Verifying Safety of Synchronous Fault-Tolerant Algorithms by Bounded Model Checking

Ilina Stoilkovska, Igor Konnov, Josef Widder, Florian Zuleger

Verifying Safety of Synchronous Fault-Tolerant Algorithms by Bounded Model Checking

Ilina Stoilkovska[1]([✉]), Igor Konnov[2], Josef Widder[1], and Florian Zuleger[1]

[1] TU Wien, Vienna, Austria
{stoilkov,widder,zuleger}@forsyte.at
[2] University of Lorraine, CNRS, Inria, LORIA,
Nancy, France
igor.konnov@inria.fr

Abstract. Many fault-tolerant distributed algorithms are designed for synchronous or round-based semantics. In this paper, we introduce the synchronous variant of threshold automata, and study their applicability and limitations for the verification of synchronous distributed algorithms. We show that in general, the reachability problem is undecidable for synchronous threshold automata. Still, we show that many synchronous fault-tolerant distributed algorithms have a bounded diameter, although the algorithms are parameterized by the number of processes. Hence, we use bounded model checking for verifying these algorithms.

The existence of bounded diameters is the main conceptual insight in this paper. We compute the diameter of several algorithms and check their safety properties, using SMT queries that contain quantifiers for dealing with the parameters symbolically. Surprisingly, performance of the SMT solvers on these queries is very good, reflecting the recent progress in dealing with quantified queries. We found that the diameter bounds of synchronous algorithms in the literature are tiny (from 1 to 4), which makes our approach applicable in practice. For a specific class of algorithms we also establish a theoretical result on the existence of a diameter, providing a first explanation for our experimental results. The encodings of our benchmarks and instructions on how to run the experiments are available at: [33].

1 Introduction

Fault-tolerant distributed algorithms are hard to design and verify. Recently, threshold automata were introduced to model, verify and synthesize asynchronous fault-tolerant distributed algorithms [19,21,24]. Owing to the well-known impossibility result [18] many distributed computing problems, including

```
1   int v:=input({0, 1})
2   bool accept:=false
3   while (true) do { // in one synchronous step
4     if (v = 1) then broadcast <ECHO>;
5     receive messages from other processes;
6     if received <ECHO> from ≥ t + 1 processes
7       then v:=1;
8     if received <ECHO> from ≥ n − t processes
9       then accept:=true;
10  }
```
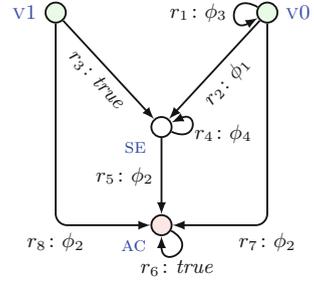

Fig. 1. Pseudo code of synchronous reliable broadcast à la [32], and its STA, with guards: $\phi_1 \equiv \#\{\text{V1}, \text{SE}, \text{AC}\} \geq t + 1 - f$ and $\phi_2 \equiv \#\{\text{V1}, \text{SE}, \text{AC}\} \geq n - t - f$ and $\phi_3 \equiv \#\{\text{V1}, \text{SE}, \text{AC}\} < t + 1$ and $\phi_4 \equiv \#\{\text{V1}, \text{SE}, \text{AC}\} < n - t$.

consensus, are not solvable in purely asynchronous systems. Thus, synchronous distributed algorithms have been extensively studied [5,26]. In this paper, we introduce *synchronous* threshold automata, and investigate their applicability and limitations for verification of synchronous fault-tolerant distributed algorithms.

An example of such a synchronous threshold automaton is given in Fig. 1 on the right; it encodes the synchronous reliable broadcast algorithm from [32]. (The pseudo code is in Fig. 1 on the left.) Its semantics is defined in terms of a counter system. For each location $\ell_i \in \{\text{V0}, \text{V1}, \text{SE}, \text{AC}\}$ (a node in the graph), we have a counter κ_i that stores the number of processes that are in ℓ_i. The system is parameterized in two ways: (i) in the number of processes n, the number of faults f, and the upper bound on the number of faults t, (ii) the expressions in the guards contain n, t, and f. Every transition moves all processes simultaneously; potentially using a different rule for each process (depicted by an edge in the figure), provided that the rule guards evaluate to true. The guards compare a sum of counters to a linear combination of parameters. For example, the guard $\phi_1 \equiv \#\{\text{V1}, \text{SE}, \text{AC}\} \geq t + 1 - f$ evaluates to true if the number of processes that are either in location V1, SE, or AC is greater than or equal to $t + 1 - f$.

Synchronous Threshold Automata (STA) model synchronous fault-tolerant distributed algorithms as follows. As processes send messages based on their current locations, we use the number of processes in given locations to test how many messages of a certain type have been sent. However, the pseudo code in Fig. 1 is predicated by received messages rather than by sent messages. This algorithm is designed to tolerate Byzantine-faulty processes, which may send spurious messages to some correct processes. Thus, the number of received messages may deviate from the number of correct processes that sent a message. For example, if the guard in line 7 evaluates to true, the $t + 1$ received messages may contain up to f messages from faulty processes. If i correct processes send <ECHO>, for $1 \leq i \leq t$, the faulty processes may "help" some correct processes to pass over the $t + 1$ threshold. In the STA, this is modeled by both the rules r_1 and r_2 being enabled. Thus, the assignment v:=1 in line 7 is modeled by the rule

Table 1. A long execution of reliable broadcast and the short representative

Process	σ_0	σ_1	σ_2	...	σ_{t+1}	σ_{t+2}	σ_{t+3}
1	V1	SE	SE	...	SE	SE	AC
2	V0	V0	SE	...	SE	SE	AC
...				...			
$t+1$	V0	V0	V0	...	SE	SE	AC
...				...			
$n-f$	V0	V0	V0	...	V0	SE	AC

Process	σ'_0	σ'_1	σ'_2
1	V1	SE	AC
2	V0	SE	AC
...		...	
$t+1$	V0	SE	AC
...		...	
$n-f$	V0	SE	AC

r_2, guarded by ϕ_2. The implicit "else" branch between lines 7 and 8 is modeled by the rule r_1, guarded by ϕ_3. As the effect of the f faulty processes on the correct processes is captured by the guards, we model only the correct processes explicitly, so that a system consists of $n-f$ copies of the STA.

Contributions. We start by introducing synchronous threshold automata (STA) and the counter systems they define.

1. We show that parameterized reachability checking of STA is undecidable.
2. We introduce an SMT-based procedure for finding the diameter of the counter system associated with an STA, i.e., the number of steps in which every configuration of the counter system is reachable. By knowing the diameter, we use bounded model checking as a complete verification method [11,14,22].
3. For a class of STA that captures several algorithms such as the broadcast algorithm in Fig. 1, we prove that a diameter is always bounded. The diameter is a function of the number of guard expressions and the longest path in the automaton, that is, it is independent of the parameters.
4. We implemented our technique, by running Z3 [29] and CVC4 [7] as back-end SMT solvers, and evaluated it by applying it to several distributed algorithms from the literature: Benchmarks that tolerate Byzantine faults from [8–10, 32], benchmarks that tolerate crashes from [13,26,30], and benchmarks that tolerate send omissions from [10,30].
5. We are the first to automatically verify the Byzantine and send omission benchmarks. For the crash benchmarks, our method performs significantly better than the abstraction-based method in [3]. By tweaking the constraints on the parameters n, t, f, we introduce configurations with more faults than expected, for which our technique automatically finds a counterexample.

2   Overview of Our Approach

Bounded Diameter. Consider Fig. 1: the processes execute the send, receive, and local computation steps in lock-step. One iteration of the loop is expressed as an STA edge that connects the locations before and after an iteration (i.e., the STA models the loop body of the pseudo code). The location SE encodes that v = 1 and accept is false. That is, SE is the location in which processes send

<ECHO> in every round. If a process sets `accept` to true, it goes to location AC. The location where v is 1 is encoded by V1, and the where v is 0 by V0.

An example execution is depicted in Table 1 on the left. We run $n - f$ copies of the STA in Fig. 1. Observe that the guards of the rules r_1 and r_2 are both enabled in the configuration σ_0. One STA uses r_2 to go to SE while the others use the self-loop r_1 to stay in V0. As both rules remain enabled, in every round one more automaton can go to SE. Hence, configuration σ_{t+1} has $t + 1$ correct STA in location SE and rule r_1 becomes disabled. Then, all remaining STA go to SE and then finally to AC. This execution depends on the parameter t, which implies that the length of this execution is unbounded for increasing values of the parameter t. (We note that we can obtain longer executions, if some STA use rule r_4). On the right, we see an execution where all STA take r_2 immediately. That is, while configuration σ_{t+3} is reached by a long execution on the left, it is reached in just two steps on the right (observe $\sigma'_2 = \sigma_{t+3}$). We are interested in whether there is a natural number k (which does not depend on the parameters n, t and f) such that we can always shorten executions to executions of length $\leq k$. (By length, we mean the number of transitions in an execution.) In such a case we say that the STA has *bounded diameter*. In Sect. 5.1 we introduce an SMT-based procedure that enumerates candidates for the diameter bound and checks if the candidate is indeed the diameter; if it finds such a bound, it terminates. For the STA in Fig. 1, this procedure computes the diameter 2.

Threshold Automata with Traps. In Sect. 5.2, we define a fragment of STA for which we theoretically guarantee a bounded diameter. For example, the STA in Fig. 1 falls in this fragment, and we obtain a guaranteed diameter of ≤ 8. The fragment is defined by two conditions: (i) The STA has a structure that implies monotonicity of the guards: the set of locations that are used in the guards (e.g., $\{V1, SE, AC\}$) is closed under the rules, i.e., from each location within the set, the STA can reach only a location in the set. We call guards that have this property *trapped*. (ii) The STA has no cycles, except possibly self-loops.

Bounded Model Checking, Completeness and (Un-)Decidability. The existence of a bounded diameter motivates the use of bounded model checking for verifying safety properties. In Sect. 6 we give an SMT encoding for checking the violation of a safety property by executions with length up to the diameter. Crucially, this approach is complete because if an execution reaches a bad configuration, this bad configuration is already reached by an execution of bounded length. We observe that for the STA defined in this paper (with linear guards and linear constraints on the parameters), the SMT encoding results in a Presburger arithmetic formula (with one quantifier alternation). Hence, checking safety properties (that can be expressed in Presburger arithmetic) is decidable for STA with bounded diameter. We also experimentally demonstrate in Sect. 7 that current SMT solvers can handle these quantified formulae well. On the contrary, we show in Sect. 4 that the parameterized reachability problem is undecidable for general STA. This implies that there are STA with unbounded diameter.

```
1   best := input_value
2   for each round 1 through ⌊t/k⌋ + 1 do {
3       broadcast best;
4       receive values b₁,...bₗ from others;
5       best := min {b₁,...bₗ};
6   }
7   choose best
```
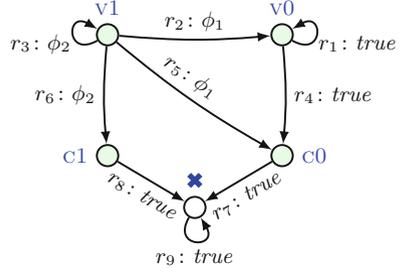

Fig. 2. Pseudo code of *FloodMin* from [13], and STA encoding its loop body, for $k = 1$, with guards: $\phi_1 \equiv \#\{v0, c0\} > 0$ and $\phi_2 \equiv \#\{v0\} = 0$.

Threshold Automata with Untrapped Guards. The *FloodMin* algorithm in Fig. 2 solves the k-set agreement problem. This algorithm is ran by n replicated processes, up to t of which may fail by crashing. For simplicity of presentation, we consider the case when $k = 1$, which turns k-set agreement into consensus. In Fig. 2, on the right, we have the STA that captures the loop body. The locations c0 and c1 correspond to the case when a process is crashing in the current round and may manage to send the value 0 and 1 respectively; the process remains in the crashed location "✖" and does not send any messages starting with the next round. We observe that the guard $\#\{v0, c0\} > 0$ is not trapped, and our result about trapped guards does not apply. Nevertheless, our SMT-based procedure can find a diameter of 2. In the same way, we automatically found a bound on the diameter for several benchmarks from the literature. It is remarkable that the diameter for the transition relation of the loop body (without the loop condition) is bounded by a constant, independent of the parameters.

Bounded Model Checking of Algorithms with Clean Rounds. The number of loop iterations $\lfloor t/k \rfloor + 1$ of the *FloodMin* algorithm has been designed such that it ensures (together with the environment assumption of at most t crashes) that there is at least one *clean* round in which at most $k - 1$ processes crashed. The correctness of the *FloodMin* algorithm relies on the occurrence of such a clean round. We make use of the existence of clean rounds by employing the following two-step methodology for the verification of safety properties: (i) we find all reachable clean-round configurations, and (ii) check if a bad configuration is reachable from those configurations. Detailed description of this methodology can be found in Sect. 6. Our method requires the encoding of a clean round as input (e.g., for Fig. 2 that no STA are in c0 and c1). We leave detecting and encoding clean rounds automatically from the fault environment for future work.

3   Synchronous Threshold Automata

We introduce the syntax of synchronous threshold automata and give some intuition of the semantics, which we will formalize as counter systems below.

A *synchronous threshold automaton* is the tuple $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$, where $\mathcal{L}$ is a finite set of locations, $\mathcal{I} \subseteq \mathcal{L}$ is a non-empty set of initial locations, Π is a finite set of parameters, $\mathcal{R}$ is a finite set of rules, RC is a resilience condition, and χ is a counter invariant, defined in the following. We assume that the set Π of parameters contains at least the parameter n, denoting the number of processes. We call the vector $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_{|\Pi|} \rangle$ the *parameter vector*, and a vector $\mathbf{p} = \langle p_1, \ldots, p_{|\Pi|} \rangle$ is an *instance of* $\boldsymbol{\pi}$, where $\pi_i \in \Pi$ is a parameter, and $p_i \in \mathbb{N}$ is a natural number, for $1 \leq i \leq |\Pi|$, such that $\mathbf{p}[\pi_i] = p_i$ is the value assigned to the parameter π_i in the instance $\mathbf{p}$ of $\boldsymbol{\pi}$. The set of *admissible instances of* $\boldsymbol{\pi}$ is defined as $P_{RC} = \{\mathbf{p} \in \mathbb{N}^{|\Pi|} \mid \mathbf{p}$ is an instance of $\boldsymbol{\pi}$ and $\mathbf{p}$ satisfies $RC\}$. The mapping $N : P_{RC} \rightarrow \mathbb{N}$ maps an admissible instance $\mathbf{p} \in P_{RC}$ to the number $N(\mathbf{p})$ of processes that participate in the algorithm, such that $N(\mathbf{p})$ is a linear combination of the parameter values in $\mathbf{p}$.

For example, for the STA in Fig. 1, $RC \equiv n > 3t \wedge t \geq f$, hence a vector $\mathbf{p} \in \mathbb{N}^{|\Pi|}$ is an admissible instance of the parameter vector $\boldsymbol{\pi} = \langle n, t, f \rangle$, if $\mathbf{p}[n] > 3\mathbf{p}[t] \wedge \mathbf{p}[t] \geq \mathbf{p}[f]$. Furthermore, for this STA, $N(\mathbf{p}) = \mathbf{p}[n] - \mathbf{p}[f]$. For the STA in Fig. 2, $RC \equiv n > t \wedge t \geq f$, hence the admissible instances satisfy $\mathbf{p}[n] > \mathbf{p}[t] \wedge \mathbf{p}[t] \geq \mathbf{p}[f]$, and we have $N(\mathbf{p}) = \mathbf{p}[n]$.

We introduce *counter atoms* of the form $\psi \equiv \#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b$, where $L \subseteq \mathcal{L}$ is a set of locations, $\#L$ denotes the total number of processes currently in the locations $\ell \in L$, $\boldsymbol{a} \in \mathbb{Z}^{|\Pi|}$ is a vector of coefficients, $\boldsymbol{\pi}$ is the parameter vector, and $b \in \mathbb{Z}$. We will use the counter atoms for expressing guards and predicates in the verification problem. In the following, we will use two abbreviations: $\#L = \boldsymbol{a} \cdot \boldsymbol{\pi} + b$ for the formula $(\#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b) \wedge \neg(\#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b + 1)$, and $\#L > \boldsymbol{a} \cdot \boldsymbol{\pi} + b$ for the formula $\#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b + 1$.

A *rule* $r \in \mathcal{R}$ is the tuple $(from, to, \varphi)$, where $from, to \in \mathcal{L}$ are locations, and φ is a guard whose truth value determines if the rule r is executed. The guard φ is a Boolean combination of counter atoms. We denote by Ψ the set of counter atoms occurring in the guards of the rules $r \in \mathcal{R}$.

The *counter invariant* χ is a Boolean combination of counter atoms $\#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b$, where each atom occurring in χ restricts the number of processes allowed to populate the locations in $L \subseteq \mathcal{L}$.

Counter Systems. The counter atoms are evaluated over tuples $(\kappa, \mathbf{p})$, where $\kappa \in \mathbb{N}^{|\mathcal{L}|}$ is a vector of *counters*, and $\mathbf{p} \in P_{RC}$ is an admissible instance of $\boldsymbol{\pi}$. For a location $\ell \in \mathcal{L}$, the counter $\kappa[\ell]$ denotes the number of processes that are currently in the location ℓ. A counter atom $\psi \equiv \#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b$ is *satisfied* in the tuple $(\kappa, \mathbf{p})$, that is $(\kappa, \mathbf{p}) \models \psi$, iff $\sum_{\ell \in L} \kappa[\ell] \geq \boldsymbol{a} \cdot \mathbf{p} + b$. The semantics of the Boolean connectives is standard.

A *transition* is a function $t : \mathcal{R} \rightarrow \mathbb{N}$ that maps a rule $r \in \mathcal{R}$ to a factor $t(r) \in \mathbb{N}$, denoting the number of processes that act upon this rule. Given an instance $\mathbf{p}$ of $\boldsymbol{\pi}$, we denote by $T(\mathbf{p})$ the set $\{t \mid \sum_{r \in \mathcal{R}} t(r) = N(\mathbf{p})\}$ of transitions whose rule factors sum up to $N(\mathbf{p})$.

Given a tuple $(\kappa, \mathbf{p})$ and a transition t, we say that t is *enabled* in $(\kappa, \mathbf{p})$, if

1. for every $r \in \mathcal{R}$, such that $t(r) > 0$, it holds that $(\kappa, \mathbf{p}) \models r.\varphi$, and
2. for every $\ell \in \mathcal{L}$, it holds that $\kappa[\ell] = \sum_{r \in \mathcal{R} \wedge r.from = \ell} t(r)$.

The first condition ensures that processes only use rules whose guards are satisfied, and the second that every process moves in an enabled transition.

Observe that each transition $t \in T(\mathbf{p})$ defines a unique tuple $(\kappa, \mathbf{p})$ in which it is enabled. We call the *origin* of a transition $t \in T(\mathbf{p})$ the tuple $o(t) = (\kappa, \mathbf{p})$, such that for every $\ell \in \mathcal{L}$, we have $o(t).\kappa[\ell] = \sum_{r \in \mathcal{R} \wedge r.from=\ell} t(r)$. Similarly, each transition defines a unique tuple $(\kappa, \mathbf{p})$ that is the result of applying the transition in its origin. We call the *goal* of a transition $t \in T(\mathbf{p})$ the tuple $g(t) = (\kappa, \mathbf{p})$, such that for every $\ell \in \mathcal{L}$, we have $g(t).\kappa[\ell] = \sum_{r \in \mathcal{R} \wedge r.to=\ell} t(r)$.

We now define a counter system, for a given $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$, and an admissible instance $\mathbf{p} \in P_{RC}$ of the parameter vector $\boldsymbol{\pi}$.

Definition 1. *A counter system w.r.t. $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$ and an admissible instance $\mathbf{p} \in P_{RC}$ is the tuple $\mathsf{CS}(STA, \mathbf{p}) = (\Sigma(\mathbf{p}), I(\mathbf{p}), R(\mathbf{p}))$, where*

- *$\Sigma(\mathbf{p}) = \{\sigma = (\kappa, \mathbf{p}) \mid \sum_{\ell \in \mathcal{L}} \sigma.\kappa[\ell] = N(\mathbf{p}) \text{ and } \sigma \models \chi\}$ are the configurations;*
- *$I(\mathbf{p}) = \{\sigma \in \Sigma(\mathbf{p}) \mid \sum_{\ell \in \mathcal{I}} \sigma.\kappa[\ell] = N(\mathbf{p})\}$ are the initial configurations;*
- *$R(\mathbf{p}) \subseteq \Sigma(\mathbf{p}) \times T(\mathbf{p}) \times \Sigma(\mathbf{p})$ is the transition relation, with $\langle \sigma, t, \sigma' \rangle \in R(\mathbf{p})$, if σ is the origin and σ' the goal of t. We write $\sigma \xrightarrow{t} \sigma'$, if $\langle \sigma, t, \sigma' \rangle \in R(\mathbf{p})$.*

We restrict ourselves to deadlock-free counter systems, i.e., counter systems where the transition relation is total (every configuration has a successor). A sufficient condition for deadlock-freedom is that for every location $\ell \in \mathcal{L}$, it holds that $\chi \rightarrow \bigvee_{r \in \mathcal{R} \wedge r.from=\ell} r.\varphi$. This ensures that it is always possible to move out of every location, as there is at least one outgoing rule per location whose guard is satisfied.

To simplify the notation, in the following we write $\sigma[\ell]$ to denote $\sigma.\kappa[\ell]$.

Paths and Schedules in a Counter System. We now define paths and schedules of a counter system, as sequences of configurations and transitions, respectively.

Definition 2. *A path in the counter system $\mathsf{CS}(STA, \mathbf{p}) = (\Sigma(\mathbf{p}), I(\mathbf{p}), R(\mathbf{p}))$ is a finite sequence $\{\sigma_i\}_{i=0}^k$ of configurations, such that for every two consecutive configurations σ_{i-1}, σ_i, for $0 < i \leq k$, there exists a transition $t_i \in T(\mathbf{p})$ such that $\sigma_{i-1} \xrightarrow{t_i} \sigma_i$. A path $\{\sigma_i\}_{i=0}^k$ is called an execution if $\sigma_0 \in I(\mathbf{p})$.*

Definition 3. *A schedule is a finite sequence $\tau = \{t_i\}_{i=1}^k$ of transitions $t_i \in T(\mathbf{p})$, for $0 < i \leq k$. We denote by $|\tau| = k$ the length of the schedule τ.*

A schedule $\tau = \{t_i\}_{i=1}^k$ is feasible if there is a path $\{\sigma_i\}_{i=0}^k$ such that $\sigma_{i-1} \xrightarrow{t_i} \sigma_i$, for $0 < i \leq k$. We call σ_0 the origin, and σ_k the goal of τ, and write $\sigma_0 \xrightarrow{\tau} \sigma_k$.

4   Parameterized Reachability and Its Undecidability

We show that the following problem is undecidable in general, by reduction from the halting problem of a two-counter machine (2CM) [28]. Such reductions are common in parameterized verification, e.g., see [12].

Definition 4 (Parameterized Reachability). *Given a formula φ, that is, a Boolean combination of counter atoms, and $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$, the* parameterized reachability *problem is to decide whether there exists an admissible instance $\mathbf{p} \in P_{RC}$, such that in the counter system $\mathsf{CS}(STA, \mathbf{p})$, there is an initial configuration $\sigma \in I(\mathbf{p})$, and a feasible schedule τ, with $\sigma \xrightarrow{\tau} \sigma'$ and $\sigma' \models \varphi$.*

To prove undecidability, we construct a synchronous threshold automaton $STA_{\mathcal{M}}$, such that every counter system induced by it simulates the steps of a 2CM executing a program P. The STA has a single parameter – the number n of processes, and the invariant $\chi = true$. The idea is that each process plays one of two roles: either it is used to encode the control flow of the program P (*controller* role), or to encode the values of the registers in unary, as in [17] (*storage* role). Thus, $STA_{\mathcal{M}}$ consists of two parts – one per each role.

Our construction allows multiple processes to act as controllers. Since we assume that 2CM is deterministic, all the controllers behave the same. For each instruction of the program P, in the controller part of $STA_{\mathcal{M}}$, there is a single location (for 'jump if zero' and 'halt') or a pair of locations (for 'increment' and 'decrement'), and a special *stuck* location. In the storage part of $STA_{\mathcal{M}}$, there is a location for each register, a store location, and auxiliary locations. The number of processes in a register location encodes the value of the register in 2CM.

An increment (resp. decrement) of a register is modeled by moving one process from (resp. to) the store location to (resp. from) the register location. The guards on the rules in the controller part check if the storage processes made a transition that truly models a step of 2CM; in this case, the controllers move on to the next location, otherwise they move to the stuck location. For example, to model a 'jump if zero' for register A, the controllers check if $\#\{\ell_A\} = 0$, where ℓ_A is the storage location corresponding to register A. The main invariant which ensures correctness is that every transition in every counter system induced by $STA_{\mathcal{M}}$ either faithfully simulates a step of the 2CM, or moves all of the controllers to the stuck location.

Let ℓ_{halt} be the halting location in the controller part of $STA_{\mathcal{M}}$. The formula $\varphi \equiv \neg(\#\{\ell_{halt}\} = 0)$ states that the controllers have reached the halting location. Thus, the answer to the parameterized reachability question given the formula φ and $STA_{\mathcal{M}}$ is positive iff 2CM halts, which gives us undecidability.

5   Bounded Diameter Oracle

5.1   Computing the Diameter Using SMT

Given an STA, the diameter is the maximal number of transitions needed to reach all possible configurations in every counter system induced by the STA, and an admissible instance $\mathbf{p} \in P_{RC}$. We adapt the definition of diameter from [11].

Definition 5 (Diameter). *Given an $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$, the* diameter *is the smallest number d such that for every $\mathbf{p} \in P_{RC}$ and every path $\{\sigma_i\}_{i=0}^{d+1}$*

of length $d + 1$ in $\mathsf{CS}(STA, \mathbf{p})$, there exists a path $\{\sigma'_j\}_{j=0}^{e}$ of length $e \leq d$ in $\mathsf{CS}(STA, \mathbf{p})$, such that $\sigma_0 = \sigma'_0$ and $\sigma_{d+1} = \sigma'_e$.

Thus, the diameter is the smallest number d that satisfies the formula:

$$\forall \mathbf{p} \in P_{RC}. \, \forall \sigma_0, \ldots, \sigma_{d+1}. \, \forall t_1, \ldots, t_{d+1}. \, \exists \sigma'_0, \ldots, \sigma'_d. \, \exists t'_1, \ldots, t'_d.$$

$$Path(\sigma_0, \sigma_{d+1}, d+1) \to (\sigma_0 = \sigma'_0) \land Path(\sigma'_0, \sigma'_d, d) \land \bigvee_{i=0}^{d} \sigma'_i = \sigma_{d+1} \quad (1)$$

where $Path(\sigma_0, \sigma_d, d)$ is a shorthand for the formula $\bigwedge_{i=0}^{d-1} R(\sigma_i, t_{i+1}, \sigma_{i+1})$, and $R(\sigma, t, \sigma')$ is a predicate which evaluates to true whenever $\sigma \xrightarrow{t} \sigma'$. Since we assume deadlock-freedom, we are able to encode the path $Path(\sigma'_0, \sigma'_d, d)$ of length d, even if the disjunction $\bigvee_{i=0}^{d} \sigma'_i = \sigma_{d+1}$ holds for some $i \leq d$.

Formula (1) gives us the following procedure to determine the diameter:

1. initialize the candidate diameter d to 1;
2. check if the negation of the formula (1) is unsatisfiable;
3. if yes, then output d and terminate;
4. if not, then increment d and jump to step 2.

If the procedure terminates, it outputs the diameter, which can be used as completeness threshold for bounded model checking. We implemented this procedure, and used a back-end SMT solver to automate the test in step 2.

5.2   Bounded Diameter for a Fragment of STA

In this section, we show that for a specific fragment of STA, we are able to give a theoretical bound on the diameter, similar to the asynchronous case [20,21].

The STA that fall in this fragment are *monotonic* and *1-cyclic*. An STA is monotonic iff every counter atom changes its truth value at most once in every path of a counter system induced by the STA and an admissible instance $\mathbf{p} \in P_{RC}$. This implies that every schedule can be partitioned into finitely many sub-schedules, that satisfy a property we call *steadiness*. We call a schedule *steady* if the set of rules whose guards are satisfied does not change in all of its transitions. We also give a sufficient condition for monotonicity, using *trapped* counter atoms, defined below. In a 1-cyclic STA, the only cycles that can be formed by its rules are self-loops. Under these two conditions, we guarantee that for every steady schedule, there exists a steady schedule of bounded length, that has the same origin and goal. We show that this bound depends on the counter atoms Ψ occurring in the guards of the STA, and the length of the longest path in the STA, denoted by c. The main result of this section is stated by the theorem:

Theorem 1. *For every feasible schedule τ in a counter system $\mathsf{CS}(STA, \mathbf{p})$, where STA is monotonic and 1-cyclic, and $\mathbf{p} \in P_{RC}$, there exists a feasible schedule τ' of length $O(|\Psi|c)$, such that τ and τ' have the same origin and goal.*

To prove Theorem 1, we start by defining monotonic STA.

Definition 6 (Monotonic STA). *An automaton $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$ is monotonic iff for every path $\{\sigma_i\}_{i=0}^{k}$ in the counter system $\mathsf{CS}(STA, \mathbf{p})$, for $\mathbf{p} \in P_{RC}$, and every counter atom $\psi \in \Psi$, we have $\sigma_i \models \psi$ implies $\sigma_j \models \psi$, for $0 \leq i < j < k$.*

To show that we can partition a schedule into finitely many sub-schedules, we need the notion of a context. A *context* of a transition $t \in T(\mathbf{p})$ is the set $\mathcal{C}_t = \{\psi \in \Psi \mid o(t) \models \psi\}$ of counter atoms ψ satisfied in the origin $o(t)$ of the transition t. Given a feasible schedule τ, the point i is a *context switch*, if $\mathcal{C}_{t_{i-1}} \neq \mathcal{C}_{t_i}$, for $1 < i \leq |\tau|$.

Lemma 1. *Every feasible schedule τ in a counter system induced by a monotonic STA has at most $|\Psi|$ context switches.*

Proof. Let $\tau = \{t_i\}_{i=1}^{k}$ be a feasible schedule and Ψ the set of counter atoms appearing on the rules of the monotonic STA. For every $\psi \in \Psi$, there is at most one context switch i, for $0 < i \leq k$, such that $\psi \notin \mathcal{C}_{t_{i-1}}$ and $\psi \in \mathcal{C}_{t_i}$. $\qquad\square$

Sufficient Condition for Monotonicity. We introduce trapped counter atoms.

Definition 7. *A set $L \subseteq \mathcal{L}$ of locations is called a* trap, *iff for every $\ell \in L$ and every $r \in \mathcal{R}$ such that $\ell = r.from$, it holds that $r.to \in L$.*
A counter atom $\psi \equiv \#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b$ is trapped *iff the set L is a trap.*

Lemma 2. *Let $\psi \equiv \#L \geq \boldsymbol{a} \cdot \boldsymbol{\pi} + b$ be a trapped counter atom, σ a configuration such that $\sigma \models \psi$, and t a transition enabled in σ. If $\sigma \xrightarrow{t} \sigma'$, then $\sigma' \models \psi$.*

Corollary 1. *Let $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$ be an automaton such that all its counter atoms are trapped. Then STA is monotonic.*

Steady Schedules. We define the notion of steadiness, similarly to [20].

Definition 8. *A schedule $\tau = \{t_i\}_{i=1}^{k}$ is* steady, *if $\mathcal{C}_{t_i} = \mathcal{C}_{t_j}$, for $0 < i < j \leq k$.*

We now focus on shortening steady schedules. That is, given a steady schedule, we construct a schedule of bounded length with the same origin and goal.

Observe that $STA = (\mathcal{L}, \mathcal{I}, \Pi, \mathcal{R}, RC, \chi)$ can be seen as a directed graph G_{STA}, with vertices corresponding to the locations $\ell \in \mathcal{L}$, and edges corresponding to the rules $r \in \mathcal{R}$. We denote by c the length of the longest path between two nodes in the graph G_{STA}, and call it the *longest chain* of STA. If G_{STA} contains only cycles of length one, then STA is called *1-cyclic*.

To shorten steady schedules, in addition to monotonicity, we require that the STA are also 1-cyclic. In the following, we assume that the schedules we shorten come from counter systems induced by monotonic and 1-cyclic STA. Intuitively, if a given schedule is longer than the longest chain of the STA, then in some transition of the schedule some processes followed a rule which is a self-loop. As processes may follow self-loops at different transitions, we cannot shorten the given schedule by eliminating transitions as a whole. Instead, we deconstruct the original schedule into sequences of process steps, which we call *runs*, shorten the runs, and reconstruct a new shorter schedule from the shortened runs. The main challenge is to show that the newly obtained schedule is feasible and steady.

Schedules as Multisets of Runs. We proceed by defining runs and showing that each schedule can be represented by a multiset of runs.

We call a *run* the sequence $\varrho = \{r_i\}_{i=1}^{k}$ of rules, for $r_i \in \mathcal{R}$, such that $r_i.to = r_{i+1}.from$, for $0 < i < k$. We denote by $\varrho[i] = r_i$ the i-th rule in the run ϱ, and by $|\varrho|$ the length of the run. The following lemma shows that a feasible schedule can be deconstructed into a multiset of runs.

Lemma 3. *For every feasible schedule $\tau = \{t_i\}_{i=1}^{k}$, there exists a multiset $(\mathcal{P}, m)$, where*

1. $\mathcal{P}$ *is a set of runs ϱ of length k, and*
2. $m : \mathcal{P} \to \mathbb{N}$ *is a multiplicity function, such that for every location $\ell \in \mathcal{L}$, it holds that $\sum_{r.from=\ell} t_i(r) = \sum_{\varrho[i].from=\ell} m(\varrho)$, for $0 < i \leq k$.*

A multiset $(\mathcal{P}, m)$ of runs of length k defines a schedule $\tau = \{t_i\}_{i=1}^{k}$ of length k, and we have $t_i(r) = \sum_{\varrho[i]=r} m(\varrho)$, for every rule $r \in \mathcal{R}$ and $0 < i \leq k$.

For the counter systems of STA, which are both monotonic and 1-cyclic, we show that their steady schedules can be shortened, so that their length does not exceed the longest chain c (that is, the length of the longest path in the STA).

Lemma 4. *Let τ be a steady feasible schedule in a counter system induced by a monotonic and 1-cyclic STA. If $|\tau| > c + 1$, then there exists a steady feasible schedule τ' such such that $|\tau'| = |\tau| - 1$, and τ, τ' have the same origin and goal.*

Proof (Sketch). If $\tau = \{t_i\}_{i=1}^{k+1}$, with $|\tau| = k + 1 > c + 1$, is a steady schedule, then $\mathcal{C}_{t_1} = \mathcal{C}_{t_k}$, and its prefix $\theta = \{t_i\}_{i=1}^{k}$ is a steady and feasible schedule, with $k > c$. By Lemma 3, there is a multiset $(\mathcal{P}, m)$ of runs of length k describing θ. Since $k > c$, and c is the longest chain in the STA, which is 1-cyclic, it must be the case that every run in $\mathcal{P}$ contains at least one self-loop. Construct a new multiset $(\mathcal{P}', m')$ of runs of length $k - 1$, such that each $\varrho' \in \mathcal{P}'$ is obtained by some $\varrho \in \mathcal{P}$ by removing one occurrence of a self-loop rule. The multiset $(\mathcal{P}', m')$ defines the schedule $\theta' = \{t'_1\}_{i=1}^{k-1}$. Because of the monotonicity and steadiness of θ, and because we only remove self-loops (which go from and to the same location) when we build θ' from θ, the feasibility is preserved, that is, it holds that $g(t'_{i-1}) = o(t'_i)$, for $1 < i < k$, and that no guards false in θ become true in θ'. Furthermore, it is easy to check that θ' has the same origin and goal as θ. As the goal of θ' is the origin of t_{k+1}, construct a schedule $\tau' = \{t'_i\}_{i=1}^{k}$, where $t'_k = t_{k+1}$. As τ is steady, the transitions t_1 and t_{k+1} have the same contexts. From $o(t_1) = o(t'_1)$ and $o(t_{k+1}) = o(t'_k)$, we get that t'_1 and t'_k have the same contexts, which, together with the monotonicity, implies that τ' is steady.   □

As a consequence of Lemmas 1 and 4, we obtain Theorem 1, which tells us that for any feasible schedule, there exists a feasible schedule of length $O(|\Psi|c)$. This bound does not depend on the parameters, but on the number of context switches and the longest chain c, which are properties of the STA.

6   Bounded Model Checking of Safety Properties

Once we obtain the diameter bound d (either using the procedure from Sect. 5.1, or by Theorem 1), we use it as a completeness threshold for bounded model checking. For the algorithms that we verify, we express the violations of their safety properties as reachability queries on bounded executions. The length of the bounded executions depends on d, and on whether the algorithm was designed such that it is assumed that there is a clean round in every execution.

Checking Safety for Algorithms that do not Assume a Clean Round. Here, we search for violations of safety properties in executions of length $e \leq d$, by checking satisfiability of the formula:

$$\exists \mathbf{p} \in P_{RC}. \exists \sigma_0, \ldots, \sigma_e. \exists t_1, \ldots, t_e.\ Init(\sigma_0) \wedge Path(\sigma_0, \sigma_e, e) \wedge Bad(\sigma_e) \quad (2)$$

where the predicate $Init(\sigma)$ encodes that σ is an initial configuration, together with the constraints imposed on the initial configuration by the safety property, and $Bad(\sigma)$ encodes the bad configuration, which, if reachable, violates safety.

For example, the algorithm in Fig. 1 has to satisfy the safety property *unforgeability*: If no process sets v to 1 initially, then no process ever sets `accept` to true. In our encoding, we check executions of length $e \leq d$, whose initial configuration has the counter $\kappa[\text{V1}] = 0$. In a bad configuration, the counter $\kappa[\text{AC}] > 0$. Thus, to find violations of unforgeability, in formula (2), we set:

$$Init(\sigma_0) \equiv \sigma_0[\text{V0}] + \sigma_0[\text{V1}] = N(\mathbf{p}) \wedge \sigma_0[\text{V1}] = 0$$
$$Bad(\sigma_e) \equiv \sigma_e[\text{AC}] > 0$$

Checking Safety for Algorithms with a Clean Round. We check for violations of safety in executions of length $e \leq 2d$, where $e = e_1 + e_2$ such that: (i) we find all reachable clean-round configurations in an execution of length e_1, for $e_1 \leq d$, such that the last configuration σ_{e_1} satisfies the clean round condition, and (ii) we check if a bad configuration is reachable from σ_{e_1} by a path of length $e_2 \leq d$. That is, we check satisfiability of the formula:

$$\exists \mathbf{p} \in P_{RC}. \exists \sigma_0, \ldots, \sigma_e. \exists t_1, \ldots, t_e.\ Init(\sigma_0) \wedge Path(\sigma_0, \sigma_{e_1}, e_1)$$
$$\wedge\ Clean(\sigma_{e_1}) \wedge Path(\sigma_{e_1}, \sigma_e, e_2) \wedge Bad(\sigma_e) \quad (3)$$

where the predicate $Clean(\sigma)$ encodes the clean round condition.

For example, one of the safety properties that the *FloodMin* algorithm for $k = 1$ (Fig. 2) has to satisfy, is *k-agreement*, which requires that at most k different values are decided. In the original algorithm, the processes decide after $\lfloor t/k \rfloor + 1$ rounds, such that at least one of them is the clean round, in which at most $k - 1$ processes crash. In our encoding, we check paths of length $e \leq 2d$. We enforce the clean round condition by asserting that the sum of counters of the locations C0, C1 are $k - 1 = 0$ in the configuration σ_{e_1}. The property

1-agreement is violated if in the last configuration both the counters $\kappa[\text{V0}]$ and $\kappa[\text{V1}]$ are non-zero. That is, to check 1-agreement, in formula (3) we set:

$$Init(\sigma_0) \equiv \sigma_0[\text{V0}] + \sigma_0[\text{V1}] + \sigma_0[\text{C0}] + \sigma_0[\text{C1}] = N(\mathbf{p})$$
$$Clean(\sigma_{e_1}) \equiv \sigma_{e_1}[\text{C0}] + \sigma_{e_1}[\text{C1}] = 0$$
$$Bad(\sigma_e) \equiv \sigma_e[\text{V0}] > 0 \land \sigma_e[\text{V1}] > 0$$

7 Experimental Evaluation

The algorithms that we model using STA and verify by bounded model checking are designed for different fault models, which in our case are crashes, send omissions or Byzantine faults. We now proceed by introducing our benchmarks. Their encodings, together with the implementations of the procedures for finding the diameter and applying bounded model checking are available at [1].

Algorithms without a Clean Round Assumption. We consider three variants of the synchronous reliable broadcast algorithm, whose STA are monotonic and 1-cyclic (i.e., Theorem 1 applies). These algorithms assume different fault models:

- `rb`, [31] (Fig. 1): reliable broadcast with at most t Byzantine faults;
- `rb_hybrid`, [10]: reliable broadcast with at most t hybrid faults: at most b Byzantine and at most s send omissions, with $t = b + s$;
- `rb_omit`, [10]: reliable broadcast with at most t send omissions.

Algorithms with a Clean Round. We encode several algorithms from this class, that solve the consensus or k-set agreement problem:

- `fair_cons` [30], `floodset` [26]: consensus with crash faults;
- `floodmin`, for $k \in \{1, 2\}$ [26] (Fig. 2): k-set agreement with crash faults;
- `kset_omit`, for $k \in \{1, 2\}$ [30]: k-set agreement with send omission faults;
- `phase_king` [8,9], `phase_queen` [8]: consensus with Byzantine faults.

These algorithms have a structure similar to the one depicted in Fig. 2, with the exception of `phase_king` and `phase_queen`. Their loop body consists of several message exchange steps, which correspond to multiple rounds, grouped in a *phase*. In each phase, a designated process acts as a coordinator.

Computing the Diameter. We implemented the procedure from Sect. 5.1 in Python. The implementation uses a back-end SMT solver (currently, `z3` and `cvc4`). Our tool computed diameter bounds for all of our benchmarks, even for those for which we do not have a theoretical guarantee. Our experiments reveal extremely low values for the diameter, that range between 1 and 4. The values for the diameter and the time needed to compute them are presented in Table 2.

Table 2. Results for our benchmarks, available at [1]: $|\mathcal{L}|, |\mathcal{R}|, |\Psi|$, RC are the number of locations, rules, atomic guards, and resilience condition in each STA; d is the diameter computed using SMT, c is the longest chain of the algorithms whose STA are monotonic and 1-cyclic; τ is the time (in seconds) to compute the diameter using SMT; T, SMT is the time to check reachability using the diameter computed using the SMT procedure from Sect. 5.1; $T, Theorem\ 1$ the time to check reachability using the bound obtained by Theorem 1. For the cases where Theorem 1 is not applicable, we write $(-)$. The experiments were run on a machine with Intel(R) Core(TM) i5-4210U CPU and 4GB of RAM, using `z3-4.8.1` and `cvc4-1.6`.

algorithm	$|\mathcal{L}|$	$|\mathcal{R}|$	$|\Psi|$	RC	d	τ z3	τ cvc4	T, SMT z3	T, SMT cvc4	c	$T, Thm.\ 1$ z3	$T, Thm.\ 1$ cvc4
rb	4	8	4	$n > 3t$	2	0.27	0.99	0.08	0.08	2	0.42	0.86
rb_hybrid	8	16	4	$n > 3b + 2s$	2	1.16	37.6	0.09	0.15	2	0.67	1.73
rb_omit	8	16	4	$n > 2t$	2	0.43	2.47	0.09	0.14	2	0.58	1.43
fair_cons	11	20	2	$n > t$	2	0.97	10.9	0.27	0.47	$-$	$-$	$-$
floodmin, $k = 1$	5	9	2	$n > t$	2	0.21	0.86	0.18	0.29	$-$	$-$	$-$
floodmin, $k = 2$	7	16	4	$n > t$	2	0.53	7.43	0.22	0.52	$-$	$-$	$-$
floodset	7	14	4	$n > t$	2	0.36	3.01	0.21	0.49	$-$	$-$	$-$
kset_omit, $k = 1$	4	6	2	$n > t$	1	0.08	0.09	0.04	0.03	$-$	$-$	$-$
kset_omit, $k = 2$	6	12	4	$n > t$	1	0.17	0.27	0.04	0.07	$-$	$-$	$-$
phase_king	34	72	12	$n > 3t$	4	12.9	50.5	1.41	5.12	$-$	$-$	$-$
phase_queen	24	42	8	$n > 4t$	3	1.78	17.7	0.36	1.92	$-$	$-$	$-$

Checking the Algorithms. We have implemented another Python function which encodes violations of the safety properties as reachability properties on paths of bounded length, as described in Sect. 6, and uses a back-end SMT solver to check their satisfiability. Table 2 contains the results that we obtained by checking reachability for our benchmarks, using the diameter bound computed using the procedure from Sect. 5.1, and diameter bound from Theorem 1, for algorithms whose STA are monotonic and 1-cyclic.

To our knowledge, we are the first to verify the listed algorithms that work with send omission, Byzantine and hybrid faults. For the algorithms with crash faults, our approach is a significant improvement to the results obtained using the abstraction-based method from [3].

Counterexamples. Our tool found a bug in the version of the `phase_king` algorithm that was given in [8], which was corrected in the version of the algorithm in [9]. The version from [8] had the wrong threshold '$> n - t$' in one guard, while the one in [9] had '$\geq n - t$' for the same guard. To test our tool, we produced erroneous encodings for our benchmarks, and checked them. For `rb`, `rb_hybrid`, `rb_omit`, `phase_king`, and `phase_queen`, we tweaked the resilience condition, and introduced more faults than expected by the algorithm, e.g., by setting $f > t$ (instead of $f \leq t$) in the STA in Fig. 1. For `fair_cons`, `floodmin`, `floodset`, and `kset_omit`, we checked executions without a clean round. For all of the erroneous encodings, our tool produces counterexamples in seconds.

8   Discussion and Related Work

Parameterized verification of synchronous and partially synchronous distributed algorithms has recently gained attention. Both models have in common that distributed computations are organized in rounds and processes (conceptually) move in lock-step. For partially synchronous consensus algorithms, the authors of [15] introduced a consensus logic and (semi-)decision procedures. Later, the authors of [27] introduced a language for partially synchronous consensus algorithms, and proved cut-off theorems specialized to the properties of consensus: agreement, validity, and termination. Concerning synchronous algorithms, the authors of [3] introduced an abstraction-based model checking technique for crash-tolerant synchronous algorithms with existential guards. In contrast to their work, we allow more general guards that contain linear expressions over the parameters, e.g., $n - t$. Our method offers more automation, and our experimental evaluation shows that our technique is faster than the technique [3].

We introduce a *synchronous* variant of threshold automata, which were proposed in [21] for asynchronous algorithms. Several extensions of this model were recently studied in [23], but the synchronous case was not considered. STA extend the guarded protocols by [16], in which a process can check only if a sum of counters is different from 0 or n. Generalizing the results from [16] to STA is not straightforward. In [2], safety of finite-state transition systems over infinite data domains was reduced to backwards reachability checking using a fixpoint computation, as long as the transition systems are well-structured. It would be interesting to put our results in this context. A decidability result for liveness properties of parameterized timed networks was obtained in [4], employing linear programming for the analysis of vector addition systems with a parametric initial state. We plan to investigate the use of similar ideas for analyzing liveness properties of STA.

The 1-cyclicity condition is reminiscent of flat counter automata [25]. In Fig. 3, we show a possible translation of an STA to a counter automaton (similar to the translation for asynchronous threshold automata from [23]). We note
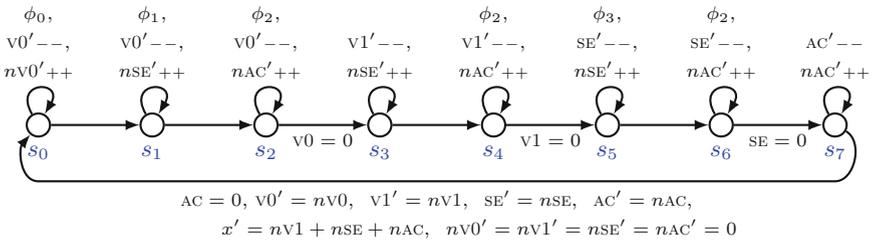

Fig. 3. A counter automaton for the STA in Fig. 1, with $\phi_0 \equiv x < t + 1$, $\phi_1 \equiv x + f \geq t + 1$, $\phi_2 \equiv x + f \geq n - t$, $\phi_3 \equiv x < n - t$, where x counts the number of processes in locations V1, SE, AC; and n, t, f are counters for the parameters. On a path from s_0 to s_7, the counters $\ell \in \{$V0, V1, SE, AC$\}$ are emptied, while the counters $n\ell$ are populated. This models the transitions from one location to another in the current round.

that the counter automaton is not flat, due to the presence of the outer loop, which models a transition to the next round. By knowing a bound d on the diameter (e.g., by Theorem 1), one can flatten the counter automaton by unfolding the outer loop d times. We also experimented with FAST [6] on two of our benchmarks: `rb` and `floodmin` for $k = 1$, depicted in Figs. 1 and 2 respectively. FAST terminated on `rb`, but took significantly longer than our tool on the same machine (i.e., hours rather than seconds). FAST ran out of memory when checking `floodmin`.

Our experiments show that STA that are neither monotonic, nor 1-cyclic still may have bounded diameters. Finding other classes of STA for which one could derive the diameter bounds is a subject of future work. Although we considered only reachability properties in this work—which happened to be challenging—we are going to investigate completeness thresholds for liveness in the future.

References

1. Experiments. https://github.com/istoilkovska/syncTA
2. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.: General decidability theorems for infinite-state systems. In: LICS, pp. 313–321 (1996)
3. Aminof, B., Rubin, S., Stoilkovska, I., Widder, J., Zuleger, F.: Parameterized model checking of synchronous distributed algorithms by abstraction. In: Dillig, I., Palsberg, J. (eds.) Verification, Model Checking, and Abstract Interpretation. LNCS, vol. 10747, pp. 1–24. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73721-8_1
4. Aminof, B., Rubin, S., Zuleger, F., Spegni, F.: Liveness of parameterized timed networks. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 375–387. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47666-6_30
5. Attiya, H., Welch, J.: Distributed Computing, 2nd edn. Wiley, Hoboken (2004)
6. Bardin, S., Leroux, J., Point, G.: FAST extended release. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 63–66. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_9
7. Barrett, C., et al.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
8. Berman, P., Garay, J.A., Perry, K.J.: Asymptotically optimal distributed consensus. Technical report, Bell Labs (1989). http://plan9.bell-labs.co/who/garay/asopt.ps
9. Berman, P., Garay, J.A., Perry, K.J.: Towards optimal distributed consensus (extended abstract). In: FOCS, pp. 410–415 (1989)
10. Biely, M., Schmid, U., Weiss, B.: Synchronous consensus under hybrid process and link failures. Theor. Comput. Sci. **412**(40), 5602–5630 (2011)
11. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49059-0_14
12. Bloem, R., et al.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2015)

13. Chaudhuri, S., Herlihy, M., Lynch, N.A., Tuttle, M.R.: Tight bounds for k-set agreement. J. ACM **47**(5), 912–943 (2000)
14. Clarke, E.M., Kroening, D., Ouaknine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 85–96. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_9
15. Drăgoi, C., Henzinger, T.A., Veith, H., Widder, J., Zufferey, D.: A logic-based framework for verifying consensus algorithms. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 161–181. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54013-4_10
16. Emerson, E.A., Namjoshi, K.S.: Automatic verification of parameterized synchronous systems. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 87–98. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61474-5_60
17. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. Int. J. Found. Comput. Sci. **14**(4), 527–550 (2003)
18. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
19. Konnov, I., Lazić, M., Veith, H., Widder, J.: A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In: POPL, pp. 719–734 (2017)
20. Konnov, I., Veith, H., Widder, J.: SMT and POR beat counter abstraction: parameterized model checking of threshold-based distributed algorithms. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 85–102. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_6
21. Konnov, I.V., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. Inf. Comput. **252**, 95–109 (2017)
22. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: Zuck, L.D., Attie, P.C., Cortesi, A., Mukhopadhyay, S. (eds.) VMCAI 2003. LNCS, vol. 2575, pp. 298–309. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36384-X_24
23. Kukovec, J., Konnov, I., Widder, J.: Reachability in parameterized systems: all flavors of threshold automata. In: CONCUR, pp. 19:1–19:17 (2018)
24. Lazić, M., Konnov, I., Widder, J., Bloem, R.: Synthesis of distributed algorithms with parameterized threshold guards. In: OPODIS. LIPIcs, vol. 95, pp. 32:1–32:20 (2017)
25. Leroux, J., Sutre, G.: Flat counter automata almost everywhere!. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 489–503. Springer, Heidelberg (2005). https://doi.org/10.1007/11562948_36
26. Lynch, N.: Distributed Algorithms. Morgan Kaufman, Burlington (1996)
27. Marić, O., Sprenger, C., Basin, D.A.: Cutoff bounds for consensus algorithms. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 217–237. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_12
28. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall Inc., Upper Saddle River (1967)
29. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
30. Raynal, M.: Fault-Tolerant Agreement in Synchronous Message-Passing Systems. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2010)

31. Srikanth, T.K., Toueg, S.: Optimal clock synchronization. J. ACM **34**(3), 626–645 (1987)
32. Srikanth, T., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib. Comput. **2**, 80–94 (1987)
33. Stoilkovska, I., Konnov, I., Widder, J., Zuleger, F.: Artifact and instructions to generate experimental results for TACAS 2019 paper: Verifying Safety of Synchronous Fault-Tolerant Algorithms by Bounded Model Checking (artifact). Figshare (2019). https://doi.org/10.6084/m9.figshare.7824929.v1