MemDAG from the baseline. The final two groups give the average and median number of R2U2 instructions emitted for that formula and the size of the SABRE encoding in bytes.

Method	Egglog Time	Gurobi Time	Par-2 Score
	avg (# MO)	avg (# TO)	avg
I-AC	3.109s (75)	170.528s (233)	2402.898s
I-No-AC	0.013s (0)	61.772s (47)	348.044s
G-AC	0.089s (85)	N/A	522.267s
G-No-AC	0.029s (0)	N/A	0.029s

Fig. 10. Average runtimes and Par-2 scores for the four modes show that the G-No-AC approach is the cheapest approach. The first and second columns are average runtimes for all non-time/memory-out instances. (#TO/#MO) indicates the number of time/memory-outs. EGGLOG never experienced a timeout and GUROBI never experienced a memory-out. The third column is the Par-2 score for each method, computed as the average of all instances with time/memory-outs counting as 2 times the timeout, in this case $2 \cdot 3600s = 7200s$. Both AC modes and No-AC modes use the same rewrites for EqSat respectively. We believe that G-AC saw 10 more MOs than I-AC despite using the same rewrites because G-AC also uses EGGLOG for extraction, meaning more data is embedded into the e-graph during EqSat (e.g., $wpd'(n)$ for each e-node n).

time (WCET) of R2U2 monitors by decreasing the number of emitted instructions. The bottom histograms in Figure 7 show that the majority of formulas that had any difference from the baseline saw reductions in the number of instructions. Although mostly modest reductions ($\sim$1-4 instructions removed), some formulas saw up to 14 instructions removed.

RQ4: The optimizations extend beyond R2U2 and reduce the memory requirements for other monitoring tools like SABRE in many instances. Figure 9 details this improvement, showing the average SABRE encoding size in bytes drop from 6250.91 (baseline) to around 5200 for the greedy methods. Figure 7 (top right) shows that the ILP methods do not perform well with respect to SABRE monitor size, sometimes more than doubling the encoding size, although the greedy methods perform decently.

RQ5: There is a direct negative correlation between e-graph size and ILP extraction performance. The Figure 8 (right) visually confirms this relationship: as the number of e-nodes approaches 10^3 and beyond, GUROBI timeouts become frequent (marked by triangles), particularly when using AC rewrites. Across all four modes, roughly 71% of all formulas had e-graphs with less than 100 e-nodes and 100 e-classes, of which only around 5% experienced a time/memory-out.

VII. CONCLUSION

We now have multiple ways to optimize an MLTL formula for RV monitoring, starting with EqSat then extracting with either the ILP or greedy method. If the ILP method is too expensive or an ILP solver is not available, the greedy method is still effective and very cheap. In the rare case the greedy method produces a formula more expensive than the input, one can drop the optimization and use the input formula. Our

results demonstrate that the presented toolchain addresses the problem of configuring RV monitors on resource-constrained systems by automatically reducing the resource requirements for MLTL formulas, without input from domain engineers.

Our work enables exploring similar optimizations for other temporal logics such as LTL, LTL_f, STL, etc. and applying similar approaches to other problems such as model checking or synthesis. For instance, applying EqSat to temporal logic formulas with the objective of minimizing their automata representations may prove effective. We may also consider extending the method to richer logics that include type information on the temporal operators (e.g., MLTL Multi-type [21]) where type information might enable further, domain-specific rewrites. This approach may also be useful for optimizing other temporal logic-based monitors such as AERIAL [6] or HYDRA [39]. While the AC rules were often prohibitively expensive, we might explore relaxing the saturation part of EqSat, i.e., running the rewrites over the e-graph for a bounded number of iterations instead.

REFERENCES

[1] Aurandt, A., Johannsen, C., Katis, A., Mavridou, A., Rozier, K.Y., Jones, P.H.: From Natural Language Requirements to Runtime Monitors for Resource-Constrained Systems: Integrating FRET and R2U2 (Under Submission) (2026)

[2] Aurandt, A., Jones, P.H., Rozier, K.Y.: Runtime verification triggers real-time, autonomous fault recovery on the cysat-i. In: Proceedings of the 14th NASA Formal Methods Symposium (NFM 2022). Lecture Notes in Computer Science (LNCS), vol. 13260. Springer, Cham, Caltech, California, USA (May 2022). https://doi.org/10.1007/978-3-031-06773-0_45

[3] Aurandt, A., Jones, P.H., Rozier, K.Y.: Towards a safe, verified runtime monitor for embedded systems: R2u2 in embedded rust. In: NASA Formal Methods Symposium. pp. 31–53. Springer (2025)

[4] Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., Tang, A.: Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. IEEE Transactions on Software Engineering **41**(7), 620–638 (2015)

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

[6] Basin, D.A., Krstic, S., Traytel, D.: Aerial: Almost event-rate independent algorithms for monitoring metric regular properties. In: RV-CuBES. pp. 29–36 (2017)

[7] Bozzano, M., Cimatti, A., Fernandes Pires, A., Jones, D., Kimberly, G., Petri, T., Robinson, R., Tonetta, S.: 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. pp. 518–535. Springer (2015)

[8] Cauwels, M., Hammer, A., Hertz, B., Jones, P.H., Rozier, K.Y.: Integrating runtime verification into an automated uas traffic management system. In: European Conference on Software Architecture. pp. 340–357. Springer (2020)

[9] Chen, C., Hu, G., Zuo, D., Yu, C., Ma, Y., Zhang, H.: E-syn: E-graph rewriting with technology-aware cost functions for logic synthesis. In: Proceedings of the 61st ACM/IEEE Design Automation Conference. pp. 1–6 (2024)

[10] Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: Tessla: temporal stream-based specification language. In: Brazilian Symposium on Formal Methods. pp. 144–162. Springer (2018)

[11] Coward, S., Constantinides, G.A., Drane, T.: Automating constraint-aware datapath optimization using e-graphs. In: 2023 60th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2023)

[12] Dabney, J.B., Badger, J.M., Rajagopal, P.: Adding a verification view for an autonomous real-time system architecture. In: Proceedings of SciTech Forum. p. Online. 2021-0566, AIAA (January 2021). https://doi.org/https://doi.org/10.2514/6.2021-0566

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

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

[15] Eker, S.: Associative-commutative rewriting on large terms. In: International Conference on Rewriting Techniques and Applications. pp. 14–29. Springer (2003)

[16] Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., Rozier, K.Y.: Model checking at scale: Automated air traffic control design space exploration. In: International Conference on Computer Aided Verification. pp. 3–22. Springer (2016)

[17] Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and bayesian network reasoners on-board fpgas: flight-certifiable system health management for embedded systems. In: International Conference on Runtime Verification. pp. 215–230. Springer (2014)

[18] Giannakopoulou, D., Mavridou, A., Rhein, J., Pressburger, T., Schumann, J., Shi, N.: Formal requirements elicitation with FRET. In: International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020). No. ARC-E-DAA-TN77785 (2020)

[19] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), https://www.gurobi.com

[20] Hammer, A., Cauwels, M., Hertz, B., Jones, P.H., Rozier, K.Y.: Integrating runtime verification into an automated uas traffic management system. Innovations in Systems and Software Engineering: A NASA Journal (July 2021). https://doi.org/10.1007/s11334-021-00407-5

[21] Hariharan, G., Kempa, B., Wongpiromsarn, T., Jones, P., Rozier, K.: Mltl multi-type: A typed logic for cyber-physical systems. ACM Transactions on Embedded Computing Systems **25**(2), 1–22 (2026)

[22] Hertz, B., Luppen, Z., Rozier, K.Y.: Integrating runtime verification into a sounding rocket control system. In: NASA Formal Methods Symposium. pp. 151–159. Springer (2021)

[23] Huang, X., Seyster, J., Callanan, S., Dixit, K., Grosu, R., Smolka, S.A., Stoller, S.D., Zadok, E.: Software monitoring with controllable overhead. International Journal on Software Tools for Technology Transfer **14**(3), 327–347 (2012)

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

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

[26] Johannsen, C., Jones, P.H., Rozier, K.Y., Wongpiromsarn, T.: Scalable MLTL Runtime Monitoring and Satisfiability via Bit-Vector Encoding. In: Formal Methods in Computer-Aided Design (FMCAD). TU Wien Academic Press (2025)

[27] Johannsen, C.G.: Dynamic set reasoning: Specifying and optimizing monitor encodings. Master's thesis, Iowa State University (2024)

[28] Joshi, R., Nelson, G., Randall, K.: Denali: A goal-directed super-optimizer. In: Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation. p. 304–314. PLDI '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/512529.512566, https://doi.org/10.1145/512529.512566

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

[30] Kosaian, K., Wang, Z., Sloan, E., Rozier, K.Y.: Formalizing mltl formula progression in isabelle/hol. In: International Conference on Intelligent Computer Mathematics. pp. 371–391. Springer (2025)

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

[32] Medhat, R., Bonakdarpour, B., Kumar, D., Fischmeister, S.: Runtime monitoring of cyber-physical systems under timing and memory constraints. ACM Transactions on Embedded Computing Systems (TECS) **14**(4), 1–29 (2015)

[33] Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2u2: monitoring and diagnosis of security threats for unmanned aerial systems. Formal Methods in System Design **51**(1), 31–61 (2017)

[34] Nelson, C.G.: Techniques for Program Verification. Ph.D. thesis, Stanford, CA, USA (1980), aAI8011683

[35] Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. Journal of the ACM (JACM) **27**(2), 356–364 (1980)

[36] Okubo, N.: Using R2U2 in JAXA program. Electronic correspondence (November–December 2020), series of emails and zoom call from JAXA to PI with technical questions about embedding R2U2 into an autonomous satellite mission with a provable memory bound of 200KB

[37] Panchekha, P., Sanchez-Stern, A., Wilcox, J.R., Tatlock, Z.: Automatically improving accuracy for floating point expressions. Acm Sigplan Notices **50**(6), 1–11 (2015)

[38] Perez, I., Dedden, F., Goodloe, A.: Copilot 3. Tech. rep. (2020)

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

[40] Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 357–372. Springer (2014)

[41] Suciu, D., Wang, Y.R., Zhang, Y.: Semantic foundations of equality saturation. arXiv preprint arXiv:2501.02413 (2025)

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

[43] Wang, Y.R., Hutchison, S., Leang, J., Howe, B., Suciu, D.: Spores: sum-product optimization via relational equality saturation for large scale linear algebra. arXiv preprint arXiv:2002.07951 (2020)

[44] Wang, Z., Guzman, L.P.G., Rozier, K.Y.: West: Interactive validation of mission-time linear temporal logic (mltl). Science of Computer Programming p. 103365 (2025)

[45] Wang, Z., Rosentrater, A.E., Aurandt, A., Johannsen, C., Rozier, K.Y.: An Integrated Ecosystem for Mission-time LTL Specification Elicitation and Validation. In: International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation (To Appear) (2026)

[46] Willsey, M., Nandi, C., Wang, Y.R., Flatt, O., Tatlock, Z., Panchekha, P.: Egg: Fast and extensible equality saturation. In: Proceedings of the ACM on Programming Languages (POPL). vol. 5, pp. 1–29. ACM New York, NY, USA (2021)

[47] Wu, C., Zhao, H., Nandi, C., Lipton, J.I., Tatlock, Z., Schulz, A.: Carpentry compiler. ACM Transactions on Graphics (TOG) **38**(6), 1–14 (2019)

[48] Yang, Y., Phothilimthana, P., Wang, Y., Willsey, M., Roy, S., Pienaar, J.: Equality saturation for tensor graph superoptimization. Proceedings of Machine Learning and Systems **3**, 255–268 (2021)

[49] Zhang, Y., Wang, Y.R., Flatt, O., Cao, D., Zucker, P., Rosenthal, E., Tatlock, Z., Willsey, M.: Better together: Unifying datalog and equality saturation. vol. 7. Association for Computing Machinery, New York, NY, USA (jun 2023). https://doi.org/10.1145/3591239, https://doi.org/10.1145/3591239

APPENDIX

Rewrite	Constraints	Bi-directional
$\neg true \mapsto false$	none	✗
$\neg false \mapsto true$	none	✗
$\varphi \wedge true \mapsto \varphi$	none	✗
$\varphi \wedge false \mapsto false$	none	✗
$\varphi \vee true \mapsto true$	none	✗
$\varphi \vee false \mapsto \varphi$	none	✗
$\neg(\varphi \wedge \psi) \mapsto (\neg\varphi) \vee (\neg\psi)$	none	✓
$\neg(\varphi \vee \psi) \mapsto (\neg\varphi) \wedge (\neg\psi)$	none	✓
$(\varphi \wedge (\psi \vee \xi)) \mapsto ((\varphi \wedge \psi) \vee (\varphi \wedge \xi))$	none	✗
$(\varphi \vee (\psi \wedge \xi)) \mapsto ((\varphi \vee \psi) \wedge (\varphi \vee \xi))$	none	✗
$\varphi \rightarrow \psi \mapsto (\neg\varphi) \vee \psi$	none	✓
$\varphi \leftrightarrow \psi \mapsto (\varphi \wedge \psi) \vee ((\neg\varphi) \wedge (\neg\psi))$	none	✓
$\varphi \wedge (\neg\varphi) \mapsto false$	none	✗
$\varphi \vee (\neg\varphi) \mapsto true$	none	✗
$\neg(\neg\varphi) \mapsto \varphi$	none	✗
$\varphi \vee \varphi \mapsto \varphi$	none	✗
$\varphi \wedge \varphi \mapsto \varphi$	none	✗
$(\Box_{[i_0,i_1]}\ \varphi) \vee (\psi\ \mathcal{R}_{[i_2,i_3]}\ \varphi) \mapsto (\Box_{[i_0,i_1]}\ \varphi) \vee (\psi\ \mathcal{R}_{[i_2,i_1]}\ \varphi)$	$(i_1 \leq i_3); (i_2 \leq i_0)$	✗
$(\varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi) \wedge (\Diamond_{[i_2,i_3]}\ \psi) \mapsto \varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi$	$(i_1 \leq i_3); (i_2 \leq i_0)$	✗
$(\Box_{[i_0,i_1]}\ \neg\varphi) \wedge (\Diamond_{[i_2,i_3]}\ \varphi) \mapsto (\Box_{[i_0,i_1]}\ \neg\varphi) \wedge (\Diamond_{[(i_1+1),i_3]}\ \varphi)$	$(i_0 \leq i_2); (i_2 \leq (i_1+1)); (i_1 < i_3)$	✗
$(\Box_{[i_0,i_1]}\ \neg\varphi) \wedge (\Diamond_{[i_2,i_3]}\ \varphi) \mapsto (\Box_{[i_0,i_1]}\ \neg\varphi) \wedge (\Diamond_{[i_2,i_0-1]}\ \varphi)$	$(i_2 < i_0); (i_0 \leq i_3); (i_3 \leq i_1)$	✗
$(\Box_{[i_0,i_1]}\ \neg\varphi) \wedge (\Diamond_{[i_2,i_3]}\ \varphi) \mapsto false$	$(i_0 \leq i_2); (i_3 \leq i_1)$	✗
$(\Box_{[i_0,i_1]}\ \varphi) \wedge (\Diamond_{[i_2,i_3]}\ \neg\varphi) \mapsto false$	$(i_0 \leq i_2); (i_3 \leq i_1)$	✗
$(\varphi \wedge \Diamond_{[i_0,i_1]}\ \psi) \rightarrow (\xi\ \mathcal{U}_{[i_0,i_2]}\ \psi \vee \Box_{[i_3,i_4]}\ \xi) \mapsto (\varphi \wedge \Diamond_{[i_0,i_1]}\ \psi) \rightarrow (\xi\ \mathcal{U}_{[i_0,i_2]}\ \psi)$	$i_2 \geq i_1; i_3 \leq i_0; i_4 \geq i_1$	✗
$(\Diamond_{[i_0,i_1]}\ \psi) \rightarrow ((\varphi\ \mathcal{U}_{[i_0,i_2]}\ \psi) \vee \Box_{[i_3,i_4]}\ \varphi) \mapsto (\Diamond_{[i_0,i_1]}\ \psi) \rightarrow (\varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi)$	$i_2 \geq i_1; i_3 \leq i_0; i_4 \geq i_1$	✗
$(\Diamond_{[i_0,i_1]}\ \psi) \wedge ((\varphi\ \mathcal{U}_{[i_0,i_2]}\ \psi) \vee \Box_{[i_3,i_4]}\ \varphi) \mapsto (\Diamond_{[i_0,i_1]}\ \psi) \wedge (\varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi)$	$i_2 \geq i_1; i_3 \leq i_0; i_4 \geq i_1$	✗
$(\psi\ \mathcal{U}_{[i_0,i_1]}\ (\varphi \wedge \psi)) \vee \Box_{[i_0,i_1]}\ \psi \mapsto \varphi\ \mathcal{R}_{[i_0,i_1]}\ \psi$	none	✗
$(\Box_{[i_0,i_1]}\ \varphi) \wedge (\Box_{[i_2,i_3]}\ \psi) \mapsto \Box_{[i_4,i_5]}\ ((\Box_{[i_0-i_4,i_1-i_5]}\ \varphi) \wedge (\Box_{[i_2-i_4,i_3-i_5]}\ \psi))$	$i_4 = min(i_0, i_2); i_5 = i_4 + min(i_1 - i_0, i_3 - i_2)$	✗
$(\Diamond_{[i_0,i_1]}\ \varphi) \vee (\Diamond_{[i_2,i_3]}\ \psi) \mapsto \Diamond_{[i_4,i_5]}\ ((\Diamond_{[i_0-i_4,i_1-i_5]}\ \varphi) \vee (\Diamond_{[i_2-i_4,i_3-i_5]}\ \psi))$	$i_4 = min(i_0, i_2); i_5 = i_4 + min(i_1 - i_0, i_3 - i_2)$	✗
$\Box_{[i_0,i_1]}\ \Diamond_{[i_2,i_2]}\ \varphi \mapsto \Box_{[(i_2+i_0),(i_2+i_1)]}\ \varphi$	none	✗
$\Diamond_{[i_0,i_1]}\ \Box_{[i_2,i_2]}\ \varphi \mapsto \Diamond_{[(i_2+i_0),(i_2+i_1)]}\ \varphi$	none	✗
$\Box_{[i_0,i_0]}\ \Diamond_{[i_1,i_2]}\ \varphi \mapsto \Diamond_{[(i_0+i_1),(i_0+i_2)]}\ \varphi$	none	✗
$\Diamond_{[i_0,i_0]}\ \Box_{[i_1,i_2]}\ \varphi \mapsto \Box_{[(i_0+i_1),(i_0+i_2)]}\ \varphi$	none	✗
$\Box_{[i_0,i_1]}\ false \mapsto false$	none	✗
$\Box_{[i_0,i_1]}\ true \mapsto true$	none	✗
$\Diamond_{[i_0,i_1]}\ false \mapsto false$	none	✗
$\Diamond_{[i_0,i_1]}\ true \mapsto true$	none	✗
$(\Diamond_{[i_0,i_1]}\ \varphi) \wedge (\Diamond_{[i_2,i_3]}\ \psi) \mapsto \Diamond_{[i_4,i_4]}\ ((\Diamond_{[(i_0-i_4),(i_1-i_4)]}\ \varphi) \wedge (\Diamond_{[(i_2-i_4),(i_3-i_4)]}\ \psi))$	$i_4 = min(i_0, i_2)$	✗
$(\Box_{[i_0,i_1]}\ \varphi) \vee (\Box_{[i_2,i_3]}\ \psi) \mapsto \Box_{[i_4,i_4]}\ ((\Box_{[(i_0-i_4),(i_1-i_4)]}\ \varphi) \vee (\Box_{[(i_2-i_4),(i_3-i_4)]}\ \psi))$	$i_4 = min(i_0, i_2)$	✗
$\Diamond_{[i_0,i_1]}\ \varphi \mapsto \neg(\Box_{[i_0,i_1]}\ \neg\varphi)$	none	✗
$\Box_{[i_0,i_1]}\ \varphi \mapsto \neg(\Diamond_{[i_0,i_1]}\ \neg\varphi)$	none	✗
$\varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi \mapsto \neg(\neg\varphi\ \mathcal{R}_{[i_0,i_1]}\ \neg\psi)$	none	✗
$\varphi\ \mathcal{R}_{[i_0,i_1]}\ \psi \mapsto \neg(\neg\varphi\ \mathcal{U}_{[i_0,i_1]}\ \neg\psi)$	none	✗
$\neg(\Diamond_{[i_0,i_1]}\ \varphi) \mapsto \Box_{[i_0,i_1]}\ (\neg\varphi)$	none	✓
$\neg(\Box_{[i_0,i_1]}\ \varphi) \mapsto \Diamond_{[i_0,i_1]}\ (\neg\varphi)$	none	✓
$\neg(\varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi) \mapsto (\neg\varphi)\ \mathcal{R}_{[i_0,i_1]}\ (\neg\psi)$	none	✓
$\neg(\varphi\ \mathcal{R}_{[i_0,i_1]}\ \psi) \mapsto (\neg\varphi)\ \mathcal{U}_{[i_0,i_1]}\ (\neg\psi)$	none	✓
$\Box_{[i_0,i_1]}\ \Box_{[i_2,i_3]}\ \varphi \mapsto \Box_{[(i_0+i_2),(i_1+i_3)]}\ \varphi$	none	✗
$\Diamond_{[i_0,i_1]}\ \Diamond_{[i_2,i_3]}\ \varphi \mapsto \Diamond_{[(i_0+i_2),(i_1+i_3)]}\ \varphi$	none	✗
$(\Box_{[i_0,i_1]}\ \varphi) \wedge (\Box_{[i_2,i_3]}\ \varphi) \mapsto \Box_{[i_0,max(i_1,i_3)]}\ \varphi$	$(i_0 \leq i_2); (i_2 \leq (i_1+1))$	✗
$(\Diamond_{[i_0,i_1]}\ \varphi) \vee (\Diamond_{[i_2,i_3]}\ \varphi) \mapsto \Diamond_{[i_0,max(i_1,i_3)]}\ \varphi$	$(i_0 \leq i_2); (i_2 \leq (i_1+1))$	✗
$(\Box_{[i_0,i_1]}\ \varphi) \vee (\Box_{[i_2,i_3]}\ \varphi) \mapsto \Box_{[i_2,i_3]}\ \varphi$	$(i_2 \geq i_0); (i_3 \leq i_1)$	✗
$(\Diamond_{[i_0,i_1]}\ \varphi) \wedge (\Diamond_{[i_2,i_3]}\ \varphi) \mapsto \Diamond_{[i_2,i_3]}\ \varphi$	$(i_2 \geq i_0); (i_3 \leq i_1)$	✗
$(\Diamond_{[i_0,i_1]}\ \varphi) \vee (\Box_{[i_0,i_1]}\ \neg\varphi) \mapsto true$	none	✗
$\Diamond_{[i_0,i_0]}\ \varphi \mapsto \Box_{[i_0,i_0]}\ \varphi$	none	✓
$true\ \mathcal{U}_{[i_0,i_1]}\ \varphi \mapsto \Diamond_{[i_0,i_1]}\ \varphi$	none	✓
$false\ \mathcal{R}_{[i_0,i_1]}\ \varphi \mapsto \Box_{[i_0,i_1]}\ \varphi$	none	✓
$\Diamond_{[i_0,i_0]}\ (\varphi\ \mathcal{U}_{[i_1,i_2]}\ \psi) \mapsto (\varphi\ \mathcal{U}_{[(i_1+i_0),(i_2+i_0)]}\ \psi)$	none	✓
$(\Diamond_{[i_2,i_2]}\ \varphi)\ \mathcal{U}_{[i_0,i_1]}\ (\Diamond_{[i_2,i_2]}\ \psi) \mapsto (\varphi\ \mathcal{U}_{[(i_0+i_2),(i_1+i_2)]}\ \psi)$	none	✓
$\Diamond_{[i_0,i_0]}\ (\varphi\ \mathcal{R}_{[i_1,i_2]}\ \psi) \mapsto (\varphi\ \mathcal{R}_{[(i_1+i_0),(i_2+i_0)]}\ \psi)$	none	✓
$(\Diamond_{[i_2,i_2]}\ \varphi)\ \mathcal{R}_{[i_0,i_1]}\ (\Diamond_{[i_2,i_2]}\ \psi) \mapsto (\varphi\ \mathcal{R}_{[(i_0+i_2),(i_1+i_2)]}\ \psi)$	none	✓
$(\varphi\ \mathcal{U}_{[i_0,i_1]}\ \psi) \wedge (\xi\ \mathcal{U}_{[i_0,i_2]}\ \psi) \mapsto ((\varphi \wedge \xi)\ \mathcal{U}_{[i_0,i_1]}\ \psi)$	$i_1 \leq i_2$	✗
$\varphi\ \mathcal{U}_{[i_0,i_1]}\ (\Box_{[0,i_2]}\ \varphi) \mapsto \Box_{[i_0,(i_0+i_2)]}\ \varphi$	none	✗
$\varphi\ \mathcal{U}_{[i_0,i_1]}\ (\Diamond_{[0,i_2]}\ \varphi) \mapsto \Diamond_{[i_0,(i_0+i_2)]}\ \varphi$	none	✗
$\varphi\ \mathcal{U}_{[i_0,i_0]}\ \psi \mapsto \Diamond_{[i_0,i_0]}\ \psi$	none	✗
$\Box_{[0,0]}\ \varphi \mapsto \varphi$	none	✗
$\Diamond_{[0,0]}\ \varphi \mapsto \varphi$	none	✗
$\varphi\ \mathcal{U}_{[0,0]}\ \psi \mapsto \psi$	none	✗
$\varphi\ \mathcal{R}_{[0,0]}\ \psi \mapsto \psi$	none	✗