Towards a Safe, Verified Runtime Monitor for Embedded Systems: R2U2 in Embedded Rust*

Alexis Aurandt[0000−0003−2008−673X],
Phillip H. Jones[0000−0002−8220−7552], and Kristin Yvonne Rozier[0000−0002−6718−2828]

Iowa State University, USA {aurandt,phjones,kyrozier}@iastate.edu

Abstract. Stream-based runtime monitors can effectively verify specified system behavior in a real-time online manner, but the effectiveness of these monitors relies heavily on complying with the system's timing and resource constraints and the correctness of the monitor's implementation. The R2U2 runtime monitoring framework provides real-time guarantees and a resource-aware architecture; however, we further reduce R2U2's overhead by optimizing both Mission-time Linear Temporal Logic (MLTL) and past-time MLTL (ptMLTL) operators and their corresponding instruction formats. We evaluate our optimizations on a suite of benchmarks and observe a significant decrease in latency and memory requirements. To improve the correctness guarantees of R2U2, we manually transpile the previous C version to *safe* embedded Rust and verify the correctness with hand-constructed proofs, testing, and code verification with Verus. We specifically target safe *embedded* Rust (i.e., no_std) to allow for deployment on embedded platforms with bare-metal environments (e.g., microcontrollers), and we provide complete proofs for all of R2U2's operators and verify the Rust code implementation of 25 of these operators with Verus code contracts.

Keywords: Stream-based Runtime Monitor · Mission-time Linear Temporal Logic · Past-time Mission-time Linear Temporal Logic · R2U2 · Embedded Rust · Verus.

1 Introduction

Stream-based runtime monitors analyze an input stream of system data against a set of system requirements and produce an output stream of corresponding verdicts. These monitors enable the detection of requirement violations in a real-time online manner, enabling appropriate mitigation actions to be taken [8,22]. When online runtime monitors are executing onboard an existing system, they must fit within the system's timing and resource constraints to produce verdicts in real time. The integration of runtime monitors also raises the question, "How can we trust that the runtime monitor is correct?" as incorrect violation detections can negatively impact a system [9,23].

The R2U2 (Realizable, Responsive, Unobtrusive Unit) stream-based monitoring framework is known for its real-time guarantees and resource-aware architecture [26,29,42,45], and the success of R2U2 has been exemplified by its deployment on several mission-critical, resource-constrained, real-time systems [5,13,15,24,25,29,44]. R2U2 supports both past-time and future-time monitoring, but most of R2U2's recent optimizations have only been applied to R2U2's future-time monitoring capabilities (e.g., [26,29,42]). R2U2 natively encodes Mission-time Linear Temporal Logic (MLTL) as its future-time logic but encoded ptMTL [2,32], instead of ptMLTL, as its past-time logic. To aid in consistency and decrease the resource overhead of R2U2's past-time monitoring, we formally define ptMLTL and encode ptMLTL operators similar to R2U2's latest future-time optimizations in [29]. Furthermore, we decrease R2U2's resource overhead by optimizing which MLTL and ptMLTL operators are directly encoded

* Supported by NSF:CPS Award 2038903. Additional details available at *https://temporallogic.org/research/R2U2Rust*.

in R2U2 and which are encoded via their semantic equivalent, and we further reduce memory requirements by refining the format of the instructions R2U2 reasons over.

R2U2's software implementation was previously only written in C. Given that both C and C++ are unsafe languages that contain several memory safety vulnerabilities (e.g., [18,35,50]), R2U2's C implementation is susceptible to memory safety issues, e.g., reading or writing beyond array bounds. Embedded Rust has become a popular alternative to C to eradicate such issues with C code [39,47,48]. Therefore, to avoid the memory safety vulnerabilities that exist in R2U2's C implementation, we manually transpile from C to *safe* embedded Rust.

To the best of our knowledge, RTLola [11,21] and TeSSLa [28] are the only other runtime verification frameworks that support a Rust implementation, but both of these tools only compile to Rust code that utilizes the Rust standard library (i.e., std). Rust's standard library requires an operating system, which isn't feasible on embedded systems with bare-metal environments as required by some applications [5,19,24]; therefore, we specifically target *embedded* Rust with no_std, which allows for systems with or without an operating system to utilize R2U2 [1].

To verify monitor correctness, the VeriMon [9,10,43] and Vydra [40] monitors were formalized and verified in Isabelle/HOL and are executable using Isabelle-generated OCaml code. In [12] and [14], executable OCaml code was extracted from a runtime monitor specified in Coq. While utilizing proof assistants to produce verified monitors ensures correctness, these techniques automatically generate code in languages not typically utilized for real deployment (e.g., OCaml, Scala, Haskell) and may lack in optimizations that a human programmer would implement. On the other hand, the developers of Copilot recently introduced the CopilotVerifier, which generates What4 SMT queries to provide a mathematical proof verifying that the original Copilot specification and the compiled C monitor are bisimilar [46]. Additionally, Lola specifications can be generated into imperative Rust code and verified with generated Prusti code contracts in [21]. Both Copilot and Lola compile specification(s) into a monitor in an imperative language (i.e., C or Rust) such that these techniques require verification of any newly compiled monitor, which can be costly as in [21]. R2U2 differs in that it is a static monitor that can interpret any MLTL/ptMLTL specification(s) at runtime, which makes it "more challenging" and "problematic" to verify [21].

To verify the correctness of R2U2's implementation, we construct complete proofs for all of R2U2's directly encoded operators. Through hand-constructing these proofs, we found several errors in the previous implementation of R2U2, which have now been corrected. To verify the correctness of the Rust code implementation, we verify that the Rust code matches all conditions in the proof using deductive code verification. We examined utilizing Creusot [17], Prusti [3,4], and Verus [33,34] for verifying R2U2's code and decided upon utilizing Verus due to its applicability and usability in verifying R2U2's implementation.

Our contributions include **(1)** syntax, semantics, and propagation delay semantics of ptMLTL (Section 2), **(2)** new encoding of ptMLTL operators (Section 3.3), **(3)** complete proofs for all of R2U2's directly encoded operators (Section 3.2 and 3.3) **(4)** new implementation of R2U2 in safe embedded Rust as a publicly available crate, [1] **(5)** significant latency and memory reductions of R2U2 (Section 3, 3.4, and 3.5), and **(6)** deductive code verification with Verus (Section 4).

2 R2U2 Overview

2.1 Mission-Time Linear Temporal Logic (MLTL) [36,42]

MLTL (or ptMLTL) is a variant of LTL (or ptLTL) over finite traces with temporal intervals that are bounded, closed, and discrete. MLTL and ptMLTL express the most commonly utilized future and past-time fragments of MTL [2,32] and STL [37].

[1] *https://crates.io/crates/r2u2_core*

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

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

where $p \in \mathcal{AP}$ is an atom, ψ and ξ are MLTL formulas, and I is a closed interval $[lb,ub]$ where lb and ub denote the lower and upper bound, respectively, such that $lb \leq ub$ and $lb,ub \in \mathbb{N}_0$.

Definition 2. *(ptMLTL Syntax)* The syntax of a ptMLTL formula φ over a set of atomic propositions $\mathcal{AP}$ is recursively defined as:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid p \mid \neg\psi \mid \psi \wedge \xi \mid \psi \vee \xi \mid \boxminus_I \psi \mid \diamondminus_I \psi \mid \psi \,\mathcal{S}_I\, \xi \mid \psi \,\mathcal{T}_I\, \xi$$

where $p \in \mathcal{AP}$ is an atom, ψ and ξ are ptMLTL formulas, and I is a closed interval $[lb,ub]$ where lb and ub denote the lower and upper bound, respectively, such that $lb \leq ub$ and $lb,ub \in \mathbb{N}_0$.

Definition 3. *(Finite Trace)* A finite trace, denoted by π, is a finite sequence of sets of atomic propositions. The i^{th} set is denoted by $\pi(i)$ and contains the atomic propositions that are satisfied at the i^{th} time step. $|\pi|$ denotes the length of π (where $|\pi| < \infty$), and $\pi[lb,ub]$ denotes the trace segment $\pi(lb),\pi(lb+1),...,\pi(ub)$.

Definition 4. *(MLTL Semantics)* We recursively define $\pi,i \models \varphi$ (finite trace π starting from time index $i \geq 0$ satisfies, or "models" MLTL formula φ) as

- $\pi,i \models \mathsf{true}$
- $\pi,i \models p$ for $p \in \mathcal{AP}$ iff $p \in \pi(i)$
- $\pi,i \models \neg\psi$ iff $\pi,i \not\models \psi$
- $\pi,i \models \psi \wedge \xi$ iff $\pi,i \models \psi$ and $\pi,i \models \xi$
- $\pi,i \models \psi \vee \xi$ iff $\pi,i \models \psi$ or $\pi,i \models \xi$
- $\pi,i \models \Box_{[lb,ub]} \psi$ iff $|\pi| \leq i+lb$ [2] or $\forall j \in [i+lb,i+ub]$, $\pi,j \models \psi$
- $\pi,i \models \Diamond_{[lb,ub]} \psi$ iff $|\pi| > i+lb$ [2] and $\exists j \in [i+lb,i+ub]$ such that $\pi,j \models \psi$
- $\pi,i \models \psi \,\mathcal{U}_{[lb,ub]}\, \xi$ iff $|\pi| > i+lb$ [2] and $\exists j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$ and $\forall k < j$ where $k \in [i+lb,i+ub]$, $\pi,k \models \psi$
- $\pi,i \models \psi \,\mathcal{R}_{[lb,ub]}\, \xi$ iff $|\pi| \leq i+lb$ [2] or if $\exists j \in [i+lb,i+ub]$ where $\pi,j \not\models \xi$, then $\exists k < j$ where $k \in [i+lb,i+ub]$ such that $\pi,k \models \psi$

Given two MLTL formulas ψ and ξ, they are semantically equivalent (denoted by $\psi \equiv \xi$) if and only if $\pi \models \psi \Leftrightarrow \pi \models \xi$ for all traces π. MLTL also keeps the standard operator equivalences from LTL, including $\mathsf{false} \equiv \neg\mathsf{true}$, $\psi \vee \xi \equiv \neg(\neg\psi \wedge \neg\xi)$, $\psi \rightarrow \xi \equiv \neg\psi \vee \xi$, $\psi \leftrightarrow \xi \equiv \neg(\psi \oplus \xi)$, $\neg(\psi \,\mathcal{U}_I\, \xi) \equiv (\neg\psi \,\mathcal{R}_I\, \neg\xi)$, $\neg\Diamond_I \psi \equiv \Box_I \neg\psi$, $\Diamond_I \psi \equiv (\mathsf{true} \,\mathcal{U}_I\, \psi)$, and $\Box_I \psi \equiv (\mathsf{false} \,\mathcal{R}_I\, \psi)$. Notably, MLTL discards the next ($\bigcirc$) operator since $\bigcirc\psi \equiv \Box_{[1,1]}\psi$.

Axiom 1. *(Early Evaluation of Until Operator)* Following directly from Definition 4, the MLTL formula $\psi \,\mathcal{U}_{[lb,ub]}\, \xi$ can be evaluated based on ξ alone in two cases: (1) if $\pi,i+lb \models \xi$, then $\pi,i \models \psi \,\mathcal{U}_{[lb,ub]}\, \xi$ and (2) if $\nexists j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$, then $\pi,i \not\models \psi \,\mathcal{U}_{[lb,ub]}\, \xi$. Additionally, if $\exists j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$, then $\forall m \geq j$ and $\forall n > j$ where $m,n \in [i+lb,i+ub]$, it is not necessary to know if $\pi,m \models \psi$ and if $\pi,n \models \xi$ to determine if $\pi,i \models \psi \,\mathcal{U}_{[lb,ub]}\, \xi$.

Axiom 2. *(Early Evaluation of Release Operator)* Following directly from Definition 4, the MLTL formula $\psi \mathcal{R}_{[lb,ub]} \xi$ can be evaluated based on ξ alone in two cases: (1) if $\pi,i+lb \not\models \xi$, then $\pi,i \not\models \psi \,\mathcal{R}_{[lb,ub]}\, \xi$ and (2) if $\forall j \in [i+lb,i+ub]$, $\pi,j \models \xi$, then $\pi,i \models \psi \,\mathcal{R}_{[lb,ub]}\, \xi$. Additionally, if $\exists j \in [i+lb,i+ub]$ such that $\pi,j \models \psi$ and $\pi,j \models \xi$, then $\forall m > j$ where $m \in [i+lb,i+ub]$, it is not necessary to know if $\pi,m \models \psi$ and if $\pi,m \models \xi$ to determine if $\pi,i \models \psi \,\mathcal{R}_{[lb,ub]}\, \xi$.

[2] In stream-based monitoring, we always assume that there will be an extension of the trace π.

Definition 5. *(ptMLTL Semantics)* We recursively define $\pi,i \models \varphi$ (finite trace π starting from time index $i \geq 0$ satisfies, or "models" ptMLTL formula φ) as

- $\pi,i \models \mathsf{true}$
- $\pi,i \models p$ for $p \in \mathcal{AP}$ iff $p \in \pi(i)$
- $\pi,i \models \neg\psi$ iff $\pi,i \not\models \psi$
- $\pi,i \models \psi \wedge \xi$ iff $\pi,i \models \psi$ and $\pi,i \models \xi$
- $\pi,i \models \psi \vee \xi$ iff $\pi,i \models \psi$ or $\pi,i \models \xi$
- $\pi,i \models \boxminus_{[lb,ub]} \psi$ iff $|\pi| \leq i - ub$ [2] or $\forall j \in [i-ub, i-lb]$, $\pi,j \models \psi$
- $\pi,i \models \diamondminus_{[lb,ub]} \psi$ iff $|\pi| > i - ub$ [2] and $\exists j \in [i-ub, i-lb]$ such that $\pi,j \models \psi$
- $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ iff $|\pi| > i - ub$ [2] and $\exists j \in [i-ub, i-lb]$ such that $\pi,j \models \xi$ and $\forall k > j$ where $k \in [i-ub, i-lb]$, $\pi,k \models \psi$
- $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ iff $|\pi| \leq i - ub$ [2] or if $\exists j \in [i-ub, i-lb]$ where $\pi,j \not\models \xi$, then $\exists k > j$ where $k \in [i-ub, i-lb]$ such that $\pi,k \models \psi$

Given two ptMLTL formulas ψ and ξ, they are semantically equivalent (denoted by $\psi \equiv \xi$) if and only if $\pi \models \psi \Leftrightarrow \pi \models \xi$ for all traces π. ptMLTL also keeps the standard operator equivalences from ptLTL, including $\mathsf{false} \equiv \neg\mathsf{true}$, $\psi \vee \xi \equiv \neg(\neg\psi \wedge \neg\xi)$, $\psi \rightarrow \xi \equiv \neg\psi \vee \xi$, $\psi \leftrightarrow \xi \equiv \neg(\psi \oplus \xi)$, $\neg\diamondminus_I \psi \equiv \boxminus_I \neg\psi$, $\diamondminus\psi \equiv (\mathsf{true}\ \mathcal{S}_I\ \psi)$, and $\boxminus\psi \equiv (\mathsf{false}\ \mathcal{T}_I\ \psi)$. Notably, ptMLTL discards the previous ($\ominus$) operator since $\ominus\psi \equiv \boxminus_{[1,1]}\psi$.

Axiom 3. *(Early Evaluation of Since Operator)* Following directly from Definition 5, the ptMLTL formula $\psi\ \mathcal{S}_{[lb,ub]}\ \xi$ can be evaluated based on ξ alone in two cases: (1) if $\pi,i-lb \models \xi$, then $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ (2) if $\nexists j \in [i-ub, i-lb]$ such that $\pi,j \models \xi$, then $\pi,i \not\models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$. Additionally, if $\exists j \in [i-ub, i-lb]$ such that $\pi,j \models \xi$ and $\forall k > j$ where $k \in [i-ub, i-lb]$, $\pi,k \models \psi$, then it is not necessary to know if $\pi,k \models \xi$ to determine $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$.

Axiom 4. *(Early Evaluation of Trigger Operator)* Following directly from Definition 5, the ptMLTL formula $\psi\ \mathcal{T}_{[lb,ub]}\ \xi$ can be evaluated based on ξ alone in two cases: (1) if $\pi,i-lb \not\models \xi$, then $\pi,i \not\models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ (2) if $\forall j \in [i-ub, i-lb]$, $\pi,j \models \xi$, then $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$. Additionally, if $\exists j \in [i-ub, i-lb]$ such that $\pi,j \models \psi$ and $\pi,j \models \xi$ and $\forall k > j$ where $k \in [i-ub, i-lb]$, $\pi,k \models \xi$, then it is not necessary to know if $\pi,k \models \psi$ to determine $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$.

2.2 Abstract Syntax Tree Architecture

R2U2 is a stream-based runtime monitor that reevaluates MLTL and ptMLTL formulas for each time index i. The Configuration Compiler for Property Organization (C2PO) [26] compiles MLTL and ptMLTL formula(s) for input into R2U2. C2PO decomposes the MLTL and ptMLTL formula(s) into subformula nodes represented in an Abstract Syntax Tree (AST) and optimizes the AST by applying Common Subexpression Elimination [26,29] and various rewriting rules [27]. C2PO then outputs assembly-style instructions for R2U2 to reason over. Figures 1 and 2 illustrate an AST and the compiled instructions for $(\Box_{[0,3]}\psi)\ \mathcal{U}_{[2,4]}\ \xi$, respectively.

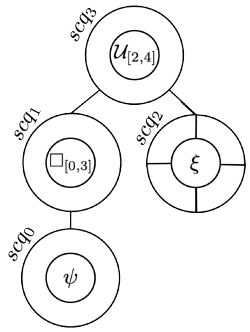


Fig. 1. AST for $(\Box_{[0,3]}\psi)\ \mathcal{U}_{[2,4]}\ \xi$

In the outputted assembly, there are configuration instructions and computation instructions. The configuration instructions are run once upon initialization to configure the AST in terms of sizing and metadata (e.g., the lower and upper bounds of temporal operators). The computation instructions are saved in a table and sequentially iterated over at each timestamp. The computation instructions are ordered such that R2U2 reasons over the AST by determining the evaluation of

each subformula node from the bottom-up and propagating verdicts to the parent node(s).

Each node of the AST computes and stores verdict-timestamp tuples $T_\varphi = (v,\tau)$ for its subformula φ, where $v \in \{\mathsf{true},\mathsf{false}\}$ and $\tau \in \mathbb{N}_0$. Each node stores the verdict-timestamp tuples in a shared connection queue (SCQ), where the SCQ is a circular buffer that overwrites verdict-timestamp tuples in a circular and aggregated manner.

Propogation Delay. To compute the SCQ size of each node in the AST, the propagation delay of each subformula must first be computed.

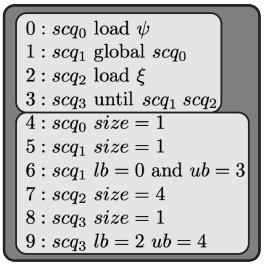


Fig. 2. Instructions compiled by C2PO for $(\Box_{[0,3]}\psi)\ \mathcal{U}_{[2,4]}\ \xi$. Instructions 0–3 are the computation instructions, and instructions 4–9 are the configuration instructions.

Definition 6. *(Propagation Delay [29])* The propagation delay of an MLTL/ptMLTL formula φ is the time between when a set of propositions $\pi(i)$ arrives and when the verdict of $\pi,i \models \varphi$ is determinable. The best-case propagation delay ($\varphi.bpd$) is its minimum time delay, and the worst-case propagation delay ($\varphi.wpd$) is its maximum time delay.

Definition 7. *(MLTL Propagation Delay Semantics [29])* Let ψ and ξ be subformulas of an MLTL formula φ where $\varphi.bpd$ and $\varphi.wpd$ are structurally defined as follows:

$$\bullet\ \varphi \in \mathcal{AP}: \begin{cases} \varphi.wpd = 0 \\ \varphi.bpd = 0 \end{cases} \qquad \bullet\ \varphi = \neg\psi: \begin{cases} \varphi.wpd = \psi.wpd \\ \varphi.bpd = \psi.bpd \end{cases}$$

$$\bullet\ \varphi = \psi \vee \xi \text{ or } \varphi = \psi \wedge \xi: \begin{cases} \varphi.wpd = max(\psi.wpd, \xi.wpd) \\ \varphi.bpd = min(\psi.bpd, \xi.bpd) \end{cases}$$

$$\bullet\ \varphi = \Box_{[lb,ub]}\psi \text{ or } \varphi = \Diamond_{[lb,ub]}\psi: \begin{cases} \varphi.wpd = \psi.wpd + ub \\ \varphi.bpd = \psi.bpd + lb \end{cases}$$

$$\bullet\ \varphi = \psi\ \mathcal{U}_{[lb,ub]}\ \xi \text{ or } \varphi = \psi\ \mathcal{R}_{[lb,ub]}\ \xi: \begin{cases} \varphi.wpd = max(\psi.wpd, \xi.wpd) + ub \\ \varphi.bpd = min(\psi.bpd, \xi.bpd) + lb \end{cases}$$

Definition 8. *(ptMLTL Propagation Delay Semantics)* Let ψ and ξ be subformulas of a ptMLTL formula φ where $\varphi.bpd$ and $\varphi.wpd$ are structurally defined as follows:

$$\bullet\ \varphi \in \mathcal{AP}: \begin{cases} \varphi.wpd = 0 \\ \varphi.bpd = 0 \end{cases} \qquad \bullet\ \varphi = \neg\psi: \begin{cases} \varphi.wpd = \psi.wpd \\ \varphi.bpd = \psi.bpd \end{cases}$$

$$\bullet\ \varphi = \psi \vee \xi \text{ or } \varphi = \psi \wedge \xi: \begin{cases} \varphi.wpd = max(\psi.wpd, \xi.wpd) \\ \varphi.bpd = min(\psi.bpd, \xi.bpd) \end{cases}$$

$$\bullet\ \varphi = \boxminus_{[lb,ub]}\psi \text{ or } \varphi = \diamondminus_{[lb,ub]}\psi: \begin{cases} \varphi.wpd = \psi.wpd - lb \\ \varphi.bpd = \psi.bpd - ub \end{cases}$$

$$\bullet\ \varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi \text{ or } \varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi: \begin{cases} \varphi.wpd = max(\psi.wpd, \xi.wpd) - lb \\ \varphi.bpd = min(\psi.bpd, \xi.bpd) - lb \end{cases}^{3}$$

[3] $\varphi.bpd$ in this case is determined by lb (not ub) as ptMLTL requires either ψ or ξ to be known at $i - lb$ to determine if $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ or $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ according to Axioms 3 and 4, respectively.

SCQ Memory Size. To minimize the required memory resources of R2U2, the SCQs are minimally sized such that the SCQ will never overwrite a verdict-timestamp tuple required by its parent node. The minimum SCQ size of an AST node φ is determined by the worst-case propagation delay of its sibling nodes and its own best-case propagation delay; in the worst case, a node φ must store verdict-timestamp tuples in its SCQ until all of φ's siblings have the same timestamp τ for these tuples to be consumed by their parent node. Therefore, the size of node φ's SCQ corresponds to the maximum timestamp mismatch between node φ and φ's siblings. Let $\mathbb{S}_\varphi$ be the set of all of φ's sibling nodes, then the size of φ's SCQ is given by the following (proof available in [51]):

$$SCQ_{size}(\varphi) = max(max\{s.wpd \mid s \in \mathbb{S}_\varphi\} - \varphi.bpd, 0) + 1 \tag{1}$$

Aggregation. A verdict-timestamp tuple $T_\varphi = (v,\tau)$ is stored in φ's SCQ using aggregation [29,42]. Aggregation occurs such that if an incoming tuple's verdict v is equivalent to the previous tuple's verdict v, then the incoming tuple's timestamp τ overwrites the previous tuple's timestamp τ. For example, if φ's SCQ contains $\{(\mathsf{true},3),(\mathsf{false},7)\}$, then $\varphi = \mathsf{false}$ for the entire timestamp interval $[4,7]$, and if φ encounters an incoming tuple $T_\varphi = (\mathsf{false},8)$, then φ's SCQ becomes $\{(\mathsf{true},3),(\mathsf{false},8)\}$. This aggregated storing of verdict-timestamp tuples allows R2U2 to easily reason over multiple timestamps (with equivalent verdicts) at once.

Booleanizer. To produce atomics for the leaf nodes of the AST, either atomics can be loaded directly into R2U2 or the Booleanizer [26] can be utilized. R2U2's Booleanizer enables boolean expressions over booleans, integers, and/or float input signals using arithmetic, bitwise, relational, and set operators (e.g., "forexactlyn" or "foratmostn"). Similar to the MLTL/ptMLTL AST, the Booleanizer decomposes the expression(s) into subexpression(s) represented in an AST and produces computation instructions for R2U2 to reason over.

3 Optimized and Proved R2U2 Implementation

Previously, R2U2 directly encoded only a subset of MLTL operators: $\neg\psi$, $\psi \wedge \xi$, $\Box_I \psi$, and $\psi \, \mathcal{U}_I \, \xi$ (as shown in yellow in Fig. 3) [26,29,42]. The full MLTL semantics were encoded by utilizing the appropriate semantic equivalents. In many cases, this required extra negation subformulas in the AST; consequently, this required extra negation instructions for R2U2 to reason over. For example, the encoding of the Release ($\mathcal{R}$) operator required four instructions

Original Formula	Previous Encoding		New Encoding	
	Formula	# of instructions	Formula	# of instructions
$\neg\psi$	$\neg\psi$	1	$\neg\psi$	1
$\psi \wedge \xi$	$\psi \wedge \xi$	1	$\psi \wedge \xi$	1
$\psi \vee \xi$	$\neg(\neg\psi \wedge \neg\xi)$	4	$\psi \vee \xi$	1
$\psi \rightarrow \xi$	$\neg(\psi \wedge \neg\xi)$	3	$\neg\psi \vee \xi$	2
$\psi \leftrightarrow \xi$	$\neg(\psi \wedge \neg\xi) \wedge (\neg(\neg\psi \wedge \xi))$	6	$\psi \leftrightarrow \xi$	1
$\psi \oplus \xi$	$\neg(\neg(\psi \wedge \neg\xi) \wedge (\neg(\neg\psi \wedge \xi))$	8	$\neg(\psi \leftrightarrow \xi)$	2
$\Diamond_I \psi$	$\mathsf{true} \, \mathcal{U}_I \, \psi$	1	$\mathsf{true} \, \mathcal{U}_I \, \psi$	1
$\Box_I \psi$	$\Box_I \psi$	1	$\mathsf{false} \, \mathcal{R}_I \, \psi$	1
$\psi \mathcal{U}_I \xi$	$\psi \, \mathcal{U}_I \, \xi$	1	$\psi \, \mathcal{U}_I \, \xi$	1
$\psi \, \mathcal{R}_I \, \xi$	$\neg(\neg\psi \, \mathcal{U}_I \, \neg\xi)$	4	$\psi \, \mathcal{R}_I \, \xi$	1
$\boxminus_I \psi$	$\boxminus_I \psi$	1	$\mathsf{true} \, \mathcal{S}_I \, \psi$	1
$\boxminus_I \psi$	$\boxminus_I \psi$	1	$\mathsf{false} \, \mathcal{T}_I \, \xi$	1
$\psi \, \mathcal{S}_I \, \xi$	$\psi \, \mathcal{S}_I \, \xi$	1	$\psi \, \mathcal{S}_I \, \xi$	1
$\psi \, \mathcal{T}_I \, \xi$	–	–	$\psi \, \mathcal{T}_I \, \xi$	1

Fig. 3. Comparison of previous [26,29,42] and new encodings of MLTL and ptMLTL formulas in R2U2. Both the previous and new encoding directly encode a subset of operators shown in yellow and blue, respectively. The formulas with indirect encodings are given by semantic equivalents.

(i.e., three extra negation instructions) since $\psi\ \mathcal{R}_I\ \xi$ was encoded as $\neg(\neg\psi\ \mathcal{U}_I\ \neg\xi)$. This increase in instructions negatively affects R2U2's timing and resource requirements; R2U2 had to reason over extra instructions, allocate additional SCQs, and store extra instructions in its table. Therefore, we directly encode a different subset of MLTL operators: $\neg\psi$, $\psi\wedge\xi$, $\psi\vee\xi$, $\psi\leftrightarrow\xi$, $\psi\ \mathcal{U}_I\ \xi$, and $\psi\ \mathcal{R}_I\ \xi$ (as shown in blue in Fig. 3). We optimized the selection of MLTL operators with direct encodings to reduce both timing and resource requirements. As shown in Figure 3, every MLTL/ptMLTL operator that previously required more than one instruction has been reduced. Since R2U2 is designed to fit within tight memory-constrained systems, the size of the extra logic also had to be considered. For example, $\square_I\psi$ is easily encoded as false $\mathcal{R}_I\ \psi$ without additional instructions; therefore, we only directly encode the $\mathcal{R}$ operator to eliminate extra redundant logic that would be required to encode both. In the rest of this section, we provide correctness proofs for the SCQ read and write operations and all encoded MLTL and ptMLTL temporal operators.[4]

3.1 Shared Connection Queues

The algorithms for the SCQ read and write operations were first presented in [29]; however, no formal proof of correctness was provided and the algorithms contained various errors. For example, if the read pointer and write pointer point to the same SCQ slot, the read operation would always return an empty verdict-timestamp tuple, which is not the desired behavior (e.g., a SCQ of size one always returned an empty tuple such that no valid tuple was ever read). Some of the errors were fixed in future releases of R2U2, but R2U2 v3.0 [26] (i.e., the latest version of R2U2) still contained errors such that the SCQ sizing given in Equation 1 required $+3$ instead of $+1$ to mask the underlying problem. Therefore, we provide the SCQ read and write operations in Algorithm 1, and the correctness of the aggregated write and aggregated read are proved in Theorems 1 and 2, respectively. Note that the correctness of the MLTL and ptMLTL operators depends greatly on the correctness of the SCQ operations.

Algorithm 1: Shared Connection Queue (SCQ) Operations for Node φ

```
1  Initialize:
2      φ.write_ptr = 0
3      φ.read_1_ptr = 0 and φ.read_2_ptr = 0
4      φ.SCQ[0] = Empty
5  function read(read_ptr, desired_time) is
       Input: Read pointer: read_ptr; Desired timestamp: desired_time
       Output: T_φ or Empty
6      if φ.SCQ[read_ptr] = Empty and read_ptr = 0 then                  // SCQ is empty
7          return Empty          // Return Empty, indicating there is no new T_φ in SCQ
8      do                                                               // Scan forward in SCQ
9          if φ.SCQ[read_ptr].τ >= desired_time then
10             return φ.SCQ[read_ptr]                  // T_φ is new; therefore return T_φ
11         read_ptr = (read_ptr + 1) % SCQ_size(φ)                      // Step forward in SCQ
12     while read_ptr ≠ φ.write_ptr;
       /* Hit write_ptr while scanning forward; take a step back                    */
13     read_ptr = (read_ptr − 1) % SCQ_size(φ)
14     return Empty              // Return Empty, indicating there is no new T_φ in SCQ
15 function write(T_φ) is
       Input: Verdict-timestamp tuple to write: T_φ
16     prev_write_ptr = (φ.write_ptr − 1) % SCQ_size(φ)  // Find the previous write pointer
       /* Check if aggregating write                                            */
17     if !(φ.SCQ[φ.write_ptr] = Empty and φ.write_ptr = 0) then        // SCQ is not empty
           /* Previous verdict matches T_φ.v                                    */
18         if φ.SCQ[prev_write_ptr].v = φ.SCQ[φ.write_ptr].v then
19             φ.write_ptr = prev_write_ptr
20     φ.SCQ[φ.write_ptr] = T_φ
21     φ.write_ptr = (φ.write_ptr + 1) % SCQ_size(φ)          // Move write pointer forward
```

[4] Proofs for the boolean connectives are available in Appendix A.

Theorem 1 (Aggregated Write to SCQ). *Given an AST subformula node φ, φ's write pointer $\varphi.write\_ptr$, and a verdict-timestamp tuple T_φ, the write function in Algorithm 1 $\big($i.e., $\varphi.write(T_\varphi)\big)$ is guaranteed to store results in strictly increasing order using aggregation such that $\varphi.write\_ptr$ is always either an empty slot or the oldest entry in the SCQ.*

Proof. Sequential writes to a SCQ are always in strictly increasing order in terms of $T_\varphi.\tau$; this directly follows from the implementation of R2U2's MLTL and ptMLTL operators (e.g., Algorithms 2 and 3). There are two ways that a tuple T_φ is written in φ's SCQ:

(1) *Aggregate Write*: If the SCQ is not completely empty, then there was a previous write to the SCQ. If a previous write exists (line 17), the previous slot in the SCQ is checked by decrementing the write pointer in a circular manner on line 16 (i.e., if the write pointer is at the first slot of the SCQ, the previous slot is the last slot of the SCQ). If the verdict is the same in the previous slot as $T_\varphi.v$ (line 18), then the write pointer is re-assigned to the previous slot on line 19. The tuple T_φ then overwrites the previous slot on line 20.

(2) *Non-aggregate Write*: If there was no previous write to the SCQ (line 17) or the previous write to the SCQ didn't contain the same verdict as the input tuple T_φ (line 18), then the tuple T_φ is simply written at the write pointer on line 20.

In both cases, the write pointer is then incremented in a circular manner on line 21 such that the write pointer is set to the next slot. Since the write pointer is always incremented in a circular manner to the slot after where the current write occurred, the value of the $write\_ptr$ at the end of execution is always either empty or the oldest value, and the value at the previous entry is always the newest value. □

Theorem 2 (Aggregated Read from SCQ). *Given an AST subformula node φ, the read pointer $read\_ptr$, and the timestamp $desired\_time$, reading from φ's SCQ as defined in Algorithm 1 $\big($i.e., $\varphi.read(read\_ptr,desired\_time)\big)$ will return the verdict-timestamp tuple T_φ iff $T_\varphi.v$ is the verdict for the entire interval $[desired\_time,T_\varphi.\tau]$. Figure 4 provides a visualization of this theorem.*

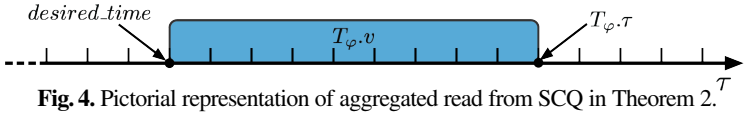


Fig. 4. Pictorial representation of aggregated read from SCQ in Theorem 2.

Proof. Each node stores two read pointers, $\varphi.read_1\_ptr$ and $\varphi.read_2\_ptr$; one for each of its possible children. When a parent node is reading a child node φ, the $desired\_time$ is always increasing such that $desired\_time$ indicates the next timestamp required for evaluation of the parent node φ; this directly follows from the implementation of R2U2's MLTL and ptMLTL operators (e.g., Algorithms 2 and 3). The minimal SCQ size in Equation 1 (proof available in [51]) guarantees that the SCQ will either contain the $desired\_time$ upon a call to $\varphi.read(read\_ptr,desired\_time)$ or will in the future.

At the beginning of each call to $\varphi.read(read\_ptr,desired\_time)$, $read\_ptr$ will point to the earliest slot that might be of interest as it will either be (1) pointing to the first slot but no value has ever been read before (initial condition), (2) pointing to the last value that was read from the previous execution (line 10) such that we will either read from this slot (due to aggregation) or a future slot, or (3) still pointing to the latest value in the SCQ from the previous execution (line 13; follows from Theorem 1).

(only-if direction) *$T_\varphi.v$ is the verdict for the entire interval $[desired\_time,T_\varphi.\tau] \rightarrow$ return T_φ*: If the SCQ is completely empty (i.e., no tuples have ever been written to the SCQ), then there are no new tuples to read; therefore, $Empty$ is returned on line 7. Based on correct SCQ sizing and behavior of $desired\_time$ and $read\_ptr$ described above, if the slot at $read\_ptr$ has a timestamp $\geq desired\_time$, then $T_\varphi.v$ is the verdict for the entire interval

$[desired\_time,T_\varphi.\tau]$ and T_φ is returned on line 10. If not, then perhaps a future slot may contain this information; therefore, $read\_ptr$ is incremented in a circular manner on line 11 such that the SCQ will be incrementally scanned to find a tuple such that the timestamp $\geq desired\_time$. If one is found (since we are incrementally scanning forward), $T_\varphi.v$ is the verdict for the entire interval $[desired\_time,T_\varphi.\tau]$ and T_φ is returned on line 10. If one is not found, the scanning will stop when the $read\_ptr$ is now pointing to the oldest value in the SCQ (i.e., $\varphi.write\_ptr$) on line 12. Then, $read\_ptr$ is decremented such that $read\_ptr$ is pointing to the latest written entry on line 13 (following from Theorem 1) and $Empty$ is returned.

(if direction) *return $T_\varphi \rightarrow T_\varphi.v$ is the verdict for the entire interval $[desired\_time,T_\varphi.\tau]$*: A tuple T_φ is only returned on line 10, which requires $T_\varphi.\tau \geq desired\_time$; this requires that either the slot at $read\_ptr$ or a slot between $read\_ptr$ and the latest entry to be $\geq desired\_time$. Since lines 8–12, incrementally scan forward in the SCQ until a tuple with timestamp $\geq desired\_time$ is found, $T_\varphi.v$ is the verdict for the entire interval $[desired\_time,T_\varphi.\tau]$. If the SCQ is completely empty, then there are no new tuples to read; therefore, $Empty$ is returned on line 7. If no entry has a timestamp $\geq desired\_time$, then $Empty$ is returned on line 14. □

3.2 MLTL Temporal Operators

In the latest version of R2U2 (i.e., v3.0 [26]) and prior versions [29,42], only the Global ($\Box$) and Until ($\mathcal{U}$) MLTL temporal operators had direct encodings, and proofs of correctness were given in [29]. Although these algorithms provided correct verdicts, the $\mathcal{U}$ operator required ***both*** ψ and ξ to be known (for an arbitrary timestamp τ) to evaluate $\psi\,\mathcal{U}_I\,\xi$. As stated in Axiom 1, there are three conditions in which $\mathcal{U}$ can be evaluated early ***without knowing both*** ψ and ξ. Since runtime monitoring requires early-as-possible identification of failures to enable effective fault recovery [5,6,29,42,51], verdicts should not be unnecessarily delayed; therefore, we rewrite the $\mathcal{U}$ algorithm in Algorithm 2 to write a verdict when sufficient data is available according to Axiom 1. We follow the same approach for the Release ($\mathcal{R}$) operator following Axiom 2.

Algorithm 2: Until Operator: $\varphi = \psi\,\mathcal{U}_{[lb,ub]}\,\xi$

```
1  Initialize:
2      φ.previous = −1                    // Initialize φ.previous; stores the last i written
3      φ.next_time = lb                   // Initialize φ.next_time; stores the next time for ψ and ξ
4  procedure Until(ψ,ξ)
       Input: Node: ψ; Node: ξ
5      T_ψ = ψ.read(φ.read_1_ptr,φ.next_time)                          // Read Node ψ
6      T_ξ = ξ.read(φ.read_2_ptr,φ.next_time)                          // Read Node ξ
7      if T_ξ ≠ Empty then                                             // New T_ξ
8          if T_ξ.v then                                               // T_ξ.v = true
9              φ.previous = T_ξ.τ − lb
10             φ.next_time = T_ξ.τ + 1
11             φ.write(true, T_ξ.τ − lb)                   // Writing T_φ = (true, T_ξ.τ − lb)
12             return
13         if T_ψ ≠ Empty then                                         // New T_ψ and T_ξ
14             τ_min = min(T_ψ.τ, T_ξ.τ)
15             φ.next_time = τ_min + 1
16             if !(T_ψ.v) then                                        // T_ψ.v = false and T_ξ.v = false
17                 φ.previous = τ_min − lb
18                 φ.write(false, τ_min − lb)              // Writing T_φ = (false, τ_min − lb)
19                 return
20         if T_ξ.τ > φ.previous + ub then   // (T_ψ = Empty or T_ψ.v = true) and T_ξ.v = false
21             φ.previous = T_ξ.τ − ub
22             φ.next_time = max(φ.next_time, φ.previous + lb + 1)
23             φ.write(false, T_ξ.τ − ub)                  // Writing T_φ = (false, T_ξ.τ − ub)
```

Theorem 3 (Correctness of the Until Operator). *Given the interval $[lb,ub]$ and two children nodes ψ and ξ, Algorithm 2 writes the tuple T_φ to φ's SCQ when sufficient data is available such that for all $i \geq 0$, $T_\varphi = (\mathsf{true},i)$ iff $\pi,i \models \psi\,\mathcal{U}_{[lb,ub]}\,\xi$.*

Proof. The $\varphi.previous$ variable stores the previous time index i that node $\varphi = \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ wrote to its SCQ. Before the first tuple is produced, $\varphi.previous$ is initialized to -1 (line 2) such that the condition on line 20 can be rewritten as $T_\xi.\tau > -1+ub$ or rather $T_\xi.\tau \geq 0+ub \equiv T_\xi.\tau \geq ub$. Whenever a verdict-timestamp tuple is written, the $\varphi.previous$ variable is updated to that timestamp (lines 9, 17, and 21).

The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized on line 3 to lb since $\psi\, \mathcal{U}_{[lb,ub]}\, \xi$ can only be evaluated when $|\pi| > i+lb$ on the interval $[i+lb,i+ub]$ where $i \geq 0$, such that $[0,lb-1]$ is never required for evaluation. The $\varphi.next\_time$ variable is then updated during execution based on what is unknown about ψ and ξ. If both T_ψ and T_ξ are not empty, then if $T_\psi.\tau \geq T_\xi.\tau$, its unknown if $\exists j \in [T_\xi.\tau+1, T_\psi.\tau]$ such that $\pi,j \models \xi$, and if $T_\psi.\tau \leq T_\xi.\tau$, then its unknown if $\forall k \in [T_\psi.\tau+1, T_\xi.\tau]$ such that $\pi,k \models \psi$. Therefore, $\varphi.next\_time$ will be updated to $min(T_\psi.\tau, T_\xi.\tau)+1$ on line 15. Because of early evaluation (following from Axiom 1), if a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to whichever is greater: $i+lb+1$ (i.e., the lb of the next evaluation) or $min(T_\psi.\tau, T_\xi.\tau)+1$ (as described above) on lines 10 and 22. Note that on line 10 this can be rewritten as just $T_\xi.\tau+1$ since $i+lb+1 \equiv (T_\xi.\tau-lb)+lb+1 \equiv T_\xi.\tau+1$, and if $T_\psi.\tau \leq T_\xi.\tau$, then $T_\psi.\tau \leq i+lb+1$. Figures 5, 6, 7, and 8 illustrate examples of how the $\varphi.next\_time$ variable is updated.

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 5–6 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time, T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time, T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi \rightarrow T_\varphi = (\mathsf{true}, i)$:
We consider all possible combinations of T_ψ and T_ξ to determine if $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$:

(1) *T_ψ and T_ξ are both empty*: There is no new information based on $\varphi.next\_time$ to evaluate if $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$; therefore, the algorithm does nothing.

(2) *T_ψ is not empty and T_ξ is empty*: There is not enough information to determine if $\exists j \in [i+lb, i+ub]$ such that $\pi, j \models \xi$, then $\forall k < j$ where $k \in [i+lb, i+ub]$ such that $\pi,k \models \psi$ (where $i = \varphi.previous+1$); therefore, the algorithm does nothing.

(3) $T_\xi.v = \mathsf{true}$ *(and T_ψ is empty or $T_\psi.v = \mathsf{false}$ or $T_\psi.v = \mathsf{true}$)*: $T_\xi.v$ is the verdict from $[\varphi.next\_time, T_\xi.\tau]$ such that $\varphi.next\_time$ is lb (initial condition) or was set by the previous execution to $\varphi.previous + lb + 1$ or $min(T_\psi.\tau, T_\xi.\tau) + 1$ as described above. Therefore, $T_\xi.v = \mathsf{true}$ at $\varphi.previous + lb + 1$ and/or $\forall k \in [\varphi.previous + lb + 1, \varphi.next\_time - 1]$, $\pi, k \models \psi$ (since the condition on line 16 was not met by the previous execution). As a result, $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ (following directly from Axiom 1) for $\varphi.previous < i \leq T_\xi.\tau - lb$; therefore, $(\mathsf{true}, T_\xi.\tau - lb)$ is written to φ's SCQ on line 11. Figure 5 provides a visualization of this case.

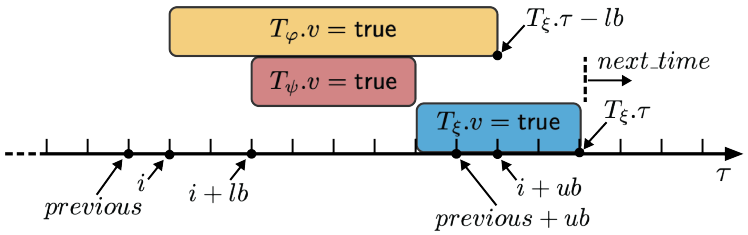


Fig. 5. Pictorial representation of $\varphi = \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for Theorem 3 Case (3).

(4) $T_\psi.v = \mathsf{false}$ *and* $T_\xi.v = \mathsf{false}$: If both $T_\psi.v$ and $T_\xi.v$ are false, then they are both guaranteed to be false from $[\varphi.next\_time, min(T_\psi.\tau, T_\xi.\tau)]$. Furthermore, $T_\xi.v$ is guaranteed to have

never been true from $[\varphi.previous+lb+1,T_\xi.\tau]$ since the result would have been written on line 11 and the $\varphi.next\_time$ and $\varphi.previous$ variables would have been updated (lines 9–10). As a result, $\pi,i \not\models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for $\varphi.previous < i \leq min(T_\psi.\tau,T_\xi.\tau)-lb$; therefore, $(\mathsf{false},min(T_\psi.\tau,T_\xi.\tau)-lb)$ is written to φ's SCQ on line 18. Figure 6 provides a visualization of this case.

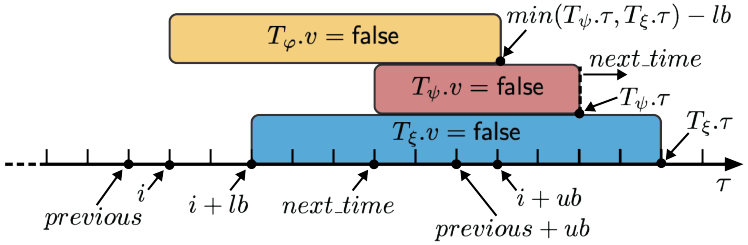


Fig. 6. Pictorial representation of $\varphi = \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for Theorem 3 Case (4).

(5) $T_\xi.v = \mathsf{false}$ *(and T_ψ is empty or $T_\psi.v = \mathsf{true}$)*: There are two sub-cases to consider:

(a) *There is enough information to determine that $\nexists j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$:* $T_\xi.v$ is guaranteed to have never been true from $[\varphi.previous+lb+1,T_\xi.\tau]$ since the result would have been written on line 11 and the $\varphi.next\_time$ and $\varphi.previous$ variables would have been updated (lines 9–10). If $T_\xi.\tau > \varphi.previous+ub$, then $\nexists j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$ where $i \in [\varphi.previous+1,T_\xi.\tau-ub]$. As a result, $\pi,i \not\models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for $\varphi.previous < i \leq T_\xi.\tau-ub$ (following directly from Axiom 1); therefore, $(\mathsf{false},T_\xi.\tau-ub)$ is written to φ's SCQ on line 23. Figure 7 provides a visualization of this case.

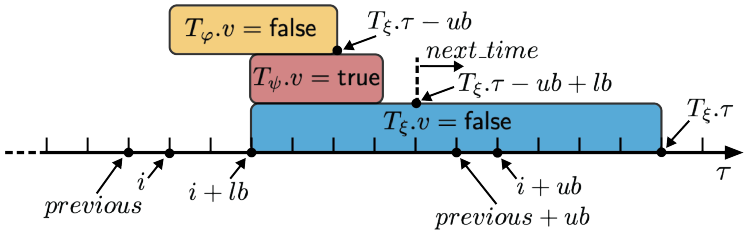


Fig. 7. Pictorial representation of $\varphi = \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for Theorem 3 Case (5)(a)

(b) *There is currently not enough information to determine if $\exists j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$:* If $T_\xi.\tau \leq \varphi.previous+ub$, then there is not enough information to guarantee that $T_\xi.v = \mathsf{false}$ from $[i+lb,i+ub]$ where $i = \varphi.previous+1$. There is still a chance that $\exists j \in [T_\xi.\tau+1,i+ub]$ such that $\pi,j \models \xi$; therefore, the algorithm does not write a tuple. Figure 8 provides a visualization of this case.

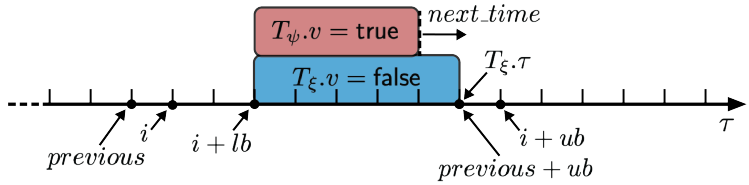


Fig. 8. Pictorial representation of $\varphi = \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for Theorem 3 Case (5)(b)

(if direction) $T_\varphi = (\mathsf{true},i) \rightarrow \pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$:
$T_\varphi = (\mathsf{true},i)$ tuples are only written to φ's SCQ on line 11, which requires that $T_\xi.v = \mathsf{true}$ is the verdict from $[\varphi.next\_time,T_\xi.\tau]$ such that $\varphi.next\_time$ is lb (initial condition) or was set by the previous execution to $\varphi.previous+lb+1$ or $min(T_\psi.\tau,T_\xi.\tau)+1$ as described

above. Therefore, $T_\xi.v = \mathsf{true}$ at $i+lb$ and/or $\forall k \in [i+lb, \varphi.next\_time]$, $\pi,k \models \psi$. As a result, $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for $\varphi.previous < i \leq T_\xi.\tau - lb$ (following directly from Axiom 1).

$T_\varphi = (\mathsf{false}, i)$ tuples are only written on lines 18 and 23 such that $T_\xi.v = \mathsf{false}$. In both cases, $T_\xi.v$ is guaranteed to have never been true from $[\varphi.previous + lb + 1, T_\xi.\tau]$ since the result would have been written on line 11 and the $\varphi.next\_time$ and $\varphi.previous$ variables would have been updated (lines 9–10). If $T_\psi.v = \mathsf{false}$, then $\pi,i \not\models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for $\varphi.previous < i \leq min(T_\psi.\tau, T_\xi.\tau) - lb$ (line 18). If $T_\psi.v \neq \mathsf{false}$ and $T_\xi.\tau > \varphi.previous + ub$, then $\nexists j \in [i+lb, i+ub]$ such that $\pi,j \models \xi$; therefore, $\pi,i \not\models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ for $\varphi.previous < i \leq T_\xi.\tau - ub$ (line 23).

There are three conditions under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$:

(1) *T_ψ and T_ξ are both empty*: There is not enough information to evaluate if $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$.
(2) *T_ψ is not empty and T_ξ is empty*: There is not enough information to determine if $\exists j \in [i+lb, i+ub]$ such that $\pi, j \models \xi$, then $\forall k < j$ where $k \in [i+lb, i+ub]$ such that $\pi,k \models \psi$; therefore, if $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ cannot be determined.
(3) *$T_\xi.v = \mathsf{false}$ and $T_\xi.\tau \leq \varphi.previous + ub$ and $T_\psi \neq \mathsf{false}$*: There is not enough information to guarantee that $T_\xi.v = \mathsf{false}$ from $[i+lb, i+ub]$ where $i = \varphi.previous + 1$. There is still a chance that $\exists j \in [T_\xi.\tau + 1, i+ub]$ such that $\pi,j \models \xi$; therefore, if $\pi,i \models \psi\, \mathcal{U}_{[lb,ub]}\, \xi$ cannot be determined.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true}, i)$ are only written to φ's SCQ iff $\pi,i \models \psi \mathcal{U}_{[lb,ub]} \xi$, and $T_\varphi = (\mathsf{false}, i)$ are only written to φ's SCQ iff $\pi,i \not\models \psi \mathcal{U}_{[lb,ub]} \xi$. □

The algorithm and proof of the Release ($\mathcal{R}$) operator follow very similarly to the Until ($\mathcal{U}$) operator and are available in Appendix B.

3.3 ptMLTL Temporal Operators

The SCQ architecture of R2U2 was developed in [29] and greatly reduced the memory requirements of the previous implementation [42] as exemplified by R2U2's integration on the heavily resource-constrained FPGA of the Robonaut2's knee joint. However, this new SCQ architecture was only applied to R2U2's MLTL operators. The past-time logic still implemented the approach in [41] which utilized single read queues and encoded ptMTL [2,32] rather than ptMLTL. Note that in ptMTL, there is no Trigger ($\mathcal{T}$) operator (see Figure 3), and the satisfaction of $\pi,i \models \psi\, \mathcal{S}_{[lb,ub]}\, \xi$ requires ψ to hold from the position where ξ holds in $[i-ub, i-lb]$ to position i, while in ptMLTL, ψ is only required to hold within the interval $[i-ub, i-lb]$ after ξ holds. Therefore, we compose new Since ($\mathcal{S}$) and Trigger ($\mathcal{T}$) algorithms that utilize the SCQ architecture and implement ptMLTL. Similar to the Until and Release operators (Section 3.2), we ensure verdicts are written as soon as sufficient data is available according to Axioms 3 and 4. The algorithm of the Since ($\mathcal{S}$) operator is available in Algorithm 3 and its corresponding proof is available in Appendix C. The algorithm and proof of the Trigger ($\mathcal{T}$) operator follow very similarly to the Since operator and are available in Appendix D.

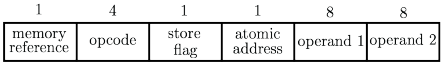
3.4 Reduction of Instruction Size

As discussed in Section 2.2, R2U2 stores its computation instructions in a table; therefore, these instructions require memory resources. We evaluated the current instruction format for both the booleanizer and temporal logic (i.e., MLTL and ptMTL) instructions as present in R2U2 v3.0 [26] and were able to reduce the memory footprint of each (Figure 9). Within the booleanizer instructions, we reduced the opcode down from 4 bytes (i.e., allows $4{,}294{,}967{,}296$ opcodes) to 1 byte (i.e, allows 256 opcodes); R2U2 only currently supports 40 different booleanizer operations,

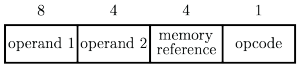
Algorithm 3: Since Operator: $\varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi$

```
1  Initialize:
2      φ.edge = −1                          // Initialize φ.edge; stores the last i where π,i ⊨ ξ
3      φ.previous = −1                      // Initialize φ.previous; stores the last i written
4      φ.next_time = 0                      // Initialize φ.next_time; stores the next time for ψ and ξ
5  procedure Since(ψ,ξ)
       Input: Node: ψ; Node: ξ
6      T_ψ = ψ.read(φ.read_1_ptr,φ.next_time)                          // Read Node ψ
7      T_ξ = ξ.read(φ.read_2_ptr,φ.next_time)                          // Read Node ξ
8      if φ.previous + 1 − lb < 0 then                                 // i − lb < 0
9          φ.previous = lb − 1
10         φ.write(false,lb − 1)                                       // Writing T_φ = (false,lb − 1)
11     if T_ξ ≠ Empty then                                             // New T_ξ
12         if T_ξ.v then                                               // T_ξ.v = true
13             φ.edge = T_ξ.τ                                          // Updating φ.edge to last true edge
14             φ.next_time = T_ξ.τ + 1                                 // Updating φ.next_time to after φ.edge
15             if T_ξ.τ > φ.previous − lb then                         // π,i − lb ⊨ ξ
16                 φ.previous = T_ξ.τ + lb
17                 φ.write(true,T_ξ.τ + lb)                            // Writing T_φ = (true,T_ξ.τ + lb)
18                 return
19         else                                                        // T_ξ.v = false
20             if φ.edge ≤ φ.previous − ub or φ.edge = −1 then          // ∄j ∈ [i − ub,T_ξ.τ],π,j ⊨ ξ
21                 φ.next_time = T_ξ.τ + 1                             // Move φ.next_time forward
22                 if T_ξ.τ > φ.previous − lb then                     // ∄j ∈ [i − ub,i − lb],π,j ⊨ ξ
23                     φ.previous = T_ξ.τ + lb
24                     φ.write(false,T_ξ.τ + lb)                       // Writing T_φ = (false,T_ξ.τ + lb)
25                     return
26             if T_ψ ≠ Empty then                                     // New T_ψ
27                 if !(T_ψ.v) then                                    // T_ψ.v = false and T_ξ.v = false
28                     φ.next_time = T_ξ.τ + 1                         // Move φ.next_time forward
29                     if T_ξ.τ > φ.previous − lb then
30                         φ.previous = T_ξ.τ + lb
31                         φ.write(false,T_ξ.τ + lb)                   // Writing T_φ = (false,T_ξ.τ + lb)
32                         return
33     if T_ψ ≠ Empty then                                             // New T_ψ
           /* ∃φ.edge ∈ [i − ub,i − lb], such that ∀k > φ.edge where k ≥ i − lb, π,k ⊨ ψ  */
34         if T_ψ.v and T_ψ.τ > φ.previous − lb and φ.edge > φ.previous − ub and φ.edge ≠ −1 then
35             φ.previous = min(T_ψ.τ + lb,φ.edge + ub)                // Limit i based on φ.edge
36             φ.next_time = max(φ.next_time,φ.previous − ub + 1)
37             φ.write(true,φ.previous)                                // Writing T_φ = (true,φ.previous)
```

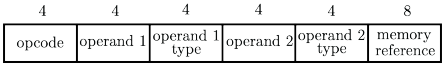
and 1 byte still allows the booleanizer to support 216 more opcodes without increasing the size. On the other hand, we increased the size of the memory reference address field from 1 (i.e., allowed for only 256 boolean instructions total) to 4 bytes. We kept the first operand field as 8 bytes to allow for possible loading of doubles, but reduced the second operand field to 4 since this field will only ever contain an address to another booleanizer instruction, which is constricted to a max of 4 bytes. Lastly, we removed the parameters to store the final result of the booleanizer (i.e., atomic address and store flag) from *every* instruction. There is now a separate store instruction for when the booleanizer has completed its computations and needs to return/store the atomic for the temporal logic operators to read. Within the temporal logic instructions, we also decreased the opcode down from 4 bytes to 1 byte, and we decreased the operand type values down from 4 bytes to 1 byte as these flags only store a value of 0, 1, or 2, indicating whether



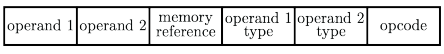
(a) Previous Booleanizer Instruction Format



(b) New Booleanizer Instruction Format



(c) Previous Temporal Logic Instruction Format

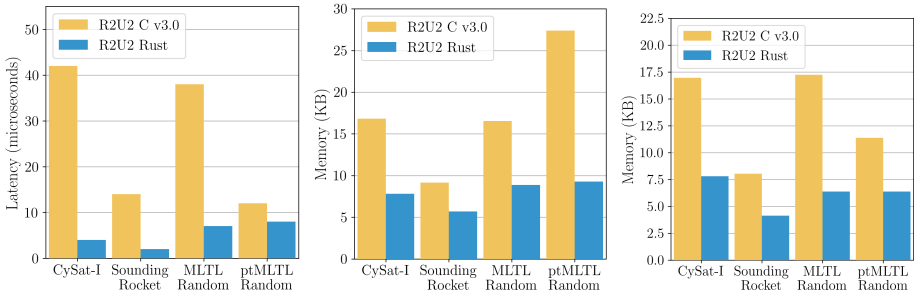


(d) New Temporal Logic Instruction Format

Fig. 9. Comparison of previous [26] and new instruction formats

the operand is an atomic, a subformula, or a constant. These simple optimizations are significant as each booleanizer instruction decreased from 28 bytes to 20 bytes and each temporal logic instruction decreased from 28 to 16 bytes.

3.5 Latency and Memory Analysis



(a) 95% tail latency (μs) (b) Total queue memory (KB) (c) Total instruction memory (KB)

Fig. 10. Comparison of latency and memory requirements between R2U2 C v3.0 [26] and R2U2 Rust.

We manually transpile R2U2 v3.0 [26] from C to safe embedded Rust and apply the optimizations and proved algorithms given in Sections 3–3.4. To evaluate the effectiveness of our optimizations, we compare the latency and memory requirements of our Rust implementation against R2U2 v3.0 on a suite of benchmarks. The benchmarks utilized are as follows: (1) 22 MLTL specifications utilized to verify the electrical power system of the CySat-I CubeSat [5], (2) 16 MLTL specifications for the Nova Somnium sounding rocket's aerobraking control system [24], (3) 35 random hand-written MLTL specifications, and (4) 35 random hand-written ptMLTL specifications.[5] In Figure 10a, we recorded the 95% tail latency (i.e., 95% of recorded latencies are less than or equal to the given latency) for each time step where $|\pi| = 1{,}000{,}000$ on a 2.8 GHz Quad-Core Intel® i7 processor with 16GB of RAM; a 1.5–10.5x decrease in latency was observed depending on the benchmark, which can mostly be accredited to the reduction of instructions through direct encodings.[5] Figure 10b reveals a 1.5–3x decrease in total queue memory size for each benchmark; this is accredited to the reduction in the number of instructions through direct encodings, removal of the two extra entries in each queue from the previous implementation of the SCQ read/write operations (Section 3.1), and modifying past-time to utilize SCQs (Section 3.3). Figure 10c indicates a 2–2.7x decrease in instruction memory size as a result of directly encoding instructions and refining R2U2's instruction format (Section 3.4). We also ran our embedded Rust implementation on a resource-constrained bare-metal STM32F3DISCOVERY microcontroller with 48MHz system clock, 256KB of flash memory, and 48KB of RAM [49], and the average latency observed is recorded in Figure 11. While the latency on the STM32F3DISCOVERY microcontroller is greater than on the Intel® i7 processor, these latencies still indicate real-time performance.

Benchmark	Clock Cycles	Time (ms)
CySat-I	257836	5.371
Sounding Rocket	117827	2.454
MLTL Random	266507	5.552
ptMLTL Random	384825	8.017

Fig. 11. Latency on STM32F3DISCOVERY

4 Verification of R2U2's Rust Implementation

We examined three Rust code verification tools to verify R2U2's Rust implementation: Creusot [17], Prusti [3,4], and Verus [33,34]. Creusot encodes a Rust application in the WhyML

[5] Benchmarks and more analysis available at *https://temporallogic.org/research/R2U2Rust*

intermediate verification language for use in Why3 [20], where Why3 directly encodes SMT queries for input into backend solvers like the Z3 SMT solver [16]; Prusti translates an entire Rust application into the Viper intermediate verification language [38] and the Viper program is verified using Viper's symbolic execution verifier (which further translates to SMT queries for Z3); and Verus encodes Rust code directly to SMT-LIB [7] for input into Z3. As a result, Verus is faster than both Creusot and Prusti as it directly encodes SMT queries, and Prusti is the slowest as it involves several additional steps, including re-verifying Rust's type checking in Viper [33]. On the other hand, Prusti was the simplest tool to run as the full tool is available as a plugin extension directly inside VSCode, compared to Creusot and Verus which each require local installations. Prusti and Creusot are also directly compatible with Rust's Cargo package manager, but currently, Verus is not. Verus is only compatible with rustc (i.e., the Rust compiler), which requires directly specifying compilation flags (including linking dependencies).

After experimenting with each tool, we found pre- and post-conditions easier to compose with Verus's 'requires' and 'ensures' blocks, compared to Prusti and Creusot's clauses. For example, Creusot could not automatically reason about our structs with a 'Default' implementation,[6] and Prusti cannot unwrap 'Option' Rust types in pre- and post-conditions.[7] As a result, we were unable to directly specify the complex specifications required for R2U2's temporal operators in Prusti or Creusot. Therefore, we utilized Verus to verify R2U2. It is important to note that there are still parts of R2U2 that could not be verified with Verus such as floats, modulo operations, and certain &mut references, and we also discovered a bug within Verus where arrays cannot be sized according to constant values.[8] While Verus has its shortcomings, we were able to overcome most of them and found it easier than Prusti or Creusot for verifying R2U2.

Verus automatically detected multiple locations within the booleanizer implementation (Section 2.2) that could result in underflow or overflow; this included operators that added, subtracted, or multiplied two integers together. For release builds, Rust will perform wrapping operations by default where the underflow and overflow bits are just ignored. To eradicate the possibility of unnoticed underflow or overflows, we specified saturating add, subtract, and multiply operators such that if the result underflows, the result will be the minimum value that can be stored in the result type, and if it overflows, the result will be the maximum value that can be stored in the result type. Since the booleanizer will eventually compare integers utilizing comparators (i.e., $>$, $<$, $\leq$, $\geq$, and $=$), saturating operations are safer. Furthermore, we added an overflow detection flag that can easily be read, reset, and mitigated by the monitored system.

We specify pre- and post-conditions for every R2U2 operator that is possible with Verus. The pre- and post-conditions in the booleanizer are directly mapped to ensure saturating operations and correct overflow detection. The pre- and post-conditions for the MLTL and ptMLTL operators directly ensure all cases and claims in the hand-constructed proofs presented in Section 3. In total, there are 487 lines of code contracts that verify a total of 25 operators. Through these pre- and post-conditions, we are able to ensure correct implementation of our algorithms in Rust. During this process, we also found Verus helpful in refining our initial hand-constructed algorithms by removing vacuous conditions. On the other hand, Verus was not able to verify all of R2U2's Rust code nor was it able to consider the correctness of the C2PO compiler; therefore, we also exhaustively test all of our MLTL operators according to the strategy in [30].

[6] *https://github.com/creusot-rs/creusot/issues/792*
[7] *https://github.com/viperproject/prusti-dev/issues/1306*
[8] *https://github.com/verus-lang/verus/issues/1334*

5 Conclusion and Future Work

We developed a new implementation of R2U2 written in safe embedded Rust that significantly decreases its previous resource overhead and provides improved guarantees of correctness through hand-constructed proofs, testing, and Verus code contracts. While we hand-constructed our proofs, we eventually hope to formalize R2U2 in a proof assistant such as Isabelle/HOL, and we also anticipate more intuitive automatic test generation that can test a wider range of *both* MLTL and ptMLTL formulas against an oracle such as [31]. While Verus has its current limitations, we look forward to the further development of Verus's capabilities and plan to incorporate more deductive code verification as features become available.

References

1. A no_std rust environment, *https://docs.rust-embedded.org/book/intro/no-std.html*
2. Alur, R., Henzinger, T.A.: Real-time Logics: Complexity and Expressiveness. In: LICS. pp. 390–401. IEEE (1990)
3. Astrauskas, V., Bílỳ, A., Fiala, J., Grannan, Z., Matheja, C., Müller, P., Poli, F., Summers, A.J.: The prusti project: Formal verification for rust. In: NASA Formal Methods Symposium. pp. 88–108. Springer (2022), *https://doi.org/10.1007/978-3-031-06773-0_5*
4. Astrauskas, V., Müller, P., Poli, F., Summers, A.J.: Leveraging rust types for modular specification and verification. Proceedings of the ACM on Programming Languages **3**(OOPSLA), 1–30 (2019), *https://doi.org/10.1145/3360573*
5. Aurandt, A., Jones, P.H., Rozier, K.Y.: Runtime verification triggers real-time, autonomous fault recovery on the cysat-i. In: NASA Formal Methods Symposium. pp. 816–825. Springer (2022), *https://temporallogic.org/research/CySat-NFM22/CySat-NFM22.pdf*
6. Aurandt, A., Jones, P.H., Rozier, K.Y., Wongpiromsarn, T.: Multimodal model predictive runtime verification for safety of autonomous cyber-physical systems. In: International Conference on Formal Methods for Industrial Critical Systems. pp. 220–244. Springer (2024), *https://temporallogic.org/research/MMPRV/MMPRV.pdf*
7. Barrett, C., Stump, A., Tinelli, C., et al.: The smt-lib standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK). vol. 13, p. 14 (2010)
8. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. Lectures on Runtime Verification: Introductory and Advanced Topics pp. 135–175 (2018). https://doi.org/https://doi.org/10.1007/978-3-319-75632-5_5
9. Basin, D., Dardinier, T., Hauser, N., Heimes, L., Huerta y Munive, J.J., Kaletsch, N., Krstić, S., Marsicano, E., Raszyk, M., Schneider, J., et al.: Verimon: a formally verified monitoring tool. In: International Colloquium on Theoretical Aspects of Computing. pp. 1–6. Springer (2022), *https://doi.org/10.1007/978-3-031-17715-6_1*
10. Basin, D., Dardinier, T., Heimes, L., Krstić, S., Raszyk, M., Schneider, J., Traytel, D.: A formally verified, optimized monitor for metric first-order dynamic logic. In: Automated Reasoning: 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I 10. pp. 432–453. Springer (2020), *https://doi.org/10.1007/978-3-030-51074-9_25*
11. Baumeister, J., Finkbeiner, B., Kohn, F., Scheerer, F.: A tutorial on stream-based monitoring. In: International Symposium on Formal Methods. pp. 624–648. Springer (2024), *https://doi.org/10.1007/978-3-031-71177-0_33*
12. Blech, J.O., Falcone, Y., Becker, K.: Towards certified runtime verification. In: International Conference on Formal Engineering Methods. pp. 494–509. Springer (2012), *https://doi.org/10.1007/978-3-642-34281-3_34*
13. Cauwels, M., Hammer, A., Hertz, B., Jones, P., Rozier, K.Y.: Integrating Runtime Verification into an Automated UAS Traffic Management System. In: DETECT. Springer, L'Aquila, Italy (September 2020), *https://r2u2.temporallogic.org/wp-content/uploads/2020/12/CHHJR20.pdf*
14. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: Runtime Verification: 20th International Conference, RV 2020, Los Angeles, CA,

USA, October 6–9, 2020, Proceedings 20. pp. 383–403. Springer (2020), *https://doi.org/10.1007/978-3-030-60508-7_21*
15. Dabney, J.B., Badger, J.M., Rajagopal, P.: Trustworthy autonomy for gateway vehicle system manager. In: 2023 IEEE Space Computing Conference (SCC). pp. 57–62. IEEE (2023). https://doi.org/https://doi.org/10.1109/SCC57168.2023.00018
16. 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), *https://doi.org/10.1007/978-3-540-78800-3_24*
17. Denis, X., Jourdan, J.H., Marché, C.: Creusot: a foundry for the deductive verification of rust programs. In: International Conference on Formal Engineering Methods. pp. 90–105. Springer (2022), *https://doi.org/10.1007/978-3-031-17244-1_6*
18. Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., et al.: The matter of heartbleed. In: Proceedings of the 2014 conference on internet measurement conference. pp. 475–488 (2014), *https://doi.org/10.1145/2663716.2663755*
19. Ehlers, R.: Efficient temporal logic runtime monitoring for tiny systems. In: International Conference on Tests and Proofs. pp. 3–21. Springer (2024), *https://www.ruediger-ehlers.de/papers/tap2024.pdf*
20. Filliâtre, J.C., Paskevich, A.: Why3—where programs meet provers. In: Programming Languages and Systems: 22nd European Symposium on Programming, ESOP 2013, Held As Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 22. pp. 125–128. Springer (2013), *https://doi.org/10.1007/978-3-642-37036-6_8*
21. Finkbeiner, B., Oswald, S., Passing, N., Schwenger, M.: Verified rust monitors for lola specifications. In: International Conference on Runtime Verification. pp. 431–450. Springer (2020), *https://doi.org/10.1007/978-3-030-60508-7_24*
22. Fisher, M., Mascardi, V., Rozier, K.Y., Schlingloff, B.H., Winikoff, M., Yorke-Smith, N.: Towards a framework for certification of reliable autonomous systems. Autonomous Agents and Multi-Agent Systems **35**, 1–65 (2021). https://doi.org/https://doi.org/10.1007/s10458-020-09487-2
23. Goodloe, A.E., Havelund, K.: High-integrity runtime verification. Computer **57**(4), 37–45 (2024), *https://doi.org/10.1109/MC.2023.3322902*
24. 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). https://doi.org/https://doi.org/10.1007/978-3-030-76384-8_10
25. Johannsen, C., Anderson, M., Burken, W., Diersen, E., Edgren, J., Glick, C., Jou, S., Kumar, A., Levandowski, J., Moyer, E., et al.: Openuas version 1.0. In: 2021 International Conference on Unmanned Aircraft Systems (ICUAS). pp. 1449–1458. IEEE (2021), *https://doi.org/10.1109/ICUAS51884.2021.9476814*
26. Johannsen, C., Jones, P., Kempa, B., Rozier, K.Y., Zhang, P.: 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. pp. 483–497. Springer (2023), *https://research.temporallogic.org/papers/JJKRZ23.pdf*
27. 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), *https://research.temporallogic.org/papers/JKJRW23.pdf*
28. Kallwies, H., Leucker, M., Schmitz, M., Schulz, A., Thoma, D., Weiss, A.: Tessla–an ecosystem for runtime verification. In: International Conference on Runtime Verification. pp. 314–324. Springer (2022), *https://doi.org/10.1007/978-3-031-17196-3_20*
29. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: 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). pp. 196–214. Lecture Notes in Computer Science (LNCS), Springer, Vienna, Austria (September 2020), *http://research.temporallogic.org/papers/KZJZR20.pdf*
30. Kempa, B.C.S.: Enumerating test cases for mltl runtime monitors. In: Engineering trust in space with runtime verification. Ph.D. thesis. Iowa State University (2024)

31. Kosaian, K., Wang, Z., Sloan, E., Rozier, K.: Formalizing mltl formula progression in isabelle/hol. arXiv preprint arXiv:2410.03465 (2024), *https://arxiv.org/abs/2410.03465*
32. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-time systems **2**(4), 255–299 (1990), *https://doi.org/10.1007/BF01995674*
33. Lattuada, A., Hance, T., Bosamiya, J., Brun, M., Cho, C., LeBlanc, H., Srinivasan, P., Achermann, R., Chajed, T., Hawblitzel, C., et al.: Verus: A practical foundation for systems verification. In: Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles. pp. 438–454 (2024), *https://doi.org/10.1145/3694715.3695952*
34. Lattuada, A., Hance, T., Cho, C., Brun, M., Subasinghe, I., Zhou, Y., Howell, J., Parno, B., Hawblitzel, C.: Verus: Verifying rust programs using linear ghost types. Proceedings of the ACM on Programming Languages **7**(OOPSLA1), 286–315 (2023), *https://doi.org/10.1145/3586037*
35. Leveson, N.G., Turner, C.S.: An investigation of the therac-25 accidents. Computer **26**(7), 18–41 (1993), *https://doi.org/10.1109/MC.1993.274940*
36. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability checking for mission-time LTL. In: International Conference on Computer Aided Verification. pp. 3–22. Springer (2019). https://doi.org/https://doi.org/10.1007/978-3-030-25543-5_1
37. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pp. 152–166. Springer (2004), *https://doi.org/10.1007/978-3-540-30206-3_12*
38. Müller, P., Schwerhoff, M., Summers, A.J.: Viper: A verification infrastructure for permission-based reasoning. In: Verification, Model Checking, and Abstract Interpretation: 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings 17. pp. 41–62. Springer (2016), *https://doi.org/10.1007/978-3-662-49122-5_2*
39. Pinho, A., Couto, L., Oliveira, J.: Towards rust for critical systems. In: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 19–24. IEEE (2019), *https://doi.org/10.1109/ISSREW.2019.00036*
40. Raszyk, M., Basin, D., Traytel, D.: Multi-head monitoring of metric dynamic logic. In: International Symposium on Automated Technology for Verification and Analysis. pp. 233–250. Springer (2020), *https://doi.org/10.1007/978-3-030-59152-6_13*
41. Reinbacher, T., Függer, M., Brauer, J.: Runtime verification of embedded real-time systems. Formal methods in system design **44**, 203–239 (2014), *https://doi.org/10.1007/s10703-013-0199-z*
42. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science (LNCS), vol. 8413, pp. 357–372. Springer-Verlag (April 2014), *https://doi.org/10.1007/978-3-642-54862-8_24*
43. Schneider, J., Basin, D., Krstić, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: Runtime Verification: 19th International Conference, RV 2019, Porto, Portugal, October 8–11, 2019, Proceedings 19. pp. 310–328. Springer (2019), *https://doi.org/10.1007/978-3-030-32079-9_18*
44. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2u2: monitoring and diagnosis of security threats for unmanned aerial systems. In: Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings. pp. 233–249. Springer (2015), *https://doi.org/10.1007/978-3-319-23820-3_15*
45. Schumann, J., Rozier, K.Y., Reinbacher, T., Mengshoel, O.J., Mbaya, T., Ippolito, C.: Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. In: PHM. pp. 381–401 (October 2013), *https://research.temporallogic.org/papers/SRRMMI15.pdf*
46. Scott, R.G., Dodds, M., Perez, I., Goodloe, A.E., Dockins, R.: Trustworthy runtime verification via bisimulation (experience report). Proceedings of the ACM on Programming Languages **7**(ICFP), 305–321 (2023), *https://doi.org/10.1145/3607841*
47. Seidel, L., Beier, J.: Bringing rust to safety-critical systems in space. In: 2024 Security for Space Systems (3S). pp. 1–8. IEEE (2024), *https://doi.org/10.23919/3S60530.2024.10592287*

48. Sharma, A., Sharma, S., Tanksalkar, S.R., Torres-Arias, S., Machiry, A.: Rust for embedded systems: Current state and open problems. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 2296–2310 (2024), *https://doi.org/10.1145/3658644.3690275*
49. STMicroelectronics: Discovery kit with STM32F303VC MCU, *https://www.st.com/resource/en/user_manual/um1570-discovery-kit-with-stm32f303vc-mcu-stmicroelectronics.pdf*
50. Weston, D.: Windows security best practices for integrating and managing security tools (2024), *https://www.microsoft.com/en-us/security/blog/2024/07/27/windows-security-best-practices-for-integrating-and-managing-security-tools/*
51. Zhang, P., Aurandt, A., Dureja, R., Jones, P.H., Rozier, K.Y.: Model predictive runtime verification for cyber-physical systems with real-time deadlines. In: International Conference on Formal Modeling and Analysis of Timed Systems. pp. 158–180. Springer (2023), *https://research.temporallogic.org/papers/ZADJR23.pdf*

A Boolean Connectives

Algorithm 4: Negation Operator: $\varphi = \neg\psi$

```
1  procedure Negation(ψ)
2      Initialize:
3          φ.next_time = 0        // Initialize φ.next_time; stores the next time for ψ
       Input: Node: ψ
4      T_ψ = ψ.read(φ.read_1_ptr, φ.next_time)                    // Read Node ψ
5      if T_ψ ≠ Empty then                                          // New T_ψ
6          if !(T_ψ.v) then                                        // T_ψ.v = false
7              φ.next_time = T_ψ.τ + 1
8              φ.write(true, T_ψ.τ)                    // Writing T_φ = (true, T_ψ.τ)
9          else                                                    // T_ψ.v = true
10             φ.next_time = T_ψ.τ + 1
11             φ.write(false, T_ψ.τ)                   // Writing T_φ = (false, T_ψ.τ)
```

Theorem 4 (Correctness of the Negation Operator). *Given the child node ψ, Algorithm 4 writes the tuple T_φ to φ's SCQ when sufficient data is available such that for all $i \geq 0$, $T_\varphi = (\mathsf{true}, i)$ iff $\pi, i \models \neg\psi$.*

Proof. The $\varphi.next\_time$ variable determines what timestamp is desired from node ψ to make the next evaluation. The $\varphi.next\_time$ variable is initialized to 0 on line 3. If a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to $i+1$ (lines 7 and 10).

The $\varphi.next\_time$ variable is then an input into the $read$ function on line 4 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time, T_\psi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi, i \models \neg\psi \rightarrow T_\varphi = (\mathsf{true}, i)$: We consider all possibilities of T_ψ to determine if $\pi, i \models \neg\psi$:

(1) T_ψ *is empty*: There is no new information based in $\varphi.next\_time$ to evaluate if $\pi, i \models \neg\psi$; therefore, the algorithm does nothing.
(2) $T_\psi.v = \mathsf{false}$: $T_\psi.v = \mathsf{false}$ for $[\varphi.next\_time, T_\psi.\tau]$ such that $\pi, i \not\models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$; therefore, $(\mathsf{true}, T_\psi.\tau)$ is written to φ's SCQ on line 8.
(3) $T_\psi.v = \mathsf{true}$: $T_\psi.v = \mathsf{true}$ for $[\varphi.next\_time, T_\psi.\tau]$ such that $\pi, i \models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$; therefore, $(\mathsf{false}, T_\psi.\tau)$ is written to φ's SCQ on line 11.

(if direction) $T_\varphi = (\mathsf{true}, i) \rightarrow \pi, i \models \neg\psi$:

$T_\varphi = (\mathsf{true}, i)$ tuples are only written to φ's SCQ on line 8, which requires that $T_\psi.v = \mathsf{false}$ from $[\varphi.next\_time, T_\psi.\tau]$; therefore, $\pi, i \not\models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$. As a result, $\pi, i \models \neg\psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$.

$T_\varphi = (\mathsf{false}, i)$ tuples are only written to φ's SCQ on line 11, which requires that $T_\psi.v = \mathsf{true}$ from $[\varphi.next\_time, T_\psi.\tau]$; therefore, $\pi, i \models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$. As a result, $\pi, i \not\models \neg\psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$.

There is only one condition under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi,i \models \neg\psi$: T_ψ is empty.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true},i)$ are only written to φ's SCQ iff $\pi,i \models \neg\psi$, and $T_\varphi = (\mathsf{false},i)$ are only written to φ's SCQ iff $\pi,i \not\models \neg\psi$. □

Algorithm 5: And Operator: $\varphi = \psi \wedge \xi$

```
1  procedure And(ψ,ξ)
2      Initialize:
3          φ.next_time = 0  // Initialize φ.next_time; stores the next time for ψ and ξ
       Input: Node: ψ; Node: ξ
4      T_ψ = ψ.read(φ.read_1_ptr, φ.next_time)                    // Read Node ψ
5      T_ξ = ξ.read(φ.read_2_ptr, φ.next_time)                    // Read Node ξ
6      if T_ψ ≠ Empty and T_ξ ≠ Empty then                        // New T_ψ and T_ξ
7          if T_ψ.v and T_ξ.v then                                // T_ψ.v = true and T_ξ.v = true
8              τ_min = min(T_ψ.τ, T_ξ.τ)
9              φ.next_time = τ_min + 1
10             φ.write(true, τ_min)                               // Writing T_φ = (true, τ_min)
11             return
12         else if !(T_ψ.v) and !(T_ξ.v) then                     // T_ψ.v = false and T_ξ.v = false
13             τ_max = max(T_ψ.τ, T_ξ.τ)
14             φ.next_time = τ_max + 1
15             φ.write(false, τ_max)                              // Writing T_φ = (false, τ_max)
16             return
17     if T_ψ ≠ Empty then                                        // New T_ψ
18         if !(T_ψ.v) then                                       // T_ψ.v = false
19             φ.next_time = T_ψ.τ + 1
20             φ.write(false, T_ψ.τ)                              // Writing T_φ = (false, T_ψ.τ)
21     if T_ξ ≠ Empty then                                        // New T_ξ
22         if !(T_ξ.v) then                                       // T_ξ.v = false
23             φ.next_time = T_ξ.τ + 1
24             φ.write(false, T_ξ.τ)                              // Writing T_φ = (false, T_ξ.τ)
```

Theorem 5 (Correctness of the And Operator). *Given the interval $[lb,ub]$ and two children nodes ψ and ξ, Algorithm 5 writes the tuple T_φ to φ's SCQ when sufficient data is available such that for all $i \geq 0$, $T_\varphi = (\mathsf{true},i)$ iff $\pi,i \models \psi \wedge \xi$.*

Proof. The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized to 0 on line 3. If a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to $i+1$ (lines 9, 14, 19, and 23).

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 4–5 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time, T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time, T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi,i \models \psi \wedge \xi \rightarrow T_\varphi = (\mathsf{true},i)$: We consider all possible combinations of T_ψ and T_ξ to determine if $\pi,i \models \psi \wedge \xi$:

(1) T_ψ *and* T_ξ *are both empty*: There is no new information based on $\varphi.next\_time$ to evaluate if $\pi,i \models \psi \wedge \xi$; therefore, the algorithm does nothing.

(2) *(*$T_\psi.v = \mathsf{true}$ *and* T_ξ *is empty) or (*T_ψ *is empty and* $T_\xi.v = \mathsf{true}$*)*: There is not enough information to determine if both $\pi,i \models \psi$ and $\pi,i \models \xi$; therefore, the algorithm does nothing.

(3) $T\psi.v = \mathsf{true}$ *and* $T_\xi.v = \mathsf{true}$: If both $T_\psi.v$ and $T_\xi.v$ are true, then they are both guaranteed to be true from $[\varphi.next\_time, min(T_\psi.\tau, T_\xi.\tau)]$. As a result, $\pi,i \models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau, T_\xi.\tau)$; therefore, $(\mathsf{true}, min(T_\psi.\tau, T_\xi.\tau))$ is written to φ's SCQ on line 10.

(4) $T_\psi.v = \mathsf{false}$ *and* $T_\xi.v = \mathsf{false}$: If both $T_\psi.v$ and $T_\xi.v$ are false, then they are both guaranteed to be false from $[\varphi.next\_time, min(T_\psi.\tau, T_\xi.\tau)]$, but one of them is false from $[\varphi.next\_time, max(T_\psi.\tau, T_\xi.\tau)]$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq max(T_\psi.\tau, T_\xi.\tau)$; therefore, $(\mathsf{false}, max(T_\psi.\tau, T_\xi.\tau))$ is written to φ's SCQ on line 15.

(5) $T_\psi.v = \mathsf{false}$ *and* T_ξ *is empty*: $T_\psi.v = \mathsf{false}$ for $[\varphi.next\_time, T_\psi.\tau]$ such that $\pi,i \not\models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$; therefore, $(\mathsf{false}, T_\psi.\tau)$ is written to φ's SCQ on line 20.

(6) T_ψ *is empty and* $T_\xi.v = \mathsf{false}$: $T_\xi.v = \mathsf{false}$ for $[\varphi.next\_time, T_\xi.\tau]$ such that $\pi,i \not\models \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$; therefore, $(\mathsf{false}, T_\xi.\tau)$ is written to φ's SCQ on line 24.

(if direction) $T_\varphi = (\mathsf{true}, i) \rightarrow \pi,i \models \psi \wedge \xi$:

$T_\varphi = (\mathsf{true}, i)$ tuples are only written to φ's SCQ on line 10, which requires that both $T_\psi.v = \mathsf{true}$ and $T_\xi.v = \mathsf{true}$; therefore, $\pi,i \models \psi$ and $\pi,i \models \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau, T_\xi.\tau)$. As a result, $\pi,i \models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau, T_\xi.\tau)$.

$T_\varphi = (\mathsf{false}, i)$ tuples are only written to φ's SCQ on lines 15, 20, and 24:

(1) *line 15*: It is guaranteed that both $T_\psi.v$ and $T_\xi.v$ are false from $[\varphi.next\_time, min(T_\psi.\tau, T_\xi.\tau)]$, but one of them is false from $[\varphi.next\_time, max(T_\psi.\tau, T_\xi.\tau)]$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq max(T_\psi.\tau, T_\xi.\tau)$.

(2) *line 20*: $T_\psi.v = \mathsf{false}$ for $[\varphi.next\_time, T_\psi.\tau]$ such that $\pi,i \not\models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$.

(3) *line 24*: $T_\xi.v = \mathsf{false}$ for $[\varphi.next\_time, T_\xi.\tau]$ such that $\pi,i \not\models \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$.

There are three conditions under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi,i \models \psi \wedge \xi$:

(1) T_ψ *and* T_ξ *are both empty*: There is not enough information to evaluate if $\pi,i \models \psi \wedge \xi$.

(2) $T_\psi.v = \mathsf{true}$ *and* T_ξ *is empty*: There is not enough information to determine if $\pi,i \models \xi$; therefore if $\pi,i \models \psi \wedge \xi$ cannot be determined.

(3) T_ψ *is empty and* $T_\xi.v = \mathsf{true}$: There is not enough information to determine if $\pi,i \models \psi$; therefore if $\pi,i \models \psi \wedge \xi$ cannot be determined.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true}, i)$ are only written to φ's SCQ iff $\pi,i \models \psi \wedge \xi$, and $T_\varphi = (\mathsf{false}, i)$ are only written to φ's SCQ iff $\pi,i \not\models \psi \wedge \xi$. □

Algorithm 6: Or Operator: $\varphi = \psi \vee \xi$

```
1  procedure Or(ψ,ξ)
2      Initialize:
3          φ.next_time = 0 // Initialize φ.next_time; stores the next time for ψ and ξ
       Input: Node: ψ; Node: ξ
4      T_ψ = ψ.read(φ.read_1_ptr, φ.next_time)                    // Read Node ψ
5      T_ξ = ξ.read(φ.read_2_ptr, φ.next_time)                    // Read Node ξ
6      if T_ψ ≠ Empty and T_ξ ≠ Empty then                        // New T_ψ and T_ξ
7          if T_ψ.v and T_ξ.v then                                // T_ψ.v = true and T_ξ.v = true
8              τ_max = max(T_ψ.τ, T_ξ.τ)
9              φ.next_time = τ_max + 1
10             φ.write(true, τ_max)                               // Writing T_φ = (true, τ_max)
11             return
12         else if !(T_ψ.v) and !(T_ξ.v) then                     // T_ψ.v = false and T_ξ.v = false
13             τ_min = min(T_ψ.τ, T_ξ.τ)
14             φ.next_time = τ_min + 1
15             φ.write(false, τ_min)                              // Writing T_φ = (false, τ_min)
16             return
17     if T_ψ ≠ Empty then                                        // New T_ψ
18         if T_ψ.v then                                          // T_ψ.v = true
19             φ.next_time = T_ψ.τ + 1
20             φ.write(true, T_ψ.τ)                               // Writing T_φ = (true, T_ψ.τ)
21     if T_ξ ≠ Empty then                                        // New T_ξ
22         if T_ξ.v then                                          // T_ξ.v = true
23             φ.next_time = T_ξ.τ + 1
24             φ.write(true, T_ξ.τ)                               // Writing T_φ = (true, T_ξ.τ)
```

Theorem 6 (Correctness of the Or Operator). *Given the interval* $[lb,ub]$ *and two children nodes* ψ *and* ξ*, Algorithm 6 writes the tuple* T_φ *to* φ*'s SCQ when sufficient data is available such that for all* $i \geq 0$*,* $T_\varphi = (\mathsf{true},i)$ *iff* $\pi,i \models \psi \vee \xi$*.*

Proof. The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized to 0 on line 3. If a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to $i+1$ (lines 9, 14, 19, and 23).

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 4–5 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time,T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time,T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi,i \models \psi \vee \xi \rightarrow T_\varphi = (\mathsf{true},i)$: We consider all possible combinations of T_ψ and T_ξ to determine if $\pi,i \models \psi \vee \xi$:

(1) T_ψ *and* T_ξ *are both empty*: There is no new information based on $\varphi.next\_time$ to evaluate if $\pi,i \models \psi \vee \xi$; therefore, the algorithm does nothing.
(2) *(*$T_\psi.v = \mathsf{false}$ *and* T_ξ *is empty) or (*T_ψ *is empty and* $T_\xi.v = \mathsf{false}$*)*: There is not enough information to determine if either $\pi,i \models \psi$ or $\pi,i \models \xi$; therefore, the algorithm does nothing.
(3) $T\psi.v = \mathsf{true}$ *and* $T_\xi.v = \mathsf{true}$: If both $T_\psi.v$ and $T_\xi.v$ are true, then they are both guaranteed to be true from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$, but one of them is true from $[\varphi.next\_time,max(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq max(T_\psi.\tau,T_\xi.\tau)$; therefore, $(\mathsf{true},max(T_\psi.\tau,T_\xi.\tau))$ is written to φ's SCQ on line 10.
(4) $T_\psi.v = \mathsf{false}$ *and* $T_\xi.v = \mathsf{false}$: If both $T_\psi.v$ and $T_\xi.v$ are false, then they are both guaranteed to be false from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \not\models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$; therefore, $(\mathsf{false},min(T_\psi.\tau,T_\xi.\tau))$ is written to φ's SCQ on line 15.
(5) $T_\psi.v = \mathsf{true}$ *and* T_ξ *is empty*: $T_\psi.v = \mathsf{true}$ for $[\varphi.next\_time,T_\psi.\tau]$ such that $\pi,i \models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$. As a result, $\pi,i \models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$; therefore, $(\mathsf{true},T_\psi.\tau)$ is written to φ's SCQ on line 20.
(6) T_ψ *is empty and* $T_\xi.v = \mathsf{true}$: $T_\xi.v = \mathsf{true}$ for $[\varphi.next\_time,T_\xi.\tau]$ such that $\pi,i \models \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$. As a result, $\pi,i \models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$; therefore, $(\mathsf{true},T_\xi.\tau)$ is written to φ's SCQ on line 24.

(if direction) $T_\varphi = (\mathsf{true},i) \rightarrow \pi,i \models \psi \vee \xi$:

$T_\varphi = (\mathsf{false},i)$ tuples are only written to φ's SCQ on lines 10, 20, and 24:

(1) *line 10*: It is guaranteed that both $T_\psi.v$ and $T_\xi.v$ are true from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$, but one of them is true from $[\varphi.next\_time,max(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq max(T_\psi.\tau,T_\xi.\tau)$.
(2) *line 20*: $T_\psi.v = \mathsf{true}$ for $[\varphi.next\_time,T_\psi.\tau]$ such that $\pi,i \models \psi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$. As a result, $\pi,i \models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq T_\psi.\tau$.
(3) *line 24*: $T_\xi.v = \mathsf{true}$ for $[\varphi.next\_time,T_\xi.\tau]$ such that $\pi,i \models \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$. As a result, $\pi,i \models \psi \vee \xi$ for $\varphi.next\_time \leq i \leq T_\xi.\tau$.

$T_\varphi = (\mathsf{false},i)$ tuples are only written to φ's SCQ on line 15, which requires that both $T_\psi.v = \mathsf{false}$ and $T_\xi.v = \mathsf{false}$; therefore, $\pi,i \not\models \psi$ and $\pi,i \not\models \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$. As a result, $\pi,i \not\models \psi \wedge \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$.

There are three conditions under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi,i \models \psi \vee \xi$:

(1) T_ψ *and* T_ξ *are both empty*: There is not enough information to evaluate if $\pi,i \models \psi \vee \xi$.
(2) $T_\psi.v = \mathsf{false}$ *and* T_ξ *is empty*: There is not enough information to determine if $\pi,i \models \xi$; therefore if $\pi,i \models \psi \vee \xi$ cannot be determined.

(3) T_ψ *is empty and* $T_\xi.v = \mathsf{false}$: There is not enough information to determine if $\pi,i \models \psi$; therefore if $\pi,i \models \psi \vee \xi$ cannot be determined.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true},i)$ are only written to φ's SCQ iff $\pi,i \models \psi \vee \xi$, and $T_\varphi = (\mathsf{false},i)$ are only written to φ's SCQ iff $\pi,i \not\models \psi \vee \xi$. □

Algorithm 7: Iff Operator: $\varphi = \psi \leftrightarrow \xi$

```
1  procedure Iff(ψ,ξ)
2      Initialize:
3          φ.next_time = 0 // Initialize φ.next_time; stores the next time for ψ and ξ
       Input: Node: ψ; Node: ξ
4      T_ψ = ψ.read(φ.read_1_ptr,φ.next_time)                    // Read Node ψ
5      T_ξ = ξ.read(φ.read_2_ptr,φ.next_time)                    // Read Node ξ
6      if T_ψ ≠ Empty and T_ξ ≠ Empty then                       // New T_ψ and T_ξ
7          τ_min = min(T_ψ.τ,T_ξ.τ)
8          φ.next_time = τ_min + 1
9          if T_ψ.v and T_ξ.v then                               // T_ψ.v = true and T_ξ.v = true
10             φ.write(true,τ_min)                               // Writing T_φ = (true,τ_min)
11         else if !(T_ψ.v) and !(T_ξ.v) then                    // T_ψ.v = false and T_ξ.v = false
12             φ.write(true,τ_min)                               // Writing T_φ = (true,τ_min)
13         else
14             φ.write(false,τ_min)                              // Writing T_φ = (false,τ_min)
```

Theorem 7 (Correctness of the Iff Operator). *Given the interval* $[lb,ub]$ *and two children nodes* ψ *and* ξ*, Algorithm 6 writes the tuple* T_φ *to* φ*'s SCQ when sufficient data is available such that for all* $i \geq 0$, $T_\varphi = (\mathsf{true},i)$ *iff* $\pi,i \models \psi \leftrightarrow \xi$.

Proof. The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized to 0 on line 3. If a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to $i+1$ (lines 8).

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 4–5 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time,T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time,T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi,i \models \psi \leftrightarrow \xi \rightarrow T_\varphi = (\mathsf{true},i)$: We consider all possible combinations of T_ψ and T_ξ to determine if $\pi,i \models \psi \leftrightarrow \xi$:

(1) T_ψ *is empty or* T_ξ *is empty*: There is not enough information to determine both if $\pi,i \models \psi$ and if $\pi,i \models \xi$; therefore, the algorithm does nothing.

(2) $T\psi.v = \mathsf{true}$ *and* $T_\xi.v = \mathsf{true}$: If both $T_\psi.v$ and $T_\xi.v$ are true, then they are both guaranteed to be true from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \models \psi \leftrightarrow \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$; therefore, $(\mathsf{true},min(T_\psi.\tau,T_\xi.\tau))$ is written to φ's SCQ on line 10.

(3) $T_\psi.v = \mathsf{false}$ *and* $T_\xi.v = \mathsf{false}$: If both $T_\psi.v$ and $T_\xi.v$ are false, then they are both guaranteed to be false from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \models \psi \leftrightarrow \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$; therefore, $(\mathsf{true},min(T_\psi.\tau,T_\xi.\tau))$ is written to φ's SCQ on line 12.

(4) *(*$T_\psi.v = \mathsf{true}$ *and* $T_\xi.v = \mathsf{false}$*) or (*$T_\psi.v = \mathsf{false}$ *and* $T_\xi.v = \mathsf{true}$*)*: Both $T_\psi.v$ and $T_\xi.v$ are guaranteed to be the verdicts from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$; therefore, either $\pi,i \not\models \psi \rightarrow \xi$ or $\pi,i \not\models \xi \rightarrow \psi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$. As a result, $\pi,i \not\models \psi \leftrightarrow \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$; therefore, $(\mathsf{false},min(T_\psi.\tau,T_\xi.\tau))$ is written to φ's SCQ on line 14.

(if direction) $T_\varphi = (\mathsf{true},i) \rightarrow \pi,i \models \psi \leftrightarrow \xi$:

$T_\varphi = (\mathsf{true},i)$ tuples are only written to φ's SCQ on lines 10 and 12:

(1) *line 10*: It is guaranteed that both $T_\psi.v$ and $T_\xi.v$ are true from $[\varphi.next\_time,$ $min(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \models \psi \leftrightarrow \xi$ for $\varphi.next\_time \leq i \leq max(T_\psi.\tau,T_\xi.\tau)$.

(2) *line 12*: It is guaranteed that both $T_\psi.v$ and $T_\xi.v$ are false from $[\varphi.next\_time, min(T_\psi.\tau,T_\xi.\tau)]$. As a result, $\pi,i \models \psi \leftrightarrow \xi$ for $\varphi.next\_time \leq i \leq max(T_\psi.\tau,T_\xi.\tau)$.

$T_\varphi = (\mathsf{false}, i)$ tuples are only written to φ's SCQ on line 14, which requires that either $T_\psi.v = \mathsf{true}$ and $T_\xi.v = \mathsf{false}$, or $T_\psi.v = \mathsf{false}$ and $T_\xi.v = \mathsf{true}$; therefore, either $\pi,i \not\models \psi \rightarrow \xi$ or $\pi,i \not\models \xi \rightarrow \psi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$. As a result, $\pi,i \not\models \psi \leftrightarrow \xi$ for $\varphi.next\_time \leq i \leq min(T_\psi.\tau,T_\xi.\tau)$.

There is only one condition under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi,i \models \psi \leftrightarrow \xi$: T_ψ is empty or T_ξ is empty.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true}, i)$ are only written to φ's SCQ iff $\pi,i \models \psi \leftrightarrow \xi$, and $T_\varphi = (\mathsf{false}, i)$ are only written to φ's SCQ iff $\pi,i \not\models \psi \leftrightarrow \xi$. □

B Release ($\mathcal{R}$) Operator

Algorithm 8: Release Operator: $\varphi = \psi \; \mathcal{R}_{[lb,ub]} \; \xi$

```
1  Initialize:
2      φ.previous = −1                    // Initialize φ.previous; stores the last i written
3      φ.next_time = lb                   // Initialize φ.next_time; stores the next time for ψ and ξ
4  procedure Release(ψ,ξ)
       Input: Node: ψ; Node: ξ
5      T_ψ = ψ.read(φ.read_1_ptr,φ.next_time)                          // Read Node ψ
6      T_ξ = ξ.read(φ.read_2_ptr,φ.next_time)                          // Read Node ξ
7      if T_ξ ≠ Empty then                                             // New T_ξ
8          if !(T_ξ.v) then                                            // T_ξ.v = false
9              φ.previous = T_ξ.τ − lb
10             φ.next_time = T_ξ.τ + 1
11             φ.write(false, T_ξ.τ − lb)                  // Writing T_φ = (false, T_ξ.τ − lb)
12             return
13         if T_ψ ≠ Empty then                                         // New T_ψ and T_ξ
14             τ_min = min(T_ψ.τ, T_ξ.τ)
15             φ.next_time = τ_min + 1
16             if T_ψ.v then                                           // T_ψ.v = true and T_ξ.v = true
17                 φ.previous = τ_min − lb
18                 φ.write(true, τ_min − lb)               // Writing T_φ = (true, τ_min − lb)
19                 return
20         if T_ξ.τ > φ.previous + ub then      // (T_ψ = Empty or T_ψ.v = false) and T_ξ.v = false
21             φ.previous = T_ξ.τ − ub
22             φ.next_time = max(φ.next_time, φ.previous + lb + 1)
23             φ.write(true, T_ξ.τ − ub)                   // Writing T_φ = (true, T_ξ.τ − ub)
```

Theorem 8 (Correctness of the Release Operator). *Given the interval $[lb, ub]$ and two children nodes ψ and ξ, Algorithm 8 writes the tuple T_φ to φ's SCQ when sufficient data is available such that for all $i \geq 0$, $T_\varphi = (\mathsf{true}, i)$ iff $\pi,i \models \psi \; \mathcal{R}_{[lb,ub]} \; \xi$.*

Proof. The $\varphi.previous$ variable stores the previous time index i that node $\varphi = \psi \; \mathcal{R}_{[lb,ub]} \; \xi$ wrote to its SCQ. Before the first tuple is produced, $\varphi.previous$ is initialized to -1 (line 2) such that the condition on line 20 can be rewritten as $T_\xi.\tau > -1 + ub$ or rather $T_\xi.\tau \geq 0 + ub \equiv T_\xi.\tau \geq ub$. Whenever a verdict-timestamp tuple is written, the $\varphi.previous$ variable is updated to that timestamp (lines 9, 17, and 21).

The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized on line 3 to lb since $\psi \; \mathcal{R}_{[lb,ub]} \; \xi$ can only be evaluated when $|\pi| > i + lb$ on the interval $[i+lb, i+ub]$ where $i \geq 0$, such that $[0, lb-1]$ is never required for evaluation. The $\varphi.next\_time$ variable is then updated during execution based on what is unknown about ψ and ξ. If both T_ψ and T_ξ are not empty, then if $T_\psi.\tau \geq T_\xi.\tau$, its unknown if $\exists j \in [T_\xi.\tau + 1, T_\psi.\tau]$ such that $\pi,j \not\models \xi$, and if $T_\psi.\tau \leq T_\xi.\tau$, then its unknown if $\exists k \in [T_\psi.\tau + 1, T_\xi.\tau]$ such that $\pi,k \models \psi$. Therefore, $\varphi.next\_time$ will

be updated to $min(T_\psi.\tau,T_\xi.\tau)+1$ on line 15. Because of early evaluation (following from Axiom 2), if a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to whichever is greater: $i+lb+1$ (i.e., the lb of the next evaluation) or $min(T_\psi.\tau,T_\xi.\tau)+1$ (as described above) on lines 10 and 22. Note that on line 10 this can be rewritten as just $T_\xi.\tau+1$ since $i+lb+1 \equiv (T_\xi.\tau-lb)+lb+1 \equiv T_\xi.\tau+1$, and if $T_\psi.\tau \leq T_\xi.\tau$, then $T_\psi.\tau \leq i+lb+1$. Figures 12, 13, 14, and 15 illustrate examples of how the $\varphi.next\_time$ variable is updated.

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 5–6 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time,T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time,T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi,i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi \rightarrow T_\varphi = (\mathsf{true},i)$:
We consider all possible combinations of T_ψ and T_ξ to determine if $\pi,i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$:

(1) *T_ψ and T_ξ are both empty*: There is no new information based on $\varphi.next\_time$ to evaluate if $\pi,i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$; therefore, the algorithm does nothing.

(2) *T_ψ is not empty and T_ξ is empty*: There is not enough information to determine if $\exists j \in [i+lb,i+ub]$ such that $\pi,j \not\models \xi$, then $\exists k < j$ where $k \in [i+lb,i+ub]$ such that $\pi,k \models \psi$ (where $i = \varphi.previous+1$); therefore, the algorithm does nothing.

(3) *$T_\xi.v = \mathsf{false}$ (and T_ψ is empty or $T_\psi.v = \mathsf{false}$ or $T_\psi.v = \mathsf{true}$)*: $T_\xi.v$ is the verdict from $[\varphi.next\_time,T_\xi.\tau]$ such that $\varphi.next\_time$ is lb (initial condition) or was set by the previous execution to $\varphi.previous + lb + 1$ or $min(T_\psi.\tau,T_\xi.\tau)+1$ as described above. Therefore, $T_\xi.v = \mathsf{false}$ at $\varphi.previous + lb + 1$ and/or $\nexists k \in [\varphi.previous + lb+1,\varphi.next\_time-1]$, $\pi,k \models \psi$ (since the condition on line 16 was not met by the previous execution). As a result, $\pi,i \not\models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ (following directly from Axiom 2) for $\varphi.previous < i \leq T_\xi.\tau-lb$; therefore, $(\mathsf{false},T_\xi.\tau-lb)$ is written to φ's SCQ on line 11. Figure 12 provides a visualization of this case.

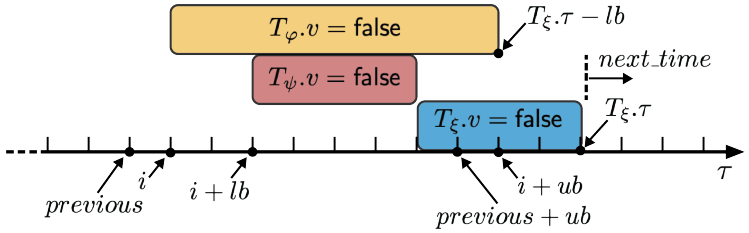


Fig. 12. Pictorial representation of $\varphi = \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for Theorem 8 Case (3).

(4) $T_\psi.v = \mathsf{true}$ *and* $T_\xi.v = \mathsf{true}$: If both $T_\psi.v$ and $T_\xi.v$ are true, then they are both guaranteed to be true from $[\varphi.next\_time,min(T_\psi.\tau,T_\xi.\tau)]$. Furthermore, $T_\xi.v$ is guaranteed to have never been false from $[\varphi.previous+lb+1,T_\xi.\tau]$ since the result would have been written on line 11 and the $\varphi.next\_time$ and $\varphi.previous$ variables would have been updated (lines 9–10). As a result, $\pi,i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq min(T_\psi.\tau,T_\xi.\tau)-lb$; therefore, $(\mathsf{true},min(T_\psi.\tau,T_\xi.\tau)-lb)$ is written to φ's SCQ on line 18. Figure 13 provides a visualization of this case.

(5) $T_\xi.v = \mathsf{false}$ *(and T_ψ is empty or $T_\psi.v = \mathsf{true}$)*: There are two sub-cases to consider:

(a) *There is enough information to determine that $\forall j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$*: $T_\xi.v$ is guaranteed to have never been false from $[\varphi.previous+lb+1,T_\xi.\tau]$ since the result would have been written on line 11 and the $\varphi.next\_time$ and $\varphi.previous$ variables would have been updated (lines 9–10). If $T_\xi.\tau > \varphi.previous+ub$, then $\forall j \in [i+lb,i+ub]$ such that $\pi,j \models \xi$ where $i \in [\varphi.previous+1,T_\xi.\tau-ub]$. As a result, $\pi,i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi.\tau-ub$ (following directly from

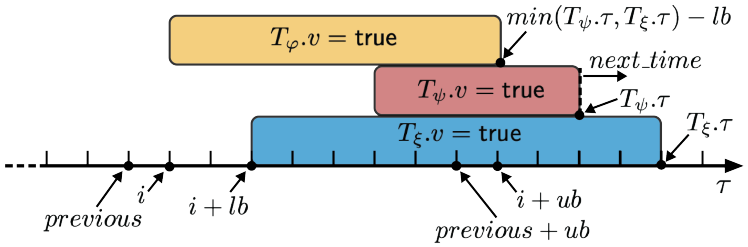


Fig. 13. Pictorial representation of $\varphi = \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for Theorem 8 Case (4).

Axiom 2); therefore, $(\text{true}, T_\xi.\tau - ub)$ is written to φ's SCQ on line 23. Figure 14 provides a visualization of this case.

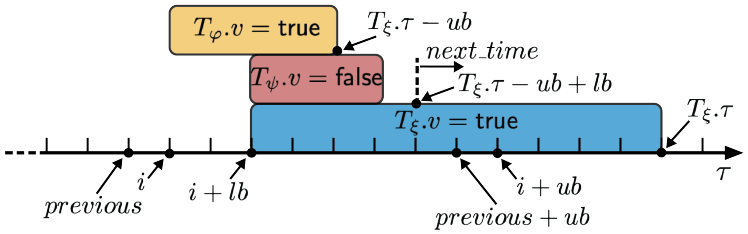


Fig. 14. Pictorial representation of $\varphi = \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for Theorem 8 Case (5)(a)

(b) *There is currently not enough information to determine if* $\exists j \in [i+lb, i+ub]$ *such that* $\pi, j \not\models \xi$: If $T_\xi.\tau \leq \varphi.previous + ub$, then there is not enough information to guarantee that $T_\xi.v = \text{true}$ from $[i+lb, i+ub]$ where $i = \varphi.previous + 1$. There is still a chance that $\exists j \in [T_\xi.\tau + 1, i+ub]$ such that $\pi, j \not\models \xi$; therefore, the algorithm does not write a tuple. Figure 15 provides a visualization of this case.

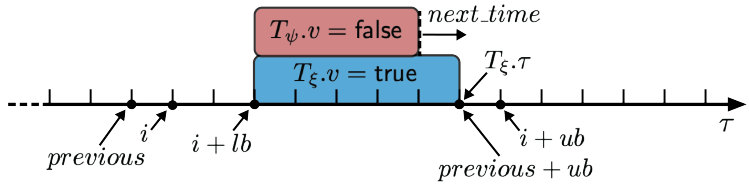


Fig. 15. Pictorial representation of $\varphi = \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for Theorem 8 Case (5)(b)

(if direction) $T_\varphi = (\text{true}, i) \rightarrow \pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$:

$T_\varphi = (\text{true}, i)$ tuples are only written on lines 18 and 23 such that $T_\xi.v = \text{true}$. In both cases, $T_\xi.v$ is guaranteed to have never been false from $[\varphi.previous + lb + 1, T_\xi.\tau]$ since the result would have been written on line 11 and the $\varphi.next\_time$ and $\varphi.previous$ variables would have been updated (lines 9–10). If $T_\psi.v = \text{true}$, then $\pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq min(T_\psi.\tau, T_\xi.\tau) - lb$ (line 18). If $T_\psi.v \neq \text{true}$ and $T_\xi.\tau > \varphi.previous + ub$, then $\forall j \in [i+lb, i+ub]$ such that $\pi, j \models \xi$; therefore, $\pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi.\tau - ub$ (line 23).

$T_\varphi = (\text{false}, i)$ tuples are only written to φ's SCQ on line 11, which requires that $T_\xi.v = \text{false}$ is the verdict from $[\varphi.next\_time, T_\xi.\tau]$ such that $\varphi.next\_time$ is lb (initial condition) or was set by the previous execution to $\varphi.previous + lb + 1$ or $min(T_\psi.\tau, T_\xi.\tau) + 1$ as described above. Therefore, $T_\xi.v = \text{false}$ at $i+lb$ and/or $\nexists k \in [i+lb, \varphi.next\_time]$, $\pi, k \models \psi$. As a result, $\pi, i \not\models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi.\tau - lb$ (following directly from Axiom 2).

There are three conditions under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$:

(1) *T_ψ and T_ξ are both empty*: There is not enough information to evaluate if $\pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$.

(2) T_ψ *is not empty and* T_ξ *is empty*: There is not enough information to determine if $\exists j \in [i+lb, i+ub]$ such that $\pi, j \not\models \xi$, then $\exists k < j$ where $k \in [i+lb, i+ub]$ such that $\pi, k \models \psi$; therefore, if $\pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ cannot be determined.

(3) $T_\xi.v = \mathsf{false}$ *and* $T_\xi.\tau \leq \varphi.previous + ub$ *and* $T_\psi \neq \mathsf{false}$: There is not enough information to guarantee that $T_\xi.v = \mathsf{true}$ from $[i+lb, i+ub]$ where $i = \varphi.previous + 1$. There is still a chance that $\exists j \in [T_\xi.\tau + 1, i+ub]$ such that $\pi, j \not\models \xi$; therefore, if $\pi, i \models \psi\ \mathcal{R}_{[lb,ub]}\ \xi$ cannot be determined.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true}, i)$ are only written to φ's SCQ iff $\pi, i \models \psi \mathcal{R}_{[lb,ub]} \xi$, and $T_\varphi = (\mathsf{false}, i)$ are only written to φ's SCQ iff $\pi, i \not\models \psi \mathcal{R}_{[lb,ub]} \xi$. □

C Correctness of the Since ($\mathcal{S}$) Operator

Theorem 9 (Correctness of the Since Operator). *Given the interval* $[lb, ub]$ *and two children nodes* ψ *and* ξ*, Algorithm 3 writes the tuple* T_φ *to* φ*'s SCQ when sufficient data is available such that for all* $i \geq 0$*,* $T_\varphi = (\mathsf{true}, i)$ *iff* $\pi, i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$*.*

Proof. The $\varphi.edge$ variable stores the latest timestamp where $T_\xi.v = \mathsf{true}$ (line 13). The $\varphi.edge$ variable is initialized to -1 (line 2) such that before $T_\xi.v = \mathsf{true}$, the conditions on lines 20 and 34 can be rewritten as $(-1 \leq \varphi.previous - ub$ or $\varphi.edge = -1) \equiv \mathsf{true}$ and $(-1 > \varphi.previous - ub$ and $\varphi.edge \neq -1) \equiv \mathsf{false}$, respectively.

The $\varphi.previous$ variable stores the previous time index i that node $\varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ wrote to its SCQ. Before the first tuple is produced, $\varphi.previous$ is initialized to -1 (line 3) such that the condition on line 8 can be rewritten as $-1+1-lb < 0 \equiv 0 - lb < 0$, the conditions on lines 15, 22, and 29 can be rewritten as $T_\xi.\tau > -1 - lb$ or rather $T_\xi.\tau \geq 0 - lb$, the condition on line 20 can be rewritten as $(\varphi.edge \leq -1 - ub$ or $\varphi.edge = -1) \equiv (\varphi.edge = -1)$, and the condition on line 34 can be rewritten as $(T_\psi.v$ and $T_\psi.\tau > -1 - lb$ and $\varphi.edge \geq 0 - ub$ and $\varphi.edge \neq -1) \equiv (T_\psi.v$ and $T_\psi.\tau \geq 0 - lb$ and $\varphi.edge \neq -1)$. Whenever a verdict-timestamp tuple is written, the $\varphi.previous$ variable is updated to that timestamp (lines 9, 16, 23, 30, and 35).

Note that if $i - ub < 0$, then $\forall i \in [0, ub-1]$, we can only consider the interval $[0, i-lb]$ for evaluation (lines 20 and 34). Line 20 checks whether $T_\xi.v$ never equals true between $[i-ub, i-lb]$; if $i - ub < 0$, then $\varphi.edge \leq \varphi.previous - ub \equiv (\varphi.edge \geq -1) \leq (\varphi.previous - ub < -1) \equiv \mathsf{false}$ since $i = \varphi.previous + 1$; therefore, line 20 is contingent on whether $\varphi.edge = -1$ (i.e., never a time yet where $T_\xi.v = \mathsf{true}$). Line 34 performs a similar check to see if $T_\xi.v = \mathsf{true}$ between $[i - ub, i - lb]$; if $i - ub < 0$, then $\varphi.edge > \varphi.previous - ub \equiv (edge \geq -1) > (\varphi.previous - ub < -1) \equiv \mathsf{true}$ since $i = \varphi.previous + 1$; therefore, line 34 is contingent on whether $\varphi.edge \neq -1$ (i.e., there was a time after or at timestamp 0 where $T_\xi.v = \mathsf{true}$).

The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized to 0 on line 4 and is then updated during execution based on what is known about ξ. When $T_\xi.v = \mathsf{true}$ (and $T_\xi.\tau < i - lb$), its unknown if $\forall k > T_\xi.\tau$ where $j \in [i-ub, i-lb]$, $\pi, k \models \psi$, and if $T_\xi.v = \mathsf{false}$ from $[i-ub, T_\xi.\tau]$ (and $T_\xi.\tau < i - lb$), its known if $\exists j \in [T_\xi.\tau + 1, i-lb]$ such that $\pi, j \models \xi$, and if both $T_\psi.v = \mathsf{false}$ and $T_\xi.v = \mathsf{false}$ (and $T_\xi.\tau < i - lb$), it is known if $\exists j \in [T_\xi.\tau + 1, i-lb]$ where $\pi, j \models \xi$ such that $\forall k > j$ where $j \in [T_\xi.\tau + 1, i-lb]$, $\pi, k \models \psi$. Therefore, $\varphi.next\_time$ will be updated to $T_\xi.\tau + 1$ on lines 14, 21, and 28. Because of early evaluation (following from Axiom 3), if a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to whichever is greater: $i - ub + 1$ (i.e., the $i - ub$ of the next evaluation) or $T_\xi.\tau + 1$ (as described above) on lines 14, 21, 28 and 36. Note that on lines 14, 21, and 28, this can be rewritten as just $T_\xi.\tau + 1$ since $i - ub + 1 \equiv (T_\xi.\tau + lb) - ub + 1$ where $lb - ub \leq 0$ such that $T_\xi.\tau + (lb - ub) + 1 \leq T_\xi.\tau + 1$. Figures 16, 17, 18, and 19 illustrate examples of how the $\varphi.next\_time$ variable is updated.

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 6–7 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time,T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time,T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi \rightarrow T_\varphi = (\mathsf{true},i)$: As an initial condition, if $0-lb<0$ (line 8), then $\forall i \in [0,lb-1]$, no interval exists from $[i-ub,i-lb]$; therefore, the algorithm automatically writes false for all time indexes less than the lower bound (line 10). We then consider all possible combinations of T_ψ and T_ξ to determine if $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$:

(1) *(T_ψ is empty or $T_\psi.\tau \leq \varphi.previous-lb$) and ($T_\xi$ is empty or $T_\xi.\tau \leq \varphi.previous-lb$)*: There is not enough information to determine if $\exists j \in [i-ub,i-lb]$ such that $\pi,j \models \xi$ and $\forall k > j$ where $k \in [i-ub,i-lb]$, $\pi,k \models \psi$ (where $i = \varphi.previous+1$); therefore, the algorithm does not write a tuple.

(2) $T_\xi.v = \mathsf{true}$ *and* $T_\xi.\tau > \varphi.previous-lb$ *and (T_ψ is empty or* $T_\psi.v = \mathsf{true}$ *or* $T_\psi.v = \mathsf{false}$*)*: $T_\xi.v$ is the verdict from $[\varphi.next\_time,T_\xi.\tau]$ such that $\varphi.next\_time \leq \varphi.previous-lb+1$ since the result would have been previously written on line 17 and the $\varphi.next\_time$ variable would have been updated (line 14). Therefore, $T_\xi.v$ is guaranteed to be true at $i-lb$ where $i \in [\varphi.previous+1,T_\xi.\tau+lb]$. As a result, $\pi,i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ (following directly from Axiom 3) for $\varphi.previous < i \leq T_\xi.\tau+lb$; therefore $(\mathsf{true},T_\xi.\tau+lb)$ is written to φ's SCQ on line 17. Figure 16 provides a visualization of this case.

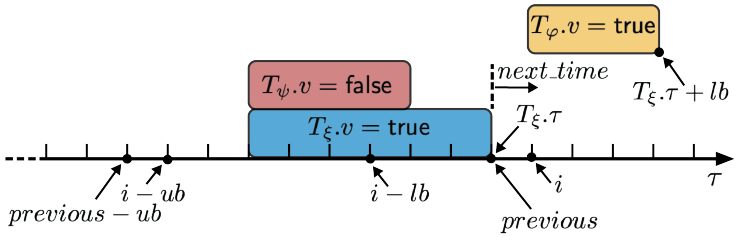


Fig. 16. Pictorial representation of $\varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ for Theorem 9 Case (2)

(3) $T_\xi.v = \mathsf{false}$ *and* $T_\xi.\tau > \varphi.previous-lb$ *and (T_ψ is empty or* $T_\psi.v = \mathsf{true}$ *or* $T_\psi.v = \mathsf{false}$*)*: There are two sub-cases to consider:

(a) *There is enough information to determine that* $\nexists j \in [i-ub,i-lb]$ *such that* $\pi,j \models \xi$: If $\varphi.edge \leq \varphi.previous-ub$ or $\varphi.edge = -1$, then $T_\xi.v$ is guaranteed to have never been true from $[max(\varphi.previous-ub+1,0),T_\xi.\tau]$ since $\varphi.edge$ would have been updated on line 13. Since $T_\xi.\tau > \varphi.previous-lb$, then $\nexists j \in [i-ub,i-lb]$ such that $\pi,j \models \xi$ where $i \in [\varphi.previous+1,T_\xi.\tau+lb]$. As a result, $\pi,i \not\models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ (following directly from Axiom 3) for $\varphi.previous < i \leq T_\xi.\tau+lb$; therefore $(\mathsf{false},T_\xi.\tau+lb)$ is written to φ's SCQ on line 24. Figure 17 provides a visualization of this case.

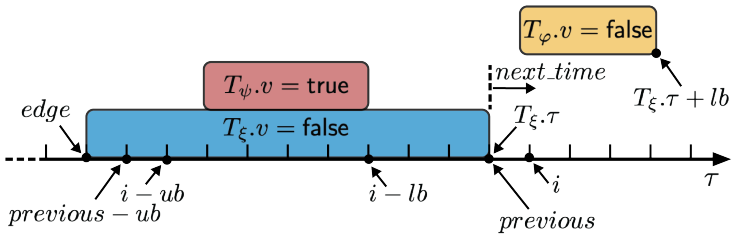


Fig. 17. Pictorial representation of $\varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ for Theorem 9 Case (3)(a)

(b) *There is currently enough information to determine* $\exists j \in [i-ub,i-lb]$ *where* $\pi,j \models \xi$: If $\varphi.edge > \varphi.previous-ub$ and $\varphi.edge \neq -1$, then $\exists j \in [i-ub,i-lb-1]$ such that

the latest value of j is $\varphi.edge = \varphi.next\_time - 1$ as set by the previous execution on lines 13 and 14. (Note that if $\pi, i-lb \models \xi$, then the result would have been written on line 17 and the $\varphi.next\_time$ and $\varphi.edge$ variables would have been updated on lines 13 and 14.) There are three sub-cases:

(b.1) *There is currently enough information to determine* $\forall k > j$ *where* $j \in [\varphi.edge+1, i-lb]$, $\pi, k \models \psi$: If $T_\psi.v = \mathsf{false}$, $T_\psi.v$ is guaranteed to be false from $[\varphi.edge+1, T_\psi.\tau]$ since $\varphi.next\_time = \varphi.edge + 1$. As a result, $\pi, i \not\models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi.\tau + lb$; therefore $(\mathsf{false}, T_\xi.\tau + lb)$ is written to φ's SCQ on line 31. Figure 18 provides a visualization of this case.

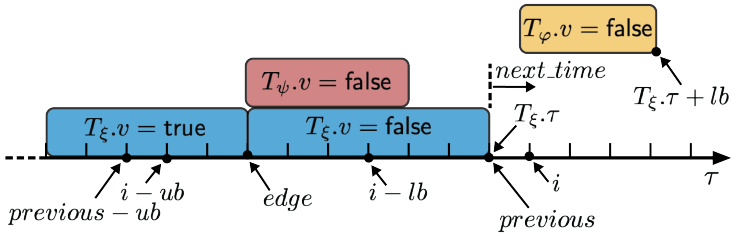


Fig. 18. Pictorial representation of $\varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ for Theorem 9 Case (3)(b.1)

(b.2) *There is currently enough information to determine* $\forall k > j$ *where* $j \in [\varphi.edge+1, i-lb]$, $\pi, k \models \psi$: If $T_\psi.v = \mathsf{true}$ and $T_\psi.\tau > \varphi.previous - lb$, then $\forall k > j$ where $k \in [\varphi.edge+1, i-lb]$, $\pi, k \models \psi$ such that $i \in [\varphi.previous+1, T_\psi.\tau+lb]$. Since it's only known that $\exists j \in [i-ub, i-lb]$ where $\pi, j \models \xi$ such that $i \in [\varphi.previous+1, (\varphi.edge + (ub-lb)) + lb]$, it's only known that $\pi, i \models \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq min(T_\psi.\tau + lb, \varphi.edge + ub)$; therefore, $(\mathsf{true}, min(T_\psi.\tau+lb, \varphi.edge+ub))$ is written to φ's SCQ on line 37. Figure 19 provides a visualization of this case.

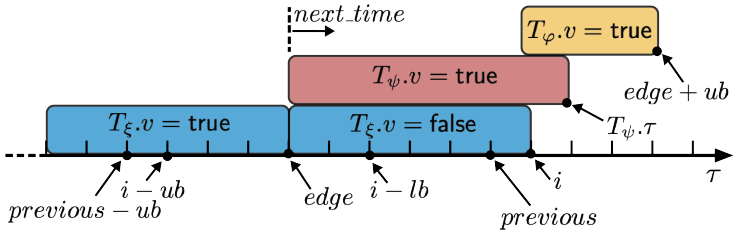


Fig. 19. Pictorial representation of $\varphi = \psi\ \mathcal{S}_{[lb,ub]}\ \xi$ for Theorem 9 Case (3)(b.2)

(b.3) *There is currently not enough information to determine if* $\forall k > j$ *where* $j \in [\varphi.edge+1, i-lb]$, $\pi, k \models \psi$: If $T_\psi.v = \mathsf{true}$ and $T_\psi.\tau < i - lb$, there is not enough information to know if $\forall k > j$ where $j \in [\varphi.edge+1, i-lb]$, $\pi, k \models \psi$; therefore, the algorithm does not write a tuple.

(4) $T_\psi.\tau \geq i-lb$ *and* (T_ξ *is empty or* $T_\xi.\tau < i-lb$): There are two sub-cases to consider:

(a) *There is enough information to determine that* $\exists j \in [i-ub, i-lb]$ *where* $\pi, j \models \xi$ *such that* $\forall k > j$ *where* $j \in [i-lb, i-lb]$, $\pi, k \models \psi$: If $\varphi.edge > \varphi.previous - ub$ and $\varphi.edge \neq -1$, then this follows directly as Case (3) (b.2) above (following directly from Axiom 3).

(b) *There is not enough information to determine if* $\exists j \in [i-ub, i-lb]$ *where* $\pi, j \models \xi$ *such that* $\forall k > j$ *where* $j \in [i-lb, i-lb]$, $\pi, k \models \psi$: If $\varphi.edge \leq \varphi.previous - ub$ or $\varphi.edge = -1$, then there is still a chance that $\exists j \in [i-ub, i-lb]$ where $\pi, j \models \xi$ in the future such that $\forall k > j$ where $j \in [i-lb, i-lb]$, $\pi, k \models \psi$; therefore, the algorithm does not write a tuple.

(if direction) $T_\varphi = (\mathsf{true}, i) \rightarrow \pi, i \models_\psi \mathcal{S}_{[lb,ub]}\ \xi$:
$T_\varphi = (\mathsf{true}, i)$ tuples are only written to φ's SCQ on lines 17 and 37:
(1) *line 17:* $T_\xi.v = \mathsf{true}$ is guaranteed to be the verdict from $[\varphi.next\_time, T_\xi.\tau]$ such that $\varphi.next\_time \leq \varphi.previous - lb + 1$ and $T_\xi.\tau \geq \varphi.previous - lb + 1$; therefore, $T_\xi.v = \mathsf{true}$ at $i - lb$ where $i \in [\varphi.previous + 1, T_\xi.\tau + lb]$. As a result, $\pi, i \models_\psi \mathcal{S}_{[lb,ub]}\ \xi$ (following directly from Axiom 3) for $\varphi.previous < i \leq T_\xi.\tau + lb$.
(2) *line 37:* It is guaranteed that $\exists j \in [\varphi.previous - ub + 1, \varphi.previous - lb]$ such that $\pi, j \models \xi$ where $j = \varphi.edge$ and that $T_\psi.v = \mathsf{true}$ is the verdict from $[\varphi.next\_time, T_\psi.\tau]$ where $\varphi.next\_time = \varphi.edge + 1$ (as set by the previous execution on line 13–14) and $T_\psi.\tau \geq \varphi.previous - lb + 1$; therefore, $\exists j \in [i - ub, i - lb]$ where $i \in [\varphi.previous + 1, (\varphi.edge + (ub - lb)) + lb)]$ and $\forall k > j$ where $k \in [i - ub, i - lb]$, $T_\psi.v = \mathsf{true}$ such that $i \in [\varphi.previous + 1, T_\psi.\tau + lb]$. As a result, $\pi, i \models_\psi \mathcal{S}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq min(T_\xi.\tau + lb, \varphi.edge + ub)$.
$T_\varphi = (\mathsf{false}, i)$ tuples are only written on lines 10, 24, and 31:
(1) *line 10:* When $i - lb < 0$ (i.e., $i < lb$), there will never exist an interval $[i - ub, i - lb]$. As a result, $\pi, i \not\models_\psi \mathcal{S}_{[lb,ub]}\ \xi$ for $i < lb$.
(2) *line 24:* $T_\xi.v$ is guaranteed to have never been true from $[\varphi.previous - lb + 1, \varphi.previous - ub + 1]$ since the $\varphi.edge$ would have been updated on line 13; therefore, $\nexists j \in [i - ub, i - lb]$ such that $\pi, j \models \xi$. As a result, $\pi, i \not\models_\psi \mathcal{S}_{[lb,ub]}\ \xi$ (following directly from Axiom 3) for $\varphi.previous < i \leq T_\xi + lb$.
(3) *line 31:* It is guaranteed that $\exists j \in [\varphi.previous - ub + 1, \varphi.previous - lb]$ such that $\pi, j \models \xi$ where $j = \varphi.edge$ and j is the latest time in $[(\varphi.previous + 1) - ub, \varphi.previous - lb]$ where $\pi, j \models \xi$, but $T_\psi.v = \mathsf{false}$ is the verdict from $[\varphi.next\_time, T_\psi.\tau]$ where $\varphi.next\_time = \varphi.edge + 1$ (as set by the previous execution on line 13–14); therefore, $\exists j \in [i - ub, i - lb]$ such that $\pi, j \models \xi$ but there is never an instance where $\forall k > j$ where $k \in [i - lb, i - ub]$, $\pi, k \models \psi$. As a result, $\pi, i \not\models_\psi \mathcal{S}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi + lb$.
There are three conditions under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi, i \models_\psi \mathcal{S}_{[lb,ub]}\ \xi$:
(1) *(T_ψ is empty or $T_\psi.\tau \leq \varphi.previous - lb$) and ($T_\xi$ is empty or $T_\xi.\tau \leq \varphi.previous - lb$)*: There is not enough information based on $\varphi.next\_time$ to evaluate if $\pi, i \models_\psi \mathcal{S}_{[lb,ub]}\ \xi$; therefore, the algorithm does not write a tuple.
(2) $T_\xi.v = \mathsf{false}$ *and* $T_\xi.\tau > i - lb$ *and* $T_\psi.v = \mathsf{true}$ *and* $T_\psi.\tau < i - lb$ *and* $\varphi.edge \geq i - ub$ *and* $\varphi.edge \neq -1$: If $\varphi.edge \geq i - ub$ and $\varphi.edge \neq -1$, then $\exists j \in [i - ub, i - lb - 1]$ such that the latest value of j is $\varphi.edge = \varphi.next\_time - 1$ as set by the previous execution on lines 13 and 14. $T_\psi.v = \mathsf{true}$ is the verdict from $[\varphi.edge + 1, T_\psi.\tau]$, but there is not enough information to determine if $\pi, k \models \psi$ where $k \in [T_\psi.\tau + 1, i - lb]$; therefore, the algorithm does nothing.
(3) *(T_ξ is Empty or $T_\xi.\tau < i - lb$) and* $T_\psi.\tau \geq i - lb$ *and* ($\varphi.edge < i - ub$ *or* $\varphi.edge = -1$): There is not enough information to guarantee that $\nexists j \in [i - ub, i - lb]$ such that $\pi, i \models \xi$, where $\forall k > j$ such that $k \in [i - ub, i - lb]$, $\pi, k \models \psi$. More specifically, it is guaranteed to be unknown if $\pi, i - lb \models \xi$ (as described in Axiom 3); therefore, the algorithm does not write a tuple.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true}, i)$ are only written to φ's SCQ iff $\pi, i \models_\psi \mathcal{S}_{[lb,ub]}\ \xi$, and $T_\varphi = (\mathsf{false}, i)$ are only written to φ's SCQ iff $\pi, i \not\models_\psi \mathcal{S}_{[lb,ub]}\ \xi$. □

D Trigger ($\mathcal{T}$) Algorithm

Theorem 10 (Correctness of the Trigger Operator). *Given the interval $[lb, ub]$ and two children nodes ψ and ξ, Algorithm 9 writes the tuple T_φ to φ's SCQ when sufficient data is available such that for all $i \geq 0$, $T_\varphi = (\mathsf{true}, i)$ iff $\pi, i \models_\psi \mathcal{T}_{[lb,ub]}\ \xi$.*

Algorithm 9: Trigger Operator: $\varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi$

```
1  Initialize:
2      φ.edge = −1                          // Initialize φ.edge; stores the last i where π,i ⊭ ξ
3      φ.previous = −1                      // Initialize φ.previous; stores the last i written
4      φ.next_time = 0                      // Initialize φ.next_time; stores the next time for ψ and ξ
5  procedure Trigger(ψ,ξ)
       Input: Node: ψ; Node: ξ
6      T_ψ = ψ.read(φ.read_1_ptr,φ.next_time)                          // Read Node ψ
7      T_ξ = ξ.read(φ.read_2_ptr,φ.next_time)                          // Read Node ξ
8      if φ.previous + 1 − lb < 0 then                                 // i − lb < 0
9          φ.previous = lb − 1
10         φ.write(true, lb − 1)                                       // Writing T_φ = (true, lb − 1)
11     if T_ξ ≠ Empty then                                             // New T_ξ
12         if !(T_ξ.v) then                                            // T_ξ.v = false
13             φ.edge = T_ξ.τ                                          // Updating φ.edge to last false edge
14             φ.next_time = T_ξ.τ + 1                                 // Updating φ.next_time to after φ.edge
15             if T_ξ.τ > φ.previous − lb then                         // π,i − lb ⊭ ξ
16                 φ.previous = T_ξ.τ + lb
17                 φ.write(false, T_ξ.τ + lb)                          // Writing T_φ = (false, T_ξ.τ + lb)
18                 return
19         else                                                        // T_ξ.v = true
20             if φ.edge ≤ φ.previous − ub or φ.edge = −1 then          // ∀j ∈ [i − ub, T_ξ.τ], π,j ⊨ ξ
21                 φ.next_time = T_ξ.τ + 1                             // Move φ.next_time forward
22                 if T_ξ.τ > φ.previous − lb then                     // ∀j ∈ [i − ub, i − lb], π,j ⊨ ξ
23                     φ.previous = T_ξ.τ + lb
24                     φ.write(true, T_ξ.τ + lb)                       // Writing T_φ = (true, T_ξ.τ + lb)
25                     return
26             if T_ψ ≠ Empty then                                     // New T_ψ
27                 if T_ψ.v then                                       // T_ψ.v = true and T_ξ.v = true
28                     φ.next_time = T_ξ.τ + 1                         // Move φ.next_time forward
29                     if T_ξ.τ > φ.previous − lb then
30                         φ.previous = T_ξ.τ + lb
31                         φ.write(true, T_ξ.τ + lb)                   // Writing T_φ = (true, T_ξ.τ + lb)
32                         return
33     if T_ψ ≠ Empty then                                             // New T_ψ
           /* ∃φ.edge ∈ [i − ub, i − lb], such that ∄k > φ.edge where k ≥ i − lb, π,k ⊨ ψ   */
34         if !(T_ψ.v) and T_ψ.τ > φ.previous − lb and φ.edge > φ.previous − ub and φ.edge ≠ −1 then
35             φ.previous = min(T_ψ.τ + lb, φ.edge + ub)               // Limit i based on φ.edge
36             φ.next_time = max(φ.next_time, φ.previous − ub + 1)
37             φ.write(false, φ.previous)                              // Writing T_φ = (false, φ.previous)
```

Proof. The $\varphi.edge$ variable stores the latest timestamp where $T_\xi.v = \text{false}$ (line 13). The $\varphi.edge$ variable is initialized to -1 (line 2) such that before $T_\xi.v = \text{false}$, the conditions on lines 20 and 34 can be rewritten as $(-1 \leq \varphi.previous - ub \text{ or } \varphi.edge = -1) \equiv \text{true}$ and $(-1 > \varphi.previous - ub \text{ and } \varphi.edge \neq -1) \equiv \text{false}$, respectively.

The $\varphi.previous$ variable stores the previous time index i that node $\varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ wrote to its SCQ. Before the first tuple is produced, $\varphi.previous$ is initialized to -1 (line 3) such that the condition on line 8 can be rewritten as $-1+1-lb<0 \equiv 0-lb<0$, the conditions on lines 15, 22, and 29 can be rewritten as $T_\xi.\tau > -1-lb$ or rather $T_\xi.\tau \geq 0-lb$, the condition on line 20 can be rewritten as $(\varphi.edge \leq -1-ub \text{ or } \varphi.edge = -1) \equiv (\varphi.edge = -1)$, and the condition on line 34 can be rewritten as $(T_\psi.v \text{ and } T_\psi.\tau > -1-lb \text{ and } \varphi.edge \geq 0-ub \text{ and } \varphi.edge \neq -1) \equiv (T_\psi.v \text{ and } T_\psi.\tau \geq 0-lb \text{ and } \varphi.edge \neq -1)$. Whenever a verdict-timestamp tuple is written, the $\varphi.previous$ variable is updated to that timestamp (lines 9, 16, 23, 30, and 35).

Note that if $i-ub<0$, then $\forall i \in [0, ub-1]$, we can only consider the interval $[0, i-lb]$ for evaluation (lines 20 and 34). Line 20 checks whether $T_\xi.v$ never equals false between $[i-ub, i-lb]$; if $i-ub<0$, then $\varphi.edge \leq \varphi.previous - ub \equiv (\varphi.edge \geq -1) \leq (\varphi.previous - ub < -1) \equiv \text{false}$ since $i = \varphi.previous + 1$; therefore, line 20 is contingent on whether $\varphi.edge = -1$ (i.e., never a time yet where $T_\xi.v = \text{false}$). Line 34 performs a similar check to see if $T_\xi.v = \text{false}$ between $[i-ub, i-lb]$; if $i-ub<0$, then $\varphi.edge > \varphi.previous - ub \equiv (edge \geq -1) >$

$(\varphi.previous - ub < -1) \equiv \mathsf{true}$ since $i = \varphi.previous + 1$; therefore, line 34 is contingent on whether $\varphi.edge \neq -1$ (i.e., there was a time after or at timestamp 0 where $T_\xi.v = \mathsf{false}$).

The $\varphi.next\_time$ variable determines what timestamp is desired from both node ψ and node ξ to make the next evaluation. The $\varphi.next\_time$ variable is initialized to 0 on line 4 and is then updated during execution based on what is known about ξ. When $T_\xi.v = \mathsf{false}$ (and $T_\xi.\tau < i - lb$), its unknown if $\exists k > T_\xi.\tau$ where $j \in [i-ub, i-lb]$, $\pi, k \models \psi$, and if $T_\xi.v = \mathsf{true}$ from $[i-ub, T_\xi.\tau]$ (and $T_\xi.\tau < i - lb$), its known if $\exists j \in [T_\xi.\tau + 1, i - lb]$ such that $\pi, j \not\models \xi$, and if both $T_\psi.v = \mathsf{true}$ and $T_\xi.v = \mathsf{true}$ (and $T_\xi.\tau < i - lb$), it is known if $\exists j \in [T_\xi.\tau + 1, i - lb]$ where $\pi, j \not\models \xi$ such that $\exists k > j$ where $j \in [T_\xi.\tau + 1, i - lb]$, $\pi, k \models \psi$. Therefore, $\varphi.next\_time$ will be updated to $T_\xi.\tau + 1$ on lines 14, 21, and 28. Because of early evaluation (following from Axiom 4), if a verdict-timestamp tuple is written, then $\varphi.next\_time$ is updated to whichever is greater: $i - ub + 1$ (i.e., the $i - ub$ of the next evaluation) or $T_\xi.\tau + 1$ (as described above) on lines 14, 21, 28 and 36. Note that on lines 14, 21, and 28, this can be rewritten as just $T_\xi.\tau + 1$ since $i - ub + 1 \equiv (T_\xi.\tau + lb) - ub + 1$ where $lb - ub \leq 0$ such that $T_\xi.\tau + (lb - ub) + 1 \leq T_\xi.\tau + 1$. Figures 20, 21, 22, and 23 illustrate examples of how the $\varphi.next\_time$ variable is updated.

The $\varphi.next\_time$ variable is then an input into the $read$ functions on lines 6–7 (defined in Algorithm 1) such that if the timestamp $\varphi.next\_time$ is available in node ψ, then $T_\psi.v$ is the verdict for the interval $[\varphi.next\_time, T_\psi.\tau]$, and if $\varphi.next\_time$ is available in node ξ, then $T_\xi.v$ is the verdict for the interval $[\varphi.next\_time, T_\xi.\tau]$ (following from Theorem 2).

(only-if direction) $\pi, i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi \rightarrow T_\varphi = (\mathsf{true}, i)$: As an initial condition, if $0 - lb < 0$ (line 8), then $\forall i \in [0, lb - 1]$, no interval exists from $[i - ub, i - lb]$; therefore, the algorithm automatically writes true for all time indexes less than the lower bound (line 10). We then consider all possible combinations of T_ψ and T_ξ to determine if $\pi, i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$:

(1) *(T_ψ is empty or $T_\psi.\tau \leq \varphi.previous - lb$) and ($T_\xi$ is empty or $T_\xi.\tau \leq \varphi.previous - lb$)*: There is not enough information to determine if $\exists j \in [i-ub, i-lb]$ such that $\pi, j \not\models \xi$ and $\exists k > j$ where $k \in [i - ub, i - lb]$, $\pi, k \models \psi$ (where $i = \varphi.previous + 1$); therefore, the algorithm does not write a tuple.

(2) $T_\xi.v = \mathsf{false}$ *and* $T_\xi.\tau > \varphi.previous - lb$ *and (T_ψ is empty or* $T_\psi.v = \mathsf{true}$ *or* $T_\psi.v = \mathsf{false}$*)*: $T_\xi.v$ is the verdict from $[\varphi.next\_time, T_\xi.\tau]$ such that $\varphi.next\_time \leq \varphi.previous - lb + 1$ since the result would have been previously written on line 17 and the $\varphi.next\_time$ variable would have been updated (line 14). Therefore, $T_\xi.v$ is guaranteed to be false at $i - lb$ where $i \in [\varphi.previous + 1, T_\xi.\tau + lb]$. As a result, $\pi, i \not\models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ (following directly from Axiom 4) for $\varphi.previous < i \leq T_\xi.\tau + lb$; therefore $(\mathsf{false}, T_\xi.\tau + lb)$ is written to φ's SCQ on line 17. Figure 20 provides a visualization of this case.

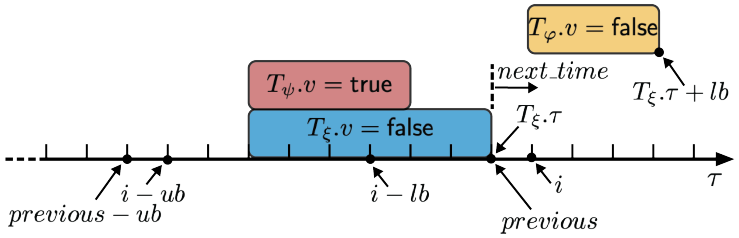


Fig. 20. Pictorial representation of $\varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for Theorem 10 Case (2)

(3) $T_\xi.v = \mathsf{true}$ *and* $T_\xi.\tau > \varphi.previous - lb$ *and (T_ψ is empty or* $T_\psi.v = \mathsf{true}$ *or* $T_\psi.v = \mathsf{false}$*)*: There are two sub-cases to consider:

(a) *There is enough information to determine that* $\forall j \in [i-ub, i-lb]$ *such that* $\pi, j \models \xi$: If $\varphi.edge \leq \varphi.previous - ub$ or $\varphi.edge = -1$, then $T_\xi.v$ is guaranteed to have never

been false from $[max(\varphi.previous-ub+1,0),T_\xi.\tau]$ since $\varphi.edge$ would have been updated on line 13. Since $T_\xi.\tau > \varphi.previous-lb$, then $\forall j \in [i-ub,i-lb]$ such that $\pi,j \models \xi$ where $i \in [\varphi.previous+1,T_\xi.\tau+lb]$. As a result, $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ (following directly from Axiom 4) for $\varphi.previous < i \leq T_\xi.\tau+lb$; therefore $(\mathsf{true},T_\xi.\tau+lb)$ is written to φ's SCQ on line 24. Figure 21 provides a visualization of this case.

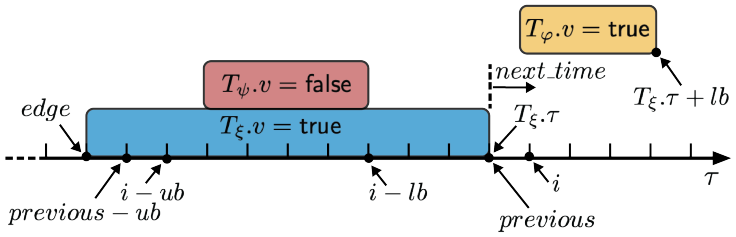


Fig. 21. Pictorial representation of $\varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for Theorem 10 Case (3)(a)

(b) *There is currently enough information to determine* $\exists j \in [i-ub,i-lb]$ *where* $\pi,j \not\models \xi$: If $\varphi.edge > \varphi.previous-ub$ and $\varphi.edge \neq -1$, then $\exists j \in [i-ub,i-lb-1]$ such that the latest value of j is $\varphi.edge = \varphi.next\_time-1$ as set by the previous execution on lines 13 and 14. (Note that if $\pi,i-lb \not\models \xi$, then the result would have been written on line 17 and the $\varphi.next\_time$ and $\varphi.edge$ variables would have been updated on lines 13 and 14.) There are three sub-cases:

(b.1) *There is currently enough information to determine* $\exists k > j$ *where* $j \in [\varphi.edge+1,i-lb]$, $\pi,k \models \psi$: If $T_\psi.v = \mathsf{true}$, $T_\psi.v$ is guaranteed to be true from $[\varphi.edge+1,T_\psi.\tau]$ since $\varphi.next\_time = \varphi.edge+1$. As a result, $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi.\tau+lb$; therefore $(\mathsf{true},T_\xi.\tau+lb)$ is written to φ's SCQ on line 31. Figure 22 provides a visualization of this case.

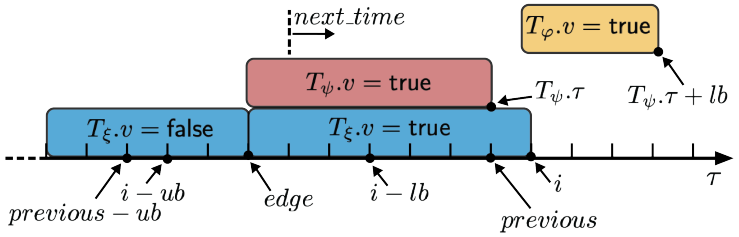


Fig. 22. Pictorial representation of $\varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for Theorem 10 Case (3)(b.1)

(b.2) *There is currently enough information to determine* $\nexists k > j$ *where* $j \in [\varphi.edge+1,i-lb]$, $\pi,k \models \psi$: If $T_\psi.v = \mathsf{false}$ and $T_\psi.\tau > \varphi.previous-lb$, then $\nexists k > j$ where $k \in [\varphi.edge+1,i-lb]$, $\pi,k \models \psi$ such that $i \in [\varphi.previous+1,T_\psi.\tau+lb]$. Since it's only known that $\exists j \in [i-ub,i-lb]$ where $\pi,j \not\models \xi$ such that $i \in [\varphi.previous+1,(\varphi.edge+(ub-lb))+lb]$, it's only known that $\pi,i \not\models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq min(T_\psi.\tau+lb,\varphi.edge+ub)$; therefore, $(\mathsf{false},min(T_\psi.\tau+lb,\varphi.edge+ub)$ is written to φ's SCQ on line 37. Figure 23 provides a visualization of this case.

(b.3) *There is currently not enough information to determine if* $\exists k > j$ *where* $j \in [\varphi.edge+1,i-lb]$, $\pi,k \models \psi$: If $T_\psi.v = \mathsf{false}$ and $T_\psi.\tau < i-lb$, there is not enough information to know if $\exists k > j$ where $j \in [\varphi.edge+1,i-lb]$, $\pi,k \models \psi$; therefore, the algorithm does not write a tuple.

(4) $T_\psi.\tau \geq i-lb$ *and* (T_ξ *is empty or* $T_\xi.\tau < i-lb$): There are two sub-cases to consider:

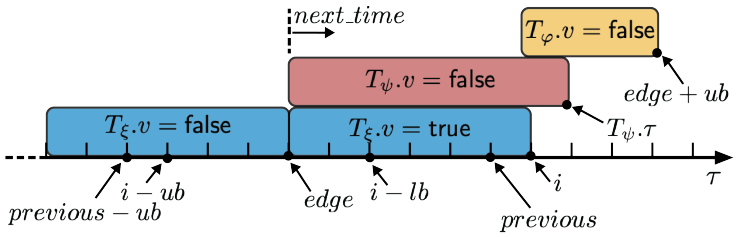


Fig. 23. Pictorial representation of $\varphi = \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for Theorem 10 Case (3)(b.2)

(a) *There is enough information to determine that* $\exists j \in [i-ub,i-lb]$ *where* $\pi,j \models \xi$ *such that* $\exists k > j$ *where* $j \in [i-lb,i-lb]$, $\pi,k \models \psi$: If $\varphi.edge > \varphi.previous - ub$ and $\varphi.edge \neq -1$, then this follows directly as Case (3) (b.2) above (following directly from Axiom 4).

(b) *There is not enough information to determine if* $\exists j \in [i-ub,i-lb]$ *where* $\pi,j \models \xi$ *such that* $\exists k > j$ *where* $j \in [i-lb,i-lb]$, $\pi,k \models \psi$: If $\varphi.edge \leq \varphi.previous - ub$ or $\varphi.edge = -1$, then there is still a chance that $\exists j \in [i-ub,i-lb]$ where $\pi,j \models \xi$ in the future such that $\exists k > j$ where $j \in [i-lb,i-lb]$, $\pi,k \not\models \psi$; therefore, the algorithm does not write a tuple.

(if direction) $T_\varphi = (\mathsf{true},i) \rightarrow \pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$:

$T_\varphi = (\mathsf{true},i)$ tuples are only written on lines 10, 24, and 31:

(1) *line 10:* When $i-lb < 0$ (i.e., $i < lb$), there will never exist an interval $[i-ub,i-lb]$. As a result, $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for $i < lb$.

(2) *line 24:* $T_\xi.v$ is guaranteed to have never been false from $[\varphi.previous-lb+1,\varphi.previous-ub+1]$ since the $\varphi.edge$ would have been updated on line 13; therefore, $\forall j \in [i-ub,i-lb]$ such that $\pi,j \models \xi$. As a result, $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ (following directly from Axiom 4) for $\varphi.previous < i \leq T_\xi + lb$.

(3) *line 31:* It is guaranteed that $\exists j \in [\varphi.previous-ub+1,\varphi.previous-lb]$ such that $\pi,j \not\models \xi$ where $j = \varphi.edge$ and j is the latest time in $[(\varphi.previous+1)-ub,\varphi.previous-lb]$ where $\pi,j \models \xi$, but $T_\psi.v = \mathsf{true}$ is the verdict from $[\varphi.next\_time,T_\psi.\tau]$ where $\varphi.next\_time = \varphi.edge+1$ (as set by the previous execution on line 13–14); therefore, if $\exists j \in [i-ub,i-lb]$ such that $\pi,j \models \xi$, then $\exists k > j$ where $k \in [i-lb,i-ub]$, $\pi,k \models \psi$. As a result, $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq T_\xi + lb$.

$T_\varphi = (\mathsf{false},i)$ tuples are only written to φ's SCQ on lines 17 and 37:

(1) *line 17:* $T_\xi.v = \mathsf{false}$ is guaranteed to be the verdict from $[\varphi.next\_time,T_\xi.\tau]$ such that $\varphi.next\_time \leq \varphi.previous - lb + 1$ and $T_\xi.\tau \geq \varphi.previous - lb + 1$; therefore, $T_\xi.v = \mathsf{false}$ at $i-lb$ where $i \in [\varphi.previous+1,T_\xi.\tau+lb]$. As a result, $\pi,i \not\models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ (following directly from Axiom 4) for $\varphi.previous < i \leq T_\xi.\tau + lb$.

(2) *line 37:* It is guaranteed that $\exists j \in [\varphi.previous - ub + 1,\varphi.previous - lb]$ such that $\pi,j \models \xi$ where $j = \varphi.edge$ but $T_\psi.v = \mathsf{false}$ is the verdict from $[\varphi.next\_time,T_\psi.\tau]$ where $\varphi.next\_time = \varphi.edge+1$ (as set by the previous execution on line 13–14) and $T_\psi.\tau \geq \varphi.previous-lb+1$; therefore, $\exists j \in [i-ub,i-lb]$ where $i \in [\varphi.previous+1,(\varphi.edge+(ub-lb))+lb)]$ but there is never an instance where $\exists k > j$ where $k \in [i-ub,i-lb]$, $T_\psi.v = \mathsf{true}$ such that $i \in [\varphi.previous+1,T_\psi.\tau+lb]$. As a result, $\pi,i \not\models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$ for $\varphi.previous < i \leq min(T_\xi.\tau+lb,\varphi.edge+ub)$.

There are three conditions under which no verdict-timestamp tuples were written to φ's SCQ since there is not enough information to determine if $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$:

(1) *(T_ψ is empty or $T_\psi.\tau \leq \varphi.previous-lb$) and ($T_\xi$ is empty or $T_\xi.\tau \leq \varphi.previous-lb$):* There is not enough information based on $\varphi.next\_time$ to evaluate if $\pi,i \models \psi\ \mathcal{T}_{[lb,ub]}\ \xi$; therefore, the algorithm does not write a tuple.

(2) $T_\xi.v = \mathsf{true}$ *and* $T_\xi.\tau > i-lb$ *and* $T_\psi.v = \mathsf{false}$ *and* $T_\psi.\tau < i-lb$ *and* $\varphi.edge \geq i-ub$ *and* $\varphi.edge \neq -1$: If $\varphi.edge \geq i-ub$ and $\varphi.edge \neq -1$, then $\exists j \in [i-ub, i-lb-1]$ such that the latest value of j is $\varphi.edge = \varphi.next\_time - 1$ as set by the previous execution on lines 13 and 14. $T_\psi.v = \mathsf{false}$ is the verdict from $[\varphi.edge+1, T_\psi.\tau]$, but there is not enough information to determine if $\pi, k \models \psi$ where $k \in [T_\psi.\tau+1, i-lb]$; therefore, the algorithm does nothing.

(3) *(*T_ξ *is Empty or* $T_\xi.\tau < i-lb$*) and* $T_\psi.\tau \geq i-lb$ *and* $(\varphi.edge < i-ub$ *or* $\varphi.edge = -1)$: There is not enough information to guarantee that if $\exists j \in [i-ub, i-lb]$ such that $\pi, i \not\models \xi$, then $\exists k > j$ such that $k \in [i-ub, i-lb]$, $\pi, k \models \psi$. More specifically, it is guaranteed to be unknown if $\pi, i-lb \not\models \xi$ (as described in Axiom 4); therefore, the algorithm does not write a tuple.

Verdict-timestamp tuples $T_\varphi = (\mathsf{true}, i)$ are only written to φ's SCQ iff $\pi, i \models \psi \; \mathcal{T}_{[lb,ub]} \; \xi$, and $T_\varphi = (\mathsf{false}, i)$ are only written to φ's SCQ iff $\pi, i \not\models \psi \; \mathcal{T}_{[lb,ub]} \; \xi$. □