Parameterized Verification of Asynchronous Round-Based Distributed Algorithms via Reduction to Finite-Counter Systems

Nathalie Bertrand
Univ Rennes, Inria, CNRS, IRISA, France

Pranav Ghorpade
The University of Sydney, Australia

Sasha Rubin
The University of Sydney, Australia

Abstract

Traditional model-checking techniques typically verify distributed algorithms only for a fixed number of finite-state processes. Parameterized model checking generalizes this to any number of processes, while still typically assuming that each process is finite-state. In this work, we consider asynchronous round-based distributed algorithms in which each process is infinite-state since it can execute for an infinite number of rounds. We show that the parameterized verification problem for asynchronous round-based distributed algorithms is undecidable, already for simple specifications. Nevertheless, as our main contribution, we provide a reduction to LTL model checking over finite-counter systems and prove that it is sound and complete. This enables the use of off-the-shelf, mature symbolic model checkers for finite-counter systems. We demonstrate the practical applicability of this reduction by verifying safety and liveness properties of several asynchronous round-based consensus and leader-election algorithms using the nuXmv model checker.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Modal and temporal logics; Theory of computation → Distributed algorithms

Keywords and phrases parametrized verification, asynchronous round-based distributed algorithms, finite-counter systems, LTL model checking

1 Introduction

Asynchronous distributed algorithms consist of a parameterized number of identical processes that asynchronously exchange messages with other processes and update their local states. A subclass of these is that of asynchronous round-based distributed algorithms (ARDAs), in which each process progresses through an unbounded sequence of rounds and each message is tagged with the sender's current round number. Typical examples include Ben-Or's randomized consensus algorithms [4], Bracha's consensus algorithm [9], and Raft's leader-election algorithm [35]. In the absence of synchronization, processes in such algorithms may drift across rounds, so each process must store its current round index as part of its local state. Since this index is unbounded, each process has infinitely many possible states. Consequently, the global system exhibits two orthogonal sources of unboundedness: the parameterized number of processes and the unbounded round index of each process.

Parameterized verification of distributed algorithms refers to checking their correctness for all numbers n of processes, provided that a resilience condition relating n and the maximum number t of faulty processes is satisfied (e.g., $3t < n$). Parameterized model checking provides techniques that cope with all admissible valuations of the parameters n and t, namely those satisfying the resilience condition. However, it typically assumes that processes are finite-state (see, for instance, the book [8] or the recent survey [25]) and is therefore not directly applicable to asynchronous round-based distributed algorithms. Moreover, even for fixed parameter values, the resulting system is infinite-state because of the round indices and cannot be directly handled by finite-state model checkers such as SPIN, NuSMV, or TLA+.

In this work, we introduce a modeling formalism for a broad class of ARDAs, together with a specification logic designed to express their standard correctness properties. For this formalism, we show that the parameterized verification problem is undecidable, whereas the fixed-instance verification problem is decidable. To address the parameterized case despite this undecidability, we give a sound and complete reduction to LTL model checking over finite-counter systems. Although this target problem is itself undecidable, it is well studied, and sound, necessarily incomplete, symbolic model-checking techniques for it have been developed and implemented, for example in nuXmv [10]. Thus, our reduction enables the use of existing symbolic model checkers for parameterized verification of ARDAs, while also potentially benefiting from future advances in such tools. To assess the practical applicability of the reduction, we apply it to four case studies: (i) Ben-Or's consensus with crash faults [4, Algo A], (ii) Ben-Or's consensus with Byzantine faults [4, Algo B], (iii) Bracha's consensus with Byzantine faults [9, Fig. 4], and (iv) Raft's leader-election [35, Sec. 5.2]. Using nuXmv as the back end, we verify safety and liveness properties, namely agreement, validity, termination (under some condition), and leader uniqueness. These case studies suggest that existing verification engines can efficiently handle the resulting finite-counter systems.

Brief Overview. § 2.1 introduces templates for ARDAs. A template specifies the symbolic parameters of the algorithm together with the behavior of a single process. For each admissible parameter valuation, it induces a round-based distributed transition system (RDTS) capturing the behavior of the resulting interacting processes. The parameterized semantics (parameterized RDTS) is obtained as the disjoint union of these fixed-instance RDTS over all admissible parameter valuations. § 2.2 identifies a class of *action-based* temporal properties, called *history state-count properties*, and introduces a tailored action-based logic, *history state-count logic* (HSCL). This logic is deliberately restricted so that it can be translated into state-based LTL, while remaining expressive enough to capture standard correctness conditions for consensus and leader election. § 2.3 gives a reduction from the non-halting problem for two-counter machines to the verification problem of parameterized RDTS against HSCL formulas, thereby establishing undecidability. § 3.1 gives four sound and complete transformations that reduce the parameterized RDTS to a finite-counter system while preserving history state-count properties. § 3.2 then gives two further sound and complete transformations that reduce verification of parameterized RDTS against HSCL formulas to LTL model checking over finite-counter systems. When specialized to fixed-instance semantics RDTS, our reduction yields a *one*-counter system for history state-count properties and reduces verification of RDTS against HSCL formulas to LTL model checking over *finite-state* systems. This proves that the fixed-instance verification problem is decidable. § 4 presents the case studies, and § 5 concludes the paper.

Related Work. Round-based distributed algorithms belong to the broader class of fault-tolerant distributed algorithms, which also includes algorithms in which processes are finite-state, such as broadcast algorithms. Threshold automata [27, 26] provide a successful framework for parameterized verification of such finite-state algorithms by abstracting them to finite-counter systems that can be analyzed by symbolic model checkers. This line of work was later extended to synchronous round-based distributed algorithms through synchronous threshold automata [40]. For asynchronous round-based distributed algorithms (ARDAs), a compositional approach was proposed in [6]. It reduces the verification to proof obligations for individual rounds, which are handled within the threshold-automata framework. A limitation is that the target property is not established directly: one must first identify suitable round

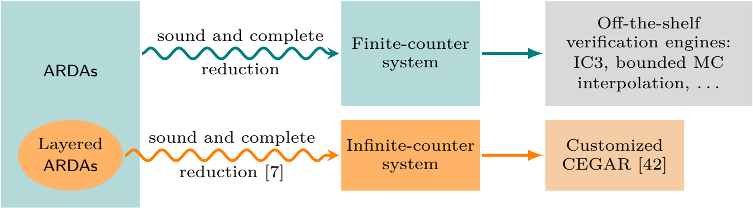


Figure 1 Our reduction improves over the verification framework for layered ARDAs by [7, 42] in both scope and the ability to reuse mature verification tools.

invariants and then argue separately that these invariants imply the desired property.

Layered threshold automata [7], together with the PyLTA prototype [42], were later proposed as a more direct framework for verifying ARDAs. However, this framework is restricted to algorithms that admit a *layered* structure, thereby excluding, for instance, most leader-election algorithms. More importantly, the corresponding verification procedure does not retain the main tooling advantage of standard threshold automata. It first gives a sound and complete abstraction to counter systems with *infinitely* many counters, and then applies a further abstraction to finite-state systems that is *incomplete*. As a result, the approach relies on a bespoke CEGAR loop rather than on off-the-shelf symbolic model checking. In contrast, our work targets a broader class of ARDAs, including leader-election algorithms, and gives a sound and complete reduction to finite-counter systems. This enables the use of mature general-purpose symbolic model checkers, as illustrated in Fig. 1.

Traditionally, parameterized verification of ARDAs has largely relied on human-guided methods. Interactive theorem provers such as Rocq, Isabelle/HOL, and TLA+ have been used to formalize distributed algorithms and establish their correctness through mechanized, expert-driven proofs [43, 11, 5]. Deductive verification frameworks such as Ivy [36] encode distributed algorithms in first-order logic and can prove correctness automatically once suitable inductive invariants are provided. However, discovering invariants remains challenging, motivating a large body of work on automatic inductive-invariant inference [21, 45, 24, 37].

For fixed-instance verification, prior work [16, 34] applies finite-state model-checking tools to prove correctness for a fixed number of processes executing a bounded number of rounds. In contrast, our setting imposes no bound on the number of rounds. The work of [13] also addresses fixed-instance verification of round-based consensus algorithms in the Heard-Of model [14], using TLC to verify fixed instances of algorithms such as OneThirdRule and UniformVoting. However, their reduction is tailored to TLA+/TLC, whose specifications are written in the Temporal Logic of Actions, whereas our reduction targets standard LTL model checking. Moreover, our framework handles a broader class of algorithms, including Byzantine fault-tolerant consensus algorithms and leader-election algorithms.

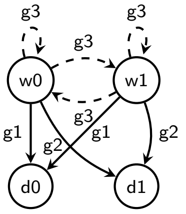
2 Modeling Asynchronous Round-Based Distributed Algorithms

This section introduces the process templates that serve as our formal model, together with the specification language and verification problems studied in the paper.

2.1 Round-based Process Templates

The syntax of our model is motivated by the typical structure of ARDAs; see Fig. 2 (left) for an example. These algorithms are parameterized; processes communicate via broadcast, with choices governed by threshold guards; and the overall behavior of each process is composed

```
Parameters: $n, t$ , with $n > 2t$
Input: $v_0 \in \{0, 1\}$
Initialize: $v := v_0$, $d :=$ Nil, $r := 0$
At round $r$:
  1. Broadcast $(v, r)$
  2. Wait until $\geq (n - t)$ messages
     received with round tag $r$
  3. Compute if some value $v'$ appears
     in $> n/2$ messages: $d := v'$; halt,
     else $v := 0$ or $v := 1$
  4. Progress $r := r + 1$
```



$\mathsf{P} = \{n, t\}$ $\mathsf{rc} = n > 2t$
$\mathcal{L} = \{\mathsf{w0}, \mathsf{w1}, \mathsf{d0}, \mathsf{d1}\}$ $\mathcal{M} = \{\mathsf{m0}, \mathsf{m1}\}$

$\mathsf{g1} = \mathtt{Quorum} \wedge \mathsf{m0} > n/2$
$\mathsf{g2} = \mathtt{Quorum} \wedge \mathsf{m1} > n/2$
$\mathsf{g3} = \mathtt{Quorum} \wedge \mathsf{m0} \leq n/2 \wedge \mathsf{m1} \leq n/2$
where $\mathtt{Quorum} = \mathsf{m0} + \mathsf{m1} \geq n - t$

$\longrightarrow$: type $= 0$
$\dashrightarrow$: type $= 1$

Bcast: $\mathsf{w0} \mapsto \mathsf{m0}, \mathsf{w1} \mapsto \mathsf{m1}$,
$\mathsf{d0} \mapsto \bot, \mathsf{d1} \mapsto \bot$

Figure 2 **Left**: Pseudo-code of a simplified round-based voting algorithm. **Right**: Corresponding process template representation. In the guards, the variable mi denotes the number of messages of type mi received by the process.

of identical, bounded, communication-closed rounds. We now expand on these three aspects. (i) ARDAs are parameterized by integer variables (typically n for number of processes and t for an upper-bound on the number of faulty processes) and are designed to work when these variables satisfy a *resilience condition*, typically expressed as a linear arithmetic formula (e.g., $t < n/2$). (ii) In ARDAs, communication is typically by *broadcast*. Moreover, since processes may be faulty, an ARDA cannot rely on receiving messages from specific processes. Accordingly, process updates are governed by *threshold guards*, which depend only on the number of received messages of each type, rather than on the identities of the senders; see step 3 in Fig. 2 (left). (iii) In ARDAs, the behavior of a process in each round follows the same high-level structure: a finite sequence of broadcast, receive, and compute steps, potentially followed by a progress step to a future round. As a result, rounds are *structurally identical*, and the control-flow within each round is *bounded*. Messages from different rounds are distinguished using *round tags*, which enforce *communication closure* within rounds: messages tagged with round r are used exclusively during computations in round r.

Exploiting the identical structure of rounds in ARDAs, our round-based process template provides a round-independent representation capturing the behavior of a *single process* within a *single round*, together with the rules governing round progress. Similar representations for ARDAs have appeared in prior work, though in more restrictive forms [7] or in a less direct form requiring additional modeling components (e.g., auxiliary locations) [6].

Before proceeding to the formal definition of templates, we illustrate it on the algorithm in Fig. 2 (left) for which Fig. 2 (right) gives a process template. The set of parameters is $\mathsf{P} = \{n, t\}$, with admissible valuations satisfying the resilience condition $\mathsf{rc} = t < n/2$. The locations $\mathcal{L} = \{\mathsf{w0}, \mathsf{w1}, \mathsf{d0}, \mathsf{d1}\}$ represent a process's position within a round: either in a *wait location* (w0 or w1, depending on its current vote) or in a *decide location* (d0 or d1, depending on its decision). These locations correspond to the *wait* and *compute* steps in the pseudo-code. The *broadcast* and *progress* steps are treated implicitly: after updating its vote, a process immediately progresses to the next round, broadcasts its new vote, and re-enters a wait location. The message types $\mathcal{M} = \{\mathsf{m0}, \mathsf{m1}\}$ represent the possible votes a process may broadcast. The transition rules, represented by edges, are labeled with a threshold guard on received messages. For example, guard g1 corresponds to the conjunction "at least $n - t$ messages are received" and "more than $n/2$ are of type m0". Transitions are also equipped with a type that specifies how the round id evolves: type 0 indicates that the process remains in its current round, while in general type k indicates that it progresses by k rounds. Our example only has type 0 and type 1 transitions represented by solid and dashed edges respectively. More generally, round-tag increases of up to $b \in \mathbb{N}$ are allowed, where

b is referred to as the jump bound. A broadcast function Bcast maps each location to the message sent while entering it: here, for $i = 1, 2$, wi maps to mi, and di maps to $\bot$ (i.e., no broadcast). As exhibited by this example, we assume that the subgraph formed by the solid edges (i.e., those with $\mathsf{type} = 0$) is acyclic.

▶ **Definition 1** (Round-based process templates). *Let* $b \in \mathbb{N}$ *be a* jump bound. *A* round-based process template*, or simply a* template*, is a tuple* $\langle \mathsf{P}, \mathsf{rc}, \mathcal{L}, \mathcal{I}, \mathcal{M}, \mathsf{Bcast}, \mathsf{Rules} \rangle$ *where:*

- P *is a finite set of* symbolic parameters*, including* n*, the number of processes;*
- $\mathsf{rc} \in \mathsf{LinArith}(\mathsf{P})$ *is a* resilience condition*;*
- $\mathcal{L}$ *is a finite set of* locations*, and* $\mathcal{I} \subseteq \mathcal{L}$ *is the set of* initial locations*;*
- $\mathcal{M}$ *is a finite set of* message types*;*
- $\mathsf{Bcast} : \mathcal{L} \to \mathcal{M} \cup \{\bot\}$ *is the* broadcast function*;*
- Rules *is a finite set of* transition rules*, where transition rule* ρ *is a record of the form* $\rho = \{\mathsf{from} \in \mathcal{L},\ \mathsf{to} \in \mathcal{L},\ \mathsf{type} \in [\![0, b]\!],\ \mathsf{guard} \in \mathsf{LinArith}(\mathcal{M} \cup \mathsf{P})\}$*. The set* Rules *is required to satisfy the following conditions:*
  1. *the subgraph induced by the rules* ρ *with* $\rho.\mathsf{type} = 0$ *is acyclic;*
  2. *no rule* ρ *with* $\rho.\mathsf{type} = 0$ *has* $\rho.\mathsf{to} \in \mathcal{I}$*;*
  3. *for all rules* $\rho, \rho' \in \mathsf{Rules}$ *such that* $\rho.\mathsf{type} = 0$ *and* $\rho'.\mathsf{from}$ *is reachable from* $\rho.\mathsf{to}$ *by a finite sequence of type-*0 *rules, the guard* $\rho'.\mathsf{guard}$ *is monotone in the message variables occurring in* $\rho.\mathsf{guard}$.

In this definition, rc specifies the admissible parameter valuations; $\mathcal{L}$ and $\mathcal{M}$ denote, respectively, the locations a process may occupy and the message types it may send in a round; and each rule specifies a source location, a target location, a round increment of at most b, and the guard under which it is enabled. Condition 1 ensures that each process visits each location at most once within a round. Condition 2 ensures that processes enter initial locations only at the start of a new round. Condition 3 is a technical monotonicity condition adapted from [41]. It is needed for the completeness proof of the received-message abstraction, one of the six transformations presented in § 3. Importantly, the overall reduction remains sound even without this condition. Moreover, as argued in [41], the monotonicity condition is often implicitly assumed by distributed algorithms designers, and are thus satisfied by a wide range of distributed algorithms, such as the ones from [9, 4, 38, 12, 39].

Informally, Condition 3 ensures that once a process takes a type-0 rule, increasing the message counts that appear in that rule's guard cannot disable any rule that may be taken later in the same round. For example, if $\rho.\mathsf{guard} = \mathsf{m}_1 + \mathsf{m}_2 < \mathsf{n}/2$, then every rule ρ' whose source location is reachable from $\rho.\mathsf{to}$ by a finite sequence of type-0 rules, and hence may be taken after ρ within the same round, must have a guard that is monotone in m_1 and m_2. Here, a guard $g \in \mathsf{LinArith}(X)$ is *monotone* in a set $Y \subseteq X$ if, for all valuations $\mu, \mu' : X \to \mathbb{N}$, whenever μ satisfies g, $\mu'(y) \geq \mu(y)$ for all $y \in Y$, and $\mu'(x) = \mu(x)$ for all $x \in X \setminus Y$, it also holds that μ' satisfies g.

While a template describes the symbolic parameters and the behavior of a single process, the semantics of several interacting processes is represented by an infinite-state transition system. We first define the fixed-instance semantics, for a given parameter valuation, and then lift it to the parameterized semantics over all admissible parameter valuations.

Given a template $\mathcal{T}$ and a parameter valuation $\nu : \mathsf{P} \to \mathbb{N}$ such that $\nu \models \mathsf{rc}$, the system of $\nu(n)$ identical processes executing $\mathcal{T}$ is represented by the action-labeled transition system $\mathsf{RDTS}(\mathcal{T}, \nu) = \langle S_\nu, I_\nu, Act_\nu, Tr_\nu \rangle$. Its semantics is sketched here and detailed in App. A.1. Elements of S_ν are *global states* (or configurations), each composed of a process state and a network state. The *process state* $\mathsf{PState} : [\![1, \nu(n)]\!] \to \langle \mathsf{loc} : \mathcal{L}, \mathsf{rd} : \mathbb{N}, \mathsf{rcvMsg} : \mathcal{M} \times \mathbb{N} \to \mathbb{N} \rangle$ stores for each process: its current round, location, and multiset of received message types. The latter records how many messages of a given type were received with a given round tag. The *network state* $\mathsf{NState} : \mathcal{M} \times \mathbb{N} \to \mathbb{N}$ collects the number of messages of each type and round tag that were broadcast. In every configuration, the number of received messages (in process state) is pointwise bounded by the number of broadcast messages (in network state). Initial global states I_ν are those where all processes are in initial locations at round 0 with empty set of received messages, and the network state contains no message. Actions in Act_ν are of two types: (i) a reception $\mathsf{Receive}(i, \langle m, r \rangle)$, where process i receives a message of type m tagged with round r, or (ii) an update $\mathsf{Update}(i, \rho, r)$, where process i updates its state according to rule ρ of the template at round r. A message reception is enabled for a process if its count of received messages (of that type and round tag) is strictly less than the number of similar broadcast messages in the network state; the process then increments its message count accordingly. An update $\mathsf{Update}(i, \rho, r)$ is enabled if process i is at location $\rho.\mathsf{frm}$ in round r, and its received messages with round tag r satisfy the guard $\rho.\mathsf{guard}$; the process location and its round are then updated to $\rho.\mathsf{to}$ and $r + \rho.\mathsf{type}$, and the process broadcasts a message $\mathsf{Bcast}(\rho.\mathsf{to})$ (unless $\rho.\mathsf{to} = \perp$) tagged with round $r + \rho.\mathsf{type}$ on the network state. An *execution* of $\mathsf{RDTS}(\mathcal{T}, \nu)$ is a finite or infinite alternating sequence of configurations and transitions starting from an initial configuration.

To reason about all admissible parameter valuations simultaneously, we lift the fixed-instance transition system RDTS to its parameterized counterpart $\mathsf{PRDTS}(\mathcal{T})$, which takes the disjoint union of all $\mathsf{RDTS}(\mathcal{T}, \nu)$ for $\nu \models \mathsf{rc}$ (see App. A.2).

2.2 History State-Count Properties and Logic

The choice of a property class is central to any verification framework, as it determines both the expressiveness of the specifications and the feasibility of the reductions. We therefore need a specification formalism that balances three requirements: (i) expressiveness, to capture standard correctness conditions for ARDAs, as enumerated in Fig. 3 (left); (ii) reduction-compatibility, to establish a sound and complete reduction from parameterized RDTS to finite-counter systems; (iii) LTL-translatability, so that the final verification problem can be expressed as standard LTL model checking.

Previously introduced specification formalisms for threshold automata fall short in expressing standard correctness properties of ARDAs. Specifically, $\mathsf{ELTL}_{\mathsf{FT}}$ [26] lacks quantification over rounds and thus cannot express standard properties such as agreement. The extension with universal round quantifiers [6] enables reasoning about properties local to individual rounds, but it still cannot express consensus properties that inherently span multiple rounds, such as agreement. Finally, [7] does not define a formal specification logic and relies on assumptions about its successive abstractions that guarantee preservation of correctness.

These limitations motivate our choice of specification formalism. We propose a semantic class of *history state-count* (HSC) properties, chosen to satisfy expressiveness and reduction-compatibility. We also introduce HSC logic (HSCL), a syntactic fragment of temporal logic whose formulas denote a subset of HSC properties and are designed to be LTL-translatable while retaining expressiveness. In contrast to prior state-based approaches [26, 7], the HSC properties are action-based. This distinction is important for our reduction: although relevant action histories could, in principle, be encoded by auxiliary state predicates, the specifications and reduction are most naturally formulated in terms of the occurrence of transitions. Thus, instead of taking a restricted fragment of state-based LTL as the specification language, we use action-based HSCL and later translate it into LTL over the reduced finite-counter system.

2.2.1 History State-Count Properties

A History State-Count (HSC) property applies to history state-count traces of executions. For an execution π, its *history state-count trace* (hsc-trace) is a function $\mathsf{hsc}(\pi) : \mathcal{L} \times \mathbb{N} \to \mathbb{N}$ that records, along π, the aggregate number of process visits to each location in every round. Formally, for a location ℓ and a round tag r, $\mathsf{hsc}(\pi)(\ell, r)$ denotes the number of update actions, with some rule ρ and round tag r' taken along π such that $\rho.\mathsf{to} = \ell$ and $r' + \rho.\mathsf{type} = r$. Hence, the hsc-trace is entirely determined by the multiset of pairs (ρ, r) corresponding to the update actions occurring along the execution. Since each process can visit any location at most once per round (Condition 1, Def. 1), the hsc-trace equivalently captures, for every pair of a location and a round tag, the number of processes that have visited it.

For a template $\mathcal{T}$ with locations $\mathcal{L}$, a *history state-count* (HSC) property is a family $\mathcal{P} = \{\mathcal{P}_\nu \subseteq \mathsf{Dom} \mid \nu : \mathsf{P} \to \mathbb{N}\}$, where $\mathsf{Dom} = \{f : \mathcal{L} \times \mathbb{N} \to \mathbb{N}\}$. We write $\mathsf{Traces}(\mathsf{RDTS}(\mathcal{T}, \nu)) = \{\mathsf{hsc}(\pi) \mid \pi \text{ is an execution of } \mathsf{RDTS}(\mathcal{T}, \nu)\}$. We write $\mathsf{RDTS}(\mathcal{T}, \nu) \models \mathcal{P}$ iff $\mathsf{Traces}(\mathsf{RDTS}(\mathcal{T}, \nu)) \subseteq \mathcal{P}_\nu$, and define $\mathsf{PRDTS}(\mathcal{T}) \models \mathcal{P}$ iff, for all valid parameter valuations ν, $\mathsf{RDTS}(\mathcal{T}, \nu) \models \mathcal{P}$.

2.2.2 History State-Count Logic (HSCL)

Informally, HSCL has two types of atomic constraints: (i) *universal, round-local constraints* of the form $\forall r.\alpha_r$, which, for each round r, allow one to bound a weighted sum of visits to locations in that round by a threshold; and (ii) *cumulative constraints*, written β, which allow one to bound the corresponding weighted sum aggregated over all rounds. Formulas in HSCL are Boolean combinations (using $\neg, \wedge$) of these two types. Note that existential quantification over rounds is expressible as $\neg\forall r.\alpha_r \equiv \exists r.\neg\alpha_r$.

▶ **Definition 2** (HSC). *Let $\mathcal{X}$ be a finite set of locations and $\mathcal{Y}$ a finite set of parameters. Formulas of* HSCL *follow the grammar:*

$$\varphi \ ::= \ \forall r.\, \alpha_r \ \mid \ \beta \ \mid \ \neg\varphi \ \mid \ (\varphi \wedge \varphi)$$

$$\textit{with } \alpha_r \ ::= \ \sum_{\ell \in \mathcal{X}} c_\ell \cdot \kappa(\ell, r) \ \le \ t, \qquad \beta \ ::= \ \sum_{\ell \in \mathcal{X}} c_\ell \cdot \sum_{r \in \mathbb{N}} \kappa(\ell, r) \ \le \ t$$

where $c_\ell \in \mathbb{N}$ are non-negative weights*; $\kappa(\ell, r)$ is a* variable *(for $\ell \in \mathcal{X}$, $r \in \mathbb{N}$) interpreted as the number of process-visits to ℓ in round r; and the* threshold *t is a linear term over $\mathcal{Y}$ of the form $t = b_0 + \sum_{y \in \mathcal{Y}} b_y \cdot y$, with $b_0, b_y \in \mathbb{Z}$. We call instances of α_r* round-local atoms *and instances of β* cumulative atoms[1].

For a process template $\mathcal{T}$ with a location set $\mathcal{L}$ and a parameter set P, we interpret $\mathcal{X}$ as a subset of $\mathcal{L}$ and $\mathcal{Y}$ as a subset of P. A HSCL formula φ represents the HSC property $\mathcal{P}^\varphi = \{\mathcal{P}^\varphi_\nu \mid \nu : \mathsf{P} \to \mathbb{N}\}$ where for a parameter valuation ν, set $\mathcal{P}^\varphi_\nu = \{f : \mathcal{L} \times \mathbb{N} \to \mathbb{N} \mid f, \nu \models \varphi\}$. Here, $f, \nu \models \varphi$ iff φ evaluates to true under the substitution of each threshold term t in φ with its value under ν (i.e., $t[\mathcal{Y} \leftarrow \nu]$), and of each variable $\kappa(x, r)$ with $f(x, r)$. We write $\mathsf{RDTS}(\mathcal{T}, \nu) \models \varphi$ if $\mathsf{RDTS}(\mathcal{T}, \nu) \models \mathcal{P}^\varphi$,and $\mathsf{PRDTS}(\mathcal{T}) \models \varphi$ if $\mathsf{PRDTS}(\mathcal{T}) \models \mathcal{P}^\varphi$.

HSCL can express the five standard correctness properties listed in Fig. 3. The same figure gives the corresponding HSCL formulas and their LTL translations. We discuss the translation from HSCL to LTL in § 3.2. As we will see, HSCL formulas infact translate to

---

[1] Although the sum in β may diverge, since $c_\ell \ge 0$ and t is finite, satisfaction of β is determined by a finite partial sum.

Agreement: Either no correct process decides on 0, or no correct process decides on 1.
$\mathtt{A} := \forall r.\kappa(\mathsf{d0}, r) \le 0 \ \lor \forall r.\kappa(\mathsf{d1}, r) \le 0$ $\qquad \mathsf{LTL}(\mathtt{A}) \ = \mathbf{G}\big(\mathsf{local}(\mathsf{d0}) \le 0\big) \ \lor \ \mathbf{G}\big(\mathsf{local}(\mathsf{d1}) \le 0\big)$

Validity: If no correct process starts with 0, then no correct process decides on 0.
$\mathtt{V} := \forall r.\kappa(\mathsf{d0}, r) \le 0$ $\qquad \mathsf{LTL}(\mathtt{V}) \ = \mathbf{G}\big(\mathsf{local}(\mathsf{d0}) \le 0\big)$

Termination: Eventually, all correct processes decide.
$\mathtt{T} := \neg(\sum_r \kappa(\mathsf{d0}, r) + \kappa(\mathsf{d1}, r) \le \mathtt{N}_\mathsf{c} - 1)$ $\qquad \mathsf{LTL}(\mathtt{T}) \ = \neg\mathbf{G}\big(\mathsf{cumul}(\mathsf{d0}) + \mathsf{cumul}(\mathsf{d1}) \le \mathtt{N}_\mathsf{c}\big)$

Restricted Termination: If one correct process decides, then all correct processes decide.
$\mathtt{RT} := \neg(\sum_r \kappa(\mathsf{d0}, r) + \kappa(\mathsf{d1}, r) \le 0) \implies \mathtt{T}$ $\qquad \mathsf{LTL}(\mathtt{RT}) \ = \neg\mathbf{G}\big(\mathsf{cumul}(\mathsf{d0}) + \mathsf{cumul}(\mathsf{d1}) \le 0\big) \implies \mathsf{LTL}(\mathtt{T})$

Leader Uniqueness: At most one leader process per round.
$\mathtt{LU} := \forall r.\kappa(\mathsf{ldr}, r) \le 1$ $\qquad \mathsf{LTL}(\mathtt{LU}) \ = \mathbf{G}\big(\mathsf{local}(\mathsf{ldr}) \le 1\big)$

Figure 3 Expressing correctness properties of ARDAs in HSCL and their LTL translations (see § 3.2). $\mathcal{X} = \{\mathsf{d0}, \mathsf{d1}, \mathsf{ldr}\}$ denote decision-0, decision-1, and leader locations, respectively, and $\mathcal{Y} = \{\mathtt{N}_\mathsf{c}\}$ denotes the number of correct processes. Formulas for (restricted) termination assume that processes halt after making a decision.

restricted LTL properties, namely obligation properties [33]. These five properties guided the design of HSCL, whose syntax is tailored to encode them effectively. Although HSCL is intentionally minimal, our reduction to a finite-counter system applies more broadly to arbitrary HSC properties. Thus, the logic is not intended as a general-purpose specification language, but rather as a concise intermediate representation that facilitates translation to LTL.

2.3 Verification Problems

We can now define the four verification problems studied in this work. They are obtained by combining a choice of semantics, fixed-instance or parameterized, with a choice of specification formalism: either an arbitrary HSC property or a property expressed in HSCL.

P1 Fixed-instance HSC-Verification Given a template $\mathcal{T}$, an HSC property $\mathcal{P}$, and a parameter valuation ν with $\nu \models \mathsf{rc}$, does $\mathsf{RDTS}(\mathcal{T}, \nu) \models \mathcal{P}$?

P2 Parameterized HSC-Verification Given a template $\mathcal{T}$ and an HSC property $\mathcal{P}$, does $\mathsf{PRDTS}(\mathcal{T}) \models \mathcal{P}$?

P3 Fixed-instance HSCL-Verification Given a template $\mathcal{T}$, an HSCL formula φ, and a parameter valuation ν with $\nu \models \mathsf{rc}$, does $\mathsf{RDTS}(\mathcal{T}, \nu) \models \varphi$?

P4 Parameterized HSCL-Verification Given a template $\mathcal{T}$ and an HSCL formula φ, does $\mathsf{PRDTS}(\mathcal{T}) \models \varphi$?

Note that P1 and P2 are mathematical problems, whereas P3 and P4 are computational decision problems. The reductions developed in the next section relate these four problems to model checking problems over finite-state, one-counter, and finite-counter systems; see Table 1. In particular, the reduction for P3 yields LTL model checking over a finite-state system, and is therefore decidable. By contrast, the reduction for P4 yields LTL model checking over a finite-counter system. Here, a finite-counter system [2], is a symbolic representation of a transition system whose states are valuations of finitely many variables, called *counters*, ranging over the non-negative integers, and whose set of initial states and transition relation are definable by linear-arithmetic formulas over these counters. For model checking over

---

[2] finite-counter systems are instances of Presburger counter systems [17] with quantifier-free predicates

such systems, we allow LTL atoms to be linear-arithmetic formulas over the counters; see App. A.3 for formal definitions.

LTL model checking over finite-counter systems is undecidable in general, thus the reduction above does not by itself give a decision procedure for P4. We next show that P4 is indeed undecidable, by extending the undecidability result of [40] from synchronous round-based distributed algorithms to our asynchronous setting.

▶ **Theorem 3.** *Parameterized* HSCL-*Verification is undecidable.*

Proof. We reduce from the non-halting problem for two-counter machines (known to be undecidable [8]). Let $\mathcal{C}$ be a two-counter machine with a finite set of control states S, an initial state s_{init}, a halting state s_{halt}, and a finite set of commands Δ. Each command in Δ has the form (s, op, s'), where op is an operation of the form $\mathsf{inc}(c_i)$, $\mathsf{dec}(c_i)$, or $\mathsf{iszero}(c_i)$ for $i \in \{1, 2\}$. Commands with operations $\mathsf{dec}(c_i)$ and $\mathsf{iszero}(c_i)$ are enabled only when $c_i > 0$ and $c_i = 0$, respectively. From $\mathcal{C}$ we construct a round-based process template $\mathcal{T}_\mathcal{C}$ and an HSCL formula $\varphi_\mathcal{C}$ such that $\mathcal{C}$ has no halting execution iff $\mathsf{PRDTS}(\mathcal{T}_\mathcal{C}) \models \varphi_\mathcal{C}$.

The construction maintains the following invariant. The r-th configuration (s, v_1, v_2) of an execution of $\mathcal{C}$ is represented by a global configuration in which all processes have just jumped to round r, with one process, referred to as the controller, in location l_s, and v_i many processes in location l_{c_i} (for $i = 1, 2$), and the rest of the processes are in l_{res} ("reserve"). Thus, initially, one process is in $l_{s_{\mathsf{init}}}$ and all other processes are in l_{res}.

A transition in $\mathcal{C}$ from configuration (s, v_1, v_2) on a command, say, $(s, \mathsf{inc}(c_1), s')$, is simulated as follows. First, the controller moves from l_s to an intermediate location $l_{(s,\mathsf{inc}(c_1),s')}$ in the same round and broadcasts message m_{inc,c_1}; this message enables a process to move from l_{res} to another intermediate location l_{inc,c_1} of the same round and broadcast the message m_{ack}; this acknowledgment then allows all processes to move to their corresponding locations in the next round, i.e., the controller to $l_{s'}$, the process in l_{inc,c_1} to l_{c_1}, and the remaining processes to stay in their current locations. Upon entering the new round, each non-controller process broadcasts a ready message indicating whether it is in l_{c_1}, l_{c_2}, or l_{res}. Once the controller enters the next round and receives all $n-1$ ready messages, the simulated transition is complete. The other commands are simulated similarly; in particular, the ready-messages counts are used to test the enabledness of decrement and zero-test commands.

The template semantics is an over-approximation of the intended simulation. For example, several processes in l_{res} may receive m_{inc,c_1} and move to l_{inc,c_1} in the same round, thereby increasing the value of c_1 by more than one. The formula $\varphi_\mathcal{C} := \neg\psi_\mathcal{C}$ where $\psi_\mathcal{C}$ conjuncts conditions ensuring well-formedness of the simulation with a halting condition: in each round, at most one process may visit some location l_s with $s \in S$, at most one process may visit an intermediate location l_{inc,c_i} or l_{dec,c_i}, and some process eventually visits $l_{s_{\mathsf{halt}}}$. Thus, $\varphi_\mathcal{C}$ holds on all executions of $\mathsf{PRDTS}(\mathcal{T}_\mathcal{C})$ iff no well-formed simulated execution of $\mathcal{C}$ reaches s_{halt}.

The full definition of $T_\mathcal{C}$ and $\varphi_\mathcal{C}$ is given in App. A.4. ◀

Table 1 Reductions to model-checking (MC) problems over one-counter, finite-counter, and finite-state systems. All reductions are sound and complete.

Problem	**Reduced problem**	**Ref.**
P1 Fixed-instance HSC-Verification	MC of one-counter systems w.r.t. HSC	Thm. 9
P2 Parameterized HSC-Verification	MC of finite-counter systems w.r.t. HSC	Thm. 10
P3 Fixed-instance HSCL-Verification	MC of finite-state systems w.r.t. LTL	Thm. 13
P4 Parameterized HSCL-Verification	MC of finite-counter systems w.r.t. LTL	Thm. 14

3 Reductions

This section presents the reductions summarized in Table 1. They are obtained by a sequence of sound and complete transformation steps. Due to space constraints, we present only the intuition behind each step and defer the formal definitions and proofs to the App. B.

3.1 Reductions for HSC Properties

In this subsection, we present a reduction pipeline for HSC properties, shown in Fig. 4. The pipeline is sound and complete for HSC properties, and reduces fixed-instance semantics to one-counter systems and parameterized semantics to finite-counter systems. At a high level, the counters of the reduced systems record, for each relevant round, (i) the number of processes at each location and (ii) the number of messages of each type that have been broadcast. For the fixed-instance semantics, given a template $\mathcal{T}$ and valuation ν, the system $\mathsf{RDTS}(\mathcal{T}, \nu)$ reduces to a finite-counter system in which all counters except one are bounded by the number of processes $\nu(n)$; hence, the target is a *one-counter system*. For the parameterized semantics, we exploit the fact that our reductions for fixed instances guarantee that the set of counters is identical across parameter valuations. Taking the disjoint union over all admissible ν thus yields a *finite-counter system*. The reduction pipeline consists of four steps. At each step X, we denote by $\mathsf{RDTS}^X(\mathcal{T}, \nu) = \langle S^X_\nu, I^X_\nu, Act^X_\nu, Tr^X_\nu \rangle$ the fixed-instance semantics after the step X.

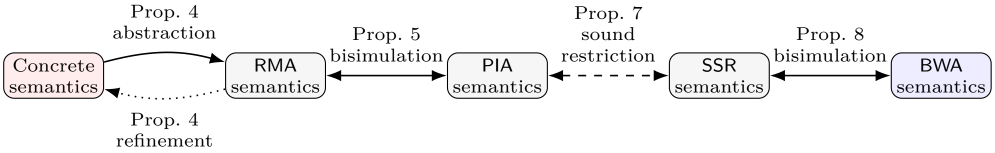


Figure 4 Reduction pipeline for fixed-instance and parameterized verification of HSC properties. All steps are sound and complete. RMA stands for received-message abstraction, PIA for process-identity abstraction, SSR for semi-synchronous restriction, and BWA for bounded-window abstraction. In the fixed-instance (resp. parameterized) setting, the target semantics BWA is a one-counter (resp. finite-counter) system.

Step 1. Received-Message Abstraction (RMA)

The first step exploits the fact that the hsc-trace of an execution is insensitive to Receive transitions. Consequently, receive transitions can be treated as stutter steps. In the abstract system $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$, the receive transitions are dropped and the elements of S^{RMA}_ν remain pairs of process and network states, except that the process state is simplified to $\mathsf{PState} : [\![1, \nu(n)]\!] \to \langle \mathsf{loc} : \mathcal{L}, \mathsf{rd} : \mathbb{N} \rangle$, omitting rcvMsg. Since Update transitions in $\mathsf{RDTS}(\mathcal{T}, \nu)$ are enabled only when their guards are justified by received messages, enabledness must now be redefined: in the abstract system, guards are evaluated *existentially* with respect to subsets of messages recorded in the network state NState. While abstraction of received messages has appeared in prior work [41, 7] in a state-based form, we give here an action-based formulation that treats receive transitions as stuttering steps. This yields refinement and forward simulation in the sense of Lynch et al. [32].

With Receive transitions as stutter steps, we establish a forward simulation [32, § 3.2] between the concrete semantics and the RMA semantics, proving that RMA is sound for HSC properties. The completeness of RMA is more subtle. The abstraction discards the concrete record of which messages a process has already *received and used* to satisfy previous guards. As a result, in the abstract enabledness relation, the witness for one update may be $R_1 \leq \mathsf{NState}$, while the witness for the very next update may be a different $R_2 \leq \mathsf{NState}$, without respecting the concrete monotonicity constraint that the second guard must hold on a super-set of messages already used (i.e., $R_2 \supseteq R_1$). Consequently, the abstraction may suggest that two updates can be carried out consecutively, even though no message-delivery schedule in the concrete system can realize them. Condition 3 in Def. 1 is imposed to exclude this phenomenon: once a process takes a type-0 rule, increasing the message counts used to justify its guard cannot disable any rule that may be taken later in the same round. Under this condition, the abstract witnesses for future updates can always be arranged consistently with a concrete receive schedule. Thus, with Receive transitions as stutter steps, we establish a refinement [32, § 3.1] between the RMA semantics to the concrete semantics, proving that RMA is complete for HSC properties.

▶ **Proposition 4** (Soundness and Completeness of RMA). *For every template $\mathcal{T}$ and parameter valuation ν:* $\mathsf{Traces}(\mathsf{RDTS}(\mathcal{T},\nu)) = \mathsf{Traces}(\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T},\nu))$.

Step 2. Process-Identity Abstraction (PIA)

The second step exploits the fact that the hsc-trace of an execution is insensitive to the identities of the processes performing Update transitions. Consequently, process identities can be dropped, resulting in an abstraction in which the exact states of processes are no longer recorded, only their counts. In the resulting abstract system $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$, the elements of S_ν^{PIA} remain pairs of process and network states, except that the process state is simplified to $\mathsf{PState} : \mathcal{L} \times \mathbb{N} \to \mathbb{N}$. Here $\mathsf{PState}(\ell, r)$ denote the number of processes in location ℓ of round r. Elements of Act_ν^{PIA} are of the form $\mathsf{Update}(\rho, r)$, again omitting process identities. Since the seminal paper [20], this counting abstraction is classical in the verification of systems composed of identical anonymous processes. The process-identity abstraction induces an action-based bisimulation [2, Def. 7.15] between $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T},\nu)$ and $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$, which entails the following result:

▶ **Proposition 5** (Soundness and Completeness of PIA). *For every template $\mathcal{T}$ and parameter valuation ν,* $\mathsf{Traces}(\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)) = \mathsf{Traces}(\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T},\nu))$.

The system $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$ is a counter system with *infinitely many* counters. Indeed, each round $r \in \mathbb{N}$ introduces $|\mathcal{L}| + |\mathcal{M}|$ counters: a location counter $\mathsf{PState}(\ell, r)$ for every $\ell \in \mathcal{L}$ and a message counter $\mathsf{NState}(m, r)$ for every $m \in \mathcal{M}$. To obtain a finite-counter reduction, a natural question is whether all counters must be tracked at all times. Intuitively, counters become redundant once they can no longer influence future updates. To capture this, we define the *relevant window* of a configuration as the interval of round tags whose counters may be nonzero and can affect future updates. Under the PIA semantics, for a configuration cfg, this window spans from the smallest ($r_{\mathsf{min}}(\mathsf{cfg})$) to the largest ($r_{\mathsf{max}}(\mathsf{cfg})$) round occupied by a process in cfg. A uniformly bounded relevant window across all configurations enables a direct reduction to a finite number of counters, as discussed in Step 4. However, due to asynchrony, the *round drift* $r_{\mathsf{max}}(\mathsf{cfg}) - r_{\mathsf{min}}(\mathsf{cfg})$ is unbounded across configurations of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$, preventing such a reduction. To this end, our next step constrains the enabledness of updates to enforce a uniformly bounded relevant window while preserving soundness for HSC properties.

Step 3. Semi-Synchronous Restriction (SSR)

The third step exploits the fact that the hsc-trace of an execution is insensitive to the order of Update transitions. Consequently, verification can be restricted to a subset of executions of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ such that every execution has a counterpart within this subset that executes the same multiset of Update actions. To achieve this, we impose a *semi-synchronous restriction* (SSR), which permits an action $a = \mathsf{Update}(\rho, r)$ from a configuration cfg only if its target round $\mathsf{tgt}(a) = r + \rho.\mathsf{type}$ is at least $r_{\mathsf{max}}(\mathsf{cfg})$, i.e., the highest round currently occupied by a process in cfg, also called the *frontier round* of cfg. The SSR therefore restricts executions by forbidding transitions that would move a process below the frontier round. As a result, since round jumps are bounded by b, the set of relevant counters in a configuration cfg is confined to the interval $[\![r_{\mathsf{max}}(\mathsf{cfg}) - b,\, r_{\mathsf{max}}(\mathsf{cfg})]\!]$, which has a uniform size of $b{+}1$ across all configurations. To establish the soundness of SSR for HSC properties, we use commutativity arguments similar to the one used in [13].

An execution $\pi = \mathsf{cfg}_1\, a_1\, \mathsf{cfg}_2\, a_2 \cdots$ of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ is called *semi-synchronous* iff its target sequence $\mathsf{tgt}(a_1)\mathsf{tgt}(a_2)\ldots$ is nondecreasing. This notion precisely characterizes the executions of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)$. The equivalence follows from the observation that, for every transition $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_{\nu}^{\mathsf{PIA}}$, the frontier round evolves as $r_{\mathsf{max}}(\mathsf{cfg}') = \max(r_{\mathsf{max}}(\mathsf{cfg}), \mathsf{tgt}(a))$.

▶ **Lemma 6.** *In* $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$*, for every execution* π *there exists a semi-synchronous execution* π' *that starts from the same initial configuration as* π *and contains the same multiset of update actions.*

The proof proceeds by transforming any execution π into a semi-synchronous execution through adjacent swaps of out-of-order update actions, thereby eliminating inversions in the target-round sequence. Each swap is justified by the *commutativity property*: two consecutive updates commute whenever the source round of the later differs from the target round of the earlier. To ensure convergence, swaps are performed in a disciplined manner that gives a monotone sequence of prefixes converging to a well-defined limiting execution, which is semi-synchronous.

The same commutativity property gives a stronger result that further reduces nondeterminism in semi-synchronous executions, which we exploit in the experiments reported in § 4. For every execution, there exists a semi-synchronous execution (with the same initial configuration and the same multiset of update actions) with the additional property that we call *strong*: for each round r, all *jump updates* targeting r (whose source round is smaller than r) happen before all *local updates* targeting r (whose source round equals r). Moreover, all jump updates are executed as a multiset in a fixed order which can alternatively be viewed as occurring synchronously in a single step.

▶ **Proposition 7** (Soundness and Completeness of SSR). *For every template* $\mathcal{T}$ *and parameter valuation* ν*,* $\mathsf{Traces}(\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)) = \mathsf{Traces}(\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu))$.

Step 4. Bounded-Window Abstraction (BWA)

The fourth abstraction step exploits the observation that configurations need not maintain counters that can no longer influence future updates. Since in $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)$ the size of the relevant window is bounded by $b + 1$ across all configurations, the BWA forgets counters outside this window and encodes the relevant portion as a sliding window of width $b + 1$. In the resulting abstract system $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T}, \nu)$, each configuration in S_{ν}^{BWA} is a triple consisting of a frontier round $r_{\mathsf{max}} \in \mathbb{N}$, a process state $\mathsf{PState} : \mathcal{L} \times [\![0, b]\!] \to \mathbb{N}$ and a network state $\mathsf{NState} : \mathcal{M} \times [\![0, b]\!] \to \mathbb{N}$. The process and network states record counters only for rounds ranging from the current frontier round down to b rounds below. The BWA induces an *action-based bisimulation* [2, Def. 7.15] between $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu)$ and $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$, establishing the following result:

▶ **Proposition 8** (Soundness and Completeness of BWA). *For every template $\mathcal{T}$ and parameter valuation ν,* $\mathsf{Traces}(\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)) = \mathsf{Traces}(\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu))$*.*

The system $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$ is a finite-counter system. Indeed, it maintains a single frontier-round counter r_{max}, and for each depth $d \in [\![0,b]\!]$, there are $|\mathcal{L}| + |\mathcal{M}|$ counters: a location counter $\mathsf{PState}(\ell,d)$ for every $\ell \in \mathcal{L}$ and a message counter $\mathsf{NState}(m,d)$ for every $m \in \mathcal{M}$. Moreover, $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$ can be viewed as a *one-counter system* with a *nondecreasing* counter. This is because the total number of processes $\nu(n)$ is bounded, and each process can send only a bounded number of messages ($\nu(n) \times |\mathcal{L}|$) per round. Consequently, all location and message counters are bounded, and the only unbounded counter is r_{max}, which increases monotonically. Thus, from Prop. 4, 5, 7, and 8, we have:

▶ **Theorem 9.** *For every template $\mathcal{T}$, parameter valuation ν, and* HSC *property $\mathcal{P}$, the system* $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$ *is a one-counter system with a nondecreasing counter r_{max}, and* $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu) \models \mathcal{P}$ *iff* $\mathsf{RDTS}(\mathcal{T},\nu) \models \mathcal{P}$*.*

We define the parameterized BWA semantics, denoted $\mathsf{PRDTS}^{\mathsf{BWA}}(\mathcal{T})$, as the disjoint union of all fixed-instance systems $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$ over admissible parameter valuations ν. Since the set of counter variables in $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$ is independent of ν, the states of $\mathsf{PRDTS}^{\mathsf{BWA}}(\mathcal{T})$ can be represented using finitely many counters: the frontier-round counter; the location and message counters inherited from $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu)$; and an additional set of $|\mathsf{P}|$ counters encoding the parameter valuation ν. Note, however, that the location and message counters in parameterized states need not be bounded, since the parameter $\nu(n)$ itself may range over unbounded values. Recall from § 2.2.1 that the parameterized semantics satisfies an HSC property iff all its admissible fixed-instance semantics do. Hence, from Thm. 9, we have:

▶ **Theorem 10.** *For every template $\mathcal{T}$ and* HSC *property $\mathcal{P}$, the system* $\mathsf{PRDTS}^{\mathsf{BWA}}(\mathcal{T})$ *is a finite-counter system, and* $\mathsf{PRDTS}^{\mathsf{BWA}}(\mathcal{T}) \models \mathcal{P}$ *iff* $\mathsf{PRDTS}(\mathcal{T}) \models \mathcal{P}$*.*

3.2 Further Reductions for HSCL Formulas

We now present a further reduction pipeline for HSCL formulas, shown in Fig. 5.

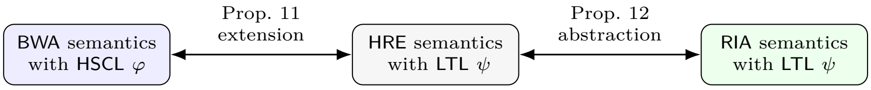


Figure 5 Further reduction pipeline for fixed-instance and parameterized verification of HSCL formulas. All steps are sound and complete. BWA stands for bounded-window abstraction, HRE for history-record extension, and RIA for round-identity abstraction. In the fixed-instance (resp. parameterized) setting, the target semantics RIA is a finite-state (resp. finite-counter) system.

Step 5. History-Record Extension (HRE)

The fifth step evaluates HSCL formulas from the state point of view by extending each BWA configuration with a "history record" that retains exactly those fragments of the execution history that are relevant to HSCL. Given an HSCL formula φ over locations $\mathcal{X}$ and parameters

$\mathcal{Y}$, we construct an extended system $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T},\nu,\varphi)$ by augmenting every state in S_ν^{BWA} with two counters for each $\ell \in \mathcal{X}$: a cumulative counter $\mathsf{cumul}(\ell)$, recording the number of visits to ℓ across all rounds; and a local counter $\mathsf{local}(\ell)$, recording the number of visits to ℓ in the current frontier round. Only the frontier round requires a dedicated local counter, since under BWA no update targets a round strictly below the frontier round. Thus, once a round is left behind, its local counts of process visits become fixed.

Observe that local counters are bounded by the total number of processes. In contrast, cumulative counters may grow unboundedly. However, for a fixed parameter valuation ν and formula φ, each cumulative counter can be capped at one more than the largest threshold appearing in the cumulative atoms of φ. Counts beyond this bound are irrelevant to satisfaction of HSCL formulas and can therefore be safely truncated. As a result, the system $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T},\nu,\varphi)$ remains a single-counter system with a nondecreasing counter r_{max}.

Next, we give a recursive procedure to translate the HSCL formula φ into an LTL formula $\mathsf{LTL}(\varphi)$. We use a variant of the standard LTL syntax in which the only temporal operator is "globally", written $\mathbf{G}$, and atomic propositions are linear-arithmetic formulas over the counter variables; see App. A.3 for the formal syntax and semantics in our setting. For universal round local atoms $\forall_r.\alpha_r$ where $\alpha_r = \sum_{\ell\in\mathcal{X}} c_\ell \cdot \kappa(\ell, r) \leq t$, define the translation $\mathsf{LTL}(\forall_r.\alpha_r) = \mathbf{G}(\sum_{\ell\in\mathcal{X}} c_\ell \cdot \mathsf{local}(\ell) \leq t)$. For cumulative atoms $\beta = \sum_{\ell\in\mathcal{X}} c_\ell \cdot \sum_r \kappa(\ell, r) \leq t$, define the translation $\mathsf{LTL}(\beta) = \mathbf{G}(\sum_{\ell\in\mathcal{X}} c_\ell \cdot \mathsf{cumul}(\ell) \leq t)$. The Boolean cases are handled compositionally: $\mathsf{LTL}(\neg\varphi) = \neg\mathsf{LTL}(\varphi)$ and $\mathsf{LTL}(\varphi \wedge \varphi') = \mathsf{LTL}(\varphi) \wedge \mathsf{LTL}(\varphi')$. Fig. 3 provides LTL translations for core HSCL formulas.

▶ **Proposition 11** (Soundness and Completeness of HRE). *For every* HSCL *formula* φ, $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T},\nu,\varphi) \models \mathsf{LTL}(\varphi)$ *iff* $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T},\nu) \models \varphi$.

Step 6. Round-Identity Abstraction (RIA)

The sixth and final step exploits the fact that the properties expressible in HSCL are insensitive to the round identifiers. Consequently, the frontier-round counter $r_{\max}$ can be dropped from HRE configurations. The resulting abstract system $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ induces a *state-based bisimulation* [2, Def. 7.7] with respect to $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T},\nu,\varphi)$, giving:

▶ **Proposition 12** (Soundness and Completeness of RIA). *For every* HSCL *formula* φ, $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi) \models \mathsf{LTL}(\varphi)$ *iff* $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T},\nu,\varphi) \models \mathsf{LTL}(\varphi)$.

Note that the system $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ is a finite-counter system, all of whose counters are bounded by a finite value determined by the parameter valuation ν, and is therefore a finite-state system. By Thm. 9 and Prop. 11 and 12, we have:

▶ **Theorem 13.** *For every template $\mathcal{T}$, parameter valuation ν, and an* HSCL *formula φ, the system* $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ *is a finite-state system, and* $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi) \models \mathsf{LTL}(\varphi)$ *iff* $\mathsf{RDTS}(\mathcal{T},\nu) \models \varphi$.

The parameterized RIA semantics, denoted $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi)$, is defined as the disjoint union of all fixed-instance systems $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ over all admissible parameter valuations ν. Since the set of counters in $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ does not depend on ν, each state of $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi)$ can be represented with finitely many counters: those of $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ together with an additional $|\mathsf{P}|$ counters encoding the parameter valuation ν. Hence, $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi)$ is a finite-counter system. The atoms of the LTL formula $\mathsf{LTL}(\varphi)$ are interpreted over these parameter counters and over the local/cumulative visit counters of $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi)$. By the definition of the disjoint union, $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi) \models \mathsf{LTL}(\varphi)$ iff, for all $\nu \models \mathsf{rc}$, $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi) \models \mathsf{LTL}(\varphi)$. Therefore, by Thm. 13, we obtain:

Table 2 Case-study results with the IC3 engine of nuXmv. Columns b, $|\mathcal{L}|$, $|\mathsf{Rules}|$, and rc give the jump bound, number of locations, number of rules, and resilience condition of the process template. For each property in Fig. 3, the rest of columns report verification time in seconds, with the maximal IC3 depth in parentheses. For T, the entry reports the time and depth to find a counterexample.

Protocol	b	$\|\mathcal{L}\|$	$\|\mathsf{Rules}\|$	rc	A	V	RT	T	LU
Ben-Or (Crash)	1	9	26	$n > 2t$	1.4s (13)	0.4s (9)	3.1s (8)	0.5s (3)	–
Ben-Or (Byzantine)	1	10	27	$n > 5t$	7.0s (11)	1.2s (7)	4.3s (7)	0.6s (3)	–
Bracha (Byzantine)	1	12	31	$n > 3t$	14.0s (14)	1.8s (8)	6.5s (11)	0.7s (3)	–
Raft leader election	2	11	25	$n > 2t$	–	–	–	–	1.8s (8)

▶ **Theorem 14.** *For every template* $\mathcal{T}$ *and* HSCL *formula* φ*, the system* $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi)$ *is a finite-counter system, and* $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi) \models \mathsf{LTL}(\varphi)$ *iff* $\mathsf{PRDTS}(\mathcal{T}) \models \varphi$.

The above six transformation steps establish the correctness of our reduction. App. D gives a direct construction of the reduced finite-state and finite-counter system, $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T},\nu,\varphi)$ and $\mathsf{PRDTS}^{\mathsf{RIA}}(\mathcal{T},\varphi)$, from the input template $\mathcal{T}$ and formula φ.

4 Case Studies

We evaluate the practical usefulness of our reduction on four case studies: (i) Ben-Or's consensus with crash faults [4, Algo A], (ii) Ben-Or's consensus with Byzantine faults [4, Algo B], (iii) Bracha's consensus with Byzantine faults [9, Fig. 4], and (iv) Raft's leader-election [35, Sec. 5.2]. The anonymous repository [1] provides source files and instructions to reproduce all experiments. For each case study, the repository includes an `.smv` file whose initial commented block specifies the corresponding process template, followed by the RIA semantics used as input to nuXmv. The models abstract randomization by nondeterminism; as a result, termination is no longer guaranteed, although the remaining properties are unaffected. Byzantine faults are modeled explicitly using dedicated fault locations, from which faulty processes may nondeterministically broadcast messages of arbitrary types. The model checker nuXmv is called via its IC3 engine (`check_ltlspec_ic3`) to check whether the translated LTL specifications hold on the RIA semantics. Table 2 reports, for each case study and property, the verification time and the maximal IC3 depth. Across all experiments, IC3 reaches shallow depths and terminates in under 15 seconds. To stress-test our reduction, we introduced intentional faults into the algorithm models by weakening the resilience condition or altering key guard predicates. As expected, for these faulty variants nuXmv produced counterexamples within seconds.

5 Conclusion

Asynchronous round-based distributed algorithms underpin many modern distributed systems, including many consensus algorithms at the core of blockchain technologies [44]. Our reduction offers a formal step toward their verification via mature symbolic model checkers and suggests a number of challenging extensions.

First, an important direction is to extend our reduction to incorporate probabilistic behavior and so allow verification of almost-sure termination. While existing probabilistic model checkers [29, 15] can efficiently verify almost-sure reachability conditions on *finite* MDPs, obtaining such a model is far from immediate, even for a fixed parameter valuation. The main challenge arises in the SSR step as a probabilistic semantics requires to reorder not only a trace, but the whole execution tree.

Second, a compelling challenge is to extend our reduction to handle algorithms whose number of locations within a round grows with the round index, as is the case for variants of Paxos [30, 31] and DAG-based consensus [3, 22, 23, 19, 18] (where the number of locations also grows with the number of processes). Interestingly, the reduction we presented in this paper does not assume that the set of locations within each round is finite; however, the reduced system would then have infinitely many counters.

References

1 Anonymous. Case study repository, 2025. URL: `https://github.com/AnonPaperX/Parameterized-Verification-of-Round-Based-Distributed-Algorithms`.

2 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* The MIT Press, 2008.

3 Leemon Baird and Atul Luykx. The hashgraph protocol: Efficient asynchronous BFT for high-throughput distributed ledgers. In *Proceedings of the 2020 International Conference on Omni-layer Intelligent Systems (COINS 2020)*, pages 1–7. IEEE, 2020. `doi:10.1109/COINS49042.2020.9191430`.

4 Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Procedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC 1983)*, pages 27–30. Association for Computing Machinery, 1983. `doi:10.1145/800221.806707`.

5 Nathalie Bertrand, Pranav Ghorpade, Sasha Rubin, Bernhard Scholz, and Pavle Subotić. Reusable formal verification of dag-based consensus protocols. In *Proceedings of the 17th International Symposium on NASA Formal Methods (NFM 2025)*, pages 138–158. Springer-Verlag, 2025. `doi:10.1007/978-3-031-93706-4_9`.

6 Nathalie Bertrand, Igor Konnov, Marijana Lazić, and Josef Widder. Verification of Randomized Consensus Algorithms Under Round-Rigid Adversaries. In *Proceedings of the 30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.33`.

7 Nathalie Bertrand, Bastien Thomas, and Josef Widder. Guard Automata for the Verification of Safety and Liveness of Distributed Algorithms. In *Proceeding of the 32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CONCUR.2021.15`.

8 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. `doi:10.2200/S00658ED1V01Y201508DCT013`.

9 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. `doi:10.1016/0890-5401(87)90054-X`.

10 Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV 2014)*, pages 334–342. Springer International Publishing, 2014.

11 Saksham Chand, Yanhong A. Liu, and Scott D. Stoller. Formal verification of multi-paxos for distributed consensus. In *Proceedings of the 21st International Symposium on Formal Methods (FM 2016)*, volume 9995 of *Lecture Notes in Computer Science*, pages 119–136, 2016. `doi:10.1007/978-3-319-48989-6\_8`.

12 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996. `doi:10.1145/226643.226647`.

13 Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In *Proceedings of the 3rd International*

Workshop on Reachability Problems (RP 2009), pages 93–106. Springer Berlin Heidelberg, 2009. `doi:10.1007/978-3-642-04420-5\_10`.

14 Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 04 2009. `doi:10.1007/s00446-009-0084-6`.

15 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker, 2017. URL: `https://arxiv.org/abs/1702.04311`, `arXiv:1702.04311`.

16 Giorgio Delzanno, Michele Tatarek, and Riccardo Traverso. Model checking paxos in spin. In *Proceedings 5th International Symposium on Games, Automata, Logics and Formal Verification (GandALF 2014)*, volume 161 of *EPTCS*, pages 131–146, 2014. `doi:10.4204/EPTCS.161.13`.

17 Stéphane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Model-checking ctl* over flat presburger counter systems. *J. Appl. Non Class. Logics*, 20(4):313–344, 2010. URL: `https://doi.org/10.3166/jancl.20.313-344`, `doi:10.3166/JANCL.20.313-344`.

18 Fantom Foundation. Lachesis aBFT, 2024. URL: `https://docs.fantom.foundation/technology/lachesis-abft`.

19 Adam Gagol, Damian Lesniak, Damian Straszak, and Michal Swietek. Aleph: Efficient atomic broadcast in asynchronous networks with Byzantine nodes. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT 2019)*, pages 214–228. ACM, 2019. `doi:10.1145/3318041.3355467`.

20 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992. `doi:10.1145/146637.146681`.

21 Aman Goel and Karem Sakallah. On symmetry and quantification: A new approach to verify distributed protocols. In *Proceedings of the 13th International Symposium on NASA Formal Methods (NFM 2021)*, page 131–150, Berlin, Heidelberg, 2021. Springer-Verlag. `doi:10.1007/978-3-030-76384-8_9`.

22 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *Proceedings of 40th ACM Symposium on Principles of Distributed Computing (PODC 2021)*, pages 165–175. ACM, 2021. `doi:10.1145/3465084.3467905`.

23 Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In *Proceedings of the 37th International Symposium on Distributed Computing, (DISC 2023)*, volume 281 of *LIPIcs*, pages 26:1–26:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.DISC.2023.26`.

24 Jason R. Koenig, Oded Padon, Sharon Shoham, and Alex Aiken. Inferring invariants with quantifier alternations: Taming the search space explosion. In *TACAS (1)*, volume 13243 of *Lecture Notes in Computer Science*, pages 338–356. Springer, 2022.

25 Igor Konnov, Marijana Lazic, Ilina Stoilkovska, and Josef Widder. Survey on parameterized verification with threshold automata and the Byzantine model checker. *Logical Methods in Computer Science*, 19(1), 2023. `doi:10.46298/LMCS-19(1:5)2023`.

26 Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*, page 719–734. Association for Computing Machinery, 2017. `doi:10.1145/3009837.3009860`.

27 Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In *CAV (1)*, volume 9206 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2015.

28 Igor Konnov, Josef Widder, Francesco Spegni, and Luca Spalazzi. Accuracy of message counting abstraction in fault-tolerant distributed algorithms. In *Proceedings of the 18th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2017)*, pages 347–366. Springer International Publishing, 2017.

29 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

30 Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998. `doi:10.1145/279227.279229`.

31 Leslie Lamport. Fast Paxos. *Distributed Computing*, 19(2):79–103, 2006. `doi:10.1007/s00446-006-0005-x`.

32 Nancy Lynch and Frits Vaandrager. Forward and backward simulations i.: untimed systems. *Information and Computation*, 121(2):214–233, 1995. `doi:10.1006/inco.1995.1134`.

33 Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *PODC*, pages 377–410. ACM, 1990.

34 Tatsuya Noguchi, Tatsuhiro Tsuchiya, and Tohru Kikuno. Safety verification of asynchronous consensus algorithms with model checking. In *IEEE 18th Pacific Rim International Symposium on Dependable Computing (PRDC 2012)*, pages 80–88. IEEE Computer Society, 2012. `doi:10.1109/PRDC.2012.24`.

35 Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference*, pages 305–320, 2014.

36 Oded Padon, Kenneth L. McMillan, Aurojit Panda, Shmuel Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2016)*, 2016. URL: `https://api.semanticscholar.org/CorpusID:17141487`.

37 William Schultz, Edward Ashton, Heidi Howard, and Stavros Tripakis. Scalable, interpretable distributed protocol verification by inductive proof slicing. *CoRR*, abs/2404.18048, 2024. URL: `https://doi.org/10.48550/arXiv.2404.18048`, `arXiv:2404.18048`, `doi:10.48550/ARXIV.2404.18048`.

38 Yee Jiun Song and Robbert van Renesse. Bosco: One-Step Byzantine Asynchronous Consensus. In Gadi Taubenfeld, editor, *Distributed Computing*, pages 438–450, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

39 T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Comput.*, 2(2):80–94, 1987. `doi:10.1007/BF01667080`.

40 Ilina Stoilkovska, Igor Konnov, Josef Widder, and Florian Zuleger. Verifying safety of synchronous fault-tolerant algorithms by bounded model checking. In Tomás Vojnar and Lijun Zhang, editors, *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019)*, Lecture Notes in Computer Science, pages 357–374. Springer, 2019. `doi:10.1007/978-3-030-17465-1\_20`.

41 Ilina Stoilkovska, Igor Konnov, Josef Widder, and Florian Zuleger. Eliminating message counters in threshold automata. In *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis (ATVA 2020)*, pages 196–212. Springer-Verlag, 2020. `doi:10.1007/978-3-030-59152-6_11`.

42 Bastien Thomas and Ocan Sankur. PyLTA: A verification tool for parameterized distributed algorithms. In *Proceedings of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2023)*, pages 28–35. Springer Nature Switzerland, 2023.

43 Doug Woos, James R. Wilcox, Steve Anton, Zachary Tatlock, Michael D. Ernst, and Thomas Anderson. Planning for change in a formal verification of the Raft consensus protocol. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2016)*, pages 154–165. ACM, 2016. `doi:10.1145/2854065.2854081`.

44 Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020. `doi:10.1109/COMST.2020.2969706`.

45 Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, and Gabriel Ryan. DistAI: Data-Driven automated invariant learning for distributed protocols. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 405–421. USENIX Association, July 2021. URL: `https://www.usenix.org/conference/osdi21/presentation/yao`.

A Details on Section 2

A.1 Fixed-instance semantics

Let $\mathcal{T} = \langle \mathsf{P}, \mathsf{rc}, \mathcal{L}, \mathcal{I}, \mathcal{M}, \mathsf{Bcast}, \mathsf{Rules} \rangle$ be a template and $\nu : \mathsf{P} \to \mathbb{N}$ a *parameter valuation* such that $\nu \models \mathsf{rc}$. In the system consisting of $\nu(n)$ identical processes, each process execute the behavior specified by $\langle \mathcal{L}, \mathcal{I}, \mathcal{M}, \mathsf{Bcast}, \mathsf{Rules} \rangle$.

The *fixed-instance semantics* of the template $\mathcal{T}$ for a parameter valuation ν, denoted $\mathsf{RDTS}(\mathcal{T}, \nu)$, is an action-labeled transition system:

$$\mathsf{RDTS}(\mathcal{T}, \nu) \; = \; \langle S_\nu, \; I_\nu, \; Act_\nu, \; Tr_\nu \rangle,$$

where S_ν denotes the set of *configurations*, $I_\nu \subseteq S_\nu$ the set of *initial configurations*, Act_ν the *action set*, and $Tr_\nu \subseteq S_\nu \times Act_\nu \times S_\nu$ the *transition relation*.

Configurations. A *configuration* is a pair $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$ consisting of a process state and a network state. The process state is a function $\mathsf{PState} : [\![1, \nu(n)]\!] \to \langle \mathsf{loc} : \mathcal{L}, \mathsf{rd} : \mathbb{N}, \mathsf{rcvMsg} : \mathcal{M} \times \mathbb{N} \to \mathbb{N} \rangle$ that assigns to each process a location in $\mathcal{L}$, a round number in $\mathbb{N}$, and a multiset of received messages without sender identities. Specifically, $\mathsf{rcvMsg}(m, r)$ records how many times a message of type $m \in \mathcal{M}$ was received with round tag r. This representation abstracts away sender identities and retains only message counts — a standard abstraction known to preserve the temporal properties typically verified for fault-tolerant distributed algorithms [28]. The reason is that correctness properties typically depend on the control locations occupied by processes, rather than on the specific messages they send or receive. Moreover, the guards that determine these locations are threshold-based and independent of process identifiers.

The network state $\mathsf{NState} : \mathcal{M} \times \mathbb{N} \to \mathbb{N}$ records how many messages of each type and round tag were broadcast, irrespective of the sender's identity. The acyclicity assumption in Definition 1 ensures that no process visits the same location more than once per round. Consequently, each process broadcasts at most $|\mathcal{L}|$ messages in a round, and the range of NState is therefore bounded above by $\nu(n) \times |\mathcal{L}|$. Moreover, if all processes are in a round with number less than r (formally, $\forall i : \mathsf{PState}(i).\mathsf{rd} < r$), then $\mathsf{NState}(\cdot, r)$ must be the zero function $\mathbf{0}$, mapping every argument to 0. Finally, in every configuration, the number of received messages is *pointwise bounded* by the number of broadcast messages:

$$\forall i, \; \forall (m, r) \in \mathcal{M} \times \mathbb{N} : \quad \mathsf{PState}(i).\mathsf{rcvMsg}(m, r) \;\leq\; \mathsf{NState}(m, r). \tag{1}$$

A configuration $\langle \mathsf{PState}, \mathsf{NState} \rangle$ is *initial* if every process is in an initial location at round 0 with an empty set of received messages, and the network contains no broadcasts. Formally: for all $i \in [\![1, \nu(n)]\!]$, $\mathsf{PState}(i).\mathsf{loc} \in \mathcal{I}$, $\mathsf{PState}(i).\mathsf{rd} = 0$, and $\mathsf{PState}(i).\mathsf{rcvMsg} = \mathbf{0}$; and $\mathsf{NState} = \mathbf{0}$.

Actions and Transitions. A process $i \in [\![1, \nu(n)]\!]$ can perform two types of actions: (i) $\mathsf{Recv}(i, \langle m, r \rangle)$, which represents the reception of a message of type m tagged with round r; (ii) $\mathsf{Update}(i, \rho, r)$, which represents updating its location by applying rule $\rho \in \mathsf{Rules}$ at round $r \in \mathbb{N}$. These form the action set Act_ν, in which each action represents a step executed by a single process. The transition relation Tr_ν consists of all triples $(\mathsf{cfg}, a, \mathsf{cfg}')$ such that $a \in Act_\nu$ is *enabled* in the configuration $\mathsf{cfg} \in S_\nu$ (denoted $\mathsf{enabled}(\mathsf{cfg}, a)$), and executing a at cfg yields cfg' (denoted $\mathsf{effect}(\mathsf{cfg}, a) = \mathsf{cfg}'$). We now formally define enabled and effect for *receive* and *update* actions.

Receive. In a configuration $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$, the action $\mathsf{Recv}(i, \langle m, r \rangle)$ is enabled when process i has not yet received all broadcasts of type m tagged with round r. Formally,

$$\mathsf{enabled}(\mathsf{cfg}, \mathsf{Recv}(i, \langle m, r \rangle)) := \mathsf{PState}(i).\mathsf{rcvMsg}(m, r) < \mathsf{NState}(m, r).$$

Executing $\mathsf{Recv}(i, \langle m, r \rangle)$ delivers one message of type m tagged with round r to process i and leaves all other components unchanged. Formally,

$\mathsf{effect}(\mathsf{cfg}, \mathsf{Recv}(i, \langle m, r \rangle)) := \langle \mathsf{PState}', \mathsf{NState}' \rangle$, where:

- For every process $j \neq i$, the state remains unchanged: $\mathsf{PState}'(j) = \mathsf{PState}(j)$.
- Location and round component of process i state remain unchanged: $\mathsf{PState}'(i).\mathsf{loc} = \mathsf{PState}(i).\mathsf{loc}$ and $\mathsf{PState}'(i).\mathsf{rd} = \mathsf{PState}(i).\mathsf{rd}$.
- The receive component of process i state is incremented at $\langle m, r \rangle$: $\mathsf{PState}'(i).\mathsf{rcvMsg}(m, r) = \mathsf{PState}(i).\mathsf{rcvMsg}(m, r) + 1$ and for all $\langle m', r' \rangle \neq \langle m, r \rangle$, $\mathsf{PState}'(i).\mathsf{rcvMsg}(m', r') = \mathsf{PState}(i).\mathsf{rcvMsg}(m', r')$.
- The network state remains unchanged: $\mathsf{NState}' = \mathsf{NState}$.

Update. In a configuration $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$, the action $\mathsf{Update}(i, \rho, r)$ is enabled when process i is in location $\rho.\mathsf{frm}$ at round r, and its received messages in round r satisfy the guard $\rho.\mathsf{guard}$. Formally,

$$\begin{aligned}\mathsf{enabled}(\mathsf{cfg}, \mathsf{Update}(i, \rho, r)) := \;& \mathsf{PState}(i).(\mathsf{loc}, \mathsf{rd}) = (\rho.\mathsf{frm}, r) \text{ and} \\ & \mathsf{PState}(i).\mathsf{rcvMsg}(\_, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu].\end{aligned}$$

Executing $\mathsf{Update}(i, \rho, r)$ has two effects. First, process i moves from location $\rho.\mathsf{frm}$ to $\rho.\mathsf{to}$ and updates its round counter to $r + \rho.\mathsf{type}$. Second, it broadcasts a message of type $\mathsf{Bcast}(\rho.\mathsf{to})$, tagged with round $r + \rho.\mathsf{type}$. Formally,

$\mathsf{effect}(\mathsf{cfg}, \mathsf{Update}(i, \rho, r)) = \langle \mathsf{PState}', \mathsf{NState}' \rangle$, where:

- For every process $j \neq i$, the state remains unchanged: $\mathsf{PState}'(j) = \mathsf{PState}(j)$.
- The location component of process i is updated to $\rho.\mathsf{to}$: $\mathsf{PState}'(i).\mathsf{loc} = \rho.\mathsf{to}$.
- The round component of process i state is incremented by $\rho.\mathsf{type}$: $\mathsf{PState}(i)'.\mathsf{rd} = r + \rho.\mathsf{type}$
- The receive component of process i remains unchanged: $\mathsf{PState}'(i).\mathsf{rcvMsg}(m, r) = \mathsf{PState}(i).\mathsf{rcvMsg}(m, r)$.
- The network state is incremented at $\langle \mathsf{Bcast}(\rho.\mathsf{to}), r + \rho.\mathsf{type} \rangle$: $\mathsf{NState}'(m', r') = \mathsf{NState}(m', r') + 1$ if $\langle m', r' \rangle = \langle \mathsf{Bcast}(\rho.\mathsf{to}), r + \rho.\mathsf{type} \rangle$, and $\mathsf{NState}'(m', r') = \mathsf{NState}(m', r')$ otherwise.

An *execution* of $\mathsf{RDTS}(\mathcal{T}, \nu)$ is a finite or infinite alternating sequence $\pi = \mathsf{cfg}_0 a_1 \mathsf{cfg}_1 a_2 \ldots$, such that $\mathsf{cfg}_0 \in I_\nu$ is an initial configuration and, for all k, $(\mathsf{cfg}_k, a_{k+1}, \mathsf{cfg}_{k+1}) \in Tr_\nu$. We adopt this definition of executions uniformly for all subsequent semantics and do not restate it.

A.2 Parameterized semantics

The parameterized semantics is a disjoint union of all fixed-instance semantics $\mathsf{RDTS}(\mathcal{T},\nu)$ for $\nu \models \mathsf{rc}$. Fix a template $\mathcal{T}$, and let $\mathsf{Val} = \{\nu : \mathsf{P} \to \mathbb{N} \mid \nu \models \mathsf{rc}\}$ denote the set of all admissible parameter valuations. The parameterized semantics of $\mathcal{T}$ is an action-labeled transition system, called the *parameterized* RDTS (PRDTS), defined as

$$\mathsf{PRDTS}(\mathcal{T}) = \langle S,\ I,\ Act,\ Tr\rangle,$$

which is the *disjoint union* of $\mathsf{RDTS}(\mathcal{T},\nu)$ over all $\nu \in \mathsf{Val}$, i.e.:

$$S = \bigcup_{\nu\in\mathsf{Val}} \{\nu\} \times S_\nu, \quad I = \bigcup_{\nu\in\mathsf{Val}} \{\nu\} \times I_\nu, \quad Act = \bigcup_{\nu\in\mathsf{Val}} \{\nu\} \times Act_\nu,$$

$$Tr = \left\{(\nu,\mathsf{cfg}) \xrightarrow{(\nu,a)} (\nu,\mathsf{cfg}') \;\middle|\; \nu \in \mathsf{Val},\ (\mathsf{cfg},a,\mathsf{cfg}') \in Tr_\nu\right\}.$$

For every execution $\pi = (\nu,\mathsf{cfg}_0)(\nu,a_1)(\nu,\mathsf{cfg}_1)\ldots$ of $\mathsf{PRDTS}(\mathcal{T})$, let $\pi_{\downarrow_\nu} = \mathsf{cfg}_0 a_1 \mathsf{cfg}_1 \ldots$. Clearly $\pi_{\downarrow_\nu}$ is an execution of $\mathsf{RDTS}(\mathcal{T},\nu)$. Moreover, for every execution π of $\mathsf{RDTS}(\mathcal{T},\nu)$, there exists execution π' of $\mathsf{PRDTS}(\mathcal{T})$ such that $\pi = \pi'_{\downarrow_\nu}$.

A.3 LTL Model Checking over Finite-Counter Systems

We use a notion of finite-counter system that can be directly represented as a nuXmv model. A *finite-counter system* $\mathcal{C}$ is a tuple $(V, \mathsf{INVAR}, \mathsf{INIT}, \mathsf{TRAN})$, where $V = \{x_1,\ldots,x_k\}$ is a finite set of variables (also called counters) ranging over $\mathbb{N}$, INVAR and INIT are linear-arithmetic formulas over $x_1,\ldots,x_k$, and TRAN is a linear-arithmetic formula over $x_1,\ldots,x_k,x'_1,\ldots,x'_k$. Here, x'_i denotes the next-state copy of x_i.

A finite-counter system $\mathcal{C} = (V, \mathsf{INVAR}, \mathsf{INIT}, \mathsf{TRAN})$ induces a transition system $\mathcal{S}_C := (Q, Q_0, \to)$, where Q is the set of valuations $s : V \to \mathbb{N}$ satisfying INVAR, $Q_0 \subseteq Q$ is the set of valuations satisfying INIT, and $\to \subseteq Q \times Q$ contains exactly the pairs (s,s') satisfying TRAN, i.e., those for which TRAN is true after substituting each x_i by $s(x_i)$ and each x'_i by $s'(x_i)$. Intuitively, INVAR defines the admissible counter values, INIT defines the admissible initial counter values, and TRAN relates current counter values to their possible next values.

We use a fragment of the syntax of LTL [2], defined according to the following grammar:

$$\varphi ::= \mathit{true} \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{G}\varphi$$

where $\mathbf{G}$ is the "globally" operator (we do not use the "next" or "until" operators), and p varies over the *atomic propositions* which are taken to be the linear-arithmetic formulas over V (rather than abstract symbols). Note that we can interpret atoms in states of the transition system $\mathcal{S}_\mathcal{C}$ (this implicitly defines a labeling function mapping each state to the set of linear-arithmetic formulas true in that state, but we will not use this labeling function explicitly).

We inductively define what it means for an infinite sequence $\pi = \pi_0\pi_1\cdots$, with $\pi_i : V \to \mathbb{N}$ for all i, to *satisfy* an LTL formula, as usual [2]:

- $\pi \models \mathit{true}$;
- $\pi \models p$ if the atom p is true under the valuation π_0;
- $\pi \models \varphi_1 \wedge \varphi_2$ if $\pi \models \varphi_1$ and $\pi \models \varphi_2$;
- $\pi \models \neg\varphi$ if $\pi \not\models \varphi$;
- $\pi \models \mathbf{G}\varphi$ if for all $j \geq 0$, it holds that $\pi_j\pi_{j+1}\cdots \models \varphi$.

A *path* of $\mathcal{S}_\mathcal{C}$ is a sequence $\pi = s_0 s_1 s_2 \cdots$ such that $s_i \in Q$ for all i, $s_0 \in Q_0$, and $(s_i, s_{i+1}) \in \rightarrow$ for every i. A *maximal* path is either an infinite path or a finite path that ends in a *terminal* state s, i.e., for which there is no state $s' \in Q$ such that $(s, s') \in \rightarrow$. A finite maximal path $s_0 s_1 \cdots s_n$ is said to *satisfy* φ iff its infinite stuttering extension $s_0 s_1 \cdots s_n s_n s_n \cdots$ satisfies φ.

Finally, given a finite-counter system $\mathcal{C}$ and an LTL formula φ, we write $\mathcal{C} \models \varphi$, and say that $\mathcal{C}$ *satisfies* φ, iff every maximal path of the induced transition system $\mathcal{S}_\mathcal{C}$ satisfies φ.

A.4 Undecidability of Parameterized HSCL-Verification

We give the detailed construction of $\mathcal{T}_\mathcal{C}$ and $\varphi_\mathcal{C}$ used in the proof of Theorem 3.
The template $\mathcal{T}_\mathcal{C}$ has jump bound $b = 1$ and is defined as follows.

- **Controller locations and messages.** For each control state $s \in S$, there is a location l_s, with no broadcast message. For each command $\mathsf{cmd} = (s, \mathsf{op}, s') \in \Delta$, there is an intermediate location l_{cmd}. If $\mathsf{op} = \mathsf{inc}(c_i)$, then l_{cmd} broadcasts m_{inc,c_i}; if $\mathsf{op} = \mathsf{dec}(c_i)$, then it broadcasts m_{dec,c_i}; and if $\mathsf{op} = \mathsf{iszero}(c_i)$, then it broadcasts m_{ack}.
- **Non-controller locations and messages.** There are three main non-controller locations l_{c_1}, l_{c_2}, and l_{res}, which broadcast a corresponding ready messages m_{rdy,c_1}, m_{rdy,c_2}, and $m_{\mathsf{rdy},\mathsf{res}}$. For each $i \in \{1, 2\}$ and each $x \in \{\mathsf{inc}, \mathsf{dec}\}$, there is an intermediate location l_{x,c_i}, which broadcasts m_{ack}.
- **Initial locations.** The initial locations are $l_{s_{\mathsf{init}}}$ and l_{res}.
- **Type-0 rules.** For each command $\mathsf{cmd} = (s, \mathsf{op}, s') \in \Delta$, the template has a rule $l_s \rightarrow l_{\mathsf{cmd}}$ with guard $m_{\mathsf{rdy},c_1} + m_{\mathsf{rdy},c_2} + m_{\mathsf{rdy},\mathsf{res}} = n - 1$. Additionally, if $\mathsf{op} = \mathsf{iszero}(c_i)$, then the guard is conjugated with $m_{\mathsf{rdy},c_i} = 0$. For each $i \in \{1, 2\}$, there is a rule $l_{\mathsf{res}} \rightarrow l_{\mathsf{inc},c_i}$ with guard $m_{\mathsf{inc},c_i} > 0$, and a rule $l_{c_i} \rightarrow l_{\mathsf{dec},c_i}$ with guard $m_{\mathsf{dec},c_i} > 0$.
- **Type-1 rules.** All type-1 rules have guard $m_{\mathsf{ack}} > 0$. For each command $\mathsf{cmd} = (s, \mathsf{op}, s') \in \Delta$, there is a rule $l_{\mathsf{cmd}} \rightarrow l_{s'}$. For each $i \in \{1, 2\}$, there are rules $l_{\mathsf{inc},c_i} \rightarrow l_{c_i}$, $l_{\mathsf{dec},c_i} \rightarrow l_{\mathsf{res}}$, $l_{c_i} \rightarrow l_{c_i}$, and $l_{\mathsf{res}} \rightarrow l_{\mathsf{res}}$.

The formula is $\varphi_\mathcal{C} := \neg(\varphi_{\mathsf{ctrl}} \wedge \varphi_{\mathsf{step}} \wedge \varphi_{\mathsf{halt}})$, where

$$\varphi_{\mathsf{ctrl}} := \forall r.\ \sum_{s \in S} \kappa(l_s, r) \leq 1, \qquad \varphi_{\mathsf{step}} := \forall r.\ \sum_{i \in \{1,2\}} \big(\kappa(l_{\mathsf{inc},c_i}, r) + \kappa(l_{\mathsf{dec},c_i}, r)\big) \leq 1,$$

and $\varphi_{\mathsf{halt}} := \neg(\forall r.\ \kappa(l_{s_{\mathsf{halt}}}, r) \leq 0)$. Here, φ_{ctrl} enforces that at most one process visits a controller location in each round, φ_{step} enforces that at most one process performs an increment/decrement move in each round, and φ_{halt} requires that the halting location is eventually visited.

B Details on Section 3

B.1 Receive-message abstraction

The system $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$ removes the per-process records of received messages and eliminates Receive actions, while deeming $\mathsf{Update}(i, \rho, r)$ enabled iff there exists a receive multiset $R : \mathcal{M} \times \mathbb{N} \rightarrow \mathbb{N}$ such that (i) $0 \leq R \leq \mathsf{NState}$ (pointwise), and (ii) if process i had received exactly R, the same update would be permitted in the concrete semantics. Equivalently, guards are evaluated *existentially* with respect to subset of the NState.

Projection. For a configuration $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$, we define the RMA projection $\alpha_{\mathsf{RMA}}(\mathsf{cfg}) = \langle \mathsf{PState}_{\mathsf{RMA}},\ \mathsf{NState}_{\mathsf{RMA}} \rangle$, where

$$\mathsf{PState}_{\mathsf{RMA}} : [\![1, \nu(n)]\!] \rightarrow \langle \mathsf{loc} : \mathcal{L}, \mathsf{rd} : \mathbb{N} \rangle$$

is given, for each process i, by

$$\mathsf{PState}_{\mathsf{RMA}}(i) = \langle \mathsf{loc} \mapsto \mathsf{PState}(i).\mathsf{loc},\ \mathsf{rd} \mapsto \mathsf{PState}(i).\mathsf{rd}\rangle,$$

and $\mathsf{NState}_{\mathsf{RMA}} = \mathsf{NState}$.

The RMA semantics $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu) = \langle S^{\mathsf{RMA}}_\nu, I^{\mathsf{RMA}}_\nu, Act^{\mathsf{RMA}}_\nu, Tr^{\mathsf{RMA}}_\nu\rangle$, where $S^{\mathsf{RMA}}_\nu = \{\alpha_{\mathsf{RMA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in S_\nu\}$, $I^{\mathsf{RMA}}_\nu = \{\alpha_{\mathsf{RMA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu\}$, the actions in Act^{RMA}_ν are exactly the $\mathsf{Update}(i, \rho, r)$ actions of the concrete system, and the transition relation Tr^{RMA}_ν is determined by the following enabledness and effect conditions. For $\mathsf{cfg}_{\mathsf{RMA}} = \langle \mathsf{PState}_{\mathsf{RMA}}, \mathsf{NState}_{\mathsf{RMA}}\rangle$ and $a = \mathsf{Update}(i, \rho, r)$,

$$\begin{aligned}\mathsf{enabled}_{\mathsf{RMA}}(\mathsf{cfg}_{\mathsf{RMA}}, a) = \mathsf{PState}_{\mathsf{RMA}}(i).(\mathsf{loc},\ \mathsf{rd}) = (\rho.\mathsf{frm}, r)\ \wedge \\ \exists R:\ 0 \le R \le \mathsf{NState}_{\mathsf{RMA}}\ \wedge\ R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu].\end{aligned}$$

The abstract effect updates the process's location and round exactly as in the corresponding concrete effect, and modifies $\mathsf{NState}_{\mathsf{RMA}}$ in the same way (see Appendix A.1). For $\pi = \mathsf{cfg}_0 a_1 \mathsf{cfg}_1 a_2 \ldots$ an execution of $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$, we define its hsc-trace exactly as in § 2.2.1 , i.e.,

$$\mathsf{hsc}(\pi)(\ell, r) = \big|\{\, k \mid a_k = \mathsf{Update}(i, \rho, r'),\ \rho.\mathsf{to} = \ell,\ r = r' + \rho.\mathsf{type}\,\}\big|.$$

Below we establish a forward simulation between the concrete semantics and the RMA semantics.

▶ **Lemma 15** (Forward simulation). *For every configuration* $\mathsf{cfg} \in S_\nu$*, if* $\mathsf{cfg} \in I_\nu$ *then* $\alpha_{\mathsf{RMA}}(\mathsf{cfg}) \in I^{\mathsf{RMA}}_\nu$*. Moreover if* $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu$*, the following hold:*

1. *If* $a = \mathsf{Recv}(i, \langle m, r\rangle)$*, then* $\alpha_{\mathsf{RMA}}(\mathsf{cfg}) = \alpha_{\mathsf{RMA}}(\mathsf{cfg}')$*.*
2. *If* $a = \mathsf{Update}(i, \rho, r)$*, then* $(\alpha_{\mathsf{RMA}}(\mathsf{cfg}), a, \alpha_{\mathsf{RMA}}(\mathsf{cfg}')) \in Tr^{\mathsf{RMA}}_\nu$*.*

Proof. 1. A receive transition changes only the received messages (rcvMsg) by process i; α_{RMA} forgets exactly this component thus $\alpha_{\mathsf{RMA}}(\mathsf{cfg}) = \alpha_{\mathsf{RMA}}(\mathsf{cfg}')$.
2. In the $\mathsf{RDTS}(\mathcal{T}, \nu)$, $a = \mathsf{Update}(i, \rho, r)$ enabledness requires at cfg:

$$\mathsf{PState}(i).(\mathsf{loc}, \mathsf{rd}) = (\rho.\mathsf{frm}, r) \text{ and } \mathsf{PState}(i).\mathsf{RecMsg}(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu].$$

By equation 1, $\mathsf{PState}(i).\mathsf{RecMsg} \le \mathsf{NState}$. Since the α_{RMA} projection preserves $\mathsf{PState}.(\mathsf{loc}, \mathsf{rd})$ and NState, $R = \mathsf{PState}(i).\mathsf{RecMsg}$ (at cfg) witnesses abstract enabledness. The abstract effect mirrors the concrete one on the components it keeps. Therefore $(\alpha_{\mathsf{RMA}}(\mathsf{cfg}), a, \alpha_{\mathsf{RMA}}(\mathsf{cfg}')) \in Tr^{\mathsf{RMA}}_\nu$. ◀

Next, we establish Refinement between the RMA semantics and the concrete semantics using Condition 3 from Def. 1.

▶ **Lemma 16** (Refinement). *For every template* $\mathcal{T}$*, there exists a mapping* $\beta_{\mathsf{RMA}} : S^{\mathsf{RMA}}_\nu \to S_\nu$ *such that:*

1. *For every* $\mathsf{cfg}_{\mathsf{RMA}} \in I^{\mathsf{RMA}}_\nu$*, we have that* $\beta_{\mathsf{RMA}}(\mathsf{cfg}_{\mathsf{RMA}}) \in I_\nu$*.*
2. *For every* $(\mathsf{cfg}_{\mathsf{RMA}}, a, \mathsf{cfg}'_{\mathsf{RMA}}) \in Tr^{\mathsf{RMA}}_\nu$*, there exists a concrete fragment* $\mathsf{cfg}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_n} \mathsf{cfg}_{n+1}$ *with* $\beta_{\mathsf{RMA}}(\mathsf{cfg}_{\mathsf{RMA}}) = \mathsf{cfg}_1$*,* $\beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}}) = \mathsf{cfg}_{n+1}$*, where for some* $1 \le j \le n$ *:* $a_j = a$ *and for all* $i \neq j$*,* a_i *is* Recv *action.*

Sketch. For an RMA configuration $\mathsf{cfg}_{\mathsf{RMA}} = \langle \mathsf{PState}_{\mathsf{RMA}}, \mathsf{NState}_{\mathsf{RMA}}\rangle$, define the refinement mapping $\beta_{\mathsf{RMA}}(\mathsf{cfg}_{\mathsf{RMA}}) = \mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState}\rangle$ with (i) $\mathsf{NState} = \mathsf{NState}_{\mathsf{RMA}}$, (ii) for process i $\mathsf{PState}(i).(\mathsf{loc}, \mathsf{rd}) = \mathsf{PState}_{\mathsf{RMA}}(i).(\mathsf{loc}, \mathsf{rd})$, and (iii) for process i and round r':

- $\mathsf{PState}(i).\mathsf{rcvMsg}(\cdot, r') = \mathsf{NState}_{\mathsf{RMA}}(\cdot, r')$ if $r' < \mathsf{PState}_{\mathsf{RMA}}(i).\mathsf{rd}$
- $\mathsf{PState}(i).\mathsf{rcvMsg}(\cdot, r') = 0$ if $r' > \mathsf{PState}_{\mathsf{RMA}}(i).\mathsf{rd}$
- $\mathsf{PState}(i).\mathsf{rcvMsg}(m, \mathsf{PState}_{\mathsf{RMA}}(i).\mathsf{rd}) = \mathsf{NState}_{\mathsf{RMA}}(m, \mathsf{PState}(i).\mathsf{rd})$ if $m \in M_i(\mathsf{cfg}_{\mathsf{RMA}})$ and 0 otherwise, where $M_i(\mathsf{cfg}_{\mathsf{RMA}})$ denotes the set of message variables that occur in the guards of type-0 rules leading (via type-0 rules) to $\mathsf{PState}_{\mathsf{RMA}}(i).\mathsf{loc}$.

To see that β_{RMA} satisfies Part 1, note that by Def. 1 no type-0 rule targets initial control locations. Consequently, for every $\mathsf{cfg}_{\mathsf{RMA}} \in I_\nu^{\mathsf{RMA}}$, it follows that $M_i(\mathsf{cfg}_{\mathsf{RMA}}) = \emptyset$. The key observation for Part 2 is that if an update $a = \mathsf{Update}(i, \rho, r)$ is enabled in $\mathsf{cfg}_{\mathsf{RMA}}$, then by Condition 3 in Def. 1, $\rho.\mathsf{guard}$ is monotone in the variables $M_i(\mathsf{cfg}_{\mathsf{RMA}})$. Hence, assuming all such messages have been received cannot disable the update. For variables outside $M_i(\mathsf{cfg}_{\mathsf{RMA}})$, receive counts are zero and thus an exact number of these can be received to execute a. Finally, after applying a additional messages can be received to reach $\beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}})$, for $(\mathsf{cfg}_{\mathsf{RMA}}, a, \mathsf{cfg}'_{\mathsf{RMA}}) \in Tr_\nu^{\mathsf{RMA}}$. Full details are given in Appendix C.1. ◀

B.1.1 Proof of Proposition 4

Proof. *Soundness.* By Lemma 15, for every execution π of $\mathsf{RDTS}(\mathcal{T}, \nu)$ there exits execution π_{RMA} of $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$ with the *same* sequence of Update actions. Hence $\mathsf{hsc}(\pi_{\mathsf{RMA}}) = \mathsf{hsc}(\pi)$. *Completeness.* By Lemma 16, for every execution π_{RMA} of $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$ there exits execution π of $\mathsf{RDTS}(\mathcal{T}, \nu)$ with the *same* sequence of Update actions, interleaved with Recv actions. Since Recv actions are stuttering steps for hsc-trace, we obtain $\mathsf{hsc}(\pi) = \mathsf{hsc}(\pi_{\mathsf{RMA}})$. ◀

B.2 Process-identity abstraction

In the system $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$, the $\mathsf{PState}_{\mathsf{RMA}}$ component of the RMA configuration is abstracted: instead of recording the exact location and round of each process, it records only the *counts* of processes at each location and round.

Projection. For a RMA configuration $\mathsf{cfg}_{\mathsf{RMA}} = \langle \mathsf{PState}_{\mathsf{RMA}}, \mathsf{NState}_{\mathsf{RMA}} \rangle$, we define the PIA projection $\alpha_{\mathsf{PIA}}(\mathsf{cfg}_{\mathsf{RMA}}) = \big\langle \mathsf{PState}_{\mathsf{PIA}}, \mathsf{NState}_{\mathsf{PIA}} \big\rangle$, where $\mathsf{PState}_{\mathsf{PIA}} : \mathcal{L} \times \mathbb{N} \to \mathbb{N}$ is given by

$$\mathsf{PState}_{\mathsf{PIA}}(\ell, r) \;=\; \big|\{\, i \in [\![1, N]\!] : \mathsf{PState}_{\mathsf{RMA}}(i).(\mathsf{loc}, \mathsf{rd}) = (\ell, r) \,\}\big|,$$

and $\mathsf{NState}_{\mathsf{PIA}} = \mathsf{NState}_{\mathsf{RMA}}$.

The PIA semantics

$$\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu) = \langle S_\nu^{\mathsf{PIA}}, I_\nu^{\mathsf{PIA}}, Act_\nu^{\mathsf{PIA}}, Tr_\nu^{\mathsf{PIA}} \rangle,$$

where $S_\nu^{\mathsf{PIA}} = \{\, \alpha_{\mathsf{PIA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in S_\nu^{\mathsf{RMA}} \,\}$, $I_\nu^{\mathsf{PIA}} = \{\, \alpha_{\mathsf{PIA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu^{\mathsf{RMA}} \,\}$, the action set $Act_\nu^{\mathsf{PIA}} = \{\, \mathsf{Update}(\rho, r) \mid \rho \in \mathsf{Rules},\ r \in \mathbb{N} \,\}$ (no process ids), and the transition relation Tr_ν^{PIA} is determined by the following enabledness and effect conditions.

For $\mathsf{cfg}_{\mathsf{PIA}} = \alpha_{\mathsf{PIA}}(\mathsf{cfg}_{\mathsf{RMA}})$ for some RMA configuration $\mathsf{cfg}_{\mathsf{RMA}}$, the abstract action $a = \mathsf{Update}(\rho, r)$ is enabled iff there exists a process i at $\mathsf{cfg}_{\mathsf{PIA}}$ $(\rho.\mathsf{frm}, r)$ in $\mathsf{cfg}_{\mathsf{RMA}}$ such that $\mathsf{Update}(i, \rho, r)$ is enabled in $\mathsf{cfg}_{\mathsf{RMA}}$. Equivalently, for $\mathsf{cfg}_{\mathsf{PIA}} = \langle \mathsf{PState}_{\mathsf{PIA}}, \mathsf{NState}_{\mathsf{PIA}} \rangle$,

$$\begin{aligned}\mathsf{enabled}_{\mathsf{PIA}}\big(\mathsf{cfg}_{\mathsf{PIA}}, a\big) = {} & \mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{frm},\, r) > 0 \;\;\wedge \\ & \exists R :\; 0 \le R \le \mathsf{NState}_{\mathsf{PIA}} \;\wedge\; R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu].\end{aligned}$$

Upon executing a at $\mathsf{cfg}_{\mathsf{PIA}}$, the $\mathsf{PState}_{\mathsf{PIA}}$ is updated to reflect move of one process from $(\rho.\mathsf{frm}, r)$ to $(\rho.\mathsf{to},\, r + \rho.\mathsf{type})$, and $\mathsf{NState}_{\mathsf{PIA}}$ is updated exactly as in the RMA semantics (see Appendix A.1). Formally,

$\mathsf{effect}_{\mathsf{PIA}}(\mathsf{cfg}_{\mathsf{PIA}},\, \mathsf{Update}(\rho, r)) = \langle \mathsf{PState}'_{\mathsf{PIA}},\, \mathsf{NState}'_{\mathsf{PIA}} \rangle$, where:

- $\mathsf{PState}'_{\mathsf{PIA}}(\ell, r') = \mathsf{PState}_{\mathsf{PIA}}(\ell, r') - 1$ if $(\ell, r') = (\rho.\mathsf{frm}, r)$; $\mathsf{PState}'_{\mathsf{PIA}}(\ell, r') = \mathsf{PState}_{\mathsf{PIA}}(\ell, r') + 1$ if $(\ell, r') = (\rho.\mathsf{to},\ r + \rho.\mathsf{type})$; and $\mathsf{PState}'_{\mathsf{PIA}}(\ell, r') = \mathsf{PState}_{\mathsf{PIA}}(\ell, r')$ otherwise.
- $\mathsf{NState}'_{\mathsf{PIA}}(m', r') = \mathsf{NState}_{\mathsf{PIA}}(m', r') + 1$ if $(m', r') = (\mathsf{Bcast}(\rho.\mathsf{to}),\ r + \rho.\mathsf{type})$; and $\mathsf{NState}'_{\mathsf{PIA}}(m', r') = \mathsf{NState}_{\mathsf{PIA}}(m', r')$ otherwise.

For $\pi = \mathsf{cfg}_0 a_1 \mathsf{cfg}_1 a_2 \ldots$ an execution of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$, define its hsc-trace analogously to as defined in § 2.2.1, except that here updates no longer carry process identifiers:

$$\mathsf{hsc}(\pi)(\ell, r) \;=\; \left|\{\, k \mid a_k = \mathsf{Update}(\rho, r'),\ \rho.\mathsf{to} = \ell,\ r = r' + \rho.\mathsf{type} \,\}\right|.$$

Define the relabeling $\delta : Act_\nu^{\mathsf{RMA}} \to Act_\nu^{\mathsf{PIA}}$ by $\delta(\mathsf{Update}(i, \rho, r)) = \mathsf{Update}(\rho, r)$. We now establish an action-based bisimulation, modulo δ, between the RMA and PIA semantics.

▶ **Lemma 17** (Action-based bisimulation). **1.** $\forall\, \mathsf{cfg} \in I_\nu^{\mathsf{RMA}} : \alpha_{\mathsf{PIA}}(\mathsf{cfg}) \in I_\nu^{\mathsf{PIA}}$ *and* $\forall\, \mathsf{cfg}_{\mathsf{PIA}} \in I_\nu^{\mathsf{PIA}} : \exists\, \mathsf{cfg} \in I_\nu^{\mathsf{RMA}}$ *with* $\alpha_{\mathsf{PIA}}(\mathsf{cfg}) = \mathsf{cfg}_{\mathsf{PIA}}$.

2. *For all* $\mathsf{cfg} \in S_\nu^{\mathsf{RMA}}$ *it holds:*

$\Rightarrow$ *If* $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{RMA}}$, *then* $(\alpha_{\mathsf{PIA}}(\mathsf{cfg}), \delta(a), \alpha_{\mathsf{PIA}}(\mathsf{cfg}')) \in Tr_\nu^{\mathsf{PIA}}$.

$\Leftarrow$ *If* $(\alpha_{\mathsf{PIA}}(\mathsf{cfg}), a', \mathsf{cfg}'_{\mathsf{PIA}}) \in Tr_\nu^{\mathsf{PIA}}$, *then* $\exists\, \mathsf{cfg}' \in S_\nu^{\mathsf{RMA}}, a \in Act_\nu^{\mathsf{RMA}} : (\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{RMA}}$, $\alpha_{\mathsf{PIA}}(\mathsf{cfg}') = \mathsf{cfg}'_{\mathsf{PIA}}$, *and* $\delta(a) = a'$.

Proof. By definition of $I_\nu^{\mathsf{PIA}} = \{\alpha_{\mathsf{PIA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu^{\mathsf{RMA}}\}$, thus we have part 1 of the lemma.

($\Rightarrow$) Let $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{RMA}}$ with $a = \mathsf{Update}(i, \rho, r)$. Enabledness of a in RMA semantics requires that in cfg:

- $\mathsf{PState}_{\mathsf{RMA}}(i).(\mathsf{loc}, \mathsf{rd}) = (\rho.\mathsf{frm}, r)$, hence $\mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{frm}, r) > 0$ in $\alpha_{\mathsf{PIA}}(\mathsf{cfg})$.
- there exists R with $0 \le R \le \mathsf{NState}_{\mathsf{RMA}}$ and $R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu]$.

Since $\mathsf{NState}_{\mathsf{PIA}} = \mathsf{NState}_{\mathsf{RMA}}$ under α_{PIA}, the same R witnesses enabledness of $a' = \delta(a) = \mathsf{Update}(\rho, r)$ in $\alpha_{\mathsf{PIA}}(\mathsf{cfg})$. For the effect, a moves one process from $(\rho.\mathsf{frm}, r)$ to $(\rho.\mathsf{to}, r{+}\rho.\mathsf{type})$ and (if applicable) broadcasts to NState. Thus $\alpha_{\mathsf{PIA}}(\mathsf{cfg}')$ decreases $\mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{frm}, r)$ by 1, increases $\mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{to}, r + \rho.\mathsf{type})$ by 1, and updates NState exactly as in $\mathsf{effect}_{\mathsf{PIA}}$. Hence $(\alpha_{\mathsf{PIA}}(\mathsf{cfg}), a', \alpha_{\mathsf{PIA}}(\mathsf{cfg}')) \in Tr_\nu^{\mathsf{PIA}}$.

($\Leftarrow$) Let $(\alpha_{\mathsf{PIA}}(\mathsf{cfg}), a', \mathsf{cfg}'_{\mathsf{PIA}}) \in Tr_\nu^{\mathsf{PIA}}$ with $a' = \mathsf{Update}(\rho, r)$. Enabledness of a' in PIA semantics requires that in $\alpha_{\mathsf{PIA}}(\mathsf{cfg})$:

- $\mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{frm}, r) > 0$, hence there exists i with $\mathsf{PState}_{\mathsf{RMA}}(i).(\mathsf{loc}, \mathsf{rd}) = (\rho.\mathsf{frm}, r)$ in cfg.
- there exists R with $0 \le R \le \mathsf{NState}_{\mathsf{PIA}}$ and $R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu]$.

Since $\mathsf{NState}_{\mathsf{PIA}} = \mathsf{NState}_{\mathsf{RMA}}$ under α_{PIA}, the same R witnesses enabledness of $a = \mathsf{Update}(i, \rho, r)$ in cfg. Let cfg' be the RMA configuration upon executing $a = \mathsf{Update}(i, \rho, r)$ from cfg. Executing a moves one process from $(\rho.\mathsf{frm}, r)$ to $(\rho.\mathsf{to}, r + \rho.\mathsf{type})$ and (if applicable) broadcasts to NState. Thus $\alpha_{\mathsf{PIA}}(\mathsf{cfg}')$ decreases $\mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{frm}, r)$ by 1, increases $\mathsf{PState}_{\mathsf{PIA}}(\rho.\mathsf{to}, r{+}\rho.\mathsf{type})$ by 1, and updates NState exactly as in $\mathsf{effect}_{\mathsf{PIA}}$. Hence $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{RMA}}$ with $\delta(a) = a'$ and $\alpha_{\mathsf{PIA}}(\mathsf{cfg}') = \mathsf{cfg}'_{\mathsf{PIA}}$. ◀

B.2.1 Proof of Proposition 5

Proof. Let $\pi = \mathsf{cfg}_1 a_1 \mathsf{cfg}_2 a_2 \cdots$ be an execution of $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$. Define its projection

$$\Delta(\pi) = \alpha_{\mathsf{RMA}}(\mathsf{cfg}_1)\ \delta(a_1)\ \alpha_{\mathsf{RMA}}(\mathsf{cfg}_2)\ \delta(a_2)\ \ \cdots\ .$$

By the direction ($\Rightarrow$) of Lem. 17, $\Delta(\pi)$ is an execution of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$. Moreover $\mathsf{hsc}(\Delta(\pi)) = \mathsf{hsc}(\pi)$. Thus, soundness follows. For completeness, apply direction ($\Leftarrow$) of Lem. 17: for every execution π_{PIA} of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ there exists an execution π of $\mathsf{RDTS}^{\mathsf{RMA}}(\mathcal{T}, \nu)$ such that $\Delta(\pi) = \pi_{\mathsf{PIA}}$. ◀

B.3 Semi-synchronous restriction

B.3.1 SSR

The system $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu) = \langle S_\nu^{\mathsf{SSR}}, I_\nu^{\mathsf{SSR}}, Act_\nu^{\mathsf{SSR}}, Tr_\nu^{\mathsf{SSR}}\rangle$, where $S_\nu^{\mathsf{SSR}} = S_\nu^{\mathsf{PIA}}$, $I_\nu^{\mathsf{SSR}} = I_\nu^{\mathsf{PIA}}$, $Act_\nu^{\mathsf{SSR}} = Act_\nu^{\mathsf{PIA}}$, and

$$Tr_\nu^{\mathsf{SSR}} = \{(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{PIA}} \mid \mathsf{tgt}(a) \geq r_{\mathsf{max}}(\mathsf{cfg})\}.$$

Equivalently, enabledness and effect are given by

$$\mathsf{enabled}_{\mathsf{SSR}}(\mathsf{cfg}, a) = \mathsf{enabled}_{\mathsf{PIA}}(\mathsf{cfg}, a) \wedge \mathsf{tgt}(a) \geq r_{\mathsf{max}}(\mathsf{cfg})$$

and $\mathsf{effect}_{\mathsf{SSR}}(\mathsf{cfg}, a) = \mathsf{effect}_{\mathsf{PIA}}(\mathsf{cfg}, a)$.

The hsc-trace of executions of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu)$ is defined identically to that of executions of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$ (see Appendix B.2). Since the action sets of the two systems coincide, this is well defined.

Recall that every rule $\rho \in \mathsf{Rules}$ satisfies $\rho.\mathsf{type} \leq b$, where $b \in \mathbb{N}$ is the *jump bound* of the template $\mathcal{T}$. Hence, in a configuration cfg, no update $\mathsf{Update}(\rho, r)$ with $r < r_{\mathsf{max}}(\mathsf{cfg}) - b$ is enabled under SSR. Consequently, in configuration cfg all counters below $r_{\mathsf{max}}(\mathsf{cfg}) - b$ can be discarded, while counters above $r_{\mathsf{max}}(\mathsf{cfg})$ are zero by definition. Thus the relevant window of any SSR configuration cfg is $[\![r_{\mathsf{max}}(\mathsf{cfg}) - b, r_{\mathsf{max}}(\mathsf{cfg})]\!]$, of fixed size $b + 1$, uniformly bounded across all configurations. For an update action $a = \mathsf{Update}(\rho, r)$, let $\mathsf{tgt}(a) = r + \rho.\mathsf{type}$.

An execution $\pi = \mathsf{cfg}_1\, a_1\, \mathsf{cfg}_2\, a_2 \cdots$ of the $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$ is called *semi-synchronous* if its target sequence $\mathsf{tgt}(a_1)\mathsf{tgt}(a_2)\ldots$ is non-decreasing. This notion precisely characterizes executions of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu)$, as formalized and proved in following lemma.

▶ **Lemma 18.** *An execution* $\pi = \mathsf{cfg}_1 a_1 \mathsf{cfg}_2 a_2 \ldots$ *of* $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$ *is semi synchronous iff it is also an execution of* $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu)$.

Proof. To prove the lemma we first prove following claim:

▷ Claim 19. For any $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{PIA}}$, $r_{\mathsf{max}}(\mathsf{cfg}') = \max\big(r_{\mathsf{max}}(\mathsf{cfg}), \mathsf{tgt}(a)\big)$.

Proof of claim. By construction of PIA an action $a = \mathsf{Update}(\rho, r)$ modifies only counters at rounds r and $\mathsf{tgt}(a) > r$. If $\mathsf{tgt}(a) < r_{\mathsf{max}}(\mathsf{cfg})$, then $r_{\mathsf{max}}(\mathsf{cfg}') = r_{\mathsf{max}}(\mathsf{cfg})$. If $\mathsf{tgt}(a) \geq r_{\mathsf{max}}(\mathsf{cfg})$, the transition increments the $\rho.\mathsf{to}$ location counter at round $\mathsf{tgt}(a)$, making it positive, so $r_{\mathsf{max}}(\mathsf{cfg}') = \mathsf{tgt}(a)$. In both cases the claim holds.

Now suppose π is semi-synchronous execution of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T},\nu)$, i.e., $\mathsf{tgt}(a_1) \leq \mathsf{tgt}(a_2) \leq \cdots$. Since every $\mathsf{cfg} \in I_\nu^{\mathsf{PIA}}$ satisfies $r_{\mathsf{max}}(\mathsf{cfg}) = 1$, we have $\mathsf{tgt}(a_1) \geq r_{\mathsf{max}}(\mathsf{cfg}_1)$. By the claim $r_{\mathsf{max}}(\mathsf{cfg}_2) = \mathsf{tgt}(a_1)$. Inductively, if $r_{\mathsf{max}}(\mathsf{cfg}_i) = \mathsf{tgt}(a_{i-1})$ then $\mathsf{tgt}(a_i) \geq \mathsf{tgt}(a_{i-1}) = r_{\mathsf{max}}(\mathsf{cfg}_i)$, so the SSR enabledness restriction holds at step i. Thus π is an execution of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu)$.

Conversely, suppose π is an execution of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T},\nu)$. Then by definition $\mathsf{tgt}(a_i) \geq r_{\mathsf{max}}(\mathsf{cfg}_i)$ for all i. The above claim gives $r_{\mathsf{max}}(\mathsf{cfg}_{i+1}) = \mathsf{tgt}(a_i)$, which implies $\mathsf{tgt}(a_{i+1}) \geq r_{\mathsf{max}}(\mathsf{cfg}_{i+1}) = \mathsf{tgt}(a_i)$. Therefore $\mathsf{tgt}(a_1) \leq \mathsf{tgt}(a_2) \leq \cdots$, i.e., the target sequence is non-decreasing, so π is semi-synchronous. ◀

To prove Lemma 6 we use the following claim whose proof follows from the definition of Tr_ν^{PIA} and can be found in Appendix C.2.

▷ Claim 20 (Commutativity). Let $a = \mathsf{Update}(\rho, r)$ and $a' = \mathsf{Update}(\rho', r')$, and suppose we have $(\mathsf{cfg}, a, \mathsf{cfg}^*), (\mathsf{cfg}^*, a', \mathsf{cfg}') \in Tr_\nu^{\mathsf{PIA}}$. If $\mathsf{tgt}(a) \neq r'$, then there exists $\mathsf{cfg}^\dagger$ with $(\mathsf{cfg}, a', \mathsf{cfg}^\dagger), (\mathsf{cfg}^\dagger, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{PIA}}$.

B.3.2 Proof of Lemma 6.

Proof. Let $\pi = \mathsf{cfg}_1 a_1 \mathsf{cfg}_2 a_2 \cdots$ be an execution of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$, where each $a_i = \mathsf{Update}(\rho_i, r_i)$ and $\mathsf{tgt}(a_i) = r_i + \rho_i.\mathsf{type}$. We transform π into a semi-synchronous execution by repeatedly commuting out-of-order updates until the sequence of target rounds becomes non-decreasing.

Inductive reordering. Write $\pi_{-1} = \pi$. For $r = 0, 1, \ldots$ define π_r from π_{r-1} as follows. Scan π_{r-1} left-to-right and identify (in increasing order of their indices) all updates whose target is r. There are finitely many such updates by finiteness of processes and acyclicity of type 0 updates. For each such update, repeatedly apply the Commutativity Claim to swap it left over any immediately preceding update whose target is $> r$, stopping once the preceding update (if any) has target $\leq r$. Perform these swaps in the order the target-r updates are encountered so their relative order is preserved. Denote the resulting execution by π_r.

Invariants. For every $r \geq 0$ there exists an index m_r such that: (i) the prefix execution $\pi_r[1..m_r]$ consists *exactly* of all updates with target $\leq r$, arranged in non-decreasing order of their targets, and for each fixed target the original relative order is preserved; (ii) every update in the suffix $\pi_r[m_r + 1..\,]$ has target $> r$; (iii) π_r is a valid PIA execution starting from cfg_1; and (iv) π_r has the same multiset of update actions as π. Items (iii)–(iv) hold because each step uses only swaps justified by the Commutativity Claim; items (i)–(ii) hold by construction.

Limit execution. For $n \in \mathbb{N}$, let r_0 be the maximum target round among the first n updates of π. Then for all $r > r_0$, the executions π_r and π_{r_0} agree on the prefix of length n. thus the prefix of length n stabilizes. Consequently, the pointwise limit $\pi' = \lim_{r \to \infty} \pi_r$ exists and is a valid execution, obtained by taking, for each n, the stabilized prefix of length n. ◀

B.3.3 Proof of Proposition 7

Proof. By Lemma 6, for every execution π of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$, there exists semi-synchronous execution π' of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ such that $\mathsf{hsc}(\pi') = \mathsf{hsc}(\pi)$. Furthermore, by Lemma 18, the executions of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)$ are exactly the semi-synchronous executions of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$. Thus the set of hsc-traces of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ and $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)$ is identical. ◀

B.3.4 Strong SSR

The SSR restriction is sufficient to obtain a reduced finite-counter semantics. However, we can further decrease non-determinism on interleaving by introducing a stronger restriction while preserving soundness.

Let θ be a total order on the set of jump rules $\{\rho \in \mathsf{Rules} \mid \rho.\mathsf{type} > 0\}$. An execution $\pi = \mathsf{cfg}_0, a_1, \mathsf{cfg}_1, a_2 \ldots$ of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ with $a_i = \mathsf{Update}(\rho_i, r_i)$ is called *θ-strong semi-synchronous* if: (i) it is semi-synchronous, i.e., for all $i < j$, $\mathsf{tgt}(a_i) \leq \mathsf{tgt}(a_j)$, and (ii) whenever $\mathsf{tgt}(a_i) = \mathsf{tgt}(a_j)$ and $\rho_j.\mathsf{type} > 0$, we have $\rho_i.\mathsf{type} > 0$ and $\rho_i \leq_\theta \rho_j$. Clearly, every θ-strong semi-synchronous execution is semi-synchronous. The converse is also true:

▶ **Lemma 21** (Soundness of strong SSR). *Let θ be a total order on jump rules. For every semi-synchronous execution π of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ there exists a θ-strong semi-synchronous execution π' of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ that starts from the same initial configuration as π and contains the same multiset of update actions.*

Proof. The proof follows from the Commutativity claim. In a semi-synchronous execution, within the fragment whose updates targets round r, a local update targeting round r can always be commuted past a jump update targeting r, so all jumps can be placed before locals.

Furthermore, jump updates targeting the same round commute among themselves. Thus, by reordering them according to θ, we obtain the desired execution. ◀

The corresponding strong SSR semantics extends the SSR semantics. Informally, configurations additionally record the last executed rule, and enabledness is further restricted as follows: (i) if the last rule was local, then no jump to the current frontier round is permitted; (ii) if the last rule was a jump, then no jump to the current frontier round with a smaller (w.r.t. θ) rule is permitted. Strong SSR substantially reduces nondeterminism: (i) it enforces that all jumps to a round r occur before any locals at r, and (ii) it quotients away permutations of jump updates targeting the same round by fixing their order according to θ. Since soundness holds for any choice of θ, the strong SSR semantics can be viewed as if all processes *synchronously jump* to round r, followed by an *asynchronous execution of local updates* at r, before moving on to the next round.

B.4 Bounded-window abstraction

Recall that in SSR the relevant window of a configuration cfg is $[\![r_{\mathsf{max}}(\mathsf{cfg}) - b,\ r_{\mathsf{max}}(\mathsf{cfg})]\!]$, of fixed size $b{+}1$. The bounded-window abstraction (BWA) drops all counters outside this window and encodes the remainder as a sliding window of width $b{+}1$ anchored at the frontier $r_{\mathsf{max}}(\mathsf{cfg})$; that is, it stores $r_{\mathsf{max}}(\mathsf{cfg})$ together with the counters for rounds $r_{\mathsf{max}}(\mathsf{cfg}) - b \leq r \leq r_{\mathsf{max}}(\mathsf{cfg})$.

Projection. For a SSR configuration $\mathsf{cfg}_{\mathsf{SSR}} = \langle \mathsf{PState}_{\mathsf{SSR}},\ \mathsf{NState}_{\mathsf{SSR}} \rangle$, we define the SSR projection

$$\alpha_{\mathsf{BWA}}(\mathsf{cfg}_{\mathsf{SSR}}) = \big\langle \mathsf{PState}_{\mathsf{BWA}},\ \mathsf{NState}_{\mathsf{BWA}},\ r_{\mathsf{max}} \big\rangle,$$

where $\mathsf{PState}_{\mathsf{BWA}} : \mathcal{L} \times [\![0, b]\!] \to \mathbb{N}$ is given by $\mathsf{PState}_{\mathsf{BWA}}(\ell, d) = \mathsf{PState}_{\mathsf{SSR}}(\ell, r_{\mathsf{max}}(\mathsf{cfg}) - d)$, $\mathsf{NState}_{\mathsf{BWA}} : \mathcal{M} \times [\![0, b]\!] \to \mathbb{N}$ is given by $\mathsf{NState}_{\mathsf{BWA}}(m, d) = \mathsf{NState}_{\mathsf{SSR}}(m, r_{\mathsf{max}}(\mathsf{cfg}) - d)$, and $r_{\mathsf{max}} \in \mathbb{N}$ is given by $r_{\mathsf{max}} = r_{\mathsf{max}}(\mathsf{cfg}_{\mathsf{SSR}})$.

The BWA semantics

$$\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T}, \nu) = \langle S_\nu^{\mathsf{BWA}}, I_\nu^{\mathsf{BWA}}, Act_\nu^{\mathsf{BWA}}, Tr_\nu^{\mathsf{BWA}} \rangle,$$

where $S_\nu^{\mathsf{BWA}} = \{\alpha_{\mathsf{BWA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in S_\nu^{\mathsf{SSR}}\}$, $I_\nu^{\mathsf{BWA}} = \{\alpha_{\mathsf{BWA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu^{\mathsf{SSR}}\}$, and $Act_\nu^{\mathsf{BWA}} = Act_\nu^{\mathsf{SSR}}$. The transition relation is the existential lift of SSR semantics: $(\mathsf{cfg}_{\mathsf{BWA}}, a, \mathsf{cfg}'_{\mathsf{BWA}}) \in Tr_\nu^{\mathsf{BWA}}$ iff there exist $\mathsf{cfg}_{\mathsf{SSR}}, \mathsf{cfg}'_{\mathsf{SSR}} \in S_\nu^{\mathsf{SSR}}$ with

$$\alpha_{\mathsf{BWA}}(\mathsf{cfg}_{\mathsf{SSR}}) = \mathsf{cfg}_{\mathsf{BWA}},\ \alpha_{\mathsf{BWA}}(\mathsf{cfg}'_{\mathsf{SSR}}) = \mathsf{cfg}'_{\mathsf{BWA}},\ \text{and } (\mathsf{cfg}_{\mathsf{SSR}}, a, \mathsf{cfg}'_{\mathsf{SSR}}) \in Tr_\nu^{\mathsf{SSR}}.$$

The hsc-trace of executions of $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T}, \nu)$ is defined identically to that of executions of $\mathsf{RDTS}^{\mathsf{PIA}}(\mathcal{T}, \nu)$ (see Appendix B.2). Since the action sets of the two systems coincide, this is well defined.

Before establishing action based bisimulation between BWA and SSR semantics, we establish following property of projection α_{BWA} :

▷ Claim 22. Let $\mathsf{cfg}, \hat{\mathsf{cfg}} \in S_\nu^{\mathsf{SSR}}$ and $a \in Act_\nu^{\mathsf{SSR}}$. If $\alpha_{\mathsf{BWA}}(\mathsf{cfg}) = \alpha_{\mathsf{BWA}}(\hat{\mathsf{cfg}})$, then in SSR semantics:

1. a is enabled at cfg iff it is enabled at $\hat{\mathsf{cfg}}$.
2. if a is enabled, then $\alpha_{\mathsf{BWA}}\big(\mathsf{effect}_{\mathsf{SSR}}(\mathsf{cfg}, a)\big) = \alpha_{\mathsf{BWA}}\big(\mathsf{effect}_{\mathsf{SSR}}(\hat{\mathsf{cfg}}, a)\big)$.

Proof. Assume $\alpha_{\mathsf{BWA}}(\mathsf{cfg}) = \alpha_{\mathsf{BWA}}(\hat{\mathsf{cfg}})$. Then $r_{\mathsf{max}}(\mathsf{cfg}) = r_{\mathsf{max}}(\hat{\mathsf{cfg}})$ and, for all $r \geq r_{\mathsf{max}}(\mathsf{cfg}) - b$, the round r counters, $\mathsf{PState}(\ell, r)$ and $\mathsf{NState}(m, r)$ coincide in cfg and $\hat{\mathsf{cfg}}$.

1. For any $a = \mathsf{Update}(\rho, r)$, a is enabled at cfg implies $r \geq r_{\mathsf{max}}(\mathsf{cfg}) - b$, Since the enabledness of a depends only on the round r counters, a is enabled in cfg iff it is enabled in $\hat{\mathsf{cfg}}$.
2. When enabled, a updates only round r and round $\mathsf{tgt}(a)$ counters and since $\mathsf{tgt}(a) \geq r$ for all rounds $r' \geq r_{\mathsf{max}}(\mathsf{cfg}) - b$ the round r' counters coincide in $\mathsf{cfg}' = \mathsf{effect}_{\mathsf{SSR}}(\mathsf{cfg}, a)$ and $\hat{\mathsf{cfg}}' = \mathsf{effect}_{\mathsf{SSR}}(\hat{\mathsf{cfg}}, a)$. Moreover

$$r_{\mathsf{max}}(\mathsf{cfg}')) = \max\{r_{\mathsf{max}}(\mathsf{cfg}), \mathsf{tgt}(a)\} = \max\{r_{\mathsf{max}}(\hat{\mathsf{cfg}}), \mathsf{tgt}(a)\} = r_{\mathsf{max}}(\hat{\mathsf{cfg}}').$$

Thus cfg' and $\hat{\mathsf{cfg}}'$ agree on r_{max} and the round r' counters for $r' \geq r_{\mathsf{max}}(\mathsf{cfg}')) - b \geq r_{\mathsf{max}}(\mathsf{cfg}) - b$. Therefore the updated window counters match as well, showing $\alpha_{\mathsf{BWA}}(\mathsf{cfg}') = \alpha_{\mathsf{BWA}}(\hat{\mathsf{cfg}}')$. ◀

▶ **Lemma 23** (Action-based bisimulation). *It holds that:*

1. $\forall\ \mathsf{cfg} \in I_\nu^{\mathsf{SSR}} : \alpha_{\mathsf{BWA}}(\mathsf{cfg}) \in I_\nu^{\mathsf{BWA}}$ *and* $\forall\ \mathsf{cfg}_{\mathsf{BWA}} \in I_\nu^{\mathsf{BWA}} : \exists\, \mathsf{cfg} \in I_\nu^{\mathsf{SSR}}$ *with* $\alpha_{\mathsf{BWA}}(\mathsf{cfg}) = \mathsf{cfg}_{\mathsf{BWA}}$.
2. *For all* $\mathsf{cfg} \in S_\nu^{\mathsf{SSR}}$ *it holds:*
   - $\Rightarrow$ *If* $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{SSR}}$*, then* $(\alpha_{\mathsf{BWA}}(\mathsf{cfg}), a, \alpha_{\mathsf{BWA}}(\mathsf{cfg}')) \in Tr_\nu^{\mathsf{BWA}}$.
   - $\Leftarrow$ *If* $(\alpha_{\mathsf{BWA}}(\mathsf{cfg}), a, \mathsf{cfg}'_{\mathsf{BWA}}) \in Tr_\nu^{\mathsf{BWA}}$*, then* $\exists\ \mathsf{cfg}' \in S_\nu^{\mathsf{SSR}} : (\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{SSR}}$ *and* $\alpha_{\mathsf{BWA}}(\mathsf{cfg}') = \mathsf{cfg}'_{\mathsf{BWA}}$.

Proof. (1) holds immediately since $I_\nu^{\mathsf{BWA}} = \{\alpha_{\mathsf{BWA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu^{\mathsf{SSR}}\}$. Direction ($\Rightarrow$) holds by the definition of Tr_ν^{BWA} as an existential lift of Tr_ν^{SSR}.

For direction ($\Leftarrow$), let $\mathsf{cfg} \in S_\nu^{\mathsf{SSR}}$ with $\alpha_{\mathsf{BWA}}(\mathsf{cfg}) = \mathsf{cfg}_{\mathsf{BWA}}$ and assume $(\mathsf{cfg}_{\mathsf{BWA}}, a, \mathsf{cfg}'_{\mathsf{BWA}}) \in Tr_\nu^{\mathsf{BWA}}$. Then by definition of Tr_ν^{BWA}, there exist $\mathsf{cfg}^\dagger, \mathsf{cfg}^{\dagger\prime} \in S_\nu^{\mathsf{SSR}}$ with $(\mathsf{cfg}^\dagger, a, \mathsf{cfg}^{\dagger\prime}) \in Tr_\nu^{\mathsf{SSR}}$, $\alpha_{\mathsf{BWA}}(\mathsf{cfg}^\dagger) = \mathsf{cfg}_{\mathsf{BWA}}$, and $\alpha_{\mathsf{BWA}}(\mathsf{cfg}^{\dagger\prime}) = \mathsf{cfg}'_{\mathsf{BWA}}$. Since cfg and $\mathsf{cfg}^\dagger$ have the same BWA projection, above Claim implies that a is also enabled at cfg and that

$$\alpha_{\mathsf{BWA}}(\mathsf{effect}_{\mathsf{SSR}}(\mathsf{cfg}, a)) = \alpha_{\mathsf{BWA}}(\mathsf{effect}_{\mathsf{SSR}}(\mathsf{cfg}^\dagger, a)) = \mathsf{cfg}'_{\mathsf{BWA}}.$$

Taking $\mathsf{cfg}' = \mathsf{effect}_{\mathsf{SSR}}(\mathsf{cfg}, a)$ proves the direction. ◀

B.4.1 Proof of Proposition 8

Proof. By Lem. 23 α_{BWA} is an action-based bisimulation between $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)$ and $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T}, \nu)$. Thus, for every execution π of $\mathsf{RDTS}^{\mathsf{SSR}}(\mathcal{T}, \nu)$ there exists an execution π_{BWA} of $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T}, \nu)$ with the same sequence of action labels, and hence $\mathsf{hsc}(\pi_{\mathsf{BWA}}) = \mathsf{hsc}(\pi)$. The converse direction follows symmetrically. ◀

B.5 History-record extension

Formally, $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T}, \nu, \varphi) = \langle S_\nu^{\mathsf{HRE}}, I_\nu^{\mathsf{HRE}}, Act_\nu^{\mathsf{HRE}}, Tr_\nu^{\mathsf{HRE}} \rangle$, where S_ν^{HRE} is the set of HRE configurations, I_ν^{HRE} the set of initial HRE configurations, $Act_\nu^{\mathsf{HRE}} = Act_\nu^{\mathsf{BWA}}$, and Tr_ν^{HRE}, the transition relation obtained by extending Tr_ν^{BWA} with the corresponding updates to the history record.

HRE **Configurations.** An HRE configuration is of the form

$$\mathsf{cfg}_{\mathsf{HRE}} = \langle \mathsf{cfg}_{\mathsf{BWA}}, \mathsf{local}, \mathsf{cumul} \rangle$$

with $\mathsf{cfg}_{\mathsf{BWA}} \in S_\nu^{\mathsf{BWA}}$ and $\mathsf{local}, \mathsf{cumul} : \mathcal{X} \to \mathbb{N}$. For $\ell \in \mathcal{X}$, $\mathsf{local}(\ell)$ counts visits to ℓ in the frontier round r_{max} of $\mathsf{cfg}_{\mathsf{BWA}}$, and $\mathsf{cumul}(\ell)$ counts total visits to ℓ up to r_{max}. We refer to counters local and cumul as history record counters.

For a formula φ, define $t_{\max}(\varphi,\nu)$ as the largest numeric value attained by any threshold in φ under ν. Formally, let $T(\varphi) \subseteq \mathsf{LinTrm}(\mathcal{Y})$ denote the set of thresholds occurring in φ (for example, in Figure 3-Left, we have $T(\mathtt{A}) = \{0\}$ and $T(\mathtt{T}) = \{\mathsf{N_c}\}$), then

$$t_{\max}(\varphi,\nu) = \max\{\, t[\mathcal{Y} \leftarrow \nu] \mid t \in T(\varphi)\,\}.$$

In HRE configurations we require that for every $\ell \in \mathcal{X}$, both $\mathsf{local}(\ell)$ and $\mathsf{cumul}(\ell)$ are bounded above by $B^{\varphi}_{\nu} = 1 + \max\{0,\, t_{\max}(\varphi,\nu)\}$. This bounding is sufficient because every atomic constraint in φ has the form $\sum_{x\in\mathcal{X}} c_x A_x \le t$ where $c_x \in \mathbb{N}$ and A_x denote either a local or cumulative visit count. If $c_x = 0$, the value of A_x is irrelevant. If $c_x > 0$ and A_x takes value B^{φ}_{ν}, then the atom is permanently false regardless of further increases. Thus exact values of A_x beyond B^{φ}_{ν} never affect the truth of φ.

An HRE configuration $\mathsf{cfg}_{\mathsf{HRE}} = \langle \mathsf{cfg}_{\mathsf{BWA}}, \mathsf{local}, \mathsf{cumul}\rangle$ is initial iff $\mathsf{cfg}_{\mathsf{BWA}} \in I^{\mathsf{BWA}}_{\nu}$ and, the history record counters are initialized to match the number of processes at location ℓ in round 0, respecting the bound of B^{φ}_{ν}. Formally, for every $\ell \in \mathcal{X}$,

$$\mathsf{local}(\ell) = \mathsf{cumul}(\ell) = \min\big(\mathsf{PState}_{\mathsf{BWA}}(\ell,0), B^{\varphi}_{\nu}\big),$$

where $\mathsf{PState}_{\mathsf{BWA}}$ is the process-state component of $\mathsf{cfg}_{\mathsf{BWA}}$.

HRE **transitions.** For HRE configurations $\mathsf{cfg}_{\mathsf{HRE}} = \langle \mathsf{cfg}_{\mathsf{BWA}}, \mathsf{local}, \mathsf{cumul}\rangle$ and $\mathsf{cfg}'_{\mathsf{HRE}} = \langle \mathsf{cfg}'_{\mathsf{BWA}}, \mathsf{local}', \mathsf{cumul}'\rangle$, and an action $a = \mathsf{Update}(\rho, r)$, we have $(\mathsf{cfg}_{\mathsf{HRE}}, a, \mathsf{cfg}_{\mathsf{HRE}}) \in Tr^{\mathsf{HRE}}_{\nu}$ iff $(\mathsf{cfg}_{\mathsf{BWA}}, a, \mathsf{cfg}'_{\mathsf{BWA}}) \in Tr^{\mathsf{BWA}}_{\nu}$ and the history record counters are related as follows. Let r_{max}, r'_{max} be the frontier rounds of $\mathsf{cfg}_{\mathsf{BWA}}$ and $\mathsf{cfg}'_{\mathsf{BWA}}$ respectively. Then, for each $\ell \in \mathcal{X}$:

- The cumulative record cumul is incremented at location $\rho.\mathsf{to}$ subject to the saturation at B^{φ}_{ν}; other locations are unchanged. Formally, $\mathsf{cumul}'(\ell) = \min(\mathsf{cumul}(\ell) + 1, B^{\varphi}_{\nu})$ if $\ell = \rho.\mathsf{to}$ and $\mathsf{cumul}'(\ell) = \mathsf{cumul}(\ell)$ otherwise.
- Recall that the record local counts visits in the frontier round.
  - If the frontier-round does not advance ($r'_{\mathsf{max}} = r_{\mathsf{max}}$), then only the target location increments, subject to the saturation. Formally, $\mathsf{local}'(\ell) = \min(\mathsf{local}(\ell) + 1, B^{\varphi}_{\nu})$ if $\ell = \rho.\mathsf{to}$ and $\mathsf{local}'(\ell) = \mathsf{local}(\ell)$ otherwise.
  - If the frontier-round advances ($r'_{\mathsf{max}} > r_{\mathsf{max}}$), all local counters reset to 0 except at the new target, which is set to 1. Formally, $\mathsf{local}'(\ell) = 1$ if $\ell = \rho.\mathsf{to}$ and $\mathsf{local}'(\ell) = 0$ otherwise.
  - Note that frontier-round is never decremented in BWA semantics.

B.5.1 Proof of Proposition 11

Proof. Write $\gamma(\langle \mathsf{cfg}_{\mathsf{BWA}}, \mathsf{local}, \mathsf{cumul}\rangle) = \mathsf{cfg}_{\mathsf{BWA}}$. For every HRE execution $\pi = \mathsf{cfg}_0 \xrightarrow{a_0} \mathsf{cfg}_1 \xrightarrow{a_1} \cdots$, its γ projection

$$\gamma(\pi) = \gamma(\mathsf{cfg}_0) \xrightarrow{a_0} \gamma(\mathsf{cfg}_1) \xrightarrow{a_1} \cdots$$

is a BWA execution. Conversely, for every BWA execution $\pi = \mathsf{cfg}_0 \xrightarrow{a_0} \mathsf{cfg}_1 \xrightarrow{a_1} \cdots$, there exists HRE execution

$$\pi' = \langle \mathsf{cfg}_0, \mathsf{local}_0, \mathsf{cumul}_0\rangle \xrightarrow{a_0} \langle \mathsf{cfg}_1, \mathsf{local}_1, \mathsf{cumul}_1\rangle \xrightarrow{a_1} \cdots,$$

obtained by initializing history record counters as specified in I^{HRE}_{ν} and executing the same actions and updating the history record counters as specified by Tr^{HRE}_{ν}.

We now prove the proposition for base cases $\varphi = \forall_r\, \alpha_r$ and $\varphi = \beta$. Since the translation is preserved for $\neg$ and $\wedge$, the inductive cases follow immediately.

- **Case $\varphi = \forall_r\, \alpha_r$:** Assume $\alpha_r = \sum_\ell c_\ell \cdot \kappa(\ell, r) \leq t$. To show $\pi \models \varphi$ iff $\pi' \models \mathsf{LTL}(\varphi)$.
  - ($\Rightarrow$) Suppose $\pi \models \varphi$ but $\pi' \not\models \mathsf{LTL}(\varphi)$. Then there exists some position i in π' with
  $$\sum_{\ell \in \mathcal{X}} c_\ell \cdot \mathsf{local}_i(\ell) > t[\mathcal{Y} \leftarrow \nu].$$
  Since local counts process visits to round $r_{\mathsf{max}}(\mathsf{cfg}_i)$, this contradicts $\pi \models \forall_r\, \alpha_r$, as α_r fails for $r = r_{\mathsf{max}}(\mathsf{cfg}_i)$.
  - ($\Leftarrow$) Conversely, suppose $\pi' \models \mathsf{LTL}(\varphi)$ but $\pi \not\models \varphi$. Then there exists a round r_0 with
  $$\sum_{\ell} c_\ell \cdot \mathsf{hsc}(\pi)(\ell, r_0) > t[\mathcal{Y} \leftarrow \nu].$$
  Since the number of processes is finite and each performs only a bounded number of updates per round, there exists a last position i in π where $r_{\mathsf{max}}(\mathsf{cfg}_i) = r_0$ (and if cfg_{i+1} exists, then $r_{\mathsf{max}}(\mathsf{cfg}_{i+1}) > r_0$). At this point,
  $$\sum_{\ell \in \mathcal{X}} c_\ell \cdot \mathsf{local}_i(\ell) = \sum_{\ell \in \mathcal{X}} c_\ell \cdot \mathsf{hsc}(\pi)(\ell, r_0) > t[\mathcal{Y} \leftarrow \nu],$$
  contradicting $\pi' \models \mathsf{LTL}(\varphi)$.
- **Case $\varphi = \beta$:** Assume $\beta = \sum_{\ell \in \mathcal{X}} c_\ell \cdot \sum_r \kappa(\ell, r) \leq t$. To show $\pi \models \varphi$ iff $\pi' \models \mathsf{LTL}(\varphi)$.
  - ($\Rightarrow$) Suppose $\pi \models \varphi$ but $\pi' \not\models \mathsf{LTL}(\varphi)$. Then there exists some position i in π' with
  $$\sum_{\ell \in \mathcal{X}} c_\ell \cdot \mathsf{cumul}_i(\ell) > t[\mathcal{Y} \leftarrow \nu].$$
  Since cumul_i count exactly the total number of cumulative visits to each location till cfg_i, the count is less than or equal to the total number of cumulative visits in π, thus
  $$\sum_{\ell \in \mathcal{X}} c_\ell \cdot \sum_r \mathsf{hsc}(\pi)(\ell, r) \geq \sum_{\ell \in \mathcal{X}} c_\ell \cdot \mathsf{cumul}_i(\ell) > t[\mathcal{Y} \leftarrow \nu]$$
  contradicting $\pi \models \varphi$.
  - ($\Leftarrow$) Conversely, if $\pi' \models \mathsf{LTL}(\beta)$ then the number of process visits never exceeds $t[\mathcal{Y} \leftarrow \nu]$ at any prefix, so $\pi \models \beta$.

Thus for every execution π of $\mathsf{RDTS}^{\mathsf{BWA}}(\mathcal{T}, \nu)$ and its corresponding execution π' of $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T}, \nu, \varphi)$: $\pi \models \varphi$ iff $\pi' \models \mathsf{LTL}(\varphi)$. Quantifying over executions yields the desired equivalence at the model level. ◀

B.6 Round-identity abstraction

The system $\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T}, \nu, \varphi)$ abstracts the frontier-round identifier r_{max} from $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T}, \nu, \varphi)$. For an HRE configuration

$$\mathsf{cfg}_{\mathsf{HRE}} = \langle \mathsf{cfg}_{\mathsf{BWA}}, \mathsf{local}, \mathsf{cumul} \rangle \text{ with } \mathsf{cfg}_{\mathsf{BWA}} = \langle \mathsf{PState}_{\mathsf{BWA}}, \mathsf{NState}_{\mathsf{BWA}}, r_{\max} \rangle,$$

define the RIA projection as $\alpha_{\mathsf{RIA}}(\mathsf{cfg}_{\mathsf{HRE}}) = \langle \mathsf{PState}_{\mathsf{BWA}}, \mathsf{NState}_{\mathsf{BWA}}, \mathsf{local}, \mathsf{cumul} \rangle$. Then the RIA semantics is the system

$$\mathsf{RDTS}^{\mathsf{RIA}}(\mathcal{T}, \nu, \varphi) = \langle S_\nu^{\mathsf{RIA}}, I_\nu^{\mathsf{RIA}}, Act_\nu^{\mathsf{RIA}}, Tr_\nu^{\mathsf{RIA}} \rangle,$$

where $S_\nu^{\mathsf{RIA}} = \{\alpha_{\mathsf{RIA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in S_\nu^{\mathsf{HR}}\}$, $I_\nu^{\mathsf{RIA}} = \{\alpha_{\mathsf{RIA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu^{\mathsf{HR}}\}$, and $Act_\nu^{\mathsf{RIA}} = Act_\nu^{\mathsf{HRE}}$. The transition relation is the existential lift of HRE semantics:
$(\mathsf{cfg}_{\mathsf{RIA}}, a, \mathsf{cfg}'_{\mathsf{RIA}}) \in Tr_\nu^{\mathsf{RIA}}$ iff there exist $\mathsf{cfg}_{\mathsf{HRE}}, \mathsf{cfg}'_{\mathsf{HRE}} \in S_\nu^{\mathsf{HRE}}$ with

$$\alpha_{\mathsf{RIA}}(\mathsf{cfg}_{\mathsf{HRE}}) = \mathsf{cfg}_{\mathsf{RIA}}, \alpha_{\mathsf{RIA}}(\mathsf{cfg}'_{\mathsf{HRE}}) = \mathsf{cfg}'_{\mathsf{RIA}}, \text{ and } (\mathsf{cfg}_{\mathsf{HRE}}, a, \mathsf{cfg}'_{\mathsf{HRE}}) \in Tr_\nu^{\mathsf{HRE}}.$$

▶ **Lemma 24** (State-based bisimulation). *It holds that:*

1. $\forall\, \mathsf{cfg} \in I_\nu^{\mathsf{HRE}} : \alpha_{\mathsf{RIA}}(\mathsf{cfg}) \in I_\nu^{\mathsf{RIA}}$ *and* $\forall\, \mathsf{cfg}_{\mathsf{RIA}} \in I_\nu^{\mathsf{RIA}} : \exists\, \mathsf{cfg} \in I_\nu^{\mathsf{HRE}}$ *with* $\alpha_{\mathsf{RIA}}(\mathsf{cfg}) = \mathsf{cfg}_{\mathsf{RIA}}$.
2. *For all* $\mathsf{cfg} \in S_\nu^{\mathsf{HRE}}$ *it holds:*
   - $\Rightarrow$ *If* $(\mathsf{cfg}, a, \mathsf{cfg}') \in Tr_\nu^{\mathsf{HRE}}$, *then* $(\alpha_{\mathsf{RIA}}(\mathsf{cfg}), a, \alpha_{\mathsf{RIA}}(\mathsf{cfg}')) \in Tr_\nu^{\mathsf{RIA}}$.
   - $\Leftarrow$ *If* $(\alpha_{\mathsf{RIA}}(\mathsf{cfg}), a, \mathsf{cfg}'_{\mathsf{RIA}}) \in Tr_\nu^{\mathsf{RIA}}$, *then* $\exists\, \mathsf{cfg}' \in S_\nu^{\mathsf{HRE}}$, $a' \in Act_\nu^{\mathsf{HRE}}$ *such that* $(\mathsf{cfg}, a', \mathsf{cfg}') \in Tr_\nu^{\mathsf{HRE}}$ *and* $\alpha_{\mathsf{RIA}}(\mathsf{cfg}') = \mathsf{cfg}'_{\mathsf{RIA}}$.

Proof. Part 1 is immediate from $I_\nu^{\mathsf{RIA}} = \{\alpha_{\mathsf{RIA}}(\mathsf{cfg}) \mid \mathsf{cfg} \in I_\nu^{\mathsf{HRE}}\}$.

Direction ($\Rightarrow$) follows from the definition of Tr_ν^{RIA} as the existential lift of Tr_ν^{HRE}. For direction ($\Leftarrow$), suppose $(\alpha_{\mathsf{RIA}}(\mathsf{cfg}), a, \mathsf{cfg}'_{\mathsf{RIA}}) \in Tr_\nu^{\mathsf{RIA}}$. By definition of Tr_ν^{RIA} there exist $\mathsf{cfg}_1, \mathsf{cfg}_2 \in S_\nu^{\mathsf{HRE}}$ with

$$(\mathsf{cfg}_1, a, \mathsf{cfg}_2) \in Tr_\nu^{\mathsf{HRE}}, \alpha_{\mathsf{RIA}}(\mathsf{cfg}_1) = \alpha_{\mathsf{RIA}}(\mathsf{cfg}), \text{ and } \alpha_{\mathsf{RIA}}(\mathsf{cfg}_2) = \mathsf{cfg}'_{\mathsf{RIA}}.$$

Thus cfg and cfg_1 agree on the $\mathsf{PState}_{\mathsf{BWA}}$, $\mathsf{NState}_{\mathsf{BWA}}$ and on history record counters local and cumul; they may differ only on the frontier-round. Let r^1_{max} be the frontier-round of cfg_1, r_{max} be the frontier-round of cfg, and let $\Delta = r_{\mathsf{max}} - r^1_{\mathsf{max}}$. Suppose $a = \mathsf{Update}(\rho, r)$. Set $r' = r + \Delta$ and $a' = \mathsf{Update}(\rho, r')$. It follows from the semantics of $\mathsf{RDTS}^{\mathsf{HRE}}(\mathcal{T}, \nu, \varphi)$ that (i) a' is enabled at cfg, and (ii) executing a' at cfg results in cfg' such that $\alpha_{\mathsf{RIA}}(\mathsf{cfg}') = \mathsf{cfg}'_{\mathsf{RIA}}$. This gives the required cfg' and a'. ◀

B.6.1 Proof of Proposition 12.

The proposition follows from the Lem. 24 and the fact that α_{RIA} preserves the state labels given by the history record counters.

C Technical proofs from Section B

C.1 Proof of Lemma 16

Proof. Fix an RMA configuration $\mathsf{cfg}_{\mathsf{RMA}} = \langle \mathsf{PState}_{\mathsf{RMA}}, \mathsf{NState}_{\mathsf{RMA}} \rangle \in S_\nu^{\mathsf{RMA}}$. Define $\beta_{\mathsf{RMA}}(\mathsf{cfg}_{\mathsf{RMA}}) = \mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle \in S_\nu$ by setting $\mathsf{NState} = \mathsf{NState}_{\mathsf{RMA}}$ and, for each $i \in [\![1, N]\!]$, letting $\mathsf{PState}(i).\mathsf{loc} = \mathsf{PState}_{\mathsf{RMA}}(i).\mathsf{loc}$ and $\mathsf{PState}(i).\mathsf{rd} = \mathsf{PState}_{\mathsf{RMA}}(i).\mathsf{rd}$.

Write $\ell \rightsquigarrow_0 \ell'$ if there is a (possibly empty) path of type-0 rules from ℓ to ℓ'. Let

$$X_i(\mathsf{cfg}) = \{\rho \in \mathsf{Rules} \mid \rho.\mathsf{type} = 0 \ \wedge\ \rho.\mathsf{to} \rightsquigarrow_0 \mathsf{PState}(i).\mathsf{loc}\},$$

and for a guard ϕ, let $\mathsf{FV}_{\mathcal{M}}(\phi) \subseteq \mathcal{M}$ denote its set of message variables. Define

$$M_i(\mathsf{cfg}) = \bigcup_{\rho \in X_i(\mathsf{cfg})} \mathsf{FV}_{\mathcal{M}}(\rho.\mathsf{guard}).$$

Finally, define the received-message multiset $\mathsf{PState}(i).\mathsf{rcvMsg} : \mathcal{M} \times \mathbb{N} \to \mathbb{N}$ by

$$\mathsf{PState}(i).\mathsf{rcvMsg}(m, r) = \begin{cases} \mathsf{NState}(m, r), & r < \mathsf{PState}(i).\mathsf{rd}, \\ \mathsf{NState}(m, r), & r = \mathsf{PState}(i).\mathsf{rd} \text{ and } m \in M_i(\mathsf{cfg}), \\ 0, & \text{otherwise.} \end{cases}$$

The β_{RMA} preserves the network component, respects process locations and rounds, and constructs received message component that satisfies condition (1).

β_{RMA} satisfies Part 1 of lemma, since initially every process is in initial location of round 0, and initial locations, and by Definition 1 no type-0 rule can lead to an initial location and thus $M_i(\mathsf{cfg}) =$ for all i.

For part 2, fix an abstract transition $(\mathsf{cfg}_{\mathsf{RMA}}, a, \mathsf{cfg}'_{\mathsf{RMA}}) \in Tr_{\nu}^{\mathsf{RMA}}$ with $a = \mathsf{Update}(i_0, \rho, r)$, and let $\beta_{\mathsf{RMA}}(\mathsf{cfg}_{\mathsf{RMA}}) = \mathsf{cfg}$ and $\beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}}) = \mathsf{cfg}'$, with $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$ and $\mathsf{cfg}' = \langle \mathsf{PState}', \mathsf{NState}' \rangle$. By RMA-enabledness we have, $\mathsf{PState}(i_0).(\mathsf{loc}, \mathsf{rd}) = (\rho.\mathsf{frm}, r)$ and there exists $R : \mathcal{M} \times \mathbb{N} \to \mathbb{N}$ satisfying $0 \leq R \leq \mathsf{NState}$ such that $R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu]$.
Step 1 (Prepare with receives). Let $F = \mathsf{FV}_{\mathcal{M}}(\rho.\mathsf{guard})$. For each $m \in F$, set $\Delta(m) = \max\{0, R(m, r) - \mathsf{PState}(i_0).\mathsf{rcvMsg}(m, r)\}$. Execute exactly $\Delta(m)$ transitions $\mathsf{Recv}(i_0, \langle m, r \rangle)$ (for each $m \in F$), obtaining

$$\mathsf{cfg} \rightsquigarrow^{\mathsf{Recv}^*} \mathsf{cfg}^* = \langle \mathsf{PState}^*, \mathsf{NState} \rangle.$$

Feasibility holds because $0 \leq \Delta(m) \leq \mathsf{NState}(m, r) - \mathsf{PState}(i_0).\mathsf{rcvMsg}(m, r)$ by $R \leq \mathsf{NState}$ (see §A.1).

Note that only $\mathsf{PState}(i_0).\mathsf{rcvMsg}(\cdot, r)$ may change in cfg^* after series of process i_0 receives. Let $S = F \cap M_{i_0}(\mathsf{cfg})$. By the definition of β_{RMA}:

- For $m \in S$, $\mathsf{PState}(i_0).\mathsf{rcvMsg}(m, r) = \mathsf{NState}(m, r)$, hence $\Delta(m) = 0$ and therefore $\mathsf{PState}^*(i_0).\mathsf{rcvMsg}(m, r) \geq R(m, r)$.
- For $m \in F \setminus S$, $\mathsf{PState}(i_0).\mathsf{rcvMsg}(m, r) = 0$, hence $\Delta(m) = R(m, r)$ and therefore $\mathsf{PState}^*(i_0).\mathsf{rcvMsg}(m, r) = R(m, r)$.
- For $m \notin F$, and for all $i \neq i_0$ or $r' \neq r$, received message counts are unchanged.

Consequently, on the guard's support F, $\mathsf{PState}^*(i_0).\mathsf{rcvMsg}(\cdot, r)$ agrees with $R(\cdot, r)$ on $F \setminus S$ and dominates it on S. Since $\rho.\mathsf{guard}$ depends only on F, and by Condition 3 in Def. 1, the guard is monotone in the variables from S, replacing $R(\cdot, r)$ by $\mathsf{PState}^*(i_0).\mathsf{rcvMsg}(\cdot, r)$ preserves satisfaction: if $R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu]$, then also $\mathsf{PState}^*(i_0).\mathsf{rcvMsg}(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu]$. Hence $\mathsf{Update}(i_0, \rho, r)$ is enabled in cfg^*.
Step 2 (Apply the update). From Step 1, $a = \mathsf{Update}(i_0, \rho, r)$ is enabled in cfg^*. Apply it to obtain

$$\mathsf{cfg}^* \xrightarrow{\mathsf{Update}(i_0, \rho, r)} \mathsf{cfg}^x = \langle \mathsf{PState}^x, \mathsf{NState}^x \rangle.$$

Receives do not change NState or any $(\mathsf{loc}, \mathsf{rd})$, and in both the concrete and RMA semantics the effect of Update on these components coincides. Therefore, comparing with the abstract step $(\mathsf{cfg}_{\mathsf{RMA}}, a, \mathsf{cfg}'_{\mathsf{RMA}})$,

$$\mathsf{NState}^x = \mathsf{NState}' \qquad \text{and} \qquad \mathsf{PState}^x(i).(\mathsf{loc}, \mathsf{rd}) = \mathsf{PState}'(i).(\mathsf{loc}, \mathsf{rd}) \ \text{ for all } i.$$

Step 3 (Realign with receives). Only the receive message component may differ between cfg^x and cfg'. We reach cfg' from cfg^x by finitely many Recv steps.
Past rounds. For every process i and every $r' < \mathsf{PState}'(i).\mathsf{rd}$, $\beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}})$ requires $\mathsf{PState}'(i).\mathsf{rcvMsg}(\cdot, r') = \mathsf{NState}'(\cdot, r')$. Since $\mathsf{PState}^x(i).\mathsf{rcvMsg}(\cdot, r') \leq \mathsf{NState}'(\cdot, r')$ component-wise (Eq. 1), we can deliver the outstanding messages at each (m, r') until equality holds.
Frontier round. Let $r_i = \mathsf{PState}'(i).\mathsf{rd}$ and let $M_i(\mathsf{cfg}')$ be as in the definition of β_{RMA}; note that $M_i(\mathsf{cfg}') = M_i(\mathsf{cfg}^x)$. For each i, deliver messages at round r_i exactly for those $m \in M_i(\mathsf{cfg}^x)$ until $\mathsf{PState}(i).\mathsf{rcvMsg}(m, r_i) = \mathsf{NState}'(m, r_i)$, and deliver none for $m \notin M_i(\mathsf{cfg}^x)$. This is feasible by Eq. 1. Moreover:

- For $i \neq i_0$ we have $\mathsf{PState}^x(i) = \mathsf{PState}(i)$, hence $M_i(\mathsf{cfg}^x) = M_i(\mathsf{cfg})$ and thus for all $m \notin M_i(\mathsf{cfg}^x)$ their receive count in round r remain 0.

- For $i = i_0$:
  - If $\rho.\mathsf{type} = 0$, then by construction of M_{i_0} along type-0 paths, $M_{i_0}(\mathsf{cfg}^x) \supseteq M_{i_0}(\mathsf{cfg}) \cup \mathsf{FV}_{\mathcal{M}}(\rho.\mathsf{guard})$, so all messages delivered in Step 1 remain permitted and any additional ones required by $\beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}})$ can be added now.
  - If $\rho.\mathsf{type} > 0$, the frontier advances to $r + \rho.\mathsf{type}$, and since $\mathsf{PState}^x(i_0).\mathsf{rcvMsg}(\cdot, r + \rho.\mathsf{type}) = 0$, we simply deliver up to $\mathsf{NState}'(\cdot, r + \rho.\mathsf{type})$ on the set $M_{i_0}(\mathsf{cfg}^x)$.

Future rounds. For all i and $r'' > \mathsf{PState}'(i).\mathsf{rd}$, $\beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}})$ requires $\mathsf{PState}(i).\mathsf{rcvMsg}(\cdot, r'') = 0$. This already holds in cfg^x and is preserved by not delivering any such messages.

After finitely many Recv transitions as above, the resulting concrete state $\mathsf{cfg}^\dagger = \langle \mathsf{PState}^\dagger, \mathsf{NState}' \rangle$ satisfies $\mathsf{PState}^\dagger(i).(\mathsf{loc}, \mathsf{rd}) = \mathsf{PState}'(i).(\mathsf{loc}, \mathsf{rd})$ and $\mathsf{PState}^\dagger(i).\mathsf{rcvMsg} = \mathsf{PState}'(i).\mathsf{rcvMsg}$ for all i; hence the state reached is exactly $\mathsf{cfg}^\dagger = \beta_{\mathsf{RMA}}(\mathsf{cfg}'_{\mathsf{RMA}}) = \mathsf{cfg}'$.

We have thus constructed a concrete finite path

$$\mathsf{cfg} \ \rightsquigarrow^{\mathsf{Recv}^*} \mathsf{cfg}^* \ \xrightarrow{a} \ \mathsf{cfg}^x \ \rightsquigarrow^{\mathsf{Recv}^*} \ \mathsf{cfg}',$$

in which exactly one transition is a and all others are Recv, as required. ◀

C.2 Proof of Commutativity Claim (Appendix B.3)

Before providing the proof we recall the PIA semantics. Recall in PIA semantics, a configuration $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$ consists of $\mathsf{PState} : \mathcal{L} \times \mathbb{N} \to \mathbb{N}$ (number of processes at location/round pairs (ℓ, r)) and $\mathsf{NState} : \mathcal{M} \times \mathbb{N} \to \mathbb{N}$ (number of messages of kind m sent for round r). For an action $a = \mathsf{Update}(\rho, r)$, set

$$s(a) = (\rho.\mathsf{from}, r), \qquad \mathsf{tgt}(a) = r + \rho.\mathsf{type}, \qquad t(a) = (\rho.\mathsf{to}, \mathsf{tgt}(a)).$$

Executing a at $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState} \rangle$ produces $\mathsf{cfg}' = \langle \mathsf{PState}', \mathsf{NState}' \rangle$ with

$$\mathsf{PState}'(\ell, r') = \begin{cases} \mathsf{PState}(\ell, r') - 1, & \text{if } (\ell, r') = s(a), \\ \mathsf{PState}(\ell, r') + 1 & \text{if } (\ell, r') = t(a), \\ \mathsf{PState}(\ell, r'), & \text{otherwise.} \end{cases}$$

$$\mathsf{NState}'(m, r') = \begin{cases} \mathsf{NState}(m, r') + 1, & \text{if } (m, r') = (\mathsf{Bcast}(\rho.\mathsf{to}), \mathsf{tgt}(a)), \\ \mathsf{NState}(m, r'), & \text{otherwise.} \end{cases}$$

Action a is enabled at cfg if

- (C1) $\mathsf{PState}(s(a)) > 0$, and
- (C2) there exists a witness $R : \mathcal{M} \times \mathbb{N} \to \mathbb{N}$ with $0 \le R \le \mathsf{NState}$ (component-wise) such that $R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu]$.

We will use following two facts. Let $a = \mathsf{Update}(\rho, r)$:

1. (F1) Round locality of guards. Enabledness of a depends only on the round r slice of NState. Indeed, if $R \le \mathsf{NState}$ witnesses the guard, then so does $\widetilde{R}$ defined by $\widetilde{R}(\cdot, r) = R(\cdot, r)$ and $\widetilde{R}(\cdot, r'') = 0$ for all $r'' \neq r$.
2. (F2) Monotonicity in NState. If a is enabled under NState and $\mathsf{NState}' \ge \mathsf{NState}$ component-wise, then a remains enabled under NState' (reuse the same witness R).

Proof. Assume $(\mathsf{cfg}, a, \mathsf{cfg}^*), (\mathsf{cfg}^*, a', \mathsf{cfg}') \in Tr_\nu^{\mathsf{PIA}}$ with $\mathsf{cfg} = \langle \mathsf{PState}, \mathsf{NState}\rangle$, $\mathsf{cfg}^* = \langle \mathsf{PState}^*, \mathsf{NState}^*\rangle$, $\mathsf{cfg}' = \langle \mathsf{PState}', \mathsf{NState}'\rangle$, $a = \mathsf{Update}(\rho, r)$ and $a' = \mathsf{Update}(\rho', r')$, and assume $\mathsf{tgt}(a) \neq r'$.

Step 1: a' is enabled at cfg. Since $(\mathsf{cfg}^*, a', \mathsf{cfg}') \in Tr_\nu^{\mathsf{PIA}}$, enabledness of a' at cfg^* gives

$$\mathsf{PState}^*(s(a')) > 0 \quad \text{and} \quad \exists R^* \leq \mathsf{NState}^* \text{ with } R^*(\cdot, r') \models \rho'.\mathsf{guard}[\mathsf{P} \leftarrow \nu].$$

Enabledness condition C2 holds. Executing a from cfg to cfg^* changes NState only at $(\mathsf{Bcast}(\rho.\mathsf{to}), \mathsf{tgt}(a))$. By the hypothesis $\mathsf{tgt}(a) \neq r'$, we have $\mathsf{NState}^*(\cdot, r') = \mathsf{NState}(\cdot, r')$. By (F1), $\widetilde{R}^*$ is a valid witness for the guard of a' at cfg.
Enabledness condition C1 holds. If $s(a') \neq s(a)$, then $\mathsf{PState}^*(s(a')) = \mathsf{PState}(s(a'))$, hence $\mathsf{PState}(s(a')) > 0$. Else $s(a') = s(a)$, then $\mathsf{PState}^*(s(a)) = \mathsf{PState}(s(a)) - 1 > 0$, so $\mathsf{PState}(s(a)) \geq 2$ and in particular $\mathsf{PState}(s(a')) > 0$. Thus a' is enabled at cfg. Let $\mathsf{cfg}^\dagger = \mathsf{effect}_{\mathsf{PIA}}(\mathsf{cfg}, a')$. Then $(\mathsf{cfg}, a', \mathsf{cfg}^\dagger) \in Tr_\nu^{\mathsf{PIA}}$.

Step 2: a is enabled at $\mathsf{cfg}^\dagger$. Since $(\mathsf{cfg}, a, \mathsf{cfg}^*) \in Tr_\nu^{\mathsf{PIA}}$, enabledness of a at cfg gives

$$\mathsf{PState}(s(a)) > 0 \quad \text{and} \quad \exists R \leq \mathsf{NState} \text{ with } R(\cdot, r) \models \rho.\mathsf{guard}[\mathsf{P} \leftarrow \nu].$$

Enabledness condition C2 holds. From cfg to $\mathsf{cfg}^\dagger$, executing a' changes PState only at $s(a')$ (by -1) and $t(a')$ (by $+1$), and changes NState only by increasing the entry at $(\mathsf{Bcast}(\rho'.\mathsf{to}), \mathsf{tgt}(a'))$. Hence $\mathsf{NState}^\dagger \geq \mathsf{NState}$ component-wise, so by (F2) any witness for the guard of a at cfg remains valid at $\mathsf{cfg}^\dagger$.
Enabledness condition C1 holds. If $s(a) \neq s(a')$, then $\mathsf{PState}^\dagger(s(a)) =$
$\mathsf{PState}(s(a)) > 0$. Else $s(a) = s(a')$, then from Step 1 we had $\mathsf{PState}(s(a)) \geq 2$, so $\mathsf{PState}^\dagger(s(a)) = \mathsf{PState}(s(a)) - 1 \geq 1 > 0$. Therefore a is enabled at $\mathsf{cfg}^\dagger$, let $\mathsf{effect}_{\mathsf{PIA}}(\mathsf{cfg}^\dagger, a) = \widehat{\mathsf{cfg}}$ then $(\mathsf{cfg}^\dagger, a, \widehat{\mathsf{cfg}}) \in Tr_\nu^{\mathsf{PIA}}$.

Step 3: $\widehat{\mathsf{cfg}}$ *and* cfg' *are equal.* Both sequences aa' and $a'a$ modify exactly the same multiset of counters:

- on PState: decrement at $s(a)$ and $s(a')$, increment at $t(a)$ and $t(a')$;
- on NState: increment at $(\mathsf{Bcast}(\rho.\mathsf{to}), \mathsf{tgt}(a))$ and at $(\mathsf{Bcast}(\rho'.\mathsf{to}), \mathsf{tgt}(a'))$.

Since these updates are additions of integers (decrement is addition of the count and -1) at fixed coordinates, which commute, we obtain

$$\mathsf{effect}_{\mathsf{PIA}}(\mathsf{effect}_{\mathsf{PIA}}(\mathsf{cfg}, a), a') \;=\; \mathsf{effect}_{\mathsf{PIA}}(\mathsf{effect}_{\mathsf{PIA}}(\mathsf{cfg}, a'), a).$$

By hypothesis, since the left-hand side is cfg' and the right hand side $\widehat{\mathsf{cfg}}$, hence $\widehat{\mathsf{cfg}} = \mathsf{cfg}'$.

Thus we have found $\mathsf{cfg}^\dagger$ with $(\mathsf{cfg}, a', \mathsf{cfg}^\dagger)$ and $(\mathsf{cfg}^\dagger, a, \mathsf{cfg}')$ in Tr_ν^{PIA}, as required. ◀

D Explicit Encoding of Finite-Counter System

This appendix gives the explicit finite-counter-system construction obtained after the reduction presented in Section 3. The input is a process template

$$\mathcal{T} = \langle \mathsf{P}, \mathsf{rc}, \mathcal{L}, \mathcal{I}, \mathcal{M}, \mathsf{Bcast}, \mathsf{Rules}\rangle$$

and an HSCL formula φ. For fixed-instance verification, the input also includes a parameter valuation $\nu : \mathsf{P} \to \mathbb{N}$.

The output is a finite-counter system

$$(V, \mathsf{INVAR}, \mathsf{INIT}, \mathsf{TRAN})$$

together with the LTL formula obtained from φ, whose atomic propositions are linear-arithmetic formulas over V. Thus, the construction produces an instance of LTL model checking over a finite-counter system, which can be encoded in symbolic model checkers such as nuXmv.

In the fixed-instance case, the invariant formula INVAR additionally fixes the parameter variables according to ν. This implies that only finitely many valuations satisfy INVAR. Hence, for fixed-instance verification, the generated finite-counter system is in fact finite-state.

For a rule $\rho \in \mathsf{Rules}$, write $\rho = (\ell, \ell', k, g)$, where $\ell = \rho.\mathsf{from}$, $\ell' = \rho.\mathsf{to}$, $k = \rho.\mathsf{type}$, and $g = \rho.\mathsf{guard}$. Let $b = \max\{\rho.\mathsf{type} \mid \rho \in \mathsf{Rules}\}$. Further, let $p_1, \ldots, p_q$ be the parameter variables from P that occur in g, and let $m_1, \ldots, m_s$ be the message variables from $\mathcal{M}$ that occur in g. Let $\widehat{g}(p_1, \ldots, p_q, m_1, \ldots, m_s)$ be a quantifier-free Presburger formula equivalent to

$$\exists r_1, \ldots, r_s.\ \left(\bigwedge_{i=1}^{s} 0 \leq r_i \leq m_i\right) \wedge g[m_1 \mapsto r_1, \ldots, m_s \mapsto r_s].$$

Let $X \subseteq \mathcal{L}$ be the set of locations mentioned in φ and let K be one plus the largest threshold appearing in cumulative atoms of φ, see Def. 2 ($K = 0$ if no such atom exists).

Below, we describe each of the components of the obtained finite-counter system and the LTL formula, including some brief explanations to help with understanding.

Variables V.

The set V consists of the following non-negative integer variables:

$$\mathsf{par}_p \qquad (p \in \mathsf{P}),$$

$$\mathsf{loc}_{\ell,d} \qquad (\ell \in \mathcal{L},\ d \in \{0, \ldots, b-1\}),$$

$$\mathsf{msg}_{m,d} \qquad (m \in \mathcal{M},\ d \in \{0, \ldots, b-1\}),$$

$$\mathsf{local}_\ell, \mathsf{cumul}_\ell \qquad (\ell \in X).$$

These are called, respectively, *parameter variables, location variables, message variables, and history variables.*

Invariant formula INVAR.

The formula INVAR is the conjunction of the constraints below.

1. The parameter variables satisfy the resilience condition:

$$\mathsf{rc}\big[p \mapsto \mathsf{par}_p\big].$$

For fixed-instance verification with parameter valuation $\nu : \mathsf{P} \to \mathbb{N}$, INVAR additionally includes the constraints

$$\mathsf{par}_p = \nu(p) \qquad (p \in \mathsf{P}).$$

2. The location variables are bounded by the total number of processes:

$$H \leq \sum_{\ell \in \mathcal{L}} \sum_{d=0}^{b-1} \mathsf{loc}_{\ell,d} \leq \mathsf{par}_n$$

where $H = n$ when failures are modeled explicitly by dedicated fault locations, and $H = n - t$ when failures are modeled implicitly by allowing processes to stop participating.
3. The message variables are bounded by the maximum number of messages that can be broadcast in one round:

$$0 \leq \mathsf{msg}_{m,d} \leq |\mathcal{L}| \cdot \mathsf{par}_n \qquad (m \in \mathcal{M},\ d \in \{0, \dots, b-1\}).$$

4. The history variables are bounded as follows:

$$(0 \leq \mathsf{local}_\ell \leq \mathsf{par}_n) \wedge (0 \leq \mathsf{cumul}_\ell \leq K) \qquad (\ell \in X).$$

Initial formula INIT.

The formula INIT is the conjunction of the constraints below.
1. The initial location variables satisfy

$$\sum_{\ell \in \mathcal{I}} \mathsf{loc}_{\ell,0} = \mathsf{par}_n,$$

$$\mathsf{loc}_{\ell,0} = 0 \qquad (\ell \notin \mathcal{I}), \qquad \text{and}$$

$$\mathsf{loc}_{\ell,d} = 0 \qquad (\ell \in \mathcal{L},\ d \in \{1, \dots, b-1\}).$$

2. The initial message variables satisfy

$$\mathsf{msg}_{m,d} = 0 \qquad (m \in \mathcal{M},\ d \in \{0, \dots, b-1\}).$$

3. The initial history variables satisfy

$$\mathsf{local}_\ell = \begin{cases} \mathsf{loc}_{\ell,0} & \text{if } \ell \in \mathcal{I}, \\ 0 & \text{otherwise}, \end{cases} \qquad \mathsf{cumul}_\ell = \begin{cases} \mathsf{loc}_{\ell,0} & \text{if } \ell \in \mathcal{I}, \\ 0 & \text{otherwise}. \end{cases}$$

Transition formula TRAN.

The transition formula is the disjunction of the local-transition formula and the jump-transition formula:

$$\mathsf{TRAN} = \mathsf{TRAN}_{\mathsf{loc}} \vee \mathsf{TRAN}_{\mathsf{jump}}.$$

Local-transition formula $\mathsf{TRAN}_{\mathsf{loc}}$. Intuitively, the formula $\mathsf{TRAN}_{\mathsf{loc}}$ encodes one process taking a type-0 rule in the current frontier round. For each type-0 rule

$$\rho = (\ell, \ell', 0, g),$$

we define a clause $\mathsf{LocClause}_\rho$, and set

$$\mathsf{TRAN}_{\mathsf{loc}} = \bigvee_{\rho = (\ell, \ell', 0, g) \in \mathsf{Rules}} \mathsf{LocClause}_\rho.$$

The clause $\mathsf{LocClause}_\rho$ is the conjunction of the following constraints.
1. The enabledness constrain:

$$\mathsf{loc}_{\ell,0} > 0 \ \wedge\ \widehat{g}\big[p \mapsto \mathsf{par}_p,\ m \mapsto \mathsf{msg}_{m,0}\big].$$

2. The location variables are updated by moving one process from ℓ to ℓ' at depth 0:

$$\mathsf{loc}'_{\ell,0} = \mathsf{loc}_{\ell,0} - 1, \qquad \mathsf{loc}'_{\ell',0} = \mathsf{loc}_{\ell',0} + 1,$$

and all other location variables are unchanged.

3. If $\mathsf{Bcast}(\ell') = m \in \mathcal{M}$, then the corresponding message variable at depth 0 is incremented:

$$\mathsf{msg}'_{m,0} = \mathsf{msg}_{m,0} + 1,$$

and all other message variables are unchanged. If $\mathsf{Bcast}(\ell') = \perp$, then all message variables are unchanged.

4. If $\ell' \in X$, then the history variables for ℓ' is incremented as follows:

$$\mathsf{local}'_{\ell'} = \mathsf{local}_{\ell'} + 1, \qquad \mathsf{cumul}'_{\ell'} = \begin{cases} \mathsf{cumul}_{\ell'} + 1 & \text{if } \mathsf{cumul}_{\ell'} < K, \\ \mathsf{cumul}_{\ell'} & \text{otherwise.} \end{cases}$$

and all other history variables are unchanged. If $\ell' \notin X$, then all history variables are unchanged.

5. Finally, all parameter variables are unchanged.

Jump-transition formula $\mathsf{TRAN}_{\mathsf{jump}}$. Intuitively, the formula $\mathsf{TRAN}_{\mathsf{jump}}$ encodes a jump of the frontier round. For each jump length

$$h \in \{1, \ldots, b\},$$

we define a clause $\mathsf{JumpClause}_h$, and set

$$\mathsf{TRAN}_{\mathsf{jump}} = \bigvee_{h=1}^{b} \mathsf{JumpClause}_h.$$

The clause $\mathsf{JumpClause}_h$ uses auxiliary non-negetive integer variables

$$J_{d,\rho} \qquad (d \in \{0, \ldots, b-1\},\ \rho \in \mathsf{Rules},\ \rho.\mathsf{type} > 0).$$

The variable $J_{d,\rho}$ denotes the number of processes at depth d that take the positive-type rule ρ during the jump.

Formally, $\mathsf{JumpClause}_h$ can be viewed as existentially quantifying the auxiliary variables $J_{d,\rho}$; the resulting formula can then be made linear-arithmetic by Presburger quantifier elimination. In our encoding, we use the equivalent and more direct presentation: the variables $J_{d,\rho}$ are included in V as auxiliary variables and are updated nondeterministically by jump transitions, so that their current values serve as witnesses for the jump.

The clause $\mathsf{JumpClause}_h$ is the conjunction of the following constraints.

1. The auxiliary variables cannot move more processes than are available. For every $\lambda \in \mathcal{L}$ and $d \in \{0, \ldots, b-1\}$ writing

$$J^{\mathsf{out}}_{\lambda,d} = \sum_{\substack{\rho=(\lambda,\ell',k,g)\in\mathsf{Rules} \\ k>0}} J_{d,\rho},$$

we require

$$J^{\mathsf{out}}_{\lambda,d} \leq \mathsf{loc}_{\lambda,d} \qquad (\lambda \in \mathcal{L},\ d \in \{0, \ldots, b-1\}).$$

2. A positive-type rule may be taken only if it lands exactly in the new frontier round. Thus, for every $\rho = (\ell, \ell', k, g)$ with $k > 0$, and every $d \in \{0, \dots, b-1\}$, we require

$$J_{d,\rho} > 0 \implies d + h = k.$$

3. Whenever some processes take a rule, its guard must be enabled at the corresponding depth.

$$J_{d,\rho} > 0 \implies \widehat{g}\big[p \mapsto \mathsf{par}_p,\ m \mapsto \mathsf{msg}_{m,d}\big],$$

4. The location variables are updated by shifting the old window by length h and placing the processes that jump in the new frontier round. For every $\lambda \in \mathcal{L}$, let

$$J^{\mathsf{in}}_{\lambda} = \sum_{\substack{d \in \{0,\dots,b-1\} \\ \rho = (\ell,\lambda,k,g) \in \mathsf{Rules} \\ k > 0}} J_{d,\rho}.$$

Then

$$\mathsf{loc}'_{\lambda,0} = J^{\mathsf{in}}_{\lambda} \qquad (\lambda \in \mathcal{L}).$$

For depths skipped by the jump, we set

$$\mathsf{loc}'_{\lambda,d} = 0 \qquad (\lambda \in \mathcal{L},\ 1 \le d < h).$$

For the remaining depths, we shift the old variables and remove the processes that jumped:

$$\mathsf{loc}'_{\lambda,d} = \mathsf{loc}_{\lambda,d-h} - J^{\mathsf{out}}_{\lambda,d-h} \qquad (\lambda \in \mathcal{L},\ h \le d \le b-1).$$

5. The message variables are updated analogously. The messages in the new frontier round are exactly those broadcast by the processes that jump into that round. Thus, for every $m \in \mathcal{M}$,

$$\mathsf{msg}'_{m,0} = \sum_{\substack{\lambda \in \mathcal{L} \\ \mathsf{Bcast}(\lambda) = m}} J^{\mathsf{in}}_{\lambda}.$$

For depths skipped by the jump, we set

$$\mathsf{msg}'_{m,d} = 0 \qquad (m \in \mathcal{M},\ 1 \le d < h).$$

For the remaining depths, we shift the old message variables:

$$\mathsf{msg}'_{m,d} = \mathsf{msg}_{m,d-h} \qquad (m \in \mathcal{M},\ h \le d \le b-1).$$

6. The history variables are updated according to the processes that enter locations mentioned in the formula. For every $\lambda \in X$,

$$\mathsf{local}'_{\lambda} = J^{\mathsf{in}}_{\lambda}, \quad \text{and} \qquad \mathsf{cumul}'_{\lambda} = \begin{cases} \mathsf{cumul}_{\lambda} + J^{\mathsf{in}}_{\lambda} & \text{if } \mathsf{cumul}_{\lambda} + J^{\mathsf{in}}_{\lambda} \le K, \\ K & \text{otherwise.} \end{cases}$$

7. Finally, all parameter variables are unchanged.

Remark. The nuXmv encoding additionally requires the transition relation to be total, i.e., to have no deadlock states. We ensure this manually by adding stuttering transitions wherever necessary.

LTL Specification.

For completeness, we recall the HSCL-to-LTL translation presented Section 3.2. From HSCL formula φ, we construct an LTL formula $\mathsf{LTL}(\varphi)$ recursively. The atoms of the translated formula are linear-arithmetic formulas over the history variables of the finite-counter system, namely formulas of the form

$$\sum_{\ell \in X} c_\ell \cdot \mathsf{local}_\ell \leq t \qquad \text{or} \qquad \sum_{\ell \in X} c_\ell \cdot \mathsf{cumul}_\ell \leq t,$$

where $c_\ell \in \mathbb{N}$ are constants, $t \in \mathsf{LinTrm(P)}$, and $\mathsf{local}_\ell, \mathsf{cumul}_\ell$ are history variables.

The translation is defined as follows. See Def.2 for the syntax of HSCL. For a universal round-local atom

$$\forall r.\ \alpha_r \qquad \text{where} \qquad \alpha_r = \sum_{\ell \in X} c_\ell \cdot \kappa(\ell, r) \leq t,$$

we define

$$\mathsf{LTL}(\forall r.\ \alpha_r) = \mathbf{G}\left(\sum_{\ell \in X} c_\ell \cdot \mathsf{local}_\ell \leq t\right).$$

For a cumulative atom

$$\beta = \sum_{\ell \in X} c_\ell \cdot \sum_{r} \kappa(\ell, r) \leq t,$$

we define

$$\mathsf{LTL}(\beta) = \mathbf{G}\left(\sum_{\ell \in X} c_\ell \cdot \mathsf{cumul}_\ell \leq t\right).$$

Boolean connectives are translated compositionally:

$$\mathsf{LTL}(\neg\psi) = \neg\mathsf{LTL}(\psi), \qquad \mathsf{LTL}(\psi \wedge \psi') = \mathsf{LTL}(\psi) \wedge \mathsf{LTL}(\psi').$$