SMT and POR beat Counter Abstraction: Parameterized Model Checking of Threshold-Based Distributed Algorithms*

Igor Konnov, Helmut Veith, and Josef Widder

TU Wien (Vienna University of Technology)

Abstract. Automatic verification of threshold-based fault-tolerant distributed algorithms (FTDA) is challenging: they have multiple parameters that are restricted by arithmetic conditions, the number of processes and faults is parameterized, and the algorithm code is parameterized due to conditions counting the number of received messages. Recently, we introduced a technique that first applies data and counter abstraction and then runs bounded model checking (BMC). Given an FTDA, our technique computes an upper bound on the diameter of the system. This makes BMC complete: it always finds a counterexample, if there is an actual error. To verify state-of-the-art FTDAs, further improvement is needed. In this paper, we encode bounded executions over integer counters in SMT. We introduce a new form of offline partial order reduction that exploits acceleration and the structure of the FTDAs. This aggressively prunes the execution space to be explored by the solver. In this way, we verified safety of seven FTDAs that were out of reach before.

1 Introduction

In recent work [28] we applied bounded model checking to verify reachability properties of threshold-based fault-tolerant distributed algorithms (FTDA), which are parameterized in the number of processes n and the fraction of faults t, e.g., $n > 3t$. Moreover, we showed how to make bounded model checking complete in the parameterized case. In particular, we considered counter systems where we record for each local state, how many processes are in this state. We have one counter per local state ℓ, denoted by $\kappa[\ell]$. A process step from local state ℓ to local state ℓ' is modeled by decrementing $\kappa[\ell]$ and incrementing $\kappa[\ell']$. When δ processes perform the same step one after the other, we allow the processes to do the *accelerated step* that instantaneously changes two counters by δ. The number δ is called *acceleration factor*, it can vary within a single run.

As we focus on FTDAs, we consider specific counter systems, namely those defined by *threshold automata*. Here, transitions are guarded by *threshold guards* that compare a shared integer variable to a linear combination of parameters, e.g., $x \geq n - t$ or $x < t$, where x is a shared variable and n and t are parameters.

* Supported by the Austrian National Research Network S11403 and S11405 (RiSE), and project P27722 (PRAVDA) of the Austrian Science Fund (FWF).

Completeness of the method [28] with respect to reachability is shown by proving a bound on the diameter of the accelerated system. Inspired by Lamport's view of distributed computation as partial order on events [30], our method is in essence an *offline* partial order reduction. Instead of pruning executions that are "similar" to ones explored before [22,43,38], we use the partial order to show (offline) that every run has a similar run of bounded length. Interestingly, the bound is independent of the parameters. In combination with the data abstraction of [25], we obtained the following automated method [28]:

1. Apply a parametric data abstraction to the process code to get a finite state process description, and construct the threshold automaton (TA) [25,27].
2. Compute the diameter bound, based on the control flow of the TA.
3. Construct a system with abstract counters, i.e., a counter abstraction [39,25].
4. Perform SAT-based bounded model checking [7,16] up to the diameter bound, to check whether bad states are reached in the counter abstraction.
5. If a counterexample is found, check its feasibility and refine, if needed [13,25].

While this allowed us to automatically verify several FTDAs not verified before, there remained two bottlenecks for scalability to larger and more complex protocols: First, due to abstraction there were spurious counterexamples. Second, counter abstraction works well in practice only for processes with a few dozens of local states, but it does not scale to hundreds of local states; partly because many different interleavings result in a large search space.

To address these bottlenecks, we make two crucial contributions in this paper: First, to eliminate one of the two sources of spurious counterexamples, namely, the non-determinism added by abstract counters, we do bounded model checking using SMT solvers with linear integer arithmetic on the accelerated system, instead of SAT-based bounded model checking on the counter abstraction.

Second, we reduce the search space dramatically: We introduce the notion of an *execution schema* that is defined as a sequence of local rules of the TA. By assigning to each rule of a schema an acceleration factor (possibly 0), one obtains a run of the counter system. Hence, each schema represents infinitely many runs. We show how to construct a set of schemas whose set of reachable states coincides with the set of reachable states of the accelerated counter system.

Our construction can be seen as an aggressive partial order reduction, where each run has a similar run generated by a schema from the set. To show this, we capture the guards that are locked and unlocked in a *locking context*. Our key insight is that a bounded number of transitions changes the context in each run. For example, of all transitions increasing a variable x, at most one makes $x \geq n - t$ true, and at most one makes $x < t$ false (the parameters n and t are fixed in a run). We fix those transitions that change the context, and apply the ideas of partial order reduction to the subexecutions between these transitions.

Our experiments show that SMT solvers and schemas outperform SAT solvers and counter abstraction in parameterized verification of threshold-based FTDAs. Indeed, we verified safety of complicated FTDAs [37,40,23,41,10,18] that have not been automatically verified before. In addition we achieved dramatic speedup and reduced memory footprint on previously verified FTDAs [9,42,12] (cf. [28]).

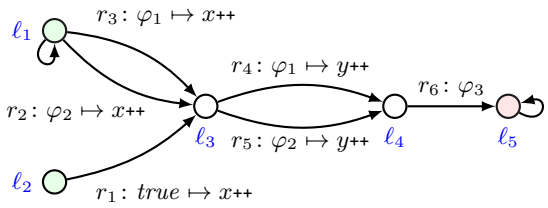

Fig. 1. An example threshold automaton.

2 A Motivating Example

Figure 1 is an example of a threshold automaton TA over two shared variables x and y and parameters n, t, and f. It is inspired by the distributed asynchronous broadcast protocol from [9], where $n - f$ correct processes concurrently execute TA, and f processes are Byzantine. As is typical for FTDAs, the parameters must satisfy a resilience condition, e.g., $n > 3t \wedge t \geq f \geq 0$ stating that less than a third of the processes is faulty. The circles depict the local states $\ell_1, \ldots, \ell_5$, two of them are the initial states ℓ_1, ℓ_2. The edges depict the rules $r_1, \ldots, r_6$ labeled with guarded commands $\varphi \mapsto \mathsf{act}$, where φ is one of the threshold guards "$\varphi_1 \colon x \geq \lceil (n+t)/2 \rceil - f$", "$\varphi_2 \colon y \geq (t+1) - f$", and "$\varphi_3 \colon y \geq (2t+1) - f$", and an action act increases the shared variables x or y by one, or zero (as in rule r_6).

Every local state ℓ_i has a non-negative counter $\boldsymbol{\kappa}[\ell_i]$ that represents the number of processes in ℓ_i. Together with the values of x, y, n, t, and f, the values of the counters constitute a configuration of the system. In the initial configuration there are $n - f$ processes in initial states, i.e., $\boldsymbol{\kappa}[\ell_1] + \boldsymbol{\kappa}[\ell_2] = n - f$, and the other counters and the shared variables x and y are zero.

The rules define the transitions of the counter system. For instance, according to the rule r_2, if in the current configuration the guard $y \geq t + 1 - f$ holds true and $\boldsymbol{\kappa}[\ell_1] \geq 5$, then five processes can instantaneously move out of the local state ℓ_1 to the local state ℓ_3, and increment x as prescribed by the action of r_2. This results in increase of x and the counter $\boldsymbol{\kappa}[\ell_3]$ by five, and counter $\boldsymbol{\kappa}[\ell_1]$ is decreased by five. When, as in this example, rule r_2 is conceptually executed by 5 processes, we denote this transition by r_2^5.

We now consider the runs more closely. As initially x and y are zero, threshold guards φ_1, φ_2, and φ_3 evaluate to false. As rules may only increase variables, these guards may eventually become true. (In this example we do not consider guards like $x < t$ that are initially true and become false, although we treat them later.) In fact, initially only r_1 is unlocked. Because r_1 increases x, it may unlock φ_1. Thus r_4 becomes unlocked. Rule r_4 increases y and thus repeated execution of r_4 (by different processes) first unlocks φ_2 and then φ_3. We call the set of conditions which evaluate to true in a configuration the *context*. For our example we observe that each run goes through the following sequence of contexts $\{\}$, $\{\varphi_1\}$, $\{\varphi_1, \varphi_2\}$, and $\{\varphi_1, \varphi_2, \varphi_3\}$. In fact, the sequence of contexts in an execution of a TA is always monotonic.

The conjunction of the guards in the context $\{\varphi_1, \varphi_2\}$ implies the guards of the rules r_1, r_2, r_3, r_4, r_5; we say they are unlocked in the context. A technical challenge addressed in this paper is to show that a fixed sequence of these rules can "capture" all schedules allowed in this context. To this end we analyze the control flow of the TA. Our example is acyclic up to self-loops, and thus the control flow establishes a partial order on the rules. We show in this paper that we can use any linear extension of this partial order as the required fixed sequence, e.g., $r_1 < r_2 < r_3 < r_4 < r_5$. (In this example we do not deal with loops, although we handle them in Section 4.1.) It remains to deal with transitions that actually change the context. In our example, only r_4 or r_5 can change the context from $\{\varphi_1, \varphi_2\}$ to $\{\varphi_1, \varphi_2, \varphi_3\}$. Therefore we append r_4, r_5 — that change the context — to the fixed sequence for the context. Thus, we obtain the following schema, where inside the curly brackets we give the contexts, and between two contexts the fixed sequences of rules. (We discuss the underlined rules below.)

$$S = \{\}\ \underline{r_1}, \underline{r_1}\ \{\varphi_1\}\ \underline{r_1}, \underline{r_3}, r_4, \underline{r_4}\ \{\varphi_1, \varphi_2\}$$
$$r_1, r_2, r_3, r_4, r_5, r_4, \underline{r_5}\ \{\varphi_1, \varphi_2, \varphi_3\}\ r_1, r_2, r_3, r_4, \underline{r_5}, \underline{r_6}\ \{\varphi_1, \varphi_2, \varphi_3\}$$

We now show how each schedule is captured by schema S. Consider, e.g., a schedule from the initial state σ_0 with $n = 5$, $t = f = 1$, $\boldsymbol{\kappa}[\ell_1] = 1$, and $\boldsymbol{\kappa}[\ell_2] = 3$. We are interested in whether there is a schedule that reaches a configuration, where all processes are in state ℓ_5. Consider the following schedule:

$$\tau \;=\; \underbrace{r_1^1}_{\tau_1}, \underbrace{r_1^1}_{t_1}, \underbrace{r_3^1, r_1^1}_{\tau_2}, \underbrace{r_4^1}_{t_2}, \underbrace{\;}_{\tau_3}, \underbrace{r_5^1}_{t_3}, \underbrace{r_6^1, r_5^1, r_5^1, r_6^1, r_6^1, r_6^1}_{\tau_4}$$

Observe that after r_1^1, r_1^1, variable $x = 2$ and φ_1 is true. Hence transition t_1 changes the context from $\{\}$ to $\{\varphi_1\}$. Similarly t_2 and t_3 change the context. Context changing transitions are marked with curly brackets. Between them we have the subschedules $\tau_1, \ldots, \tau_4$ (τ_3 is empty) marked with square brackets.

To show that this schedule is captured by the schema, we apply partial order arguments regarding distributed computations: As the guards φ_2 and φ_3 evaluate to true in τ_4, and r_5 precedes r_6 in the control flow of the TA, all transitions r_5^1 can be moved to the left in τ_4. Similarly, r_1^1 can be moved to the left in τ_2. The resulting schedule is applicable and leads to the same configuration as the original one. Further, we can accelerate the adjacent transitions with the same rule, e.g., the sequence r_5^1, r_5^1 can be transformed into r_5^2. Thus, we transform subschedules τ_i into τ_i', and arrive at the following schedule τ' that we call the representative schedule of τ. Importantly for reachability checking, if τ and τ' are applied to the same configuration, they end in the same configuration.

$$\tau' = \underbrace{r_1^1}_{\tau_1'}, \underbrace{r_1^1}_{t_1}, \underbrace{r_1^1, r_3^1}_{\tau_2'}, \underbrace{r_4^1}_{t_2}, \underbrace{\;}_{\tau_3'}, \underbrace{r_5^1}_{t_3}, \underbrace{r_5^2, r_6^4}_{\tau_4'}$$

Reconsidering schema S, we observe that the sequence of underlined rules in S matches the schedule τ'. In this paper we show that every schedule can be transformed into a representative schedule that matches one schema from a small set of

schemas. Each schema in this set corresponds to one of the monotonic sequences of contexts, and is constructed following the ideas from above. Completeness regarding reachability follows from the fact that each schedule goes through a monotonic sequence of contexts. For each schema, reachability can be expressed by an SMT formula involving both state variables and parameters.

3 Parameterized Counter Systems

We extend the framework of [28]. A threshold automaton describes a process in a concurrent system, and is a tuple $\mathsf{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$ defined below.

States. The finite set $\mathcal{L}$ contains the *local states*, and $\mathcal{I} \subseteq \mathcal{L}$ is the set of *initial states*. The finite set Γ contains the *shared variables* that range over $\mathbb{N}_0$. The finite set Π is a set of *parameter variables* that range over $\mathbb{N}_0$, and the *resilience condition* RC is a formula over parameter variables in linear integer arithmetic, e.g., $n > 3t$. The set of *admissible parameters* is $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Pi|} \colon \mathbf{p} \models RC\}$.

Rules. A rule defines a conditional transition between local states that may update the shared variables. Formally, a *rule* is a tuple $(\mathit{from}, \mathit{to}, \varphi^{\leq}, \varphi^{>}, \mathbf{u})$: The local states *from* and *to* are from $\mathcal{L}$, and capture from which local state to which a process moves. A rule is only executed if the conditions $\varphi^{\leq}$ and $\varphi^{>}$ evaluate to true. Each condition is a conjunction of guards. Each guard is defined using some shared variable $x \in \Gamma$, coefficients $a_0, \ldots, a_{|\Pi|} \in \mathbb{Z}$, and parameter variables $p_1, \ldots, p_{|\Pi|} \in \Pi$ so that $a_0 + \sum_{i=1}^{|\Pi|} a_i \cdot p_i \leq x$ and $a_0 + \sum_{i=1}^{|\Pi|} a_i \cdot p_i > x$ are a *lower guard* and *upper guard*, respectively. Let Φ^U and Φ^L be the sets of lower and upper guards. The set $\mathsf{guard}(\varphi^{\leq}) \subseteq \Phi^U$ is the set of guards used in $\varphi^{\leq}$, while the set $\mathsf{guard}(\varphi^{>}) \subseteq \Phi^L$ is the set of guards used in $\varphi^{>}$.

Rules may increase shared variables using an update vector $\mathbf{u} \in \mathbb{N}_0^{|\Gamma|}$ that is added to the vector of shared variables. Finally, $\mathcal{R}$ is the finite set of rules.

Definition 1. *Given a threshold automaton $(\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$, we define the* precedence relation $\prec_P$*: for a pair of rules $r_1, r_2 \in \mathcal{R}$, it holds that $r_1 \prec_P r_2$ if and only if $r_1.\mathit{to} = r_2.\mathit{from}$. We denote by $\prec_P^+$ the transitive closure of $\prec_P$. Further, we say that $r_1 \sim_P r_2$, if $r_1 \prec_P^+ r_2 \wedge r_2 \prec_P^+ r_1$, or $r_1 = r_2$.*

As in [28], we limit ourselves to threshold automata relevant for FTDAs, namely those where $r.\mathbf{u} = \mathbf{0}$ for all rules $r \in \mathcal{R}$ that satisfy $r \prec_P^+ r$.

Looplets. The relation $\sim_P$ defines equivalence classes of rules. An equivalence class corresponds to a loop or a single rule that is not part of a loop. Hence, we use the term looplet for one such equivalence class. For a given set of rules $\mathcal{R}$ let $\mathcal{R}/\!\sim$ be the set of equivalence classes defined by $\sim_P$. We denote by $[r]$ the equivalence class of rule r. For two classes c_1 and c_2 from $\mathcal{R}/\!\sim$ we write $c_1 \prec_C c_2$ iff there are two rules r_1 and r_2 in $\mathcal{R}$ satisfying $[r_1] = c_1$ and $[r_2] = c_2$ and $r_1 \prec_P^+ r_2$ and $r_1 \not\sim_P r_2$. As the relation $\prec_C$ is a strict partial order, there are linear extensions of $\prec_C$. Below, we fix an *arbitrary* of these linear extensions to sort transitions in a schedule: We denote by $\prec_C^{lin}$ a linear extension of $\prec_C$.

3.1 Counter Systems

Given a threshold automaton TA, a function $N\colon \mathbf{P}_{RC} \to \mathbb{N}_0$ that determines the number of processes to be modeled (typically, $N(n,t,f) = n - f$) and admissible parameter values $\mathbf{p} \in \mathbf{P}_{RC}$, we define a counter system as a transition system (Σ, I, R), that consists of the set of configurations Σ, which contain the counters and variables, the set of initial configurations I, and the transition relation R:

Configurations Σ and I. A configuration $\sigma = (\boldsymbol{\kappa}, \mathbf{g}, \mathbf{p})$ consists of a vector of *counter values* $\sigma.\boldsymbol{\kappa} \in \mathbb{N}_0^{|\mathcal{L}|}$ (for simplicity we use the convention that $\mathcal{L} = \{1, \ldots, |\mathcal{L}|\}$) a vector of *shared variable values* $\sigma.\mathbf{g} \in \mathbb{N}_0^{|\Gamma|}$, and a vector of *parameter values* $\sigma.\mathbf{p} = \mathbf{p}$. The set Σ is the set of all configurations. The set of initial configurations I contains the configurations that satisfy $\sigma.\mathbf{g} = \mathbf{0}$, $\sum_{i \in \mathcal{I}} \sigma.\boldsymbol{\kappa}[i] = N(\mathbf{p})$, and $\sum_{i \notin \mathcal{I}} \sigma.\boldsymbol{\kappa}[i] = 0$.

Transition relation R. A *transition* is a pair $t = (\mathit{rule}, \mathit{factor})$ of a rule of the TA and a non-negative integer called the *acceleration factor*, or just factor for short. For a transition $t = (\mathit{rule}, \mathit{factor})$ we refer by $t.\mathbf{u}$ to $\mathit{rule}.\mathbf{u}$, by $t.\varphi^{>}$ to $\mathit{rule}.\varphi^{>}$, etc. We say a transition t is *unlocked* in configuration σ if $\forall k \in \{0, \ldots, t.\mathit{factor} - 1\}$. $(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + k \cdot t.\mathbf{u}, \sigma.\mathbf{p}) \models t.\varphi^{\leq} \wedge t.\varphi^{>}$.

A transition t is *applicable (or enabled)* in configuration σ, if it is unlocked, and $\sigma.\boldsymbol{\kappa}[t.\mathit{from}] \geq t.\mathit{factor}$, or $t.\mathit{factor} = 0$. We say that σ' is the result of applying the enabled transition t to σ, and use the notation $\sigma' = t(\sigma)$, if

- $\sigma'.\mathbf{g} = \sigma.\mathbf{g} + t.\mathit{factor} \cdot t.\mathbf{u}$ and $\sigma'.\mathbf{p} = \sigma.\mathbf{p}$
- if $t.\mathit{from} \neq t.\mathit{to}$ then $\sigma'.\boldsymbol{\kappa}[t.\mathit{from}] = \sigma.\boldsymbol{\kappa}[t.\mathit{from}] - t.\mathit{factor}$ and $\sigma'.\boldsymbol{\kappa}[t.\mathit{to}] = \sigma.\boldsymbol{\kappa}[t.\mathit{to}] + t.\mathit{factor}$ and $\forall \ell \in \mathcal{L} \setminus \{t.\mathit{from}, t.\mathit{to}\}$. $\sigma'.\boldsymbol{\kappa}[\ell] = \sigma.\boldsymbol{\kappa}[\ell]$
- if $t.\mathit{from} = t.\mathit{to}$ then $\sigma'.\boldsymbol{\kappa} = \sigma.\boldsymbol{\kappa}$

The transition relation $R \subseteq \Sigma \times \Sigma$ of the counter system is defined as follows: $(\sigma, \sigma') \in R$ iff there is a $r \in \mathcal{R}$ and a $k \in \mathbb{N}_0$ such that $\sigma' = t(\sigma)$ for $t = (r, k)$. As updates to shared variables do not decrease their values, we obtain:

Proposition 1 ([28]). *For all configurations σ, all rules r, and all transitions t applicable to σ, the following holds:*

1. *If $\sigma \models r.\varphi^{\leq}$ then $t(\sigma) \models r.\varphi^{\leq}$*
2. *If $t(\sigma) \not\models r.\varphi^{\leq}$ then $\sigma \not\models r.\varphi^{\leq}$*
3. *If $\sigma \not\models r.\varphi^{>}$ then $t(\sigma) \not\models r.\varphi^{>}$*
4. *If $t(\sigma) \models r.\varphi^{>}$ then $\sigma \models r.\varphi^{>}$*

Schedules and paths. A *schedule* is a sequence of transitions. For a schedule τ and an index $i : 1 \leq i \leq |\tau|$, by $t[i]$ we denote the ith transition of τ, and by τ^i we denote the prefix $t[1], \ldots, t[i]$ of τ. A schedule $\tau = t_1, \ldots, t_m$ is *applicable* to configuration σ_0, if there is a sequence of configurations $\sigma_1, \ldots, \sigma_m$ with $\sigma_i = t_i(\sigma_{i-1})$ for $1 \leq i \leq m$. If there is a $t_i.\mathit{factor} > 1$, then a schedule is *accelerated*.

By $\tau \cdot \tau'$ we denote the concatenation of two schedules τ and τ'. A sequence $\sigma_0, t_1, \sigma_1, \ldots, \sigma_{k-1}, t_k, \sigma_k$ of alternating configurations and transitions is called a (finite) *path*, if transition t_i is enabled in σ_i and $\sigma_i = t_i(\sigma_{i-1})$, for $1 \leq i \leq k$. For a configuration σ_0 and a schedule τ applicable to σ, by $\mathsf{path}(\sigma_0, \tau)$ we denote the path $\sigma_0, t_1, \ldots, t_{|\tau|}, \sigma_{|\tau|}$ with $t_i = \tau[i]$ and $\sigma_i = t_i(\sigma_{i-1})$, for $1 \leq i \leq |\tau|$.

3.2 Contexts and Slices

The evaluation of the guards in the sets Φ^U and Φ^L solely defines whether certain rules are unlocked. Due to Proposition 1, we infer that when the transitions of a schedule are applied, more and more guards from Φ^U become unlocked and more and more guards from Φ^L become locked. To capture this, we define:

Definition 2. *A* context *is a pair (Ω^U, Ω^L) of subsets $\Omega^U \subseteq \Phi^U$ and $\Omega^L \subseteq \Phi^L$. We denote by Ω the pair (Ω^U, Ω^L), and by $|\Omega| = |\Omega^U| + |\Omega^L|$.*

For two contexts (Ω_1^U, Ω_1^L) and (Ω_2^U, Ω_2^L), we define that $(\Omega_1^U, \Omega_1^L) \sqsubset (\Omega_2^U, \Omega_2^L)$ if and only if $\Omega_1^U \cup \Omega_1^L \subset \Omega_2^U \cup \Omega_2^L$. Then, a sequence of contexts $\Omega_1, \ldots, \Omega_m$ is *monotonically increasing*, if $\Omega_i \sqsubset \Omega_{i+1}$ for $1 \leq i < m$. Further, a monotonically increasing sequence of contexts $\Omega_1, \ldots, \Omega_m$ is *maximal*, if $\Omega_1 = (\emptyset, \emptyset)$ and $\Omega_m = (\Phi^U, \Phi^L)$ and $|\Omega_{i+1}| = |\Omega_i| + 1$, for $1 \leq i < m$. We obtain:

Proposition 2. *Every maximal monotonically increasing sequence of contexts is of length $|\Phi^U| + |\Phi^L| + 1$. There are at most $(|\Phi^U| + |\Phi^L|)!$ such sequences.*

Definition 3. *Given a threshold automaton, we define its configuration context as a function $\omega : \Sigma \to 2^{\Phi^U} \times 2^{\Phi^L}$ that for each configuration $\sigma \in \Sigma$ gives a context (Ω^U, Ω^L) with $\Omega^U = \{\varphi \in \Phi^U : \sigma \models \varphi\}$ and $\Omega^L = \{\varphi \in \Phi^L : \sigma \not\models \varphi\}$.*

Proposition 3. *If a transition t is enabled in a configuration σ, then either $\omega(\sigma) \sqsubset \omega(t(\sigma))$, or $\omega(\sigma) = \omega(t(\sigma))$.*

We say that a schedule τ is *steady* for a configuration σ, if for every prefix τ' of τ, the context does not change, i.e., $\omega(\tau'(\sigma)) = \omega(\sigma)$.

Proposition 4. *A schedule τ is steady for a configuration σ if and only if $\omega(\sigma) = \omega(\tau(\sigma))$.*

Given a configuration σ and a schedule τ applicable to σ, we say that $\mathsf{path}(\sigma, \tau)$ *is consistent with* a sequence of contexts $\Omega_1, \ldots, \Omega_m$, if the set of indices $\{0, \ldots, |\tau|\}$ can be partitioned into m (possibly empty) disjoint sets $I_1, \ldots, I_m$ such that $\omega(\tau^i(\sigma)) = \Omega_k$, for $1 \leq k \leq m$ and $i \in I_k$.

A context defines which rules of the TA are unlocked. As we consider steady schedules, we need to understand, which rules are unlocked in that schedule:

Definition 4. *Given a threshold automaton $\mathsf{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$ and a context Ω, we define the* slice *of TA with context Ω as a threshold automaton $\mathsf{TA}|_\Omega = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}|_\Omega, RC)$, where a rule $r \in \mathcal{R}$ belongs to $\mathcal{R}|_\Omega$ if and only if $\big(\bigwedge_{\varphi \in \Omega^U} \varphi\big) \to r.\varphi^{\leq}$ and $\big(\bigwedge_{\psi \in \Phi^L \setminus \Omega^L} \psi\big) \to r.\varphi^{>}$.*

3.3 Parameterized Reachability

Given a threshold automaton TA, a *state property* B is a boolean combination of formulas that have the form $\bigwedge_{i \in Y} \boldsymbol{\kappa}[i] = 0$, for some $Y \subseteq \mathcal{L}$. The *parameterized reachability* problem is to decide whether there are parameter values $\mathbf{p} \in \mathbf{P}_{RC}$, an initial configuration $\sigma_0 \in I$, with $\sigma_0.\mathbf{p} = \mathbf{p}$, and a schedule τ satisfying that τ is applicable to σ_0, and property B holds in the final state: $\tau(\sigma_0) \models B$.

4 Main result: a complete set of schemas

We introduce the notion of a schema that is an alternating sequence of contexts and sequences of rules. A schema serves as a pattern for an infinite set of paths, and can be used to efficiently encode parameterized reachability in SMT. We show how to construct a set of schemas $\mathcal{S}$ with the following property: for each schedule τ and each configuration σ, there is a representative schedule, i.e., a schedule that if applied to σ, ends in $\tau(\sigma)$, and is generated by a schema from $\mathcal{S}$.

Definition 5. *A schema is a sequence $\Omega_0, \rho_1, \Omega_1, \ldots, \rho_m, \Omega_m$ of alternating contexts and rule sequences. Often we write $\{\Omega_0\}\rho_1\{\Omega_1\}\ldots\{\Omega_{m-1}\}\rho_m\{\Omega_m\}$ for a schema. A schema with two contexts is called simple.*

Given two schemas $S_1 = \Omega_0, \rho_1, \ldots, \rho_k, \Omega_k$ and $S_2 = \Omega'_0, \rho'_1, \ldots, \rho'_m, \Omega'_m$ with $\Omega_k = \Omega'_0$, we define their *composition* $S_1 \circ S_2$ to be the schema that is obtained by concatenation of the two sequences: $\Omega_0, \rho_1, \ldots, \rho_k, \Omega'_0, \rho'_1, \ldots, \rho'_m, \Omega'_m$.

Definition 6. *Given a configuration σ and a schedule τ applicable to σ, we say that $\mathsf{path}(\sigma, \tau)$ is generated by a simple schema $\{\Omega\}\,\rho\,\{\Omega'\}$, if the following hold:*

- *For $\rho = r_1, \ldots, r_k$ there is a monotonically increasing sequence of indices $i(1), \ldots, i(m)$, i.e., $1 \leq i(1) < \cdots < i(m) \leq k$, and there are factors $f_1, \ldots, f_m > 0$, so that schedule $(r_{i(1)}, f_1), \ldots, (r_{i(m)}, f_m) = \tau$.*
- *The first and the last states match the contexts: $\omega(\sigma) = \Omega$ and $\omega(\tau(\sigma)) = \Omega'$.*

In general, we say that $\mathsf{path}(\sigma, \tau)$ is generated by a schema S, if $S = S_1 \circ \cdots \circ S_k$ for simple schemas $S_1, \ldots, S_k$ and $\tau = \tau_1 \cdots \tau_k$ such that each $\mathsf{path}(\pi_i(\sigma), \tau_i)$ is generated by the simple schema S_i, for $\pi_i = \tau_1 \cdots \tau_{i-1}$ and $1 \leq i \leq k$.

The *language of a schema* S — denoted with $\mathcal{L}(S)$ — is the set of all paths generated by S. For a set of configurations $C \subseteq \Sigma$ and a set of schemas $\mathcal{S}$, we define the set $\mathsf{Reach}(C, \mathcal{S})$ to contain all configurations reachable from C via the paths generated by the schemas from $\mathcal{S}$, i.e., $\mathsf{Reach}(C, \mathcal{S}) = \{\tau(\sigma) \mid \sigma \in C,\ \exists S \in \mathcal{S}.\ \mathsf{path}(\sigma, \tau) \in \mathcal{L}(S)\}$. We say that a set $\mathcal{S}$ of schemas is *complete*, if: $\forall C \subseteq \Sigma.\ \{\tau(\sigma) \mid \sigma \in C,\ \tau \text{ is applicable to } \sigma\} = \mathsf{Reach}(C, \mathcal{S})$.

In [28, Thm. 1], we introduced a quantity $\mathcal{C}$ that depends on the number of conditions in a TA, and have shown that for every configuration σ and every schedule τ applicable to σ, there is a schedule τ' of length at most $d = |\mathcal{R}| \cdot (\mathcal{C} + 1) + \mathcal{C}$ that is also applicable to σ and results in $\tau(\sigma)$. Hence, by enumerating all sequences of rules of length up to d, one can construct a complete set of schemas:

Corollary 1. *For a threshold automaton, there is a complete schema set $\mathcal{S}_d$ of cardinality $|\mathcal{R}|^{|\mathcal{R}| \cdot (\mathcal{C}+1)+\mathcal{C}}$.*

Although the set $\mathcal{S}_d$ is finite, enumerating all its elements is impractical. We show that there is a complete set of schemas whose cardinality solely depends on the number of guards that syntactically occur in the TA. These numbers $|\Phi^U|$ and $|\Phi^L|$ are in practice much smaller than the number of rules $|\mathcal{R}|$:

Theorem 1. *For a threshold automaton, there is a complete schema set of cardinality at most* $(|\Phi^U| + |\Phi^L|)!$*, where the length of each schema does not exceed* $(3 \cdot (|\Phi^U| + |\Phi^L|) + 2) \cdot |\mathcal{R}|$.

Proof idea. Construct the set Z of all maximal monotonically increasing sequences of contexts. From Proposition 2, there are at most $(|\Phi^U|+|\Phi^L|)!$ maximal monotonically increasing sequences of contexts. Therefore, $|Z| \leq (|\Phi^U| + |\Phi^L|)!$. Then, for each sequence $z \in Z$, we do the following:

1. Show that for each configuration σ and each schedule τ applicable to σ and consistent with the sequence z, there is a schedule $s(\tau)$ that has a specific structure, and is also applicable to σ. We call $s(\tau)$ the representative of τ.
2. Construct a schema and show that it generates all paths of all schedules $s(\tau)$ found in (1). The length of the schema is at most $(3 \cdot (|\Phi^U| + |\Phi^L|) + 2) \cdot |\mathcal{R}|$.

To prove Theorem 1, it remains to show existence of a representative schedule and of a schema as formulated in (1)–(2). We do this below in Proposition 9 and Theorem 2 respectively. Before that we consider special cases: when all rules of a schedule belong to the same looplet, and when a schedule is steady.

4.1 Special case I: one context and one looplet

We show that for each schedule that uses only the rules from a fixed looplet and does not change its context, there exists a representative schedule of bounded length that reaches the same final state.

Proposition 5. *Fix a threshold automaton, a context* Ω*, and a looplet* $c \in (\mathcal{R}|_\Omega)/_\sim$ *in the slice* $\mathsf{TA}|_\Omega$*. Let* σ *be a configuration and* $\tau = t_1, \ldots, t_m$ *a steady schedule applicable to* σ*, with* $[t_i.rule] = c$ *for* $1 \leq i \leq |\tau|$*. There exists a representative schedule* $\mathsf{crep}_c^\Omega[\sigma, \tau]$ *with the following properties:*

a) schedule $\mathsf{crep}_c^\Omega[\sigma, \tau]$ *is applicable to* σ*, and* $\mathsf{crep}_c^\Omega[\sigma, \tau](\sigma) = \tau(\sigma)$*,*
b) the rule of each transition t *in* $\mathsf{crep}_c^\Omega[\sigma, \tau]$ *belongs to* c*, that is,* $[t.rule] = c$*,*
c) schedule $\mathsf{crep}_c^\Omega[\sigma, \tau]$ *is not longer than* $2 \cdot |c|$.

Proof idea for Proposition 5. If $|c| = 1$, then we use a single accelerated transition or the empty schedule as representative. If $|c| > 1$, the rules of the slice $\mathsf{TA}|_\Omega$ form a strongly connected component. Then, we can choose a node h, and construct two spanning trees: an *out-tree*, whose edges are pointing away from h, and an *in-tree*, whose edges are pointing to h. Using the trees, we construct two sequences of rules sorted in the topological order of the trees: the sequence $r_{\mathsf{in}}(1), \ldots, r_{\mathsf{in}}(k)$ moves processes to h, and the sequence $r_{\mathsf{out}}(1), \ldots, r_{\mathsf{out}}(m)$ distributes the processes from h to the locations. As a result, for each location ℓ in the graph, the processes are transferred from ℓ to the other locations, if $\sigma[\ell] > \tau(\sigma)[\ell]$, and additional processes arrive at ℓ, if $\sigma[\ell] < \tau(\sigma)[\ell]$.

Proposition 6. *Fix a threshold automaton, a context* Ω*, and a looplet* $c \in (\mathcal{R}|_\Omega)/_\sim$ *in the slice* $\mathsf{TA}|_\Omega$*. There exists a schema* $\mathsf{cschema}_c^\Omega$ *with the following properties: For each configuration* σ *and each steady schedule* $\tau = t_1, \ldots, t_m$ *applicable to* σ*, if* $[t_i.rule] = c$ *for* $1 \leq i \leq |\tau|$*, then* $\mathsf{path}(\sigma, \tau')$ *of the representative schedule* $\tau' = \mathsf{crep}_c^\Omega[\sigma, \tau]$ *from Proposition 5 is generated by* $\mathsf{cschema}_c^\Omega$.

Proof idea. We construct the schema using the same sequence of rules as in Proposition 5, i.e., $\mathsf{cschema}_c^\Omega = \{\Omega\}\, r_{\mathsf{in}}(1), \ldots, r_{\mathsf{in}}(k),\ r_{\mathsf{out}}(1), \ldots, r_{\mathsf{out}}(m)\, \{\Omega\}$. It follows that $\mathsf{cschema}_c^\Omega$ generates all paths of the representative schedules.

4.2 Special case II: one context and multiple looplets

In this section, we show that for each steady schedule, there exists a representative steady schedule of bounded length that reaches the same final state.

Proposition 7. *Fix a threshold automaton and a context Ω. For every configuration σ with $\omega(\sigma) = \Omega$ and every steady schedule τ applicable to σ, there exists a steady schedule $\mathsf{srep}_\Omega[\sigma, \tau]$ with the following properties:*
a) $\mathsf{srep}_\Omega[\sigma, \tau]$ *is applicable to* σ*, and* $\mathsf{srep}_\Omega[\sigma, \tau](\sigma) = \tau(\sigma)$*,*
b) $|\mathsf{srep}_\Omega[\sigma, \tau]| \leq 2 \cdot |(\mathcal{R}|_\Omega)/_\sim|$

To construct a representative schedule, we fix a context Ω of at TA, a configuration σ with $\omega(\sigma) = \Omega$, and a steady schedule τ applicable to σ. The key notion in our construction is a projection of a schedule on a set of looplets:

Definition 7. *Let $\tau = t_1, \ldots, t_k$ be a schedule and C be a set of looplets. Given an increasing sequence of indices $i(1), \ldots, i(m) \in \{1, \ldots, k\}$, i.e., $i(j) < i(j+1)$, for $1 \leq j < m$, a schedule $t_{i(1)} \ldots t_{i(m)}$ is a projection of τ on C, if each index $j \in \{1, \ldots, k\}$ belongs to $\{i(1), \ldots, i(m)\}$ if and only if $[t_j.rule] \in C$.*

In fact, each schedule τ has a unique projection on a set C. In the following, we write $\tau|_{c_1,\ldots,c_m}$ to denote the projection of τ on a set $\{c_1, \ldots, c_m\}$.

Provided that $c_1, \ldots, c_m$ are all looplets of the slice $(\mathcal{R}|_\Omega)/_\sim$ ordered with respect to $\prec_C^{lin}$, we construct the following sequences of projections on each looplet (note that π_0 is the empty schedule): $\pi_i = \tau|_{c_1} \cdot \cdots \cdot \tau|_{c_i}$ for $0 \leq i \leq m$.

Having defined $\{\pi_i\}_{0 \leq i \leq m}$, we construct the representative $\mathsf{srep}_\Omega[\sigma, \tau]$ simply as a concatenation of the representatives of each looplet:

$$\mathsf{srep}_\Omega[\sigma, \tau] = \mathsf{crep}_{c_1}^\Omega[\pi_0(\sigma), \tau|_{c_1}] \cdot \mathsf{crep}_{c_2}^\Omega[\pi_1(\sigma), \tau|_{c_2}] \cdots \mathsf{crep}_{c_m}^\Omega[\pi_{m-1}(\sigma), \tau|_{c_m}]$$

Lemma 1 (Looplet sorting). *Given a threshold automaton, a context Ω, a configuration σ, a steady schedule τ applicable to σ, and a sequence $c_1, \ldots, c_m$ of all looplets in the slice $(\mathcal{R}|_\Omega)/_\sim$ with the property $c_i \prec_C^{lin} c_j$ for $1 \leq i < j \leq m$, the following holds:*
1. *Schedule $\tau|_{c_1}$ is applicable to the configuration σ.*
2. *Schedule $\tau|_{c_2,\ldots,c_m}$ is applicable to the configuration $\tau|_{c_1}(\sigma)$.*
3. *Schedule $\tau|_{c_1} \cdot \tau|_{c_2,\ldots,c_m}$, when applied to σ, results in configuration $\tau(\sigma)$.*

Proof (of Proposition 7). By iteratively applying Lemma 1, we prove by induction that schedule $\tau|_{c_1} \cdot \cdots \cdot \tau|_{c_m}$ is applicable to σ and results in $\tau(\sigma)$. From Proposition 5, we conclude that each schedule $\tau|_{c_i}$ can be replaced by its representative $\mathsf{crep}_{c_i}^\Omega[\pi_{i-1}(\sigma), \tau|_{c_i}]$. Thus, $\mathsf{srep}_\Omega[\sigma, \tau]$ is applicable to σ and results in $\tau(\sigma)$. By Proposition 4, schedule $\mathsf{srep}_\Omega[\sigma, \tau]$ is steady, since $\omega(\sigma) = \omega(\tau(\sigma))$. □

Finally, we show that for a given context, there is a schema that generates all paths of such representative schedules.

Proposition 8. *Fix a threshold automaton and a context Ω. Let $c_1, \ldots, c_m$ be the sorted sequence of all looplets of the slice $(\mathcal{R}|_\Omega)/_\sim$, i.e., it holds that $c_1 \prec_C^{lin} \ldots \prec_C^{lin} c_m$. Schema $\mathsf{sschema}_\Omega = \mathsf{cschema}_{c_1}^\Omega \circ \mathsf{cschema}_{c_2}^\Omega \circ \cdots \circ \mathsf{cschema}_{c_m}^\Omega$ satisfies: For each configuration σ with $\omega(\sigma) = \Omega$ and each steady schedule τ applicable to σ, $\mathsf{path}(\sigma, \tau')$ of the representative $\tau' = \mathsf{srep}_\Omega[\sigma, \tau]$ is generated by $\mathsf{sschema}_\Omega$.*

Proof. As for an arbitrary configuration σ with $\omega(\sigma) = \Omega$ and a steady schedule τ applicable to σ, we constructed $\mathsf{srep}_\Omega[\sigma, \tau]$ as a sorted sequence of representatives of the looplets, all paths of $\mathsf{srep}_\Omega[\sigma, \tau]$ are generated by $\mathsf{sschema}_\Omega$. □

4.3 The general case

Using the results from Sections 4.1 and 4.2, for each configuration and each schedule (without restrictions) we construct a representative schedule.

Proposition 9. *Given a threshold automaton, a configuration σ, and schedule τ applicable to σ, there exists a schedule $\mathsf{rep}[\sigma, \tau]$ with the following properties:*
a) $\mathsf{rep}[\sigma, \tau]$ *is applicable to* σ, *and* $\mathsf{rep}[\sigma, \tau](\sigma) = \tau(\sigma)$,
b) $|\mathsf{rep}[\sigma, \tau]| \leq 2 \cdot |\mathcal{R}| \cdot (|\Phi^U| + |\Phi^L| + 1) + |\Phi^U| + |\Phi^L|$.

Proof idea. Consider the maximal monotonically increasing sequence $\Omega_0, \ldots, \Omega_m$ such that $\mathsf{path}(\sigma, \tau)$ is consistent with the sequence. Thus, τ contains at most m transitions that change their context, and schedules between these transitions are steady. By applying Proposition 7, we replace the steady schedules with their representatives and obtain $\mathsf{rep}[\sigma, \tau]$, which is applicable to σ and results in $\tau(\sigma)$. By Proposition 7, the representative of a steady schedule is not longer than $2 \cdot |\mathcal{R}|$, which together with m transitions gives us the bound $2 \cdot |\mathcal{R}| \cdot (m+1) + m$. By Proposition 2, the number m is $|\Phi^U| + |\Phi^L|$. This gives us the needed bound.

Further, given a maximal monotonically increasing sequence z of contexts, we construct a schema that generates all paths of the schedules consistent with z:

Theorem 2. *For a threshold automaton and a monotonically increasing sequence z of contexts, there exists a schema $\mathsf{schema}(z)$ that generates all paths of the representative schedules that are consistent with z, and the length of $\mathsf{schema}(z)$ does not exceed $(3 \cdot |\mathcal{R}| + 1) \cdot (|\Phi^U| + |\Phi^L|) + 2 \cdot |\mathcal{R}|$.*

Proof. Given a threshold automaton, let ρ_{all} be the sequence $r_1, \ldots, r_{|\mathcal{R}|}$ of all rules from $\mathcal{R}$, and $z = \Omega_0, \ldots, \Omega_m$ a monotonically increasing sequence of contexts. By the construction in Proposition 9, each representative schedule $\mathsf{rep}[\sigma, \tau]$ consists of the representatives of steady schedules terminated with transitions that change the context. Then, for each context Ω_i, for $0 \leq i < m$, we compose $\mathsf{sschema}_\Omega$ with $\{\Omega_i\}\ \rho_{\mathsf{all}}\ \{\Omega_{i+1}\}$. This composition generates the representative of a steady schedule and the transition changing the context from Ω_i to Ω_{i+1}. Consequently, we construct the $\mathsf{schema}(z)$ as follows:

$$(\mathsf{sschema}_{\Omega_0} \circ \{\Omega_0\}\ \rho_{\mathsf{all}}\ \{\Omega_1\}) \circ \cdots \circ (\mathsf{sschema}_{\Omega_{m-1}} \circ \{\Omega_{m-1}\}\ \rho_{\mathsf{all}}\ \{\Omega_m\}) \circ \mathsf{sschema}_{\Omega_m}$$

By inductively applying Proposition 8, we prove that $\mathsf{schema}(z)$ generates all paths of schedules $\mathsf{rep}[\sigma, \tau]$ that are consistent with the sequence z. We get the needed bound on the length of $\mathsf{schema}(z)$ by using an argument similar to Proposition 9 and by noting that we add $|\mathcal{R}|$ extra rules per context. □

Computing the Complete Set of Schemas. Our proofs show that the set of schemas is easily computed from the TA: The threshold guards are syntactic parts of the TA, and enable us to directly construct increasing sequences of contexts. To find a slice of the TA for a given context, we filter the rules with unlocked guards, i.e., check if the context contains the guard. To produce the simple schema of a looplet, we compute a spanning tree over the slice. To construct simple schemas, we do a topological sort over the looplets. For example, it takes just 30 seconds to compute the schemas in our longest experiment that runs for 4 hours.

4.4 Optimization: smaller complete sets of schemas

Entailment optimization. We say that a guard $\varphi_1 \in \Phi^U$ *entails* a guard $\varphi_2 \in \Phi^U$, if for all combinations of parameters $\mathbf{p} \in \mathbf{P}_{RC}$ and shared variables $\mathbf{g} \in \mathbb{N}_0^{|\Gamma|}$, it holds that $(\mathbf{g}, \mathbf{p}) \models \varphi_1 \to \varphi_2$. For instance, in our example, $\varphi_3 \colon y \geq (2t+1) - f$ entails $\varphi_2 \colon y \geq (t+1) - f$. If φ_1 entails φ_2, then we can omit all monotonically increasing sequences that contain a context (Ω^U, Ω^L) with $\varphi_1 \in \Omega^U$ and $\varphi_2 \notin \Omega^U$. If the number of schemas before applying this optimization is $m!$ and there are k entailments, then the number of schemas reduces from $m!$ to $(m-k)!$. A similar optimization is introduced for the guards from Φ^L.

Control flow optimization. Based on the proof of Lemma 1, we introduce the following optimization for TAs that are DAGs (possibly with self loops).

We say that a rule $r \in \mathcal{R}$ *may unlock* a lower guard $\varphi \in \Phi^U$, if there is a $\mathbf{p} \in \mathbf{P}_{RC}$ and $\mathbf{g} \in \mathbb{N}_0^{|\Gamma|}$ satisfying: $(\mathbf{g}, \mathbf{p}) \models r.\varphi^{\leq} \wedge r.\varphi^{>}$ (the rule is unlocked); $(\mathbf{g}, \mathbf{p}) \not\models \varphi$ (the guard is locked); $(\mathbf{g} + r.\mathbf{u}, \mathbf{p}) \models \varphi$ (the guard is now unlocked).

In our example, the rule r_1 may unlock the guard φ_1.

Let $\varphi \in \Phi^U$ be a guard, $r'_1, \ldots, r'_m$ be the rules that use φ, and $r_1, \ldots, r_k$ be the rules that may unlock φ. If $r_i \prec_c^{lin} r'_j$, for $1 \leq i \leq k$ and $1 \leq j \leq m$, then we exclude some sequences of contexts as follows (we call φ *forward-unlockable*). Let $\psi_1, \ldots, \psi_n \in \Phi^U$ be the guards of $r_1, \ldots, r_k$. Guard φ cannot be unlocked before $\psi_1, \ldots, \psi_n$, and thus we can omit all sequences of contexts, where φ appears in the contexts before $\psi_1, \ldots, \psi_n$. Moreover, as $\psi_1, \ldots, \psi_n$ are the only guards of the rules unlocking φ, we omit the sequences with different combinations of contexts involving φ and the guards from $\Phi^U \setminus \{\varphi, \psi_1, \ldots, \psi_n\}$. Finally, as the rules $r'_1, \ldots, r'_m$ appear after the rules $r_1, \ldots, r_k$ in the order $\prec_c^{lin}$, the rules $r'_1, \ldots, r'_m$ appear after the rules $r_1, \ldots, r_k$ in a rule sequence of every schema. Thus, we omit the combinations of the contexts involving φ and $\psi_1, \ldots, \psi_n$.

Hence, we add all forward-unlockable guards to the initial context (we still check the guards of the rules in the SMT encoding in Section 5). If the number of schemas before applying this optimization is $m!$ and there are k forward-unlocking guards, then the number of schemas reduces from $m!$ to $(m-k)!$. A similar optimization is introduced for the guards from Φ^L.

5 Checking a Schema with SMT

The encoding for a schema is obtained by decomposing the schema into a sequence of simple schemas and encoding the simple schemas. Given a simple schema $S = \{\Omega_1\}\, r_1, \ldots, r_m\, \{\Omega_2\}$, we construct an SMT formula such that every model of the formula represents a path from $\mathcal{L}(S)$, and for every path in $\mathcal{L}(S)$ there is a corresponding model of the formula. Thus, we need to model a path of $m+1$ configurations and m transitions (whose acceleration factors may be 0).

To represent a configuration σ_i, for $0 \leq i \leq m$, we introduce two vectors of SMT variables: a vector $\mathbf{k}^i = (k^i_1, \ldots, k^i_{|L|})$ to represent the process counters, a vector $\mathbf{x}^i = (x^i_1, \ldots, x^i_{|\Gamma|})$ to represent the shared variables. We call the pair $(\mathbf{k}^i, \mathbf{x}^i)$ the *layer* i, for $1 \leq i \leq m$.

A straightforward way to represent a bounded computation of length m is to encode the choice of a rule from $\mathcal{R}$ and to encode all the rules from $\mathcal{R}$ for each layer. In any case, we do not encode bounded computation but rather schemas, for which the sequence of rules $r_1, \ldots, r_m$ is fixed. We exploit this in two ways: First, instead of encoding the choice of a rule and encoding all rules, we encode for each layer i the constraints of rule r_i. Second, as this constraint may update only two counters — $r_i.\mathit{from}$ and $r_i.\mathit{to}$ — we do not need $|L|$ counter variables per layer, but only encode the two counters per layer that have actually changed. As is a common technique in bounded model checking, the counters that are not changed are "reused" from previous layers in our encoding. By doing so, we encode the schema rules with $|L| + |\Gamma| + m \cdot (2 + |\Gamma|)$ integer variables, $2m$ equations, and at most $m \cdot (|\Phi^U| + |\Phi^L|)$ inequalities over linear integer arithmetic.

6 Experiments

Implementation. We have implemented the technique in our tool ByMC (Byzantine Model Checker [2]), which integrates with an SMT solver via the interface provided by SMTLIB2. In our experiments, we used Z3 [17] as back-end solver.

Benchmarks. We revisited several asynchronous FTDAs that we evaluated in previous work [25,28]. In addition to these classic FTDAs, we considered asynchronous (Byzantine) consensus algorithms — namely, BOSCO [41], C1CS [10], and CF1S [18] — that are designed to work despite partial failure of the distributed system. All our benchmarks, their source code in our parametric extension of Promela, and the code of the threshold automata are freely available [1].

The challenge in the verification of FTDAs is the immense non-determinism caused by interleavings, asynchronous message passing, and faults. In our modeling, all these are reflected in non-deterministic choices in the Promela code. To obtain threshold automata, as required for our technique, our tool constructs a parametric interval data abstraction [25] that adds to non-determinism.

Evaluation. Table 1 summarizes our experiments conducted with nuXmv, FAST, and our new implementation. We evaluated four different tool configurations: our new implementation (SMT); our previous implementation that checks the

Input	Case	Threshold Automaton					Time, seconds				Memory, GB			
FTDA	(if more than one)	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\Phi^U\rvert$	$\lvert\Phi^L\rvert$	$\lvert\mathcal{S}\rvert$	SMT	FAST	BMC	BDD	SMT	FAST	BMC	BDD
FRB	—	6	8	1	0	1	1	1	6	6	0.1	0.1	0.1	0.1
STRB	—	7	15	3	0	4	1	1	2	2	0.1	0.1	0.1	0.1
ABA	$\frac{n+t}{2} = 2t+1$	37	180	6	0	106	18	1103	12512	8	0.1	3.5	0.8	0.1
ABA	$\frac{n+t}{2} > 2t+1$	61	392	8	0	838	294	7782	🕒	18	0.4	12.3	🕒	0.1
CBC	$\lfloor\frac{n}{2}\rfloor < n-t \wedge f = 0$	74	364	12⋆	0	1	21	⚠	12989	🕒	0.1	⚠	1.3	🕒
CBC	$\lfloor\frac{n}{2}\rfloor = n-t \wedge f = 0$	40	137	12⋆	0	1	6	⚠	132	🕒	0.1	⚠	0.3	🕒
CBC	$\lfloor\frac{n}{2}\rfloor < n-t \wedge f > 0$	115	896	17⋆	1	2	366	⚠	💣	💣	1.3	⚠	💣	💣
CBC	$\lfloor\frac{n}{2}\rfloor = n-t \wedge f > 0$	71	408	17⋆	1	2	35	⚠	🕒	🕒	0.3	⚠	🕒	🕒
NBACC	—	109	1724	6	0	106	218	💣	💣	🕒	0.5	💣	💣	🕒
NBAC	—	77	1356	6	0	106	151	💣	↻	🕒	0.3	💣	↻	🕒
NBACG	—	24	44	4	0	14	2	⚠	275	🕒	0.1	⚠	0.2	🕒
CF1S	$f = 0$	57	416	4	0	14	10	19089	12829	81	0.1	29.1	1.3	0.2
CF1S	$f = 1$	57	416	4	1	60	22	💣	5583	309	0.1	💣	0.9	0.4
CF1S	$f > 1$	98	1152	6	1	594	531	💣	🕒	49133	0.5	💣	🕒	6.0
C1CS	$f = 0$	125	1992	8	0	838	1989	💣	💣	10591	1.4	💣	💣	2.0
C1CS	$f = 1$	84	926	6	1	594	399	💣	💣	33033	0.4	💣	💣	1.0
C1CS	$f > 1$	129	2128	8	1	5808	9876	💣	☹	🕒	8.2	💣	☹	🕒
BOSCO	$\lfloor\frac{n+3t}{2}\rfloor + 1 = n-t$	58	380	6	0	106	43	💣	☹	🕒	0.1	💣	☹	🕒
BOSCO	$\lfloor\frac{n+3t}{2}\rfloor + 1 > n-t$	88	740	8	0	838	598	💣	☹	🕒	0.5	💣	☹	🕒
BOSCO	$\lfloor\frac{n+3t}{2}\rfloor + 1 < n-t$	62	420	6	0	106	46	💣	☹	🕒	0.1	💣	☹	🕒
BOSCO	$n > 5t \wedge f = 0$	134	1978	10	0	6802	13610	💣	💣	↻	9.9	💣	💣	↻
BOSCO	$n > 7t$	98	1080	8	0	838	797	💣	↻	↻	0.7	💣	↻	↻

Table 1. Summary of our experiments on AMD Opteron®6272, 32 cores, 192 GB. The symbols are: "🕒" for timeout of 24 h.; "💣" for memory overrun of 32 GB; "⚠" for BDD nodes overrun; "↻" for timeout in the refinement loop (24 h.); "☹" for spurious counterexamples due to counter abstraction. ⋆ In these cases, we used the control flow optimization from Section 4.4.

counter abstraction with nuXmv [11], either using binary decision diagrams (BDD), or SAT-based bounded model checking (BMC); and the acceleration-based tool FAST [4]. We compare our results with FAST, as TAs can be encoded with counter automata [3], which FAST receives at its input. For FAST, we give only the figures using the Mona plugin, which produced the best results in our experiments. For BMC, our tool first generates a SAT formula with nuXmv and then calls the solver Lingeling [6] to check satisfiability in non-incremental mode. This works better than the incremental mode with MiniSAT, built into nuXmv.

On large problems, our new technique works significantly better than BDD- and SAT-based model checking. BDDs work extremely well on smaller problems. Importantly, our new technique does not use abstraction refinement.

NBAC and NBACC are challenging as the model checker produces many spurious counterexamples, which are an artifact of counter abstraction losing or adding processes. When using SAT-based model checking, the individual calls to nuXmv are fast, but the abstraction-refinement loop times out, due to a large number of refinements (about 500). BDD-based model checking times out

when looking for a counterexample. Our new technique, preserves the number of proceses, and thus, there are no spurious counterexamples of this kind.

In comparison to the general-purpose acceleration tool FAST, our tool uses less memory and is faster on the benchmarks where FAST is successful.

As predicted by the distributed algorithms literature, our tool finds counterexamples, when we relax the resilience condition. In contrast to counter abstraction, our new technique gives concrete values of the parameters and shows how many processes move at each step.

Our new method uses integer counters and thus does not introduce spurious behavior caused by counter abstraction, but still has spurious counterexamples from parameterized data abstraction for complex FTDAs such as BOSCO, C1CS, NBAC, and NBACC. In these cases, we manually refine the interval domain by adding new symbolic interval borders, see [25]. We believe that these interval borders can be derived directly from the TA, so that no refinement is necessary in the first place, and leave this question to future work.

7 Discussions

We introduced a method to efficiently check reachability properties of FTDAs in a parameterized way. If $n > 7t$ as for BOSCO, even the simplest interesting case with $t = 2$ leads to a system size that is out of range of explicit state model checking. Hence, FTDAs force us to develop parameterized verification methods.

The problem we consider is concerned with parameterized model checking, for which many interesting results exist [20,19,15,14,21,26]. However, the FTDAs considered by us run under the different assumptions. In [28], we discuss the relation between partial orders in accelerated counter systems of threshold automata and the following work: compact programs [35], counter abstraction [39,5], completeness thresholds [7,16,29], partial order reduction [22,43,38,8], and Lipton's movers [34]. We also discussed their relation to counter automata. Indeed, our result entails *flattability* [33] of every counter system of threshold automata: a complete set of schemas immediately gives us a flat counter automaton. Hence, the acceleration semi-algorithms [33,3] should terminate on the systems of TAs, though it rarely happens in our experiments. Further, our execution schemas are inspired by a general notion of *semi-linear path schemas* SLPS [32,33]. We construct a small complete set of schemas and thus a provably small SLPS. Besides, in our work we distinguish counter systems and counter abstraction: the former counts processes as integers, while the latter uses counters over a finite abstract domain, e.g., $\{0, 1, many\}$ [39].

Many distributed algorithms can be specified with I/O Automata [36] or TLA+ [31]. In these frameworks, correctness is typically shown with a proof assistant, while model checking is used as a debugger on small instances. Parameterized model checking is not a concern there, except one notable result [24].

Finally, to verify all properties of FTDAs, we have to check that they are not only safe, but also progress. Liveness properties is a subject to ongoing work.

References

1. https://github.com/konnov/fault-tolerant-benchmarks/tree/master/2015
2. ByMC: Byzantine model checker (2013), http://forsyte.tuwien.ac.at/software/bymc/, accessed: Feb, 2015
3. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: Fast: acceleration from theory to practice. STTT 10(5), 401–424 (2008)
4. Bardin, S., Leroux, J., Point, G.: Fast extended release. In: CAV. LNCS, vol. 4144, pp. 63–66 (2006)
5. Basler, G., Mazzucchi, M., Wahl, T., Kroening, D.: Symbolic counter abstraction for concurrent software. In: CAV. LNCS, vol. 5643, pp. 64–78 (2009)
6. Biere, A.: Lingeling, Plingeling and Treengeling entering the SAT competition 2013. Proceedings of SAT Competition 2013; Solver and p. 51 (2013)
7. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS. LNCS, vol. 1579, pp. 193–207 (1999)
8. Bokor, P., Kinder, J., Serafini, M., Suri, N.: Efficient model checking of fault-tolerant distributed protocols. In: DSN. pp. 73–84 (2011)
9. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. J. ACM 32(4), 824–840 (1985)
10. Brasileiro, F.V., Greve, F., Mostéfaoui, A., Raynal, M.: Consensus in one communication step. In: PaCT. LNCS, vol. 2127, pp. 42–50 (2001)
11. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: CAV. LNCS, vol. 8559, pp. 334–342 (2014)
12. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. JACM 43(2), 225–267 (March 1996)
13. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
14. Clarke, E., Talupur, M., Veith, H.: Proving Ptolemy right: the environment abstraction framework for model checking concurrent systems. In: TACAS'08/ETAPS'08. pp. 33–47. Springer (2008)
15. Clarke, E., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: CONCUR 2004. vol. 3170, pp. 276–291 (2004)
16. Clarke, E.M., Kroening, D., Ouaknine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: VMCAI. LNCS, vol. 2937, pp. 85–96 (2004)
17. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 1579, pp. 337–340 (2008)
18. Dobre, D., Suri, N.: One-step consensus with zero-degradation. In: DSN. pp. 137–146 (2006)
19. Emerson, E.A., Kahlon, V.: Model checking guarded protocols. In: LICS. pp. 361–370. IEEE (2003)
20. Emerson, E., Namjoshi, K.: Reasoning about rings. In: POPL. pp. 85–94 (1995)
21. Esparza, J., Ganty, P., Majumdar, R.: Parameterized verification of asynchronous shared-memory systems. In: CAV. pp. 124–140 (2013)
22. Godefroid, P.: Using partial orders to improve automatic verification methods. In: CAV. LNCS, vol. 531, pp. 176–185 (1990)
23. Guerraoui, R.: Non-blocking atomic commit in asynchronous distributed systems with failure detectors. Distributed Computing 15(1), 17–25 (2002)
24. Jensen, H., Lynch, N.: A proof of Burns n-process mutual exclusion algorithm using abstraction. In: Steffen, B. (ed.) TACAS, LNCS, vol. 1384, pp. 409–423. Springer Berlin / Heidelberg (1998)

25. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In: FMCAD. pp. 201–209 (2013)
26. Kaiser, A., Kroening, D., Wahl, T.: Efficient coverability analysis by proof minimization. In: CONCUR. pp. 500–515 (2012)
27. Kesten, Y., Pnueli, A.: Control and data abstraction: the cornerstones of practical formal verification. STTT 2, 328–342 (2000)
28. Konnov, I., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In: CONCUR. LNCS, vol. 8704, pp. 125–140 (2014)
29. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: VMCAI. LNCS, vol. 2575, pp. 298–309 (2003)
30. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
31. Lamport, L.: Specifying systems: The TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co., Inc. (2002)
32. Leroux, J., Sutre, G.: On flatness for 2-dimensional vector addition systems with states. In: CONCUR 2004-Concurrency Theory, pp. 402–416. Springer (2004)
33. Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: ATVA. LNCS, vol. 3707, pp. 489–503 (2005)
34. Lipton, R.J.: Reduction: A method of proving properties of parallel programs. Commun. ACM 18(12), 717–721 (1975)
35. Lubachevsky, B.D.: An approach to automating the verification of compact parallel coordination programs. I. Acta Informatica 21(2), 125–169 (1984)
36. Lynch, N.: Distributed Algorithms. Morgan Kaufman (1996)
37. Mostéfaoui, A., Mourgaya, E., Parvédy, P.R., Raynal, M.: Evaluating the condition-based approach to solve consensus. In: DSN. pp. 541–550 (2003)
38. Peled, D.: All from one, one for all: on model checking using representatives. In: CAV. LNCS, vol. 697, pp. 409–423 (1993)
39. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,∞)- counter abstraction. In: CAV, LNCS, vol. 2404, pp. 93–111 (2002)
40. Raynal, M.: A case study of agreement problems in distributed systems: Non-blocking atomic commitment. In: HASE. pp. 209–214 (1997)
41. Song, Y.J., van Renesse, R.: Bosco: One-step Byzantine asynchronous consensus. In: DISC. LNCS, vol. 5218, pp. 438–450 (2008)
42. Srikanth, T., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Dist. Comp. 2, 80–94 (1987)
43. Valmari, A.: Stubborn sets for reduced state space generation. In: Advances in Petri Nets 1990, LNCS, vol. 483, pp. 491–515. Springer (1991)