Parameterized Complexity of Safety of Threshold Automata

A. R. Balasubramanian
Technische Universität München, Germany
bala.ayikudi@tum.de

Abstract

Threshold automata are a formalism for modeling fault-tolerant distributed algorithms. In this paper, we study the parameterized complexity of reachability of threshold automata. As a first result, we show that the problem becomes W[1]-hard even when parameterized by parameters which are quite small in practice. We then consider two restricted cases which arise in practice and provide fixed-parameter tractable algorithms for both these cases. Finally, we report on experimental results conducted on some protocols taken from the literature.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Logic and verification

Keywords and phrases Threshold automata, distributed algorithms, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.37

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

Acknowledgements I would like to thank Prof. Javier Esparza for useful discussions regarding the paper and Christoph Welzel and Margarete Richter for their encouragement and support. I would also like to thank the anonymous reviewers whose comments greatly improved the presentation of the paper.

1 Introduction

Threshold automata [21] are a formalism for modeling and analyzing fault-tolerant distributed algorithms. In this setup, an arbitrary but fixed number of processes execute a given protocol as specified by a threshold automaton. Verification of these systems aims to prove these protocols correct for any number of processes [4].

One of the main differences between threshold automata and other formalisms for modeling distributed protocols (like replicated systems and population protocols [1, 16, 18]) is the notion of a *threshold guard*. Roughly speaking, a threshold guard specifies certain constraints that should hold between the number of messages received and the number of participating processes, in order for a transition to be enabled. For example, a guard of the form $x \geq n/2 - t$, (where x counts the number of messages of some specified type, n is the number of processes and t is the maximum number of faulty processes), specifies that the number of messages received should be bigger than $n/2 - t$, in order for the process to proceed. This feature is important in modeling fault-tolerant distributed algorithms where, often a process can make a move only if it has received a message from a majority or two-thirds of the number of processes. In a collection of papers [26, 3, 25, 24, 23], many algorithms have been developed for verifying various properties for threshold automata. These algorithms have then been used to verify a number of protocols from the distributed computing literature [24]. It is known that the reachability problem for threshold automata is NP-complete [2].

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).
Editors: Nitin Saxena and Sunil Simon; Article No. 37; pp. 37:1–37:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Parameterized complexity [13] attempts to study decision problems that come along with a *parameter*. In parameterized complexity, apart from the size of the input n, one considers further parameters k that capture the structure of the input and one looks for algorithms that run in time $f(k) \cdot n^{O(1)}$, where f is some function dependent on k alone. The hope is to find parameters which are quite small in practice and to base the dominant running time of the algorithm on this parameter alone. Problems solvable in such a manner are called fixed-parameter tractable (FPT).

In recent years, increasing effort has been devoted to studying the parameterized complexity of problems in verification for different models [10, 11, 15, 17, 9]. Motivated by this and by the hard theoretical complexity (NP-completeness) of reachability of threshold automata, we study the parameterized complexity of the same problem, parameterized by (among other parameters) the number of threshold guards and the given safety specification. Our first result is a parameterized equivalent of NP-hardness, which shows that reachability of threshold automata is W[1]-hard. However, motivated by practical concerns, we then study two restricted cases for which we provide fixed-parameter tractable algorithms. In the first case, we restrict ourselves to threshold automata whose underlying control structure is acyclic and provide a simple algorithm which reduces the size of the automaton to a function of the considered parameters. In the second case we consider multiplicative threshold automata where the number of *fall* guards is a constant. (For a definition of fall guards, see Section 2.1.) Roughly speaking, multiplicativity means that any run over a smaller population of processes can be "lifted" to a run over a bigger population as well. For this case, we use results from Petri net theory to provide an FPT algorithm. Finally, the usefulness of these cases is shown by a preliminary implementation of our algorithms on various protocols from the benchmark in [24]. Our implementation compares favorably with ByMC, the tool developed in [24] and also with the algorithm given in [2].

2 Preliminaries

We denote the set of non-negative integers by $\mathbb{N}_0$, the set of positive integers by $\mathbb{N}_{>0}$ and the set of all non-negative rational numbers by $\mathbb{Q}_{\geq 0}$.

2.1 Threshold Automata

We introduce threshold automata, mostly following the definition and notations used in [2]. Along the way, we also illustrate the definitions on the example of Figure 2 from [25], which is a model of the Byzantine agreement protocol of Figure 1.

Environments

Threshold automata are defined relative to an *environment* $Env = (\Pi, RC, Num)$, where Π is a set of *environment variables* ranging over $\mathbb{N}_0$, $RC \subseteq \mathbb{N}_0^{\Pi}$ is a *resilience condition* over the environment variables, expressible as a linear formula, and $Num\colon RC \to \mathbb{N}_0$ is a linear function called the *number function*. Intuitively, a valuation of Π determines the number of processes of different kinds (e.g. faulty) executing the protocol, and RC describes the admissible combinations of values of environment variables. Finally, Num associates to a each admissible combination, the number of copies of the automaton that are going to run in parallel, or, equivalently, the number of processes explicitly modeled. In a Byzantine setting, faulty processes behave arbitrarily, and so we do not model them explicitly; in this case, the system consists of one copy of the automaton for every correct process. In the crash fault model, processes behave correctly until they crash and they must be modeled explicitly.

```
1  var $myval_i \in \{0,1\}$
2  var $accept_i \in \{false, true\} \leftarrow false$
3
4  while true do (in one atomic step)
5   if $myval_i = 1$
6     and not sent ECHO before
7   then send ECHO to all
8
9   if received ECHO from at least
10      $t + 1$ distinct processes
11      and not sent ECHO before
12  then send ECHO to all
13
14  if received ECHO from at least
15      $n - t$ distinct processes
16  then $accept_i \leftarrow true$
17 od
```

Figure 1 Pseudocode of a reliable broadcast protocol from [28] for a correct process i, where n and t denote the number of processes, and an upper bound on the number of faulty processes. The protocol satisfies its specification (if $myval_i = 0$ for every correct process i, then no correct process sets its *accept* variable to *true*) if $t < n/3$.

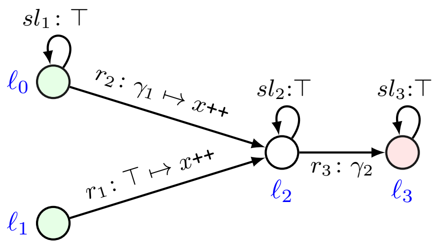

Figure 2 Threshold automaton from [25] modeling the body of the loop in the protocol from Fig. 1. Symbols γ_1, γ_2 stand for the threshold guards $x \geq (t+1) - f$ and $x \geq (n-t) - f$, where n and t are as in Fig. 1, and f is the actual number of faulty processes. The shared variable x models the number of ECHO messages sent by correct processes. Processes with $myval_i = b$ (line 1) start in location ℓ_b (in green). Rules r_1 and r_2 model sending ECHO at lines 7 and 12. The self-loop rules $sl_1, \ldots, sl_3$ are stuttering steps.

▶ **Example 1.** In the threshold automaton of Figure 2, the environment variables are n, f, and t, describing the number of processes, the number of (Byzantine) faulty processes, and the maximum possible number of faulty processes, respectively. The resilience condition is the constraint $n/3 > t \geq f$. The function *Num* is given by $Num(n,t,f) = n - f$, which is the number of correct processes.

Threshold automata

A *threshold automaton* over an environment *Env* is a tuple $\mathsf{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \mathcal{R})$, where $\mathcal{L}$ is a finite set of *local states* (or *locations*), $\mathcal{I} \subseteq \mathcal{L}$ is a nonempty subset of *initial locations*, Γ is a set of *shared variables* ranging over $\mathbb{N}_0$, and $\mathcal{R}$ is a set of *transition rules* (or just *rules*), formally described below.

A *transition rule* (or just a *rule*) is a tuple $r = (\mathit{from}, \mathit{to}, \varphi, \vec{u})$, where *from* and *to* are the *source* and *target* locations, $\varphi \subseteq \mathbb{N}_0^{\Pi \cup \Gamma}$ is a conjunction of *threshold guards* (described below), and $\vec{u} \colon \Gamma \to \{0,1\}$ is an *update*. We often let $r.\mathit{from}, r.\mathit{to}, r.\varphi, r.\vec{u}$ denote the components of r. Intuitively, r states that a process can move from *from* to *to* if the current values of Π and Γ satisfy φ, and when it moves, it updates the current valuation $\vec{\boldsymbol{g}}$ of Γ by performing the update $\vec{\boldsymbol{g}} := \vec{\boldsymbol{g}} + \vec{u}$. Since all components of $\vec{u}$ are nonnegative, the values of shared variables never decrease. A *threshold guard* φ has one of the following forms: $b \cdot x \bowtie a_0 + a_1 \cdot p_1 + \ldots + a_{|\Pi|} \cdot p_{|\Pi|}$ where $\bowtie \in \{\geq, <\}$, $x \in \Gamma$ is a shared variable, $p_1, \ldots, p_{|\Pi|} \in \Pi$ are the environment variables, $b \in \mathbb{N}_{>0}$ and $a_0, a_1, \ldots, a_{|\Pi|} \in \mathbb{Z}$ are integer coefficients. If $b = 1$, then the guard is called a simple guard. Additionally, if $\bowtie = \geq$, then the guard is called a *rise guard* and otherwise the guard is called a *fall guard*. We sometimes use $b \cdot x = a_0 + a_1 \cdot p_1 + \ldots + a_{|\Pi|} \cdot p_{|\Pi|}$ as a shorthand for $b \cdot x \geq a_0 + a_1 \cdot p_1 + \ldots + a_{|\Pi|} \cdot p_{|\Pi|} \wedge b \cdot x < (a_0 + 1) + a_1 \cdot p_1 + \ldots + a_{|\Pi|} \cdot p_{|\Pi|}$. Since shared variables are initialized to 0 and they never decrease, once a rise (resp. fall) guard becomes true (resp. false) it stays true (resp. false). We call this property *monotonicity of guards.* We let Φ^{rise}, Φ^{fall}, and Φ denote the sets of rise guards, fall guards, and all guards of TA. Finally, the *graph* of TA is the graph where the vertices are the locations and there is an edge between ℓ and ℓ' if there is a rule r in TA with $r.\mathit{from} = \ell$ and $r.\mathit{to} = \ell'$. We say that TA is acyclic if its graph is acyclic.

▶ **Example 2.** The rule r_2 of Figure 2 has ℓ_0 and ℓ_2 as source and target locations, $x \geq (t+1) - f$ as guard, and increments the value of the shared variable x by 1.

Configurations and transition relation

A *configuration* of TA is a triple $\sigma = (\vec{\boldsymbol{\kappa}}, \vec{\boldsymbol{g}}, \vec{\boldsymbol{p}})$ where $\vec{\boldsymbol{\kappa}} \colon \mathcal{L} \to \mathbb{N}_0$ describes the number of processes at each location, and $\vec{\boldsymbol{g}} \in \mathbb{N}_0^{|\Gamma|}$ and $\vec{\boldsymbol{p}} \in RC$ are valuations of the shared variables and the environment variables. In particular, $\sum_{\ell \in \mathcal{L}} \vec{\boldsymbol{\kappa}}(\ell) = \mathit{Num}(\vec{\boldsymbol{p}})$ always holds. A configuration is *initial* if $\vec{\boldsymbol{\kappa}}(\ell) = 0$ for every $\ell \notin \mathcal{I}$, and $\vec{\boldsymbol{g}} = \vec{0}$. We often let $\sigma.\vec{\boldsymbol{\kappa}}, \sigma.\vec{\boldsymbol{g}}, \sigma.\vec{\boldsymbol{p}}$ denote the components of σ.

A configuration $\sigma = (\vec{\boldsymbol{\kappa}}, \vec{\boldsymbol{g}}, \vec{\boldsymbol{p}})$ *enables* a rule $r = (\mathit{from}, \mathit{to}, \varphi, \vec{u})$ if $\vec{\boldsymbol{\kappa}}(\mathit{from}) > 0$, and $(\vec{\boldsymbol{g}}, \vec{\boldsymbol{p}})$ satisfies the guard φ, i.e., substituting $\vec{\boldsymbol{g}}(x)$ for x and $\vec{\boldsymbol{p}}(p_i)$ for p_i in φ yields a true expression, denoted by $\sigma \models \varphi$. If σ enables r, then TA can *move* from σ to the configuration $r(\sigma) = (\vec{\boldsymbol{\kappa}}', \vec{\boldsymbol{g}}', \vec{\boldsymbol{p}}')$ defined as follows: (i) $\vec{\boldsymbol{p}}' = \vec{\boldsymbol{p}}$, (ii) $\vec{\boldsymbol{g}}' = \vec{\boldsymbol{g}} + \vec{u}$, and (iii) $\vec{\boldsymbol{\kappa}}' = \vec{\boldsymbol{\kappa}} + \vec{v}_r$, where $\vec{v}_r = \vec{0}$ if $\mathit{from} = \mathit{to}$ and otherwise, $\vec{v}_r(\mathit{from}) = -1$, $\vec{v}_r(\mathit{to}) = +1$, and $\vec{v}_r(\ell) = 0$ for all other locations ℓ. We let $\sigma \to r(\sigma)$ denote that TA can move from σ to $r(\sigma)$.

Schedules and paths

A *schedule* is a (finite or infinite) sequence of rules. A schedule $\tau = r_1, \dots, r_m$ is *applicable* to configuration σ_0 if there is a sequence of configurations $\sigma_1, \dots, \sigma_m$ such that $\sigma_i = r_i(\sigma_{i-1})$ for $1 \leq i \leq m$, and we define $\tau(\sigma_0) := \sigma_m$. We let $\sigma \xrightarrow{*} \sigma'$ denote that $\tau(\sigma) = \sigma'$ for some schedule τ, and say that σ' is *reachable* from σ. Further we let $\tau \cdot \tau'$ denote the concatenation of two schedules τ and τ', and, given $\mu \geq 0$, let $\mu \cdot \tau$ the concatenation of τ with itself μ times.

A *path* or *run* is a finite or infinite sequence $\sigma_0, r_1, \sigma_1, \dots, \sigma_{k-1}, r_k, \sigma_k, \dots$ of alternating configurations and rules such that $\sigma_i = r_i(\sigma_{i-1})$ for every r_i in the sequence. If $\tau = r_1, \dots, r_{|\tau|}$ is applicable to σ_0, then we let $\mathsf{path}(\sigma_0, \tau)$ denote the path $\sigma_0, r_1, \sigma_1, \dots, r_{|\tau|}, \sigma_{|\tau|}$ with $\sigma_i = r_i(\sigma_{i-1})$, for $1 \leq i \leq |\tau|$. Similarly, if τ is an infinite schedule. Given a path $\mathsf{path}(\sigma, \tau)$, the set of all configurations in the path is denoted by $\mathsf{Cfgs}(\sigma, \tau)$.

The main focus of this paper will be the *reachability problem* and is defined as: Given TA and a set of locations $\mathcal{L}_{\mathsf{spec}} = \mathcal{L}_{=0} \cup \mathcal{L}_{>0}$ (called the specification), decide if there is a run of TA satisfying $\mathcal{L}_{\mathsf{spec}}$, i.e., decide if there is an initial configuration σ_0 such that some σ reachable from σ_0 satisfies $\sigma.\vec{\boldsymbol{\kappa}}(\ell) = 0$ for every $\ell \in \mathcal{L}_{=0}$ and $\sigma.\vec{\boldsymbol{\kappa}}(\ell) > 0$ for every $\ell \in \mathcal{L}_{>0}$. The *coverability problem* is the special case of the reachability problem where $\mathcal{L}_{=0} = \emptyset$.

2.2 Fixed-parameter tractability

We refer the reader to [13] for more information on parameterized complexity and only give the necessary definitions here. A *parameterized problem* L is a subset of $\Sigma^* \times \mathbb{N}_0$ for some alphabet Σ. A parameterized problem L is said to be *fixed-parameter tractable* (FPT) if there exists an algorithm A such that $(x, k) \in L$ iff $A(x, k)$ is true and A runs in time $f(k) \cdot |x|^{O(1)}$ for some computable function f, depending only on the *parameter* k. Given parameterized problems $L, L' \subseteq \Sigma^* \times \mathbb{N}_0$ we say that L is reducible to L' if there is an algorithm that, given an input (x, k), produces another input (x', k') in time $f(k) \cdot |x|^{O(1)}$ such that $(x, k) \in L \iff (x', k') \in L'$ and $k' \leq g(k)$ for some functions f and g depending only on k.

The parameterized clique problem is the set of all pairs (G, k) such that the graph G has a clique of size k. A parameterized problem L is said to be W[1]-hard if there is a parameterized reduction from L to the parameterized clique problem. If L is W[1]-hard and there is a parameterized reduction from L to L' then L' is W[1]-hard as well. W[1]-hardness is usually taken to be evidence that the problem does not have an FPT algorithm.

3 W[1]-hardness

We consider the reachability problem parameterized by the following parameters: $|\Phi|$ (the number of distinct guards), $|\mathcal{L}_{\mathsf{spec}}|$ (the size of the specification), $|RC|$ (the number of constraints in the resilience condition) and C (the maximum constant appearing in any of the guards of TA). (We note that if $x \in \Gamma \cup \Pi$ such that x does not appear in any of the guards in Φ or in any of the constraints in RC, then x can be removed from the input. Hence, we will always assume that $|\Gamma| + |\Pi| \leq |\Phi| + |RC|$ and for this reason, we do not consider $|\Gamma|$ and $|\Pi|$ explicitly as parameters.) In practice, all these values are quite small, roughly in the range of 10 to 25. Unfortunately, we prove the following negative result:

▶ **Theorem 3.** *Coverability (and hence reachability) for threshold automata parameterized by $|\Phi| + |\mathcal{L}_{spec}| + |RC| + C$ is $W[1]$-hard, even for acyclic automata where $|\Phi^{\text{fall}}|$ is a constant.*

Proof. We give a parameterized reduction from the *Unary Bin Packing* problem which is known to be $W[1]$-hard (See Theorem 2 of [20]) and is defined as follows:

> *Given:* A finite set of items $I = \{0, 1, 2, \ldots, w\}$, a size $size(i) \in \mathbb{N}_0$ for each $i \in I$, two positive integers B and k. (The integers $size(i)$ and B are encoded in unary)
> *Parameter:* k
> *Decide:* If there exists a partition of I into *bins* $I_1, \ldots, I_k$ such that the sum of the sizes of the items in each bin I_j is less than or equal to B

Let $size = \sum_{i \in I} size(i)$. Our parameterized reduction works as follows: We will have $k+1$ *environment variables* $c_1, c_2, \ldots, c_k, n$. Intuitively c_i will denote the sum of the sizes of the items in the i^{th} bin. The environment variable n will denote the number of processes modeled.

Further, we will have $k + 5$ *shared variables* $x_1, \ldots, x_k, access_1, access_2, access_3$ and $count_1, count_2$. The variable x_i will denote the sum of the sizes of items which **do not belong** to the i^{th} bin. The role of $count_1$ and $count_2$ will be to set up two counters whose value will be exactly $size$ and B respectively. Our construction will have three gadgets and the role of $access_1, access_2$ and $access_3$ is to ensure that exactly one process can enter the first, second and third gadgets respectively.

We will have exactly one initial location $start$ and three rules of the form $r_1 : (start, access_1 < 1, access_1{+}{+}, p_0)$, $r_2 : (start, access_2 < 1, access_2{+}{+}, q_0)$ and $r_3 : (start, access_3 < 1, access_3{+}{+}, \ell_0)$. This means that once a process fires r_1, it increments $access_1$ and hence no other process can fire r_1 in the future. Similarly for the rules r_2 and r_3. Hence these three rules ensure that at most one process can enter p_0, q_0 and ℓ_0 respectively. For the specification, we set $\mathcal{L}_{=0} = \emptyset$ and $\mathcal{L}_{>0} = \{p_{corr}, q_{corr}, \ell_{w+1}\}$, whose locations we will now explain.

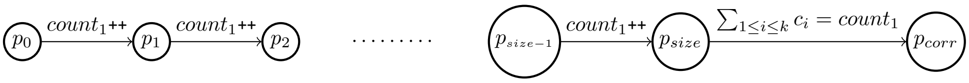

Figure 3 First gadget, which sets up the value of $count_1$ to be exactly $size$.

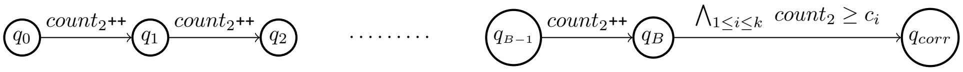

Figure 4 Second gadget, which sets up the value of $count_2$ to be exactly B.

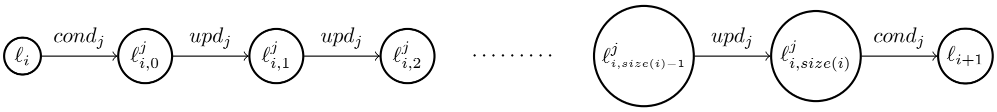

Figure 5 Third gadget, which guesses the partition. Here $cond_j$ is the condition $x_j \ge \sum_{l\ne j} c_l$ and upd_j is the update $\wedge_{l\ne j}\ x_l$++.

The first gadget is given by Figure 3 and starts from the location p_0. It increments the shared variable $count_1$ to the value $size$. This gadget then ensures that we can reach p_{corr} only if the sum of the values of the environment variables $c_1, c_2, \dots, c_k$ is exactly $size$. Notice that this gadget can be constructed in polynomial time, since each $size(i)$ is given in unary.

The second gadget is given by Figure 4 and starts from the location q_0. It increments the shared variable $count_2$ to the value B. This gadget then ensures that we can reach q_{corr} only if the values of the environment variables $c_1, c_2, \dots, c_k$ are all at most B. Notice that this gadget can be constructed in polynomial time, since B is given in unary.

The third gadget is comprised of locations $\{\ell_i\}_{0\le i\le w+1}$ and $\{\ell^j_{i,q}\}^{1\le j\le k}_{0\le i\le w, 0\le q\le size(i)}$ and is comprised of various mini-gadgets. For every $0 \le i \le w$ and $1 \le j \le k$, the third gadget has a mini-gadget as given by Figure 5.

Recall that the shared variable x_j denotes the the sum of the sizes of items which **do not** belong to the j^{th} bin. Intuitively, if a process moves from ℓ_i to ℓ_{i+1} by going through $\ell^j_{i,0}, \dots, \ell^j_{i,size(i)}$, this corresponds to putting the i^{th} item in the j^{th} bin and hence the mini-gadget increments the variables $\{x_l\}_{l\ne j}$ by the value $size(i)$. To ensure that we do not overshoot the bin size of the j^{th} bin, we have the guards $x_j \ge \sum_{l\ne j} c_l$ at the beginning and the end of the mini-gadget. Recall that the first gadget ensures that $\sum_{1\le l\le k} c_l = size$ and since x_j denotes the sum of sizes of items **not in** the j^{th} bin, the condition $x_j \ge \sum_{l\ne j} c_l$ ensures that the sum of the sizes of the items in the j^{th} bin is at most c_j. Since the second gadget forces $c_j \le B$, it follows that the test $x_j \ge \sum_{l\ne j} c_l$ ensures that the sum of the sizes **in** the j^{th} bin is at most B. Notice that, once again this gadget can be constructed in polynomial time, since each $size(i)$ is given in unary.

Let RC be $n > 1$ and let $Num(c_1, \dots, c_k, n) = n$. From the given construction it is clear that a configuration satisfying $\mathcal{L}_{\mathsf{spec}}$ is reachable iff we can partition I into k bins such that the sum of sizes of items in each bin does not exceed B.

It is clear that the reduction can be accomplished in polynomial time. Notice that the automaton is acyclic, $\mathcal{L}_{=0} = \emptyset$, $|\Phi| = O(k)$, $|\Phi^{\mathrm{fall}}| = 4$, $|RC| = 1$, $C = 1$ and $|\mathcal{L}_{\mathsf{spec}}| = 3$. Hence it is clear that $|\Phi| + |RC| + C + |\mathcal{L}_{\mathsf{spec}}| = O(k)$ and so the above reduction is indeed a parameterized reduction from the unary bin packing problem to the coverability problem. ◀

We now identify two special cases for which we give an FPT algorithm and discuss how these special cases arise in practice for a variety of distributed algorithms.

4 Acyclic threshold automata

The first case we consider is that of acyclic threshold automata, i.e., threshold automata whose underlying graph is acyclic. Except for one protocol, all the others in the benchmark of [24] are acyclic.[1] As the reduction of Theorem 3 produces acyclic threshold automata, we cannot hope for an FPT algorithm parameterized by $\{|\Phi|, |\mathcal{L}_{\mathsf{spec}}|, |RC|, C\}$. However, we show that

▶ **Theorem 4.** *Reachability of acyclic threshold automata parameterized by $|\Phi| + |\mathcal{L}_{spec}| + |RC| + C + D$ is in FPT, where D is the length of the longest path in the graph of the threshold automaton.*

Proof. Let TA be the given acyclic threshold automaton. First, we show that it is possible to incrementally "contract" the locations of TA in a bottom-up manner, while preserving the reachability property, such that, in the resulting automaton after contraction, the number of locations and rules is a function of $|\Phi| + |\mathcal{L}_{\mathsf{spec}}| + |RC| + C + D$. This then immediately implies our theorem, since the size of the whole automaton is now just a function of $|\Phi| + |\mathcal{L}_{\mathsf{spec}}| + |RC| + C + D$.

More formally, let the contraction of a subset $S = \{\ell_1, \dots, \ell_q\}$ of locations of TA be the following operation: We remove the locations $\ell_1, \dots, \ell_q$ from TA, introduce a new location ℓ_S and we replace all occurrences of $\ell_1, \dots, \ell_q$ in every rule of TA with ℓ_S. We say that a set S in TA is *good* if for every two locations $\ell, \ell' \in S$, if $(\ell, \ell'', \phi, \vec{u})$ is a rule in TA then $(\ell', \ell'', \phi, \vec{u})$ is also a rule in TA. Intuitively, this means that, for every rule that we can fire from ℓ, there is another rule we can fire from ℓ' which will have the exact same effect. Since TA is assumed to be acyclic, contracting a good set cannot introduce cycles. Let $\mathtt{Tar} = \{\ell : \ell \in \mathcal{L}_{\mathsf{spec}}\}$. The following is a very simple fact to verify:

> *Claim:* Suppose S is a good set such that $S \cap \mathtt{Tar} = \emptyset$ and let TA' be the threshold automaton obtained by contracting S in TA. Then TA satisfies $\mathcal{L}_{\mathsf{spec}}$ iff TA' does.

Given a threshold automaton TA such that D is the length of the longest path in its graph, the "layers" of TA is a partition of the locations into subsets $L_0^{\mathsf{TA}}, L_1^{\mathsf{TA}}, \dots, L_D^{\mathsf{TA}}$ such that $\ell \in L_i^{\mathsf{TA}}$ iff the longest path ending at ℓ in the graph of TA is of length i. The subset L_i^{TA} will be called the i^{th} layer of TA. We will now construct a sequence of threshold automata $\mathsf{TA}_D, \mathsf{TA}_{D-1}, \dots, \mathsf{TA}_0$ such that for each i, $|L_i^{\mathsf{TA}_i}| + |L_{i+1}^{\mathsf{TA}_i}| + \dots + |L_D^{\mathsf{TA}_i}| \leq g_i(|\Phi|, |RC|, |\mathcal{L}_{\mathsf{spec}}|, D)$ for some function g_i and such that TA_i satisfies $\mathcal{L}_{\mathsf{spec}}$ iff TA_{i+1} does.

For the base case of TA_D, we take the threshold automaton TA and consider the set $S_D := L_D^{\mathsf{TA}} \setminus \mathtt{Tar}$. We now contract S_D in TA to get a threshold automaton TA_D. Notice that S_D is a good set and by the above claim, TA_D satisfies $\mathcal{L}_{\mathsf{spec}}$ iff TA does.

For the induction step, suppose we have already constructed TA_{i+1}. For a location $\ell \in L_i^{\mathsf{TA}_{i+1}}$, define its *color* to be the set $\{(\ell', \phi, \vec{u}) : (\ell, \ell', \phi, \vec{u}) \text{ is a rule in } \mathsf{TA}_{i+1}\}$. Observe that if $\ell \in L_i^{\mathsf{TA}_{i+1}}$ and $(\ell, \ell', \phi, \vec{u})$ is a rule in TA_{i+1} then $\ell' \in L_{i+1}^{\mathsf{TA}_{i+1}} \cup L_{i+2}^{\mathsf{TA}_{i+1}} \cup \dots \cup L_D^{\mathsf{TA}_{i+1}}$. By induction hypothesis, $|L_{i+1}^{\mathsf{TA}_{i+1}} \cup L_{i+2}^{\mathsf{TA}_{i+1}} \cup \dots \cup L_D^{\mathsf{TA}_{i+1}}| \leq g_{i+1}(|\Phi|, |RC|, |\mathcal{L}_{\mathsf{spec}}|, D)$ for

[1] Some of the examples have self-loops on some locations, but since these self-loops do not update any of the shared variables, we can remove them without affecting the reachability relation.

some function g_{i+1}. It then follows that the number of possible colors is at most $2^{|\Phi|} \cdot 2^{|\Gamma|} \cdot g_{i+1}(|\Phi|, |\mathcal{L}_{\mathtt{spec}}|, D)$. Hence as long as the number of locations in $L_i^{\mathsf{TA}_{i+1}}$ is bigger than $2^{|\Phi|} \cdot 2^{|\Gamma|} \cdot g_{i+1}(|\Phi|, |\mathcal{L}_{\mathtt{spec}}|, D) + |\mathtt{Tar}|$ there will be two locations in $L_i^{\mathsf{TA}_{i+1}} \setminus \mathtt{Tar}$ which have the same color and can hence be contracted while maintaining the answer for $\mathcal{L}_{\mathtt{spec}}$. It then follows that by repeated contraction, we can finally end up at a threshold automaton TA_i such that $|L_i^{\mathsf{TA}_i}| + \cdots + |L_D^{\mathsf{TA}_i}| \leq O(2^{|\Phi|} \cdot 2^{|\Phi|+|RC|} \cdot g_{i+1}(|\Phi|, |RC|, |\mathcal{L}_{\mathtt{spec}}|, D) + |\mathtt{Tar}|)$. Taking this bound to be the function g_i, we get our required TA_i.[2]

Notice that the number of locations (and also rules) in TA_0 is only dependent on $|\Phi|, |RC|, |\mathcal{L}_{\mathtt{spec}}|$ and D. Since the reachability problem is decidable, it immediately follows that we have a parameterized algorithm for acyclic threshold automata running in time $f(|\Phi| + |RC| + |\mathcal{L}_{\mathtt{spec}}| + C + D) \cdot n^{O(1)}$ ◀

5 Threshold automata with constantly many fall guards

As a second case, we consider threshold automata in which the number of fall guards is a constant. In almost all of the benchmarks of [24], the number of fall guards is at most one. We provide some intuitive reason behind this phenomenon. In threshold automata, shared variables are usually used for two things: To record that some process has sent a message or to keep track of the number of processes which have crashed so far. If a shared variable v is used for the first purpose, then all guards containing v are typically rise guards, since we only want to check that enough messages have been received to proceed. On the other hand, if v is used to keep track of the number of crashed processes, then we will have a fall guard which allows a process to crash only if the value of v is less than the maximum number of processes allowed to crash. However, since we will only need one fall guard for this purpose, it follows that in practice we can hope to have very few fall guards in a threshold automaton.

Since the reduction of Theorem 3 produces threshold automata with constantly many fall guards, we need another restriction on this class as well, which we now describe.

▶ **Definition 5.** *A threshold automaton TA over an environment Env* $= (\Pi, RC, Num)$ *is called multiplicative if every fall guard is simple and for every* $\mu \in \mathbb{N}_{>0}$, *(i) for every rational vector* $\mathbf{p} \in \mathbb{Q}_{\geq 0}^{|\Pi|}$, *if* $RC(\mathbf{p})$ *is true then* $RC(\mu \cdot \mathbf{p})$ *is true and* $Num(\mu \cdot \mathbf{p}) = \mu \cdot Num(\mathbf{p})$ *and (ii) for every guard* $g := b \cdot x \bowtie a_0 + a_1 p_1 + \cdots + a_l p_l$ *in TA where* $\bowtie \in \{\geq, <\}$, *if* $(y, q_1, \ldots, q_l)$ *is a rational solution to* g *then* $(\mu \cdot y, \mu \cdot q_1, \ldots, \mu \cdot q_l)$ *is also a solution to* g.

To the best of our knowledge, many algorithms discussed in the literature (For example, see [8, 28, 6, 27, 19, 14, 7]), and more than two-thirds of all of the benchmarks of [24] satisfy multiplicativity. The main result of this section is

▶ **Theorem 6.** *Given a multiplicative threshold automaton TA with a constant number of fall guards and a specification* $\mathcal{L}_{spec}$, *it can be decided in time* $f(|\Phi|) \cdot n^{O(1)}$ *whether there is a run of TA satisfying* $\mathcal{L}_{spec}$.

The rest of this section is devoted to proving this result, which we do so in four parts. Let us fix a threshold automaton $\mathsf{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \mathcal{R})$, an environment $Env = (\Pi, RC, Num)$ and a specification $\mathcal{L}_{\mathtt{spec}}$ for the rest of this section. Let Φ denote the set of all guards which appear in TA.

[2] Though the function g_i as given here gives very huge bounds, we show in the experimental section that repeated contractions can sometimes reduce the number of locations by 50%. Intuitively, this is because the number of colors of a location in the benchmarks is much smaller than the worst-case analysis performed here.

First part: Decomposing paths into steady paths

First, similar to the paper [22], we show that the job of finding a path satisfying $\mathcal{L}_{\mathsf{spec}}$ can be reduced to that of finding a bounded number of concatenated “steady” paths. However, the result needs to be stated in a different manner than [22], so that later on, we could leverage the fact that the threshold automaton TA contains only constantly many fall guards.

A *context* ω is any subset of the guards of TA, i.e., $\omega \subseteq \Phi$. A rule r is said to be *activated* by a context ω if all the rise guards of r are present in ω and all the fall guards of r are **not** present in ω. The set of all rules activated by a context ω is denoted by $\mathcal{R}_\omega$.

The *context* of a configuration σ, denoted by $\omega(\sigma)$, is the set of all *rise* guards that evaluate to true and the set of all *fall* guards that evaluate to false in σ. Since the values of the shared variables can only increase along a path, it easily follows that for any configuration σ and any schedule τ applicable to σ, $\omega(\sigma) \subseteq \omega(\tau(\sigma))$.

We say that $\mathsf{path}(\sigma,\tau)$ is *ω-steady* if all the rules in the schedule τ are from $\mathcal{R}_\omega$ and for every configuration $\sigma' \in \mathsf{Cfgs}(\sigma,\tau)$, we have $\mathcal{R}_\omega \subseteq \mathcal{R}_{\omega(\sigma')}$. Intuitively, if $\mathsf{path}(\sigma,\tau)$ is ω-steady then the path only uses rules from $\mathcal{R}_\omega$. We have the following lemma.

▶ **Lemma 7.** *The specification $\mathcal{L}_{spec}$ can be satisfied by a path of TA iff there exists $K \leq |\Phi|$, configurations $\sigma_0, \sigma'_0, \dots, \sigma_K, \sigma'_K$ and contexts $\omega_0 \subsetneq \omega_1 \subsetneq \dots \subsetneq \omega_K$ such that*

- *σ_0 is an initial configuration and σ'_K satisfies $\mathcal{L}_{spec}$*
- *For every $i \leq K$, there is a ω_i-steady path $\sigma_i \xrightarrow{*} \sigma'_i$*
- *For every $i < K$, if $\mathcal{R}_{\omega_i} \subseteq \mathcal{R}_{\omega_{i+1}}$ then there is a ω_i-steady path $\sigma'_i \xrightarrow{*} \sigma_{i+1}$, otherwise $\sigma'_i \to \sigma_{i+1}$*

Proof. (Sketch.) Clearly if there exists such configurations and contexts then then there exists a path of TA which satisfies $\mathcal{L}_{\mathsf{spec}}$. To prove the other direction, suppose $\mathsf{path}(\sigma_0,\tau)$ is a path of TA which satisfies $\mathcal{L}_{\mathsf{spec}}$. Using the fact that $\omega(\sigma') \subseteq \omega(\tau'(\sigma'))$ for any configuration σ' and any schedule τ', we can decompose $\mathsf{path}(\sigma_0,\tau)$ into $\sigma_0, \tau_0, \sigma'_0, t_0, \sigma_1, \tau_1, \sigma'_1, t_1, \dots, \sigma_K, \tau_K, \sigma'_K$ such that for every i, $\omega(\sigma_i) = \omega(\sigma'_i)$, $\omega(\sigma'_i) \subsetneq \omega(\sigma_{i+1})$ and t_i is a rule of TA. We can then prove that the configurations $\sigma_0, \sigma'_0, \dots, \sigma_K, \sigma'_K$ and the contexts $\omega(\sigma_0), \dots, \omega(\sigma_K)$ satisfy the required conditions. ◀

Second part: Establishing a connection between continuous Petri nets and steady paths

Let us fix a context ω of the threshold automaton TA for the rest of this subsection. We say that a configuration σ is $\mathcal{R}_\omega$-applicable if $(\sigma.\vec{\boldsymbol{g}}, \sigma.\vec{\boldsymbol{p}})$ satisfies every guard of every rule in $\mathcal{R}_\omega$.

Continuous Petri nets

To define continuous Petri nets, we will mostly reuse the same notations from [5]. A continuous Petri net N is a tuple (P,T,F) where P is a finite set of places, T is a finite set of transitions and $F \subseteq P \times T \cup T \times P$ is the flow relation. For a transition t, let ${}^\bullet t = \{p : (p,t) \in F\}$ and $t^\bullet = \{p : (t,p) \in F\}$. A marking M of N is a function $M : P \to \mathbb{Q}_{\geq 0}$. Intuitively a marking M assigns $M(p)$ many *tokens* to each place $p \in P$. A marking is called integral if $M(p) \in \mathbb{N}_0$ for every place p. Given a marking M and a $k \in \mathbb{N}_{>0}$ let kM denote the marking $kM(p) = k \cdot M(p)$. The transition relation between two markings M and M' is defined as follows: For $\alpha \in (0,1]$ and $t \in T$, we say that $M \xrightarrow{\alpha t} M'$ if for every $p \in {}^\bullet t$, $M(p) \geq \alpha$ and $M'(p) = M(p) - \alpha$ if $p \in {}^\bullet t \setminus t^\bullet$, $M'(p) = M(p) + \alpha$ if $p \in t^\bullet \setminus {}^\bullet t$ and $M'(p) = M(p)$ otherwise. We say that $M \to M'$ if $M \xrightarrow{\alpha t} M'$ for some α and t. Finally we say that $M \xrightarrow{*} M'$ if there exists $M_1, \dots, M_{k-1}$ such that $M \to M_1 \to \dots M_{k-1} \to M'$.

Constructing continuous Petri nets from contexts

We now construct a *continuous Petri net* N_ω for the context ω as follows: For every location ℓ of TA, we will have a place p_ℓ. Similarly for every variable $x \in \Gamma \cup \Pi$, we will have a place p_x. If $r = (\ell, \ell', \phi, \vec{u})$ is a rule in $\mathcal{R}_\omega$, we will have a transition t_r where ${}^\bullet t_r = \{p_\ell\}$ and $t_r^\bullet = \{p_{\ell'}\} \cup \{p_x : \vec{u}[x] = 1\}$.

We note that N_ω tries to simulate exactly the rules of $\mathcal{R}_\omega$, but it does not check whether the corresponding guard of a rule is true before firing it. To ensure that a proper simulation is carried out by N_ω, we will restrict ourselves to only runs of N_ω over *compatible markings* which are defined as follows.

A marking M of N_ω is called a *compatible marking* if $\sum_{\ell \in \mathcal{L}} M(p_\ell) = Num(\{M(p_x) : x \in \Pi\})$ and if for every $x \in \Gamma \cup \Pi$, the assignment $x \mapsto M(p_x)$ satisfies the resilience condition RC and all the guards of all the rules in $\mathcal{R}_\omega$. Notice that to every $\mathcal{R}_\omega$-applicable configuration σ of TA we can bijectively assign a canonical *compatible integral* marking $\mathbb{B}(\sigma)$ of N_ω where $(\mathbb{B}(\sigma))(p_x) = \sigma[x]$.

▶ **Proposition 8.** *The following are true:*

- *Suppose $\sigma \xrightarrow{*} \sigma'$ is an ω-steady run of TA. Then $\mathbb{B}(\sigma) \xrightarrow{*} \mathbb{B}(\sigma')$ in N_ω.*
- *Suppose M and M' are compatible markings of N_ω such that $M \xrightarrow{*} M'$. Then there exists $\mu \in \mathbb{N}_{>0}$ such that for all $k \in \mathbb{N}_{>0}$, $\mu k M$ and $\mu k M'$ are compatible integral markings and $\mathbb{B}^{-1}(\mu k M) \xrightarrow{*} \mathbb{B}^{-1}(\mu k M')$ is an ω-steady run of TA.*

Proof. (Sketch.) The first point is obvious from the definition. For the second point, if $M := M_0 \xrightarrow{\alpha_1 t_{r_1}} M_1 \xrightarrow{\alpha_2 t_{r_2}} M_2 \dots M_{l-1} \xrightarrow{\alpha_l t_{r_l}} M_l := M'$ is a run, then by multiplying the markings by the least common multiple of the denominators of $\{\alpha_i\}_{i \le l} \cup \{M_i(p_x) : i \le l,\ x \in \mathcal{L} \cup \Gamma \cup \Pi\}$ (which we take to be μ), we can get an integral run between $\mu k M$ and $\mu k M'$. Using multiplicativity of TA, we can translate this back to a run of TA. ◀

Third part: Characterizing steady paths

It was shown in ([5], Theorems 3.6 and 3.3) that there is a logic (which the authors of [5] call *convex semi-linear Horn formulas*) characterizing reachability in continuous Petri nets, whose satisfiability can be tested **in polynomial time**. Using this result, proposition 8 and multiplicativity, we show that

▶ **Lemma 9.** *Given a context ω, in polynomial time we can construct a convex semi-linear Horn formula $\phi_\omega(\mathbf{x}, \mathbf{y})$ with $2(|\mathcal{L}| + |\Gamma| + |\Pi|)$ free variables such that*

- *If $\sigma \xrightarrow{*} \sigma'$ is an ω-steady path of TA then $\phi_\omega(\sigma, \sigma')$ is true*
- *Suppose $\phi_\omega(M, M')$ is true. Then there exists $\mu \in \mathbb{N}$ such that for all $k \in \mathbb{N}$, $\mu k M, \mu k M'$ are configurations of TA such that $\mu k M \xrightarrow{*} \mu k M'$ is an ω-steady path in TA.*

▶ **Lemma 10.** *Given a rule r of TA, in polynomial time we can construct a convex semi-linear Horn formula $\phi_r(\mathbf{x}, \mathbf{y})$ with $2(|\mathcal{L}| + |\Gamma| + |\Pi|)$ free variables such that*

- *If σ and σ' are configurations of TA such that $\sigma' = r(\sigma)$, then $\phi_r(\sigma, \sigma')$ is true.*
- *Suppose $\phi_r(M, M')$ is true. Then there exists $\mu \in \mathbb{N}$ such that for all $k \in \mathbb{N}$, $\mu k M, \mu k M'$ are configurations of TA such that $\mu k M' = (\mu k \cdot r)(\mu k M)$, i.e., $\mu k M'$ can be obtained by applying the rule r to $\mu k M$, repeatedly for μk many steps.*

Fourth part: Bringing it all together

▶ **Theorem 11.** *Given a multiplicative threshold automaton TA with constant number of fall guards and a specification $\mathcal{L}_{spec}$, it can be decided in time $f(|\Phi|) \cdot n^{O(1)}$ whether there is a run of TA satisfying $\mathcal{L}_{spec}$.*

Proof. (Sketch.) One can easily show that if we have a monotonically increasing context sequence $\omega_0 \subsetneq \omega_1 \subsetneq \cdots \subsetneq \omega_K$, the size of the set $\{j : \mathcal{R}_{\omega_j} \not\subseteq \mathcal{R}_{\omega_{j+1}}\}$ is at most $|\Phi^{\text{fall}}|$. Using this observation, we proceed as follows. We iterate over all $K \leq |\Phi|$ and over all possible monotonically increasing context sequences $\omega_0 \subsetneq \omega_1 \subsetneq \cdots \subsetneq \omega_K$ of length $K+1$ and all possible rule sequences $r_1, \ldots, r_c$ of length $c = \#\{j : \mathcal{R}_{\omega_j} \not\subseteq \mathcal{R}_{\omega_{j+1}}\}$. Note that the number of such iterations is at most $O(|\Phi| \cdot |\Phi^{\text{fall}}| \cdot |\Phi|! \cdot 2^{|\Phi|} \cdot |\mathcal{R}|^{|\Phi^{\text{fall}}|})$. Since $|\Phi^{\text{fall}}|$ is assumed to be a constant, the exponential dependence only lies upon $|\Phi|$.

A position $0 \leq l \leq K$ is called *bad* if $\mathcal{R}_{\omega_l} \not\subseteq \mathcal{R}_{\omega_{l+1}}$. Let $j_1, \ldots, j_c$ be the set of all bad positions. Using lemmas 9 and 10 we can write down the following convex semi-linear Horn formula in polynomial time:

$$\xi_0(\mathbf{x}_0, \mathbf{y}_0, \mathbf{x}_1) \wedge \xi_1(\mathbf{x}_1, \mathbf{y}_1, \mathbf{x}_2) \wedge \cdots \wedge \xi_{K-1}(\mathbf{x}_{K-1}, \mathbf{y}_{K-1}, \mathbf{x}_K) \wedge \xi_K(\mathbf{x}_K, \mathbf{y}_K) \tag{1}$$

where $\xi_K(\mathbf{x}_K, \mathbf{y}_K) = \phi_{\omega_K}(\mathbf{x}_K, \mathbf{y}_K)$ and ξ_i for $i < K$ is defined as follows: If i is a bad position, i.e., if $i = j_l$ for some $1 \leq l \leq c$, then $\xi_i(\mathbf{x}_i, \mathbf{y}_i, \mathbf{x}_{i+1}) = \phi_{\omega_i}(\mathbf{x}_i, \mathbf{y}_i) \wedge \phi_{r_l}(\mathbf{y}_i, \mathbf{x}_{i+1})$. It i not a bad position, then $\xi_i(\mathbf{x}_i, \mathbf{y}_i, \mathbf{x}_{i+1}) = \phi_{\omega_i}(\mathbf{x}_i, \mathbf{y}_i) \wedge \phi_{\omega_i}(\mathbf{y}_i, \mathbf{x}_{i+1})$

To equation (1), we also add a constraint stating that $\mathbf{x}_0$ is an initial configuration and $\mathbf{y}_K$ satisfies $\mathcal{L}_{\mathsf{spec}}$. By proposition 8 we can then easily show that, there is a run of TA satisfying $\mathcal{L}_{\mathsf{spec}}$ iff in at least one iteration, the constructed formula (1) is satisfiable. ◀

6 NP-hardness of multiplicative threshold automata

A natural question arises from the results of the previous section. Can we do better than fixed-parameter tractability and instead solve the reachability problem for multiplicative threshold automata in polynomial time? We remark that the proof of NP-hardness of reachability for threshold automata given in [2] does not produce multiplicative threshold automata and hence does not answer this question. Nevertheless, we show that it is unlikely for reachability of multiplicative threshold automata to be in polynomial time.

▶ **Theorem 12.** *Coverability (and hence reachability) for multiplicative threshold automata is NP-hard even when there are no fall guards.*

Proof. We give an easy reduction from 3-SAT. Let φ be a propositional formula with variables $x_1, \ldots, x_k$ and clauses $C_1, \ldots, C_m$. We will have $2k$ shared variables $y_1, \ldots, y_k, \bar{y_1}, \ldots, \bar{y_k}$ and one environment variable n, denoting the number of processes. Incrementing y_i ($\bar{y_i}$ resp.) corresponds to setting x_i to true (false resp). We will have $2k+1$ locations $\ell_0, \ell'_0, \ell_1, \ell'_1, \ldots, \ell'_{k-1}, \ell_k$. Between ℓ_i and ℓ'_i we will have two rules which increment y_i and $\bar{y_i}$ respectively. To ensure that all the processes increment the same variable, we have two rules from ℓ'_i to ℓ_{i+1} which test that $y_i \geq n$ and $\bar{y_i} \geq n$ respectively. Hence if one process increments y_i and another increments $\bar{y_i}$, then all the processes get stuck at ℓ'_i.

Let $var(x_i) = y_i$ and let $var(\bar{x_i}) = \bar{y_i}$. We will then have m locations $\ell_{k+1}, \ell_{k+2}, \ldots, \ell_{k+m}$ and the following rules between ℓ_{k+i-1} and ℓ_{k+i} for every $1 \leq i \leq m$: If the clause C_i is of the form $a \vee b \vee c$ then there are three rules between ℓ_{k+i-1} and ℓ_{k+i}, each checking if $var(a) \geq 1$, $var(b) \geq 1$ and $var(c) \geq 1$ respectively. Hence if either one of $var(a)$ or $var(b)$ or $var(c)$ was incremented, the processes could move from ℓ_{k+i-1} to ℓ_{k+i}, otherwise all the processes get stuck at ℓ_{k+i-1}. Finally we set the initial location to be ℓ_0 and the specification to be $\mathcal{L}_{=0} = \emptyset$ and $\mathcal{L}_{>0} = \{\ell_{k+m}\}$. It is then easy to see that φ is satisfied iff there is a run which satisfies $\mathcal{L}_{\mathtt{spec}}$. ◀

7 Experiments

We implemented the contraction procedure for the acyclic threshold automata as presented in section 4 and then used the algorithm for multiplicative threshold automata presented in section 5. To leverage the solid engineering work that has been put into modern SMT solvers, we used the Z3 solver to solve the convex semi-linear Horn formulas as well as to choose a context (and rule) sequence. We applied our implementations to all the multiplicative protocols in the latest version of the benchmark of [24], which contains various algorithms taken from the distributed computing literature. For more information on the protocols, we refer the reader to the benchmark of [24].

Table 1 The experiments were run on a machine with Intel® Core$^{\text{TM}}$ i5-7200U CPU with 7.7 GiB memory. The time limit was set to be 2 hours and the memory limit was set to be 7 GiB. TLE (MLE) means that the time limit (memory limit) exceeded for the particular benchmark.

Input	Case	Time, seconds		
	(if more than one)	This paper	Algo from [2]	ByMC
frb		0.38	0.32	0.07
frb	hand-coded TA	0.29	0.31	0.16
strb		0.44	0.43	0.14
strb	hand-coded TA	0.32	0.30	0.10
nbacg		2.92	8.43	9.71
aba	Case 1	4.49	10.26	25.6
aba	Case 2	18.29	41.92	704.9
cbc	Case 1	3579.24	MLE	MLE
cbc	Case 2	183.61	2035.5	26.37
cbc	Case 3	MLE	MLE	MLE
cbc	Case 4	MLE	MLE	MLE
cbc	hand-coded TA	3.27	0.91	0.26
cf1s	Case 1	13.81	13.53	37.09
cf1s	Case 2	12.47	16.14	186.5
cf1s	Case 3	84.95	86.98	7875
cf1s	hand-coded TA	1.75	1.31	2737.53
c1cs	Case 1	179.39	598.2	TLE
c1cs	Case 2	70.77	747.86	7119.71
c1cs	Case 3	604.91	1575.21	MLE
c1cs	hand-coded TA	4.87	6.63	TLE

Evaluation: Table 1 summarizes our results and compares them with the results obtained using ByMC, the tool presented in [24] and the algorithm from [2].

For some safety specifications, our contraction procedure was able to reduce the number of locations by more than 50% for the cbc protocol(s). This helped us save some memory, as we also noticed that running just the algorithm for multiplicative threshold automata took much more memory and the algorithm was not able to complete its execution. Our implementation compares favorably with both ByMC and the algorithm from [2] in some cases, but also performs worse in some of the hand-coded examples, the second case of cbc and the frb and strb protocols.

8 Conclusion

In this paper, we have investigated the parameterized complexity of safety in threshold automata. Though we have proved hardness results even in very restricted settings, we have also identified tractable special cases which arise in practice. A preliminary implementation of our algorithms suggest that these methods might be useful in practice as well.

For the sake of simplicity, we have only restricted to verifying safety properties in this paper. A special type of logic called $\mathsf{ELTL}_{\mathsf{FT}}$ [12] has been proposed for threshold automata which can express various safety and liveness properties. Since model checking this logic decomposes to a finite number of safety specifications (modulo some technical constraints), we believe that our algorithm for multiplicative threshold automata can be adapted to give an algorithm for model checking this logic as well.

References

1 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006. doi:10.1007/s00446-005-0138-3.

2 A. R. Balasubramanian, Javier Esparza, and Marijana Lazić. Complexity of verification and synthesis of threshold automata. In *Accepted at ATVA 2020*, 2020. URL: https://arxiv.org/abs/2007.06248.

3 Nathalie Bertrand, Igor Konnov, Marijana Lazić, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. In *CONCUR*, volume 140 of *LIPIcs*, pages 33:1–33:15, 2019.

4 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.

5 Michael Blondin and Christoph Haase. Logics for continuous reachability in petri nets and vector addition systems with states. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005068.

6 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

7 Francisco Vilar Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *PaCT*, volume 2127 of *LNCS*, pages 42–50, 2001.

8 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

9 Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. On the complexity of bounded context switching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.27.

10 Peter Chini, Roland Meyer, and Prakash Saivasan. Fine-grained complexity of safety verification. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki,*

Greece, April 14-20, 2018, Proceedings, Part II, volume 10806 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2018. doi:10.1007/978-3-319-89963-3_2.

11 Peter Chini, Roland Meyer, and Prakash Saivasan. Complexity of liveness in parameterized systems. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 37:1–37:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.37.

12 Peter Chini, Roland Meyer, and Prakash Saivasan. Liveness in broadcast networks. In *NETYS 2019, Revised Selected Papers*, pages 52–66, 2019.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 Dan Dobre and Neeraj Suri. One-step consensus with zero-degradation. In *DSN*, pages 137–146, 2006.

15 Constantin Enea and Azadeh Farzan. On atomicity in presence of non-atomic writes. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 497–514. Springer, 2016. doi:10.1007/978-3-662-49674-9_29.

16 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359. IEEE Computer Society, 1999.

17 Azadeh Farzan and P. Madhusudan. The complexity of predicting atomicity violations. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2009. doi:10.1007/978-3-642-00768-2_14.

18 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

19 Rachid Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15(1):17–25, 2002.

20 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.

21 Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR*, volume 8704 of *LNCS*, pages 125–140, 2014.

22 Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2015. doi:10.1007/978-3-319-21690-4_6.

23 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.

24 Igor Konnov and Josef Widder. Bymc: Byzantine model checker. In *ISoLA (3)*, volume 11246 of *LNCS*, pages 327–342. Springer, 2018.

25 Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL 2017*, pages 719–734, 2017.

26 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In *CONCUR*, pages 19:1–19:17, 2018.

27 Achour Mostéfaoui, Eric Mourgaya, Philippe Raipin Parvédy, and Michel Raynal. Evaluating the condition-based approach to solve consensus. In *DSN*, pages 541–550, 2003.

28 T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Dist. Comp.*, 2:80–94, 1987.