Guard Automata for the Verification of Safety and Liveness of Distributed Algorithms

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

Bastien Thomas
Université Rennes, Inria, CNRS, IRISA, France

Josef Widder
Informal Systems, Wien, Austria

Abstract

Distributed algorithms typically run over arbitrary many processes and may involve unboundedly many rounds, making the automated verification of their correctness challenging. Building on domain theory, we introduce a framework that abstracts infinite-state distributed systems that represent distributed algorithms into finite-state *guard automata*. The soundness of the approach corresponds to the Scott-continuity of the abstraction, which relies on the assumption that the distributed algorithms are *layered*. Guard automata thus enable the verification of safety and liveness properties of distributed algorithms.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Distributed algorithms

Keywords and phrases Verification, Distributed algorithms, Domain theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.15

Related Version *Full Version*: https://hal.inria.fr/hal-03283388

Funding This project has received funding from Interchain Foundation (Switzerland).

1 Introduction

Under the umbrella of *parameterized verification*, the verification of systems formed of an arbitrary number of agents executing the same code, has attracted quite some attention in the recent years, see for instance [18, 9]. Application examples range from distributed algorithms (e.g., for clock synchronization [28] or robot coordination [27]), cache-coherence protocols [25, 1], to chemical or biological systems [10]. In all cases, the systems are designed to operate correctly independently of the number of agents.

More specifically, *distributed algorithms* are central to various emblematic applications, including telecommunications, scientific computing, and Blockchain. Automatically proving the correctness of distributed algorithms is a particularly relevant, as stated by Lamport: “Model-checking algorithms prior to submitting them for publication should become the norm” [22]. The task, that the verification community has started to address, is quite challenging, since it aims at validating at once all instances of the algorithm for arbitrarily many processes.

Distributed algorithms with *threshold guards* are omni-present in solutions for consensus and agreement problems. Typically, these guards also are parameterized, e.g., if the number of processes in a distributed system is n, then it is natural to require that certain actions are taken only if a majority of processes is ready to do so; this results in a parameterized threshold expression of $n/2$. Due to Blockchain and other current applications these kinds of distributed algorithm enjoy recent attention from the algorithm design community as well as

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 15; pp. 15:1–15:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the verification community. the algorithm design community has been studying them for a long time, (see e.g., [11]) and typically provides hand-written proofs based on mathematical models without formal semantics.

For computer-aided verification the first challenge is to develop appropriate modeling formalisms that maintain all behaviors of the original algorithms on the one hand, and on the other hand are abstract and succinct to allow for efficient verification. Several approaches towards efficient verification have recently been proposed.

The threshold automata framework [20] targets asynchronous distributed algorithms with threshold guards and reductions (similar to [23, 17]) have been used to show that SMT-based bounded model checking is complete [19]. Later this framework was generalized and generalizations were analyzed regarding decidability [21], and complexity [5]. The current paper also targets threshold distributed algorithms, yet eventually provides an even coarser abstraction to represent their behaviors, thus reducing the overall verification complexity. Moreover, the semantics of distributed algorithms and the soundness of the abstraction rely on domain theory concepts, thus providing a solid mathematical framework to our work. Last but not least, our approach can handle infinite behaviours, in contrast to the threshold automata framework.

The logical fragment of the IVy toolset has also been shown to allow to model threshold guards by axiomising their semantics as quorum systems [7]. For instance, the reason for waiting for quorums of more than $n/2$ messages is that any two such quorums must intersect at one sender. IVy allows to express these quorum axioms and reduce verification to decidable fragments. Similar intuitions underlie verification results in the heard-of model (HO model) [13]. This computational model for distributed algorithms already targets a high level of abstractions that are sound for communication closed distributed algorithms [12]. Here a consensus logic was introduced in [16] that could be used for deductive verification and cut-off results where provided in [24] that reduce the parameterized verification problem to small finite instances. Compared to this line of work, the distributed algorithms we target share some similarities with these round-based communication closed models. Recently, a threshold automata framework for round-based algorithms was introduced that also uses a small counterexample property for verification in [29]. In contrast, we use domain theory, and particularly Scott continuity to be able to reason on infinite behaviors and thus to capture algorithms that do not necessarily terminate.

Other less related verification frameworks also target distributed algorithms with quite different techniques such as event B [26], array systems [4] or logic and automata theory [3].

Contributions

Using basic domain theory concepts, we provide a rigorous framework to model and verify (asynchronous) distributed algorithms. Our methodology applies to distributed algorithms that are structured in *layers* (that can be seen as a fine-grain notion of rounds), and may consist of countably many layers, thus capturing round-based distributed algorithms (with no *a priori* bound on the number of rounds).

- In Section 2, we define partially ordered transition systems, which serve to express the semantics our models.
- Section 3 introduces the low-level model of layered distributed systems to represent threshold based distributed algorithms. The state-space of layered distributed systems being infinite (and even not necessarily finitely representable), we provide several abstraction steps, up to a so-called guard abstraction. The soundness of each step is justified by

the Scott-continuity of the corresponding abstraction. Some steps are also complete, and thus do not introduce spurious behaviors.

- Finally, towards practical verification, we define in Section 4 the guard automaton, a finite-state abstraction of (cyclic) layered distributed systems. It overapproximates the set of infinite behaviors of distributed algorithms, and thus enabling the verification of safety as well as liveness properties. Its construction can be automated with the help of an SMT solver, paving the way to the automated verification of round-based threshold distributed algorithms.

2 A Fistful of Domain Theory

2.1 Mathematical Preliminaries

This section presents mathematical notions as well as notations that are used throughout the paper. In particular, it introduces partially ordered sets and Scott topology. The interested reader is referred to [2] for an thorough introduction to domain theory.

Sets and multisets. A *multiset* over a set X is an element of $\mathbb{N}^X$. Addition and inclusion over multisets are defined in a natural way. For $\xi, \xi' \in \mathbb{N}^X$ two multisets, $\xi + \xi' \in \mathbb{N}^X$ is the multiset such that for every $x \in X$, $(\xi + \xi')(x) = \xi(x) + \xi'(x)$. We write $\xi \sqsubseteq \xi'$ if for every $x \in X$, $\xi(x) \leq \xi'(x)$. Standard sets can be seen as special cases of multisets with the canonical bijection between the set of subsets of X (2^X) and the set of functions from X to $\{0, 1\}$.

Sequences. For X a set and $n \in \mathbb{N}$ a natural number, a sequence of elements of X of length n is some $u \in X^{\{0,\dots,n-1\}}$. Its length is $|u| = n$ and for $i < n$, $u(i) \in X$ denotes the letter at index i. $X^* = \bigcup_{n\in\mathbb{N}} X^{\{0,\dots,n-1\}}$ (resp. $X^+ = \bigcup_{n>0} X^{\{0,\dots,n-1\}}$) denotes the set of all *finite* (resp. finite and non-empty) sequences of elements of X. Moreover, $\overline{X^*} = X^* \cup X^{\mathbb{N}}$ is the set of finite or *infinite* sequences of X. For $u \in X^*$ a finite sequence and $v \in \overline{X^*}$ a finite or infinite sequence, we write $u \cdot v$ for the *concatenation* of u and v. For u and w two sequences, we write $u \prec w$ and say that u is a *prefix* of w if either w is finite and there exists $v \in \overline{X^*}$ such that $u \cdot v = w$ or $u = w$. For w a sequence and $i \leq |w|$, w_i is the prefix w of length i.

Closures and bounds for partially ordered sets. Let $(X, \sqsubseteq)$ be a partially ordered set, and $\xi \subset X$. The *upward-closure* of ξ is $\uparrow\xi = \{x \in X \mid \exists x' \in \xi, x' \sqsubseteq x\}$, and ξ is *upward-closed* if $\uparrow\xi = \xi$. Dually, one defines the *downward-closure* $\downarrow\xi$ and *downward-closed* sets. An element $x \in X$ is an *upper-bound* of ξ if for any element $x' \in \xi$, $x' \sqsubseteq x$. We write $\mathsf{ub}(\xi)$ for the set of upper-bounds of ξ. If it exists (it is then unique), the *greatest* element of ξ is $x \in X$ such that $x \in \xi$ and $x \in \mathsf{ub}(\xi)$. Dually, one defines the notion of *least* element by reversing the order. If it exists, the *least upper bound* of ξ is the least element of $\mathsf{ub}(\xi)$, and we denote it by $\bigsqcup \xi$. Finally ξ is *directed* if it is non-empty and if for every two elements $x, x' \in \xi$, $\mathsf{ub}(\{x, x'\}) \cap \xi \neq \emptyset$; intuitively, any finite subset of ξ has an upper-bound in ξ. An interesting particular case of directed case are completely ordered sets which are called *chains* in this context.

Directed Complete Partially ordered sets (DCPO). A DCPO is a partially ordered set $(X, \sqsubseteq)$ such that any directed subset $\xi \subset X$ has a (unique) least upper bound. These partially ordered sets are particularly important in semantics of programming languages.

The Scott Topology on DCPO. Directed complete partial orders are naturally equipped with the Scott topology. A subset ξ of a DCPO $(X, \sqsubseteq)$ is *Scott-closed* if it is *downward-closed* and if for any directed subset $\xi' \subset \xi$, $\bigsqcup \xi' \in \xi$. A subset is *Scott-open* if its complement in X is Scott-closed. Functions that are continuous for the Scott topology are called *Scott-continuous*. A function $f : X \to Y$ is *monotonous* if for any $x, x' \in X$, if $x \sqsubseteq x'$ then $f(x) \sqsubseteq f(x')$. A Scott-continuous function is always monotonous. A function $f : X \to Y$ is Scott-continuous if and only if for any directed subset $\xi \subset X$, $f(\bigsqcup(\xi)) = \bigsqcup(f(\xi))$. In this paper, a *partial* function $f : X \to Y$ is called *Scott-continuous* if its domain $\mathsf{dom}(f)$ is Scott-closed and if for any directed subset $\xi \subset \mathsf{dom}(f)$, $f(\bigsqcup \xi) = \bigsqcup f(\xi)$.

2.2 Partially Ordered Transition Systems

Building on domain theory, this section introduces a generic model for distributed transition systems, that will capture the semantics of distributed algorithms. An ordering naturally appears on sets of sent messages –that can only grow– and the asynchrony requires the order to be partial only.

▶ **Definition 1.** *A* partially ordered transition system *(POTS) is a tuple $\mathcal{O} = (X, \sqsubseteq, A)$ where:*

- *$(X, \sqsubseteq)$ forms a DCPO.*
- *A is a set of partial functions, called* actions*, from X to itself and such that for every $a \in A$ and every $x \in \mathsf{dom}(a)$, $x \sqsubseteq a(x)$.*

▶ **Definition 2.** *A* schedule *is a (finite or infinite) sequence of* actions*: $\sigma = (a_t)_{t<T}$, with $T \in \overline{\mathbb{N}}$. A schedule $\sigma = (a_t)_{t<T}$ is* applicable *at $x \in X$ if there exists a sequence $(x_t)_{t<T+1}$ with $x_0 = x$, and for every $t < T$, $x_t \in \mathsf{dom}(a_t)$ and $a_t(x_t) = x_{t+1}$. In this case, we write* $\mathsf{configs}(x, \sigma)$ *for the sequence $(x_t)_{t<T+1}$, and $x \star \sigma$ for $\bigsqcup \{x_t \mid t < T{+}1\}$.*

The above definition uses the convention that $\infty + 1 = \infty$. Note that if σ is applicable at x, then the sequence $(x_t)_{t<T+1}$ is unique. Moreover, the least upper bound $\bigsqcup \{x_t \mid t < T + 1\}$ exists because for any $t < T$, $x_t \sqsubseteq x_{t+1}$ and $\{x_t \mid t < T + 1\}$ is therefore a chain. When $\sigma = (a_t)_{t<T}$ is finite, $x \star \sigma = x \star a_0 \star \cdots \star a_{T-1}$ denotes the last element of the monotonous sequence $\mathsf{configs}(x, \sigma)$. In particular, for $a \in A$ and $x \in \mathsf{dom}(a)$, $x \star a = a(x)$. When $\sigma_t \in A^t$ is defined as the prefix of length t of σ, $x_t = x \star \sigma_t$ and it follows: $x \star \sigma = \bigsqcup \{x \star \sigma_t \mid t < T, t \in \mathbb{N}\}$.

The following lemma will be useful throughout the paper:

▶ **Lemma 3.** *For $x \in X$, the set $\mathrm{App}(x)$ of schedules applicable at x is Scott-closed for the prefix ordering and the function: $[x \star \_] : \mathrm{App}(x) \to X$ is Scott-continuous.*

▶ **Definition 4.** *An* abstraction *between POTS $\mathcal{O} = (X, \sqsubseteq, A)$ and $\mathcal{O}' = (X', \sqsubseteq, A')$ consists of*

- *a* set abstraction $\mathsf{ab}_X : X \to X'$ *which is a Scott-continuous function;*
- *a* monoid abstraction $\mathsf{ab}_A : A^* \to A'^*$ *which is a monoid morphism (with slight abuse of notation, ab_A also denotes its Scott-continuous extension $\overline{A^*} \to \overline{A'^*}$);*

both such that for every $a \in A$ and every $x \in \mathsf{dom}(a)$, $\mathsf{ab}_A(a) \in A'^*$ is applicable at $\mathsf{ab}_X(x) \in X'$ and $\mathsf{ab}_X(x \star a) = \mathsf{ab}_X(x) \star \mathsf{ab}_A(a)$.

The last condition of the definition of abstraction translates into the commutativity of the diagram in Figure 1a. The soundness of the abstraction for any (possibly infinite) schedule is stated in the following proposition and illustrated on Figure 1b.

▶ **Proposition 5.** *Let $(\mathsf{ab}_X, \mathsf{ab}_A)$ be an abstraction between $\mathcal{O} = (X, \sqsubseteq, A)$ and $\mathcal{O}' = (X', \sqsubseteq, A')$, $x \in X$ be an element, and $\sigma \in \overline{A^*}$ a schedule. If σ is applicable at x, then $\mathsf{ab}_A(\sigma)$ is applicable at $\mathsf{ab}_X(x)$ and $\mathsf{ab}_X(x \star \sigma) = \mathsf{ab}_X(x) \star \mathsf{ab}_A(\sigma)$.*

$$\begin{tikzcd}
\mathsf{dom}(a) \subset X \arrow[r, "a"] \arrow[d, "\mathsf{ab}_X"'] & X \arrow[d, "\mathsf{ab}_X"] \\
\mathsf{dom}(\mathsf{ab}_A(a)) \subset X' \arrow[r, "\mathsf{ab}_A(a)"'] & X'
\end{tikzcd}$$

(a) By Definition 4 diagram commutes for any action $a \in A$.

$$\begin{tikzcd}
X \arrow[r, "a_0"] \arrow[d, "\mathsf{ab}_X"'] \arrow[rrr, bend left, "\sigma"] & X \arrow[r, "a_1"] \arrow[d, "\mathsf{ab}_X"'] & \cdots & X \arrow[d, "\mathsf{ab}_X"'] \\
X' \arrow[r, "\mathsf{ab}_A(a_0)"] \arrow[rrr, bend right, "\mathsf{ab}_A(\sigma)"'] & X' \arrow[r, "\mathsf{ab}_A(a_1)"] & \cdots & X'
\end{tikzcd}$$

(b) By Proposition 5 diagram commutes for any schedule σ.

Figure 1 $(\mathsf{ab}_X, \mathsf{ab}_A)$ forms an abstraction between the POTS $(X, \sqsubseteq, A)$ and $(X', \sqsubseteq, A')$.

The proof of this proposition is by transfinite induction on the length of schedules: showing that the result holds for finite schedules is easy, and continuity arguments (such as Lemma 3) are then used to extend to infinite schedules.

3 Layered Distributed Systems and their Abstractions

This section introduces a low-level model for distributed algorithms, whose semantics will be expressed as a POTS. The model is structured in layers, thus restricting the application to algorithms with a specific shape. However, many distributed algorithms from the literature fall in this class, and minor modifications of other algorithms make them amenable to our techniques. The restriction to layered models is used several times in the theoretical developments that follow.

3.1 Layered Distributed Transition Systems

This section introduces *Layered Distributed Transition Systems* (LDTSs) as a model for distributed algorithms, such as the Phase King algorithm [8]. A simplified version of the algorithm is provided in Algorithm 1. This algorithm operates in rounds, each consisting of three steps:

- Broadcast a message (ℓ, m) to all process where ℓ is the round index (line 3)
- Receive the messages $(\ell, \_)$ sent in this round (line 4)
- Update the process variables according to the received messages (lines 5 to 12)

In general, such a series of three instructions, indexed by $\ell \in \mathbb{N}$, is called a *layer* and it refines the classical notion of *rounds*: for instance, in Ben-Or's consensus algorithm [6], each round comprises two layers. Note that layers are assumed to be *communication-closed* [17, 14]: the update instruction at layer ℓ only depends on received messages from the same layer.

Distributed algorithms run over a finite set of *processes*, and at every point in time, the local state of a process is defined by the valuation of its local variables. In this paper, the contents of a sent message is not particularly relevant as it can be deduced from the local state of its sender. Therefore, the communications can be encoded by guards that prevent a process from taking a transition if a condition on the state of *other processes* is not met. Formally, the syntax of layered distributed transition systems is as follows:

Algorithm 1 Inspired by the Phase King Algorithm, this algorithm is a *synchronous* algorithm targetting the resolution of binary consensus. It executes $t+1$ rounds. In round $\ell \in \{0 \dots t\}$, the local value v of each process is updated either according to the majority, or to the value of the process with id ℓ (the King process).

Process PhaseKing(n, t, id, v)**:**
  Data: n processes, $t < \frac{n}{4}$ Byzantine faults, $\mathrm{id} \in \{0 \dots n-1\}$, $v \in \{0,1\}$.
  for $\ell = 0$ *to* t **do**
    broadcast (ℓ, id, v)
    receive all the messages $(\ell, \_, \_)$
    $n_0 \leftarrow$ number of messages $(\ell, \_, 0)$ received
    $n_1 \leftarrow$ number of messages $(\ell, \_, 1)$ received
    if $n_0 > \frac{n}{2} + t$ **then**
      $v \leftarrow 0$
    else if $n_1 > \frac{n}{2} + t$ **then**
      $v \leftarrow 1$
    else
      $v \leftarrow v'$ where (ℓ, ℓ, v') is a received message
  end
  return v;

▶ **Definition 6.** *A* layered distributed transition system *(LDTS) is a tuple* $\mathcal{D} = (P, S, \mathsf{guard})$ *where:*

- P *is a* finite *set of* processes
- S *is a set of* states *partitioned in* layers*:* $S = \bigcup_{\ell \in \mathbb{N}} S_\ell$.
  For $\perp$ *a new element, set* $S^\perp = S \cup \{\perp\}$ *and for* $\ell \in \mathbb{N}$, $S^\perp_\ell = S_\ell \cup \perp$.
  The set $S^\perp$ *is partially ordered with* $s \sqsubseteq s'$ *if* $s = \perp$ *or* $s = s'$.
- $\mathsf{guard} : S^2 \to 2^{[P \to S^\perp]}$ *associates to each pair of states a guard.*
  Additionally, the following layered hypothesis *is imposed:*
  For $\ell \in \mathbb{N}$, $s \in S_\ell$ *and* $s' \in S$, $\mathsf{guard}(s, s') \in 2^{[P \to S^\perp_\ell]}$, *and if* $s' \notin S_{\ell+1}$, *then* $\mathsf{guard}(s, s') = \emptyset$.

Intuitively, for $\ell \in \mathbb{N}$, S_ℓ is the set of states a process can be in at layer ℓ, and $\perp$ is used to represent that a process has not reached that layer yet. Although trivial, the ordering on $S^\perp$ shows sufficient to represent the semantics of distributed algorithms. Moreover, the *guards* correspond to a condition on messages *received* from other processes. Having $x \in \mathsf{guard}(s, s')$ with $x(p) = \perp$ means that there are no conditions on the messages received from process p, so that a process in state s can go to s' even if it has not received any message from p.

To define the semantics of LDTS, recall that the system *a priori* runs fully asynchronously, so that processes may be in different layers[1]. However, messages may be received by processes even if the sender has later reached a layer. This means that the state of each process at each layer should be recorded in the semantics of a LDTS. An agglomeration of local states is called a *configuration*. A *full configuration* additionally stores the messages *received* by each process, as formalized below:

▶ **Definition 7.** *Let* $\mathcal{D} = (P, S, \mathsf{guard})$ *be an LDTS. A* full configuration *of* $\mathcal{D}$ *is a pair* $c^f = (\mathsf{state}(c^f), \mathsf{received}(c^f))$ *where*

- $\mathsf{state}(c^f) : P \to \overline{S^+}$ *is such that for every* $p \in P$ *and* $\ell \in \mathbb{N}$
  - *if* $\ell < |\mathsf{state}(c^f)(p)|$, *then* $\mathsf{state}(c^f)(p)(\ell) \in S_\ell$ *and the latter is the state of* p *in* ℓ*;*
  - *if* $\ell \geq |\mathsf{state}(c^f)(p)|$, *then* $\mathsf{state}(c^f)(p)(\ell) = \perp \in S^\perp_\ell$.
- $\mathsf{received}(c^f) : P \to P \to \mathbb{N} \to S^\perp$ *such that for every* $p \in P$, $\mathsf{received}(c^f)(p) \sqsubseteq \mathsf{state}(c^f)$.

[1] Synchronous systems can also be represented by LDTS, as illustrated with the Phase King algorithm.

The set of full configurations is denoted C^f. It is partially ordered with $\sqsubseteq$ defined by $c^f \sqsubseteq c^{f'}$ if $\mathsf{state}(c^f) \sqsubseteq \mathsf{state}(c^{f'})$ *pointwise with the prefix ordering on $\overline{S^+}$ and* $\mathsf{received}(c^f) \sqsubseteq \mathsf{received}(c^{f'})$ *pointwise.*

Note that $S^\perp$ is a DCPO since each of its directed subsets is finite. C^f is isomorphic to the Cartesian product $\left[(P,=) \to (\overline{S^+}, \prec)\right] \times \left[(P^2 \times \mathbb{N}, =) \to (S^\perp, \sqsubseteq)\right]$ and is therefore a DCPO too.

At a full configuration $c^f \in C^f$, two types of *actions* may happen, corresponding to receptions and internal transitions. First, a process $p \in P$ may receive a message that was sent in layer $\ell \in \mathbb{N}$ by a process $p' \in P$; this action is denoted $\mathsf{rec}\,(p,\ell,p')$. Second, a process $p \in P$ may move from a state $s \in S_\ell$ to state $s' \in S_{\ell+1}$, denoted $\mathsf{tr}\,(p,s,s')$. The effect of actions on full configurations is formally defined as follows:

▶ **Definition 8.** *The set of* actions *of an LDTS $\mathcal{D} = (P, S, \mathsf{guard})$ is*

$$A^f = \{\mathsf{rec}\,(p,p',\ell) \mid p,p' \in P, \ell \in \mathbb{N}\} \cup \bigcup_{\ell \in \mathbb{N}} \{\mathsf{tr}\,(p,s,s') \mid p \in P, s \in S_\ell, s' \in S_{\ell+1}\} \quad .$$

For $c^f \in C^f$ and $\mathsf{rec}\,(p,p',\ell) \in A^f$, the full configuration $c^{f'} = \mathsf{rec}\,(p,p',\ell)\,(c^f)$ is defined by:

- $\mathsf{state}(c^{f'}) = \mathsf{state}(c^f)$
- $\mathsf{received}(c^{f'})(p)(p')(\ell) = \mathsf{state}(c^f)(p')(\ell)$ *and* $\mathsf{received}(c^{f'})$ *equals* $\mathsf{received}(c^f)$ *elsewhere.*

For $c^f \in C^f$ and $\mathsf{tr}\,(p,s,s') \in A^f$, writing $\ell = \left|\mathsf{state}(c^f)(p)\right| - 1$, then $\mathsf{tr}\,(p,s,s')$ is enabled *at $c^f \in C^f$ if: $\ell < \infty$, $\mathsf{state}(c^f)(p)(\ell) = s$ and $\mathsf{received}(c^f)(p)(\_)(\ell) \in \mathsf{guard}(s)(s')$. In this case, the full configuration $c^{f'} = \mathsf{tr}\,(p,s,s')\,(c^f)$ is defined with:*

- $\mathsf{state}(c^{f'})(p) = \mathsf{state}(c^f)(p) \cdot s'$ *and* $\mathsf{state}(c^{f'})$ *equals* $\mathsf{state}(c^f)$ *elsewhere.*
- $\mathsf{received}(c^{f'}) = \mathsf{received}(c^f)$

Note that the reception actions are always enabled. So defined, the semantics of an LDTS is a POTS $\mathcal{O}^f_{\mathcal{D}} = \left(C^f, \sqsubseteq, A^f\right)$; in particular, the notions of schedules and abstractions apply.

▶ **Example 9.** Consider the Phase King algorithm run by three correct processes and a Byzantine one. The Byzantine process is not represented explicitly ($P = \{p_0, p_1, p_2\}$ only contains correct processes) but the guards of the LDTS account for the messages it may send. Also, the King is chosen at each round non-deterministically, abstracting process ids.

A correct process in layer ℓ may be in one of four states $S_\ell = \{v_0, v_1, k_0, k_1\}$, where k_x (resp. v_x) represents that the local value of v is $x \in \{0,1\}$ and that the process is currently King (resp. not King). A full configuration, say c^f, is depicted top-left of Figure 2. The sequence states process p_0 went through so far is $\mathsf{state}(c^f)(p_0) = v_0 \cdot k_1 \cdot v_1$. Also, $\mathsf{received}(c^f)(p_0)(p_2)(0) = v_1$ represents that process p_0 received the message that process p_2 was in state v_1 at layer 0. In contrast, p_0 does not know the state of p_2 at layer 2 (represented by a blank space instead of $\perp$ for commodity). Thus, in c^f, the message sent by process p_2 at layer 2 has yet to be received by p_0. The action $\mathsf{rec}\,(p_0,p_2,2)$ corresponding to this reception is therefore enabled at c^f. The resulting configuration $c^f \star \mathsf{rec}\,(p_0,p_2,2)$ would be identical to c^f except for $\mathsf{received}(c^f \star \mathsf{rec}\,(p_0,p_2,2))(p_0)(p_2)(2) = \mathsf{state}(c^f)(p_2)(2) = v_1$ instead of $\perp$. The reception $\mathsf{rec}\,(p_0,p_1,2)$ can also happen at $c^f \star \mathsf{rec}\,(p_0,p_2,2)$. The resulting configuration $c^{f'} = c^f \star \mathsf{rec}\,(p_0,p_2,2) \star \mathsf{rec}\,(p_0,p_1,2)$ coincides with c^f except for

$$\mathsf{received}(c^{f'})(p_0) = \begin{array}{llll} p_0: & v_0 & k_1 & v_1 \\ p_1: & v_1 & v_1 & k_1 \\ p_2: & v_1 & v_0 & v_1 \end{array}$$

Now p_0 has received more than $\frac{n}{2} + t$ messages in $\{v_1, k_1\}$ so that it updates its value to 1 in the next round. Therefore, the action $\mathsf{tr}\,(p_0, v_1, v_1)$ is enabled at $c^{f'}$ and the configuration $c^{f'} \star \mathsf{tr}\,(p_0,v_1,v_1)$ is equal to $c^{f'}$ except for $\mathsf{state}\big(c^{f'} \star \mathsf{tr}\,(p_0,v_1,v_1)\big) = v_0 \cdot k_1 \cdot v_1 \cdot v_1$.

3.2 Abstracting Received Messages

The partially ordered transition system $\mathcal{O}^f_{\mathcal{D}}$ is fine-grained and rather complex to analyze, therefore the aim of the rest of this section is to define simpler POTS, that preserve or overapproximate the semantics of $\mathcal{O}^f_{\mathcal{D}}$. The successive steps are represented in Figure 2.

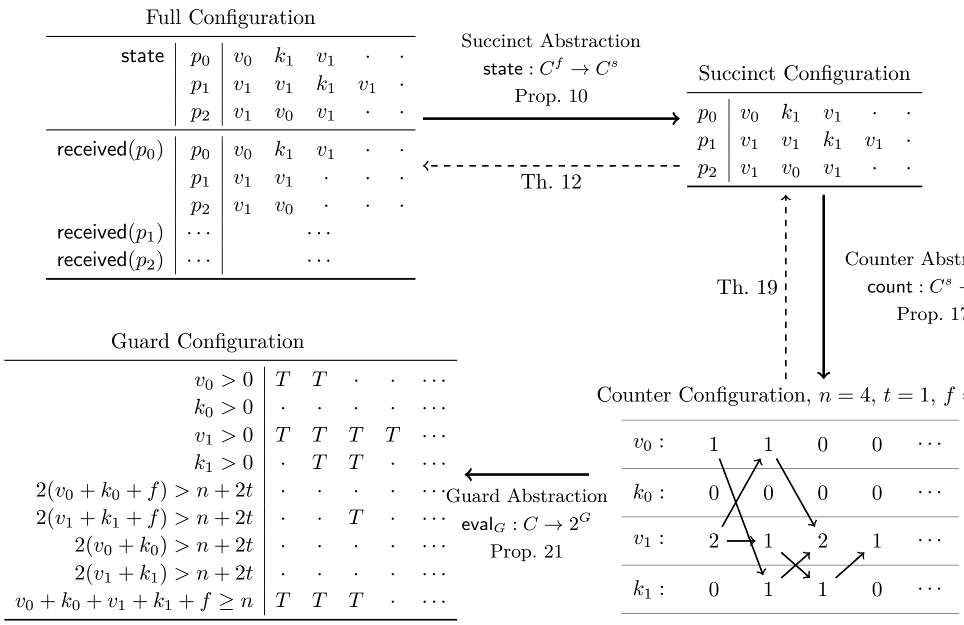


Figure 2 An illustration of the successive abstractions.

The information of messages received by each process is used to check enabledness of transitions. However, the received messages necessarily form a subset of the sent messages. Using the notion of abstraction, this section proves that received messages can be forgotten without losing any information. Instead, it suffices to require the existence of a subset of sent messages that would enable a transition. Changing views from received messages to sent ones is often implicit [21, 20] and without restrictions it may introduce spurious counter-examples (see Example 13). By imposing that each message appears in at most one guard in the transitions taken by a process, the layering hypothesis guarantees that the abstraction is complete (Theorem 12). This abstraction is then used to provide a characterization of reachable configurations (Theorem 15), including those reachable via an infinite schedule.

A *succinct configuration* is an element of $C^s = P \to \overline{S^+}$. For $c^s \in C^s$, $p \in P$, $\ell < |c^s(p)|$ and $s \in S$, $c^s(p)(\ell) = s$ means that process p is/was in state s at layer ℓ. As before, if $\ell \geq |c^s(p)|$, then $c^s(p)(\ell) = \perp$, representing that process p has not reached layer ℓ yet. So-defined, the projection $\mathsf{state} : C^f \to C^s$ abstracts C^f into C^s, so that the reception actions become useless. The set of *succinct actions* is then $A^s = \bigcup_{\ell\in\mathbb{N}} \{[p : s \to s'] \mid p \in P, s \in S_\ell, s' \in S_{\ell+1}\}$ and the monoid morphism $\mathsf{simpl} : A^{f*} \to A^{s*}$ is defined by ignoring reception actions. Formally:

- for $\mathsf{rec}\,(p, p', \ell) \in A^f$, $\mathsf{simpl}\,(\mathsf{rec}\,(p, p', \ell)) = \varepsilon$;
- for $\mathsf{tr}\,(p, s, s') \in A^f$, $\mathsf{simpl}\,(\mathsf{tr}\,(p, s, s')) = [p : s \to s']$.

One can define enabledness of a succinct action, and its effect. For a succinct configuration $c^s \in C^s$ and a succinct action $[p : s \to s'] \in A^s$, writing $\ell = |c^s(p)| - 1$, then $[p : s \to s']$ is *enabled* at c^s if $\ell < \infty$, $c^s(p)(\ell) = s$ and $c^s(\_)(\ell) \in\uparrow\mathsf{guard}(s)(s')$. In this case, $([p : s \to s'](c^s))\,(p) = c^s(p) \cdot s'$ and $([p : s \to s'](c^s))$ coincides with c^s for any other process.

The first two conditions of enabledness are analogous to the case of the full semantics (see Definition 8). The last condition however replaces the guard of the edge with its upper closure. This derives from the fact that the condition now deals with *sent messages* instead of *received* ones, and the latter can only be smaller than the former.

Altogether, the *succinct semantics* of the LDTS consists of the POTS $\mathcal{O}^s_{\mathcal{D}} = (C^s, \sqsubseteq, A^s)$, whose definition is justified by the following proposition:

▶ **Proposition 10.** *The mappings* $\mathsf{state} : C^f \to C^s$ *and* $\mathsf{simpl} : {A^f}^* \to {A^s}^*$ *define an abstraction from the full POTS* $\mathcal{O}^f_{\mathcal{D}} = \left(C^f, \sqsubseteq, A^f\right)$ *to the succinct POTS* $\mathcal{O}^s_{\mathcal{D}} = (C^s, \sqsubseteq, A^s)$.

▶ **Example 11.** Consider the succinct configuration c^s in the top right of Figure 2. It is obtained by applying state to the full configuration c^f on the left. In Example 9, the full schedule $\sigma^f = \mathsf{rec}\,(p_0, p_2, 2) \cdot \mathsf{rec}\,(p_0, p_1, 2) \cdot \mathsf{tr}\,(p_0, v_1, v_1)$ is shown to be applicable at c^f. Therefore, Proposition 10 implies that $\mathsf{simpl}(\sigma^f) = [p_0 : v_1 \to v_1]$ is applicable at c^s.

Propositions 10 and 5 entail that the succinct abstraction is *sound* in the sense that it does not remove any existing behavior, and properties that hold on every execution of the succinct model also hold on the full semantics. However, in general, abstractions are not *complete* and they may introduce new behaviors (for instance, schedules without any reception actions may be applicable in the simplification but not in the full model). Nevertheless, the succinct abstraction is complete: there always exists an applicable full schedule corresponding to each applicable succinct schedule.

▶ **Theorem 12.** *Let* $\sigma^s \in \overline{{A^s}^*}$ *be a succinct schedule applicable at an initial configuration* $c^s \in C^s$. *Then, there exists a full schedule* $\sigma^f \in \overline{{A^f}^*}$ *applicable at a full configuration* $c^f \in C^f$ *such that:* $\mathsf{state}(c^f) = c^s$, $\mathsf{simpl}(\sigma^f) = \sigma^s$, *and* $\mathsf{state}(c^f \star \sigma^f) = c^s \star \sigma^s$.

To prove Theorem 12 one transforms each action $[p : s \to s']$ into a finite schedule of the form $(\mathsf{rec}\,(p, p_u, \ell))_{u<U} \cdot \mathsf{tr}\,(p, s, s')$, carefully choosing the receptions to ensure that the last transition is enabled. To do so, the difficulties are twofold. First, the full schedule $(\mathsf{rec}\,(p, p_u, \ell))_{u<U} \cdot \mathsf{tr}\,(p, s, s')$ not only depends on $[p : s \to s']$, but also on the current configuration. Therefore one cannot define a trivial abstraction. Second, this method requires a way to control the buffers of received messages throughout the schedule. Indeed, one should avoid that a process receives too many messages to take a transition, as 'un-receiving' messages in impossible. This is where the layered structure comes into play, and ensures that when a process receives messages enabling a transition, no earlier transition required these.

▶ **Example 13.** As explained, the layering assumption is crucial in Theorem 12. Consider the *non layered* distributed transition system with four states a, b, c, x, and two processes p, p'. Let c^f be the initial full configuration with $\mathsf{state}(c^f)(p) = a$ and $\mathsf{state}(c^f)(p') = x$. Intuitively, in this counterexample, the guards are set such that the first transition $\mathsf{tr}\,(p, a, b)$ is enabled only if $\mathsf{received}(c^f)(p)(p') = x$ while the next transition $\mathsf{tr}\,(p, b, c)$ requires $\mathsf{received}(c^f)(p)(p') = \perp \neq x$. Process p would thus have to "forget" that it received a message from p' in order to take the second transition, which is impossible in the full semantics.

In contrast, the succinct semantics does not record whether p has already received the message from p' when approaching the second transition. The succinct schedule $[p : a \to b] \cdot [p : b \to c]$ is therefore applicable at $\mathsf{state}(c^f)$ which would contradict Theorem 12 for unlayered distributed transition systems. Imposing that each message appears at most in one guard along the execution of a process, the layered hypothesis prevents this type of counterexamples.

The advantage of the succinct semantics over the full one is that the guards can only become true during an execution. This monotony property, combined with the layered hypothesis, entail the possibility to check that a configuration is reachable *a posteriori*, simply by verifying that the guards of the transitions that are taken are verified in the last configuration. In particular, this avoids building explicitly the schedule at all intermediate configurations. This is formally stated in the following definition and theorem.

▶ **Definition 14.** *A succinct configuration $c^s \in C^s$ is* coherent *if for any $p \in P$ and $\ell \in \mathbb{N}$, if $c^s(p)(\ell) = s \neq \bot$ and $c^s(p)(\ell+1) = s' \neq \bot$, then $c^s(\_)(\ell) \in \uparrow \mathsf{guard}(s, s')$.*

▶ **Theorem 15.** *Let $c^s, c^{s\prime} \in C^s$ be two succinct configurations such that c^s is* coherent*. Then the following statements are equivalent:*

- *$c^s \sqsubseteq c^{s\prime}$ and $c^{s\prime}$ is coherent.*
- *There exists a (possibly infinite) schedule $\sigma^s \in \overline{A^{s*}}$ applicable at c^s such that $c^s \star \sigma^s = c^{s\prime}$.*

3.3 Counter Abstraction

The theory presented so far dealt with a fixed set P of processes. As an advantage, the guards of the edges could be any condition on the set of received messages, but as a drawback, it is impossible to represent *parameterised* systems where the number of processes is not fixed. To remedy this downside, this section introduces *layered threshold automata* (LTA). While this model is syntactically similar to threshold automata [20], its semantics in terms of a POTS is novel. Natural abstractions between the semantics of LDTS and LTA can then be presented, proving that LTA form a faithful representation of distributed algorithms, in contrast to unrestricted threshold automata.

▶ **Definition 16.** *A* Layered Threshold Automaton *(LTA) is a tuple $\mathcal{T} = (R, S, \mathsf{guard})$ where:*

- *R is a set of* parameters
- *S is a set of* states *partitioned into layers: $S = \bigcup_{i=0}^{\infty} S_i$, with S_0 the set of* initial states*.*
- *$\mathsf{guard} : S^2 \to \mathsf{PA}(S \cup R)$ associates a* guard*, in Presburger arithmetic over free variables in $S \cup R$, to each pair of states. The* layered hypothesis *assumes that for $\ell \in \mathbb{N}$, $s \in S_\ell$, and $s' \in S$, $\mathsf{guard}(s, s') \in \mathsf{PA}(S_\ell \cup R)$ and if $s' \notin S_{\ell+1}$, $\mathsf{guard}(s, s') = \mathsf{false}$.*

The guards are monotonous*, i.e. for any guard $g \in \mathsf{guard}(S^2)$, for any valuation $\rho \in \mathbb{N}^R$, $\kappa, \kappa' \in \mathbb{N}^S$, if $\kappa \leq \kappa'$ when ordered pointwise and if $\rho, \kappa \models g$, then $\rho, \kappa' \models g$ as well.*

The set of parameters R typically includes the number n of processes and an upper bound t on the number of faulty processes. Intuitively, the guards represent the conditions on *sent messages* for taking the corresponding transition. The monotony assumption therefore requires that guards in the algorithms concern received messages only, which may be any subset of the sent messages.

In the remainder of this section, $\mathcal{T} = (R, S, \mathsf{guard})$ is a fixed LTA. A *configuration c* of $\mathcal{T}$ is defined by:

- a *parameter valuation* $\mathsf{param}(c) \in R \to \mathbb{N}$ that remains constant during an execution;
- a *counting mapping* $\kappa(c) \in S \to \mathbb{N}$ where $\kappa(c)(s) = k$ means that k processes have visited the state s;
- *flow counters* $\mathsf{flow}(c) \in \left(\bigcup_{\ell \in \mathbb{N}} S_\ell \times S_{\ell+1}\right) \to \mathbb{N}$ where $\mathsf{flow}(c)(s, s') = k$ means that k processes moved from s to s'.

Moreover, processes that leave a state must have entered it, therefore, configuations should also verify the following *flow conditions*:

- **in:** for every $\ell \in \mathbb{N} \setminus \{0\}$ and every $s \in S_\ell$, $\sum_{s' \in S_{\ell-1}} \mathsf{flow}(c)(s', s) = \kappa(c)(s)$
- **out:** for every $\ell \in \mathbb{N}$ and every $s \in S_\ell$, $\sum_{s' \in S_{\ell+1}} \mathsf{flow}(c)(s, s') \leq \kappa(c)(s)$.

The set C of all configurations is equipped with the natural order $\sqsubseteq$ defined by $c \sqsubseteq c'$ if $\mathsf{param}(c) = \mathsf{param}(c')$, $\kappa(c) \leq \kappa(c')$ and $\mathsf{flow}(c) \leq \mathsf{flow}(c')$.

An action over C is an element of $A = \bigcup_{\ell \in \mathbb{N}} A_\ell$ where for $\ell \in \mathbb{N}$, $A_\ell = \{[s \to s'] \mid s \in s_\ell, s' \in S_{\ell+1}\}$. For $c \in C$, an action $[s \to s'] \in A_\ell$ is *enabled* at c if:

- $\sum_{s'' \in S_{\ell+1}} \mathsf{flow}(c)(s, s'') < \kappa(c)(s)$, and
- $\mathsf{param}(c), \kappa(c) \models \mathsf{guard}(s, s')$, written $c \models \mathsf{guard}(s, s')$ for short.

In so, the successor configuration $[s \to s'](c) = c' \in C$ is defined by:

- $\mathsf{param}(c') = \mathsf{param}(c)$
- $\mathsf{flow}(c') = \mathsf{flow}(c) + \mathbb{1}_{(s,s')}$ where $\mathbb{1}_{(s,s')}(s, s') = 1$ and $\mathbb{1}_{(s,s')}(e) = 0$ elsewhere.
- $\kappa(c') = \kappa(c) + \mathbb{1}_{s'}$ where $\mathbb{1}_{s'}(s') = 1$ and $\mathbb{1}_{s'}(s'') = 0$ elsewhere.

One can easily check that configuration c' verifies the flow conditions.

The semantics of the LTA $\mathcal{T}$ is defined as the POTS $\mathcal{O}_\mathcal{T} = (C, \sqsubseteq, A)$.

For $\rho \in \mathbb{N}^R$, the set of configurations that have ρ as parameters and n processes initially is $C_\rho = \{c \in C \mid \mathsf{param}(c) = \rho, \text{ and } \sum_{s \in S_0} \kappa(c)(s) = \rho(\mathsf{n})\}$. Let $\mathcal{O}^\rho_\mathcal{T} = (C_\rho, \sqsubseteq, A)$ denote the POTS restricted to these configurations.

There is a strong link between LTA and LDTS. More precisely, fix a valuation $\rho \in \mathbb{N}^R$. Consider P_ρ a set of $\rho(\mathsf{n})$ processes, and the LDTS $\mathcal{D}_\rho = \left(P_\rho, S, \mathsf{guard}_\rho\right)$ where the function $\mathsf{guard}_\rho \in \bigcup_{\ell \in \mathbb{N}} \left(S_\ell \times S_{\ell+1} \to 2^{\left[P_\rho \to S_\ell^\perp\right]}\right)$ is defined for every $\ell \in \mathbb{N}$, $s \in S_\ell$ and $s' \in S_{\ell+1}$ by:

$$\mathsf{guard}_\rho(s, s') = \left\{x \in P \to S^\perp \mid \rho, \left[s \mapsto \left|x^{-1}(\{s\})\right|\right] \models \mathsf{guard}(s, s')\right\} \quad .$$

Let $C^s_\rho = P_\rho \to \overline{S^+}$ denote the set of succinct configurations of $\mathcal{D}_\rho$. Consider $c^s \in C^s_\rho$ and define $\mathsf{count}_{C^s_\rho}(c^s) \in C_\rho$ with:

- $\mathsf{param}\left(\mathsf{count}_{C^s_\rho}(c^s)\right) = \rho$
- for $\ell \in \mathbb{N}$ and $s \in S_\ell$: $\kappa\left(\mathsf{count}_{C^s_\rho}(c^s)\right)(s)(\ell) = |\{p \in P_\rho \mid c^s(p)(\ell) = s\}|$
- For $\ell \in \mathbb{N}$, $s \in S_\ell$ and $s' \in S_{\ell+1}$:

$$\mathsf{flow}\left(\mathsf{count}_{C^s_\rho}(c^s)\right)(s, s') = \left|\left\{p \in P_\rho \mid \begin{array}{c} c^s(p)(\ell) = s \\ c^s(p)(\ell+1) = s' \end{array}\right\}\right|$$

Let $A^s_\rho = \bigcup_{\ell \in \mathbb{N}} \{[p : s \to s'] \mid p \in P_\rho, s \in S_\ell, s' \in S_{\ell+1}\}$ denotes the set of succinct actions of $\mathcal{D}_\rho$. Define a monoid morphism $\mathsf{count}_{A^s_\rho} : A^{s*}_\rho \to A^*$ such that for $[p : s \to s'] \in A^s_\rho$, $\mathsf{count}_{A^s_\rho}(\mathsf{tr}\,(p, s, s')) = [s \to s']$. So defined:

▶ **Proposition 17.** *The mappings $\mathsf{count}_{C^s_\rho} : C^s_\rho \to C_\rho$ and $\mathsf{count}_{A^s_\rho} : A^{s*}_\rho \to A^*$ define an abstraction from the POTS $\left(C^s_\rho, \sqsubseteq, A^s_\rho\right)$ to the counter POTS $(C_\rho, \sqsubseteq, A)$.*

Proposition 17 holds for *any* parameter valuation $\rho \in \mathbb{N}^R$. Thus, a single LTA represents *infinitely-many* LDTS, one for each parameter valuation.

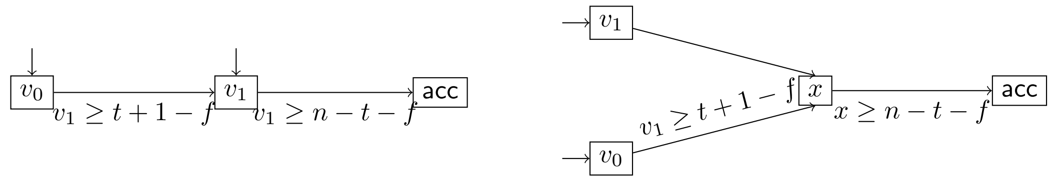
Similarly to the case of LTA, one can define *coherence* of configurations for LDTS, and obtain an equivalent of Theorem 15 at the counter abstraction level.

▶ **Definition 18.** *Configuration $c \in C$ is said* counter coherent *when for every $\ell \in \mathbb{N}$, $s \in S_\ell$ and $s' \in S_{\ell+1}$, if $\mathsf{flow}(c)(s, s') > 0$, then $c \models \mathsf{guard}(s, s')$.*

▶ **Theorem 19.** *Let $c, c' \in C_\rho$ be two configurations such that c is counter coherent. Then the following statements are equivalent:*

- *$c \sqsubseteq c'$ and c' is counter coherent;*
- *There exists a (possibly infinite) schedule $\sigma \in \overline{A^*}$ applicable at c such that $c \star \sigma = c'$.*

The flow conditions and the counter coherence can easily be encoded as a set of linear arithmetic formulas that *do not* depend on the number of processes. In particular, if the LTA is *finite*, then the resulting set of equations is finite as well, making the reachability problem decidable in this case (for initial and target states represented by linear arithmetic formulas). This can be used to verify not only safety properties, but also liveness properties as configurations represent potentially infinite behaviors and contain information about the whole execution. Theorem 19 differs from the threshold automata approach [20] because a schedule does not need to be explicitely built. In particular, the layering assumption implies that the order in which guards become true is irrelevant, which simplifies a lot the SMT queries. More importantly, our approach applies to *infinite* automata where methods based on bounding the diameter of the transition system have little chance of succeeding.



(a) Non-layered. **(b)** Layered.

Figure 3 Two threshold automata for the reliable broadcast algorithm [11].

▶ **Example 20.** Theorem 19 heavily relies on the layered hypothesis. To see that, consider the non layered model of Figure 3a. Let c be a configuration with $\mathsf{flow}(c)(v_0, v_1) > 0$. Then the counter coherence would require that $c \models v_1 \geq t+1-f$, however, this last condition may only hold because the transition was taken in the first place, resulting in spurious configurations. This can be fixed by tweaking the model in order to make it layered as seen on Figure 3b.

3.4 Guard Abstraction

Consider an LTA $\mathcal{T} = (R, S, \mathsf{guard})$. Even when S is finite, its configuration set C is infinite as the number of processes n is unbounded. When S is infinite, then C is infinite in two dimensions: it consists of infinitely many variables that may take infinitely many values. The guard abstraction presented here aims at partitioning these values into finitely many classes. The resulting model will however remain infinite, if S is.

Consider a set $G \subset \mathsf{PA}(S \cup R)$ of *monotonous guards*, that is, every $g \in G$ is a linear arithmetic formulas with free variables in $S \cup R$ such that for $\rho \in \mathbb{N}^R$ and $\kappa, \kappa' \in \mathbb{N}^S$, if $\kappa \leq \kappa'$ pointwise and if $\rho, \kappa \models g$, then $\rho, \kappa' \models g$ as well.

Intuitively, the guard abstraction only records the valuations of the guards, not the number of processes in each state. For this idea to succeed, the valuations of the guards must converge during an execution, which is guaranteed by the following proposition.

▶ **Proposition 21.** *The mapping* $\mathsf{eval}_G \colon (C, \sqsubseteq) \to \left(2^G, \subseteq\right)$ *defined by* $\mathsf{eval}_G(c) = \{g \in G \mid c \models g\}$ *is Scott-continuous.*

4 Guard Automata towards Practical Implementation

While Theorem 19 suffices to verify *finite* LTA through the counter abstraction, it falls short at capturing infinite models that arise for instance from round-based algorithms. This section introduces guard automata as a finite-state abstraction which is sound, yet, unsurprisingly, not complete in general and may introduce spurious counterexamples.

4.1 Cyclic LTA

Towards algorithmic considerations and practical implementations, the rest of the paper focuses on round-based distributed algorithms, which can be captured by cyclic LTA. Intuitively, a cyclic LTA is used to model an LTA that repeats a finite series of layers indefinitely. For $k \in \mathbb{N}_{>0}$, a *k-cyclic LTA* (k-CLTA) is a tuple $\mathcal{T}^c = (R, S^c, \mathsf{guard}^c)$ where:

- R is a *finite* set of parameters.
- S^c is a *finite* set of states partitioned into k layers $S^c = S^c_0 \cup \cdots \cup S^c_{k-1}$.
- $\mathsf{guard}^c : S^{c2} \to \mathsf{PA}(R \cup S^c)$ is a finite set of guards such that for $\ell < k$, $s^c \in S^c_\ell$ and $s^{c\prime} \in S^c$, $\mathsf{guard}^c(s^c, s^{c\prime}) \in \mathsf{PA}(R \cup S^c_\ell)$ and if $s^{c\prime} \notin S^c_{\ell+1 \bmod k}$, then $\mathsf{guard}^c(s^c, s^{c\prime}) = \mathsf{false}$.

Unfolding a k-CLTA yields an infinite-state acyclic LTA $\mathsf{unfold}\,(R, S^c, \mathsf{guard}^c)$. Formally $\mathsf{unfold}\,(R, S^c, \mathsf{guard}^c) = (R, S, \mathsf{guard})$ with:

- $S = \{(s^c, \ell) \mid \ell \in \mathbb{N}, s^c \in S^c_{\ell \bmod k}\}$
- For $\ell \in \mathbb{N}$, $s^c \in S^c_{\ell \bmod k}$ and $s^{c\prime} \in S^c_{\ell+1 \bmod k}$, $\mathsf{guard}\big((s^c, \ell), (s^{c\prime}, \ell+1)\big) = \mathsf{guard}^c(s^c, s^{c\prime})[s^{c\prime\prime} \leftarrow (s^{c\prime\prime}, \ell) \text{ for } s^{c\prime\prime} \in S^c_{\ell \bmod k}]$ meaning that any free variable $s^{c\prime\prime} \in S^c$ that appears in $\mathsf{guard}^c(s^c, s^{c\prime})$ gets replaced with $(s^{c\prime\prime}, \ell)$. In any other case, guard is false.

4.2 Guard Automaton

From the guard abstraction, one can construct a finite-state automaton that represents the set of reachable configurations of a cyclic LTA.

Let $\mathcal{T}^c = (R, S^c, \mathsf{guard}^c)$ be a k-CLTA equipped with a *finite* set of guards expressed in Presburger arithmetic: $G^c = \bigcup_{\ell<k} G^c_\ell$ such that for $\ell < k$, $G^c_\ell \in \mathsf{PA}(S^c_\ell \cup R)$. In practice, G^c will include all guards appearing in the LTA, as well as the events that need to be observed.

A CLTA can be unfolded into an infinite-state LTA, by concatenating copies of $\mathcal{T}^c$. In order for the guard abstraction to be formally defined, copies of the guards in G^c for each new layer are required. For $\ell \in \mathbb{N}$ a layer index and $g^c \in G^c_{\ell \bmod k}$ a guard, $\mathsf{unfold}^G{}_\ell(g^c) = g^c[s^c \leftarrow (s^c, \ell) \text{ for } s^c \in S^c_{\ell \bmod k}]$ denotes the guard obtained by replacing every free occurrence of a variable $s^c \in S^c_{\ell \bmod k}$ in g^c by (s^c, ℓ). The converse folding operation is defined by: $\mathsf{fold}^G{}_\ell(g) = g[(s^c, \ell) \leftarrow s^c, \text{ for } s^c \in S^c_{\ell \bmod k}]$. Finally, $G_\ell = \mathsf{unfold}^G\,(G^c_{\ell \bmod k})$ is the set of guards at layer ℓ and $G = \bigcup_{\ell \in \mathbb{N}} G_\ell$ the set of all guards.

The guard abstraction maps every configuration of $\mathsf{unfold}(\mathcal{T}^c)$ to a set of guards that hold in that configuration. Formally, $\mathsf{eval}_G : C \to 2^G$. A set of guards $\gamma \in 2^G$ can be represented with the sequence $\gamma_0\gamma_1\ldots$, where for $\ell \in \mathbb{N}$, $\gamma_\ell = \gamma \cap G_\ell$. $\mathsf{fold}^G(\gamma)$ then denotes the sequence $\mathsf{fold}^G{}_0(\gamma_0) \cdot \mathsf{fold}^G{}_1(\gamma_1) \cdot \cdots \in \big(2^{G^c}\big)^\omega$ and unfold^G is the converse operation that applies $\mathsf{unfold}^G{}_\ell$ to the elements of layer ℓ in the sequence. Doing so, a configuration $c \in C$ defines a (possibly infinite) word $\gamma^c_0\gamma^c_1\ldots$ over the *finite* alphabet $\Sigma = \bigcup_{\ell<k} 2^{G^c_\ell}$ as represented in Figure 4.

$$
\begin{array}{ccccc}
 & & \gamma_0 \subset G_0 & & \gamma^c_0 \subset G^c_0 \\
 & & \gamma_1 \subset G_1 & & \gamma^c_1 \subset G^c_1 \\
 & & \gamma_2 \subset G_2 & \xrightarrow{\mathsf{fold}^G : \prod_{\ell\in\mathbb{N}} 2^{G_\ell} \to \prod_{\ell\in\mathbb{N}} 2^{G^c_{\ell \bmod k}}} & \gamma^c_2 \subset G^c_2 \\
c \in C & \xrightarrow{\mathsf{eval}_G : C \to 2^G \approx \prod_{\ell\in\mathbb{N}} 2^{G_\ell}} & \vdots & & \vdots \\
 & & \gamma_{k-1} \subset G_{k-1} & \xleftarrow[\mathsf{unfold}^G : \prod_{\ell\in\mathbb{N}} 2^{G^c_{\ell \bmod k}} \to \prod_{\ell\in\mathbb{N}} 2^{G_\ell}]{} & \gamma^c_{k-1} \subset G^c_{k-1} \\
 & & \gamma_k \subset G_k & & \gamma^c_k \subset G^c_0 \\
 & & \gamma_{k+1} \subset G_{k+1} & & \gamma^c_{k+1} \subset G^c_1 \\
 & & \vdots & & \vdots
\end{array}
$$

Figure 4 From a configuration to a word over the finite alphabet of the guard automaton.

For $\ell < k$ a layer index, $\gamma^c \in 2^{G^c_\ell}$ and $\gamma^{c\prime} \in 2^{G^c_{\ell+1 \bmod k}}$ guard valuations of layer ℓ and the next layer, one can use an SMT solver to check whether $\gamma^{c\prime}$ is a successor γ^c. Precisely, the SMT query asks for the existence of $x \in \mathbb{N}^{S^c_\ell}$, $y \in \mathbb{N}^{S^c_{\ell+1 \bmod k}}$ and $e \in \mathbb{N}^{S^c_\ell \times S^c_{\ell+1 \bmod \mathbb{N}}}$ such that the valuation of guards (1), flow condition (2) and counter coherence (3) are verified.

$$x \models \bigwedge_{g^c \in \gamma^c} g^c \wedge \bigwedge_{g^c \in G^c_\ell \setminus \gamma^c} \neg g^c \qquad y \models \bigwedge_{g^{c\prime} \in \gamma^{c\prime}} g^{c\prime} \wedge \bigwedge_{g^{c\prime} \in G^c_{\ell+1 \bmod k} \setminus \gamma^{c\prime}} \neg g^{c\prime} \tag{1}$$

$$e, x \models \bigwedge_{s^c \in S^c_\ell} s^c \geq \sum_{s^{c\prime} \in S^c_{\ell+1 \bmod k}} \left[s^c, s^{c\prime}\right] \qquad e, y \models \bigwedge_{s^{c\prime} \in S^c_{\ell+1 \bmod k}} \sum_{s^c \in S^c_\ell} \left[s^c, s^{c\prime}\right] = s^{c\prime} \tag{2}$$

$$e, x \models \bigwedge_{(s^c, s^{c\prime}) \in S^c_\ell \times S^c_{\ell+1 \bmod \mathbb{N}}} \left[s^c, s^{c\prime}\right] > 0 \longrightarrow \mathsf{guard}^c(s^c, s^{c\prime}) \tag{3}$$

The guard automaton is a finite automaton whose language *overapproximates* the set of reachable configurations. It bears similarities with de Bruijn graphs [15] used e.g. in bioinformatics. If $E_\ell \subset 2^{G^c_\ell} \times 2^{G^c_{\ell+1 \bmod k}}$ denotes the set of all pairs $\gamma^c, \gamma^{c\prime}$ that verify conditions (1) and and (3), one can build the set $E = \bigcup_{\ell<k} E_\ell$.

▶ **Definition 22.** *The* guard automaton *of* $\mathcal{T}^c$ *is* $\mathsf{GA}_G(\mathcal{T}^c) = \left(\Sigma, E, 2^{G^c_0}, \mathsf{src}, \mathsf{dest}, \mathsf{label}\right)$ *where:*

- Σ *is both the alphabet and the set of states.*
- $2^{G^c_0} \subset \Sigma$ *is the set of initial states.*
- $E \subset \Sigma^2$ *defined above is the set of edges, equipped with* $\mathsf{src} : E \to \Sigma$ *(resp.* $\mathsf{dest} : E \to \Sigma$*) that defines the* source state *(resp.* destination state*) of every edge, and* $\mathsf{label} : E \to \Sigma$ *associates a* label *to each edge defined by* $\mathsf{label}(\gamma^c, \gamma^{c\prime}) = \gamma^c$.

An infinite run $(e_\ell)_{\ell<\infty}$ of the guard automaton defines a word $\mathsf{word}\left((e_\ell)_{\ell<\infty}\right) = \mathsf{label}(e_0) \cdot \mathsf{label}(e_1) \cdot \cdots$, and $\mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c)) \subset \Sigma^\omega$ denotes the language of $\mathsf{GA}_G(\mathcal{T}^c)$.

▶ **Example 23.** Algorithm 1 can be described by the following CLTA with $k = 1$. The parameters are $R = \{n, t, f\}$ where f denotes the actual number of Byzantine faults. States are $S^c = \{v_0, k_0, v_1, k_1\}$. The guards here only depend on the next value of v. For instance:

$$\mathsf{guard}(\_, v_0) = (v_0 + k_0 + v_1 + k_1 + f = n)$$
$$\wedge \Big(\big(2(v_0 + k_0 + f) > n + 2t\big) \vee \big((2v_0 + 2k_0 \leq n + 2t) \wedge (2v_1 + 2k_1 \leq n + 2t) \wedge (k_1 = 0) \big) \Big).$$

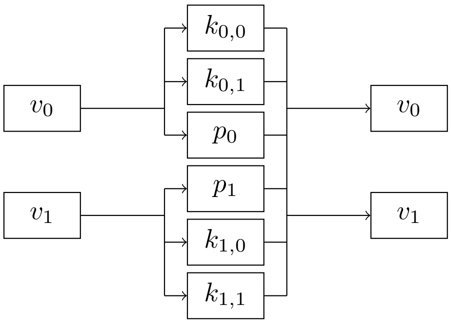
Also, $\mathsf{guard}(\_, k_0) = \mathsf{guard}(\_, v_0)$ and $\mathsf{guard}(\_, v_1) = \mathsf{guard}(\_, k_1)$ is defined symmetrically.

A configuration c of the unfolded LTA is depicted bottom-right of Figure 2, where the array contains the valuation $\kappa(c)$ and the arrows represent the flow. For example $\kappa(c)(v_1, 0) = 2$, $\mathsf{flow}(c)((v_0, 0), (k_1, 1)) = 1$ and $\mathsf{flow}(c)((v_0, 0), (v_0, 1)) = 0$.

The guard abstraction transforms c into the guard configuration bottom-left of Figure 2. Here, we chose the set of guards G^c to consist of $s > 0$ for each $s \in S^c$ and of the guards of the LTA. The alphabet Σ contains e.g., $(T \cdot T \cdot \cdots \cdot T)$. SMT queries determine whether two letters may appear successively, in order to build the guard automaton. For instance, according to the first two layers of $\mathsf{eval}_G(c)$, $(T \cdot T \cdot \cdots \cdot T)$ can be followed by $(T \cdot TT \cdot \cdots \cdot T)$. There will therefore be a transition between these two states in the guard automaton.

▶ **Theorem 24.** *Let* $c \in C$ *be a configuration of* $\mathsf{unfold}(\mathcal{T}^c)$ *and* $\mathsf{eval}_G(c) \in 2^G$ *its guard abstraction. If* c *is counter-coherent, then* $\mathsf{fold}^G\left(\mathsf{eval}_G(c)\right) \in \mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c))$.

By soundness of the guard automaton construction, a property which holds on configurations that correspond to runs of $\mathsf{GA}_G(\mathcal{T}^c)$ also holds on the configurations of $\mathsf{unfold}(\mathcal{T}^c)$. A simple verification procedure thus consists in checking that $\mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c))$ is included in a given language of correct configurations. At a first glance, it might seem that only safety properties can be checked. However, the guard automaton also represents configurations reachable by infinite schedules, making the verification of liveness properties feasible.



$P = \{n, t, f\}$ and for $x, y \in \{0, 1\}$:

$$\begin{aligned}
\mathsf{guard}(v_x, p_x) &= \mathsf{true} \\
\mathsf{guard}(v_x, k_{x,y}) &= [2(v_y + f) \geq n] \\
\mathsf{guard}(_, v_x) &= [2(p_x + k_{x,0} + k_{x,1}) > n + 2t] \vee \\
&\quad \left[\begin{array}{c} 2(p_0 + k_{0,0} + k_{0,1}) \leq n + 2t \wedge \\ 2(p_1 + k_{1,0} + k_{1,1}) \leq n + 2t \wedge \\ k_{0,x} + k_{1,x} = 0 \end{array}\right]
\end{aligned}$$

Figure 5 A 2-CLTA for the Phase King algorithm with non-deterministic choice of the king. A process in $k_{x,y}$ is king of the current round, its current value is x and it thinks the majority is y.

▶ **Example 25.** For presentation purposes, Algorithm 1 is an overly simplified version of the Phase King algorithm [8]. The latter can be faithfully encoded by the 2-CLTA $\mathcal{T}^c$ of Figure 5, where the updated value when there is no clear majority is not the king's value, but rather the majority of the values received by the king. Each round consists of two layers of communication, a first in which each process broadcasts its value, and a second in which the king broadcasts what it thinks is the majority. The set of guards at the first layer is $G_0^c = \{v_0 > 0, v_1 > 0\}$ and at the second layer G_1^c consists of $k_{0,0}+k_{1,0} > 0$, $k_{0,1}+k_{1,1} > 0$, $p_0+k_{0,0}+k_{0,1} > 0$, $p_1+k_{1,0}+k_{1,1} > 0$, $2(k_{0,0}+k_{0,1}+p_0+f) > n+2t$ and $2(k_{1,0}+k_{1,1}+p_1+f) > n+2t$.

Restricting to valuations with $\sum_{s \in S_\ell} s+f = n$ (fairness) and $k_{0,0}+k_{0,1}+k_{1,0}+k_{1,1} \leq 1$ (at most one king), the resulting guard automaton has 3 states in even layers and 11 in odd layers. Writing [*formula*] for the set of letters in 2^{G^c} for which *formula* holds, one can show:

$$\begin{aligned}
\mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c)) \subset\ & [\neg(k_{0,0} + k_{1,0} > 0) \wedge \neg(k_{0,1} + k_{1,1} > 0)]^\omega && (4)\\
& \cup \Sigma^*[(k_{0,0} + k_{1,0} > 0) \vee (k_{0,1} + k_{1,1} > 0)][\neg(p_0 + k_{0,0} + k_{0,1} > 0)]^\omega && (5)\\
& \cup \Sigma^*[(k_{0,0} + k_{1,0} > 0) \vee (k_{0,1} + k_{1,1} > 0)][\neg(p_1 + k_{1,0} + k_{1,1} > 0)]^\omega\ . && (6)
\end{aligned}$$

Therefore, either every chosen king is Byzantine (4), or all processes agree on a value after a non-Byzantine king is chosen (5 or 6).

In general, although is it sound, the guard automaton construction is not complete: the language may contain words that correspond to no configuration of the LTA. As usual for incomplete methods, heuristics can be used to remove some spurious counterexamples.

5 Conclusion

This paper presented a methodology, based on domain theory, to represent and analyze distributed algorithms. Infinite-state models are abstracted into finite-state guard automata, on which one can check safety and liveness properties.

Optimizing and benchmarking the guard automaton implementation is on our current agenda to demonstrate the applicability of our methodology to standard distributed algorithms. A more long-term research objective is to build on the current contribution to develop a rigorous framework for the verification of randomized distributed algorithms.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Zeinab Ganjei, Ahmed Rezine, and Yunyun Zhu. Verification of cache coherence protocols wrt. trace filters. In *Proceedings of the 15th International Conference on Formal Methods in Computer-Aided Design (FMCAD'15)*, pages 9–16. IEEE, 2015.

2 Samson Abramsky and Achim Jung. *Domain Theory*, volume 3 of *Handbook of Logic in Computer Science*, pages 1–168. Clarendon Press, 1994.

3 C. Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. *Information and Computation*, 259:305–327, 2018.

4 Francesco Alberti, Silvio Ghilardi, and Elena Pagani. Counting constraints in flat array fragments. In *Proceedings of the 8th International Joint Conference on Automated Reasoning (IJCAR'16)*, volume 9706 of *Lecture Notes in Computer Science*, pages 65–81, 2016.

5 A. R. Balasubramanian, Javier Esparza, and Marijana Lazić. Complexity of verification and synthesis of threshold automata. In *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis (ATVA'20)*, volume 12302 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2020.

6 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the 2nd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'83)*, pages 27–30, 1983.

7 Idan Berkovits, Marijana Lazic, Giuliano Losa, Oded Padon, and Sharon Shoham. Verification of threshold-based distributed algorithms by decomposition to decidable logics. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV'19)*, volume 11562 of *Lecture Notes in Computer Science*, pages 245–266. Springer, 2019.

8 Piotr Berman and Juan A. Garay. Cloture votes: n/4-resilient distributed consensus in t+1 rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993.

9 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.

10 Luca Bortolussi and Jane Hillston. Model checking single agent behaviours by fluid approximation. *Information and Computation*, 242:183–226, 2015.

11 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.

12 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'09)*, volume 5797 of *Lecture Notes in Computer Science*, pages 93–106, 2009.

13 Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

14 Andrei Damian, Cezara Drăgoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV'19)*, volume 11562 of *Lecture Notes in Computer Science*, pages 344–363. Springer, 2019.

15 Nicolaas G. de Bruijn. A combinatorial problem. *Indagationes Mathematicae*, 49:758–764, 1946.

16 Cezara Drăgoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A Logic-Based Framework for Verifying Consensus Algorithms. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'14)*, volume 8318 of *Lecture Notes in Computer Science*, pages 161–181, 2014.

17 Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2(3):155–173, 1982.

18 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

19 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'17)*, pages 719–734, 2017.

20 Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Information and Computation*, 252:95–109, 2017.

21 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In *Proceedings of the 29th International Conference on Concurrency Theory (CONCUR'18)*, volume 118 of *LIPIcs*, pages 19:1–19:17, 2018.

22 Leslie Lamport. Checking a multithreaded algorithm with $^{+}$CAL. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC'06)*, volume 4167 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 2006.

23 Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, 1975.

24 Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff bounds for consensus algorithms. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV'17)*, volume 10427 of *Lecture Notes in Computer Science*, pages 217–237, 2017.

25 Kenneth L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'01)*, volume 2144, pages 179–195. Springer, 2001.

26 Dominique Méry. Verification by Construction of Distributed Algorithms. In *Proceedings of the 16th International Colloquium on Theoretical Aspects of Computing (ICTAC'19)*, volume 11884 of *Lecture Notes in Computer Science*, pages 22–38. Springer, 2019.

27 Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru, and Sébastien Tixeuil. Parameterized verification of algorithms for oblivious robots on a ring. In *Proceedings of the 17th International Conference on Formal Methods in Computer Aided Design (FMCAD'17)*, pages 212–219. IEEE, 2017.

28 Ocan Sankur and Jean-Pierre Talpin. An abstraction technique for parameterized model checking of leader election protocols: Application to FTSP. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 23–40, 2017. `doi:10.1007/978-3-662-54577-5_2`.

29 Ilina Stoilkovska, Igor Konnov, Josef Widder, and Florian Zuleger. Verifying safety of synchronous fault-tolerant algorithms by bounded model checking. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'19)*, volume 11428 of *Lecture Notes in Computer Science*, pages 357–374, 2019.