On the Completeness of Bounded Model Checking for Threshold-Based Distributed Algorithms: Reachability

Igor Konnov, Helmut Veith, and Josef Widder *

Vienna University of Technology (TU Wien)

Abstract. Counter abstraction is a powerful tool for parameterized model checking, if the number of local states of the concurrent processes is relatively small. In recent work, we introduced parametric interval counter abstraction that allowed us to verify the safety and liveness of threshold-based fault-tolerant distributed algorithms (FTDA). Due to state space explosion, applying this technique to distributed algorithms with hundreds of local states is challenging for state-of-the-art model checkers. In this paper, we demonstrate that reachability properties of FTDAs can be verified by bounded model checking. To ensure completeness, we need an upper bound on the diameter, i.e., on the longest distance between states. We show that the diameters of accelerated counter systems of FTDAs, and of their counter abstractions, have a quadratic upper bound in the number of local transitions. Our experiments show that the resulting bounds are sufficiently small to use bounded model checking for parameterized verification of reachability properties of several FTDAs, some of which have not been automatically verified before.

1 Introduction

A system that consists of concurrent anonymous (identical) processes can be modeled as a counter system: Instead of recording which process is in which local state, we record for each local state, how many processes are in this state. We have one counter per local state ℓ, denoted by $\kappa[\ell]$. Each counter is bounded by the number of processes. A step by a process that goes from local state ℓ to local state ℓ' is modeled by decrementing $\kappa[\ell]$ and incrementing $\kappa[\ell']$.

We consider a specific class of counter systems, namely those that are defined by *threshold automata*. The technical motivation to introduce threshold automata is to capture the relevant properties of fault-tolerant distributed algorithms (FTDAs). FTDAs are an important class of distributed algorithms that work even if a subset of the processes fail [26]. Typically, they are parameterized in the number of processes and the number of tolerated faulty processes. These numbers of processes are parameters of the verification problem. We show

* Supported in part by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), and by the Vienna Science and Technology Fund (WWTF) grant PROSEED.

that accelerated counter systems defined by threshold automata have a diameter whose bound is independent of the bound on the counters, but depends only on characteristics of the threshold automaton. This bound can be used for parameterized model checking of FTDAs, as we confirm by experimental evaluation.

Modeling FTDAs as counter systems defined by threshold automata. A threshold automaton consists of rules that define the conditions and effects of changes to the local state of a process of a distributed algorithm. Conditions are *threshold guards* that compare the value of a shared integer variable to a linear combination of parameters, e.g., $x \geq n - t$, where x is a shared variable and n and t are parameters. This captures counting arguments which are used in FTDAs, e.g., a process takes a certain step only if it has received a message from a majority of processes. To model this, we use the shared variable x as the number of processes that have sent a message, n as the number of processes in the system, and t as the assumed number of faulty processes. The condition $x \geq n - t$ then captures a majority under the resilience condition that $n > 2t$. Resilience conditions are standard assumptions for the correctness of an FTDA. Apart from changing the local state, applying a rule can increase a shared variable, which naturally captures that a process has sent a message. Thus we consider threshold automata where shared variables are never decreased and where rules that form cycles do not modify shared variables, which is natural for modeling FTDAs.

Bounding the Diameter. For reachability it is not relevant whether we "move" processes one by one from state ℓ to ℓ'. If several processes perform the same transition one after the other, we can model this as a single update on the counters: The sequence where b processes one after the other move from ℓ to ℓ' can be encoded as a transition where $\kappa[\ell]$ is decreased by b and $\kappa[\ell']$ is increased by b. Value b is called the acceleration factor and may vary in a run depending on how many repetitions of the same transition should be captured. We call such runs of a counter system *accelerated*. The lengths of accelerated runs are the ones relevant for the diameter of the counter system.

The main technical challenge comes from the interactions of shared variables and threshold guards. We address it with the following three ideas: (i) *Acceleration* as discussed above. (ii) *Sorting*, that is, given an arbitrary run of a counter system, we can shorten it by changing the order of transitions such that there are possibly many consecutive transitions that can be merged according to (i). However, as we have arithmetic threshold conditions, not all changes of the order result in allowed runs. (iii) *Segmentation*, that is, we partition a run into segments, inside of which we can reorder the transitions; cf. (ii). In combination, these three ideas enable us to prove the main theorem: *The diameter of a counter system is at most quadratic in the number of rules; more precisely, it is bounded by the product of the number of rules and the number of distinct threshold conditions.* In particular, the diameter is independent of the parameters.

Using the Bound for Parameterized Model Checking. Parameterized model checking is concerned with the verification of concurrent or distributed systems,

where the number of processes is not a priori fixed, that is, a system is verified for all sizes. In our case, the counter systems for all values of n and t that satisfy the resilience condition should be verified. A well-known parameterized model checking technique is to map all these counter systems to a *counter abstraction*, where the counter values are not natural numbers, but range over an abstract finite domain, e.g. [29]. In [16] we developed a more general form of counter abstraction for expressions used in threshold guards, which leads, e.g., to the abstract domain of four values that capture the parametric intervals $[0, 1)$ and $[1, t + 1)$ and $[t + 1, n - t)$ and $[n - t, \infty)$. It is easy to see [16] that a counter abstraction simulates all counter systems for all parameter values that satisfy the resilience condition. The bound d on the diameter of counter systems implies a bound $\hat{d}$ on the diameter of the counter abstraction. From this and simulation follows that if an abstract state is not reachable in the counter abstraction within $\hat{d}$ steps, no concretization of this state is reachable in any of the concrete counter systems. This allows us to efficiently combine counter abstraction with *bounded model checking* [6]. Typically, bounded model checking is restricted to finding bugs that occur after a bounded number of steps of the systems. However, if one can show that within this bound every state is reachable from an initial state, bounded model checking is a complete method for verifying reachability.

2 Our approach at a glance

Figure 1 represents a threshold automaton: The circles depict the local states, and the arrows represent rules (r_1 to r_5) that define how the automaton makes transitions. Rounded corner labels correspond to conditional rules, so that the rule can only be executed if the threshold guard evaluates to true. In our example, x and y are shared variables, and n, t, and f are parameters that are assumed to satisfy the resilience condition $n \geq 2t \wedge f \leq t$. The number of processes (that each execute the automaton) depends on the parameters, in this example we assume that n processes run concurrently. Finally, rectangular labels on arrows correspond to rules that increment a shared variable. The transitions of the counter system are then defined using the rules, e.g., when rule r_2 is executed, then variable y is incremented and the counters $\kappa[\ell_3]$ and $\kappa[\ell_2]$ are updated.

Consider a counter system in which the parameter values are $n = 3$, and $t = f = 1$. Let σ_0 be the configuration where $x = y = 0$ and all counters are set to 0 except $\kappa[\ell_1] = 3$. This configuration corresponds to a concurrent system where all three processes are in ℓ_1. For illustration, we assume that in this concurrent system processes have the identifiers 1, 2, and 3, and we denote by $r_i(j)$ that process j executes rule r_i. Recall that we have anonymous (symmetric) systems, so we use the identifiers only for illustration: the transition of the counter system is solely defined by the rule being executed.

As we are interested in the diameter, we have to consider the distance between configurations in terms of length of runs. In this example, we consider the distance of σ_0 to a configuration where $\kappa[\ell_5] = 3$, that is, all three processes are in local state ℓ_5. First, observe that the rule r_5 is locked in σ_0 as $y = 0$

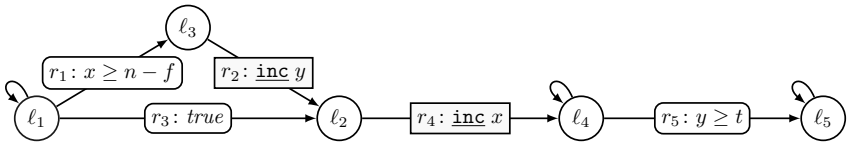

Fig. 1. Example of a Threshold Automaton

and $t = 1$. Hence, we require that rule r_2 is executed at least once so that the value of y increases. However, due to the precedence relation on the rules, before that, r_1 must be executed, which is also locked in σ_0. The sequence of transitions $\tau_1 = r_3(1), r_4(1), r_3(2), r_4(2)$ leads from σ_0 to the configuration where $\kappa[\ell_1] = 1$, $\kappa[\ell_4] = 2$, and $x = 2$; we denote it by σ_1. In σ_1, rule r_1 is unlocked, so we may apply $\tau_2 = r_1(3), r_2(3)$, to arrive at σ_2, where $y = 1$, and thus r_5 is unlocked. To σ_2 we may apply $\tau_3 = r_5(1), r_5(2), r_4(3), r_5(3)$ to arrive at the required configuration σ_3 with $\kappa[\ell_5] = 3$.

In order to exploit acceleration as much as possible, we would like to group together occurrences of the same rule. In τ_1, we can actually swap $r_4(1)$ and $r_3(2)$ as locally the precedence relation of each process is maintained, and both rules are unconditional. Similarly, in τ_3, we can move $r_4(3)$ to the beginning of the sequence τ_3. Concatenating these altered sequences, the resulting complete schedule is $\tau = r_3(1), r_3(2), r_4(1), r_4(2), r_1(3), r_2(3), r_4(3), r_5(1), r_5(2), r_5(3)$. We can group together the consecutive occurrences for the same rules r_i, and write the schedule using pairs consisting of rules and acceleration factors, that is, $(r_3, 2)$, $(r_4, 2)$, $(r_1, 1)$, $(r_2, 1)$, $(r_4, 1)$, $(r_5, 3)$.

In schedule τ, the occurrences of all rules are grouped together except for r_4. That is, in the accelerated schedule we have two occurrences for r_4, while for the other rules one occurrence is sufficient. Actually, there is no way around this: We cannot swap $r_2(3)$ with $r_4(3)$, as we have to maintain the local precedence relation of process 3. More precisely, in the counter system, r_4 would require us to decrease the counter $\kappa[\ell_2]$ at a point in the schedule where $\kappa[\ell_2] = 0$. We first have to increase the counter value by executing a transition according to rule r_2, before we can apply r_4. Moreover, we cannot move the subsequence $r_1(3), r_2(3), r_4(3)$ to the left, as $r_1(3)$ is locked in the prefix.

In this paper we characterize such cases. The issue here is that r_4 can unlock r_1 (we use the notation $r_4 \prec_U r_1$), while r_1 precedes r_4 in the control flow of the processes ($r_1 \prec_P r_4$). We coin the term *milestone* for transitions like $r_1(3)$ that cannot be moved, and show that the same issue arises if a rule r locks a threshold guard of rule r', where r precedes r' in the control flow. As processes do not decrease shared variables, we have at most one milestone per threshold guard. The sequence of transitions between milestones is called a segment. We prove that transitions inside a segment can be swapped, so that one can group transitions for the same rule in so-called batches. Each of these batches can then be replaced by a single accelerated transition that leads to the same configuration as the original batch. Hence, any segment can be replaced by an accelerated

one whose length is at most the number of rules of a process. This and the number of milestones gives us the required bound on the diameter. This bound is independent of the parameters, and only depends on the number of threshold guards and the precedence relation between the rules of the processes.

Our main result is that the bound on the diameter is independent of the parameter values. In contrast, reachability of a specific local state depends on the parameter values: for a process to reach ℓ_5, at least $n - f$ processes must execute r_4 before at least t other processes must execute r_2. That is, the system must contain at least $(n-f)+t$ processes. In case of $t > f$, we obtain $(n-f)+t > n$, which is a contradiction, and ℓ_5 cannot be reached for such parameter values. The model checking problem we are interested in is whether a given state is unreachable for all parameter values that satisfy the resilience condition.

3 Parameterized Counter Systems

3.1 Threshold Automata

A threshold automaton describes a process in a concurrent system. It is defined by its local states, the shared variables, the parameters, and by rules that define the state changes and their conditions and effects on shared variables. Formally, a *threshold automaton* is a tuple $\mathsf{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$ defined below.

States. The set $\mathcal{L}$ is the finite set of *local states*, and $\mathcal{I} \subseteq \mathcal{L}$ is the set of *initial local states*. The set Γ is the finite set of *shared variables* that range over $\mathbb{N}_0$. To simplify the presentation, we view the variables as vectors in $\mathbb{N}_0^{|\Gamma|}$. 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., $n > 3t \ \wedge \ t \geq f$. Then, we denote the set of *admissible parameters* by $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Pi|} \colon \mathbf{p} \models RC\}$.

Rules. A rule defines a conditional transition between local states that may update the shared variables. Here we define the syntax and give only informal explanations of the semantics, which is defined via counter systems in Section 3.2.

Formally, a *rule* is a tuple $(\mathit{from}, \mathit{to}, \varphi^{\leq}, \varphi^{>}, \mathbf{u})$: The local states *from* and *to* are from $\mathcal{L}$. Intuitively, they capture from which local state to which a process moves, or, in terms of counter systems, which counters decrease and increase, respectively. A rule is only executed if the conditions $\varphi^{\leq}$ and $\varphi^{>}$ evaluate to true. Each condition consists of multiple 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

$$a_0 + \sum\nolimits_{i=1}^{|\Pi|} a_i \cdot p_i \ \leq \ x \qquad \text{and} \qquad a_0 + \sum\nolimits_{i=1}^{|\Pi|} a_i \cdot p_i \ > \ x$$

are a *lower guard* and *upper guard*, respectively (both, variables and coefficients, may differ for different guards). The *condition* $\varphi^{\leq}$ is a conjunction of lower

guards, and the condition $\varphi^{>}$ is a conjunction of upper guards. Rules may increase shared variables. We model this using an update vector $\mathbf{u} \in \mathbb{N}_0^{|\Gamma|}$, which is added to the vector of shared variables, when the rule is executed. Then $\mathcal{R}$ 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 $\prec_P$*, the* unlock relation $\prec_U$*, and the* lock relation $\prec_L$ *as subsets of* $\mathcal{R} \times \mathcal{R}$ *as follows:*

1. $r_1 \prec_P r_2$ *iff* $r_1.to = r_2.from$*. We denote by* $\prec_P^+$ *the transitive closure of* $\prec_P$*. If* $r_1 \prec_P r_2 \wedge r_2 \prec_P r_1$*, or if* $r_1 = r_2$*, we write* $r_1 \sim_P r_2$*.*
2. $r_1 \prec_U r_2$ *iff there is a* $\mathbf{g} \in \mathbb{N}_0^{|\Gamma|}$ *and* $\mathbf{p} \in \mathbf{P}_{RC}$ *satisfying* $(\mathbf{g}, \mathbf{p}) \models r_1.\varphi^{\leq} \wedge r_1.\varphi^{>}$ *and* $(\mathbf{g}, \mathbf{p}) \not\models r_2.\varphi^{\leq} \wedge r_2.\varphi^{>}$ *and* $(\mathbf{g} + r_1.\mathbf{u}, \mathbf{p}) \models r_2.\varphi^{\leq} \wedge r_2.\varphi^{>}$*.*
3. $r_1 \prec_L r_2$ *iff there is a* $\mathbf{g} \in \mathbb{N}_0^{|\Gamma|}$ *and* $\mathbf{p} \in \mathbf{P}_{RC}$ *satisfying* $(\mathbf{g}, \mathbf{p}) \models r_1.\varphi^{\leq} \wedge r_1.\varphi^{>}$ *and* $(\mathbf{g}, \mathbf{p}) \models r_2.\varphi^{\leq} \wedge r_2.\varphi^{>}$ *and* $(\mathbf{g} + r_1.\mathbf{u}, \mathbf{p}) \not\models r_2.\varphi^{\leq} \wedge r_2.\varphi^{>}$*.*

Definition 2. *Given a threshold automaton* $(\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$*, we define the following quantities:* $\mathcal{C}^{\leq} = |\{r.\varphi^{\leq} \colon r \in \mathcal{R}, \exists r' \in \mathcal{R}.\ r' \not\prec_P^+ r \wedge r' \prec_U r\}|$*,* $\mathcal{C}^{>} = |\{r.\varphi^{>} \colon r \in \mathcal{R}, \exists r'' \in \mathcal{R}.\ r \not\prec_P^+ r'' \wedge r'' \prec_L r\}|$*. Finally,* $\mathcal{C} = \mathcal{C}^{\leq} + \mathcal{C}^{>}$*.*

We consider specific threshold automata, namely those that naturally capture FTDAs, where rules that form cycles do not increase shared variables.

Definition 3 (Canonical Threshold Automaton). *A threshold automaton* $(\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$ *is canonical, if* $r.\mathbf{u} = \mathbf{0}$ *for all rules* $r \in \mathcal{R}$ *that satisfy* $r \prec_P^+ r$*.*

Order on rules. The relation $\sim_P$ defines equivalence classes of rules. 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 $r_1 \not\sim_P r_2$. Observe that the relation $\prec_C$ is a strict partial order (irreflexive and transitive). Hence, there are linear extensions of $\prec_C$. Below, we fix an *arbitrary* of these linear extensions to sort transitions in a schedule:

Notation. We denote by $\prec_C^{lin}$ a linear extension of $\prec_C$.

3.2 Counter Systems

Given a threshold automaton $\mathsf{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \Pi, \mathcal{R}, RC)$, a function $N \colon \mathbf{P}_{RC} \to \mathbb{N}_0$ that formalizes the number of processes to be modeled (e.g., n), 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. A configuration $\sigma = (\boldsymbol{\kappa}, \mathbf{g}, \mathbf{p})$ consists of a vector of *counter values* $\sigma.\boldsymbol{\kappa} \in \mathbb{N}_0^{|\mathcal{L}|}$,[1] 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 $\sum_{i \notin \mathcal{I}} \sigma.\boldsymbol{\kappa}[i] = 0$.

[1] For simplicity we use the convention that $\mathcal{L} = \{1, \ldots, |\mathcal{L}|\}$.

Transition relation. A *transition* is a pair $t = (\mathit{rule}, \mathit{factor})$ of a rule of the threshold automaton 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 $\forall k \in \{0, \ldots, t.\mathit{factor} - 1\}$. $(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + k \cdot t.\mathbf{u}, \sigma.\mathbf{p}) \models t.\varphi^{\leq} \wedge t.\varphi^{>}$. For transitions t_1 and t_2 we say that the two transitions are related iff $t_1.\mathit{rule}$ and $t_2.\mathit{rule}$ are related, e.g., for $\prec_P$ we write $t_1 \prec_P t_2$ iff $t_1.\mathit{rule} \prec_P t_2.\mathit{rule}$.

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

- t is enabled in σ
- $\sigma'.\mathbf{g} = \sigma.\mathbf{g} + t.\mathit{factor} \cdot t.\mathbf{u}$
- $\sigma'.\mathbf{p} = \sigma.\mathbf{p}$
- 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. *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. A *schedule* is a sequence of transitions. A schedule $\tau = t_1, \ldots, t_m$ is called *applicable* to configuration σ_0, if there is a sequence of configurations $\sigma_1, \ldots, \sigma_m$ such that $\sigma_i = t_i(\sigma_{i-1})$ for all i, $0 < i \leq m$. A schedule $t_1, \ldots, t_m$ where $t_i.\mathit{factor} = 1$ for $0 < i \leq m$ is a *conventional schedule*. If there is a $t_i.\mathit{factor} > 1$, then a schedule is called *accelerated*.

We write $\tau \cdot \tau'$ to denote the concatenation of two schedules τ and τ', and treat a transition t as schedule. If $\tau = \tau_1 \cdot t \cdot \tau_2 \cdot t' \cdot \tau_3$, for some τ_1, τ_2, and τ_3, we say that transition t precedes transition t' in τ, and denote this by $t \rightarrow_\tau t'$.

4 Diameter of Counter Systems

In this section, we will present the outline of the proof of our main theorem:

Theorem 1. *Given a canonical threshold automaton TA and a size function N, for each $\mathbf{p}$ in $\mathbf{P}_{RC}$ the diameter of the counter system is less than or equal to $d(\mathsf{TA}) = (\mathcal{C} + 1) \cdot |\mathcal{R}| + \mathcal{C}$, and thus independent of $\mathbf{p}$.*

From the theorem it follows that for all parameter values, reachability in the counter system can be verified by exploring runs of length at most $d(\mathsf{TA})$. However, the theorem alone is not sufficient to solve the parameterized model checking problem. For this, we combine the bound with the abstraction method

in [16]. More precisely, the counter abstraction in [16] simulates the counter systems for *all* parameter values that satisfy the resilience condition. Consequently, the bound on the length of the run of the counter systems entails a bound for the counter abstraction. We exploit this in the experiments in Section 5.

4.1 Proof Idea

Given a rule r, a schedule τ and two transitions t_i and t_j, with $t_i \to_\tau t_j$, the subschedule $t_i \cdot \ldots \cdot t_j$ of τ is a *batch of rule* r if $t_\ell.rule = r$ for $i \leq \ell \leq j$, and if the subschedule is maximal, that is, $i = 1 \vee t_{i-1} \neq r$ and $j = m \vee t_{j+1} \neq r$. Similarly, we define a batch of a class c as a subschedule $t_i \cdot \ldots \cdot t_j$ where $[r_\ell] = c$ for $i \leq \ell \leq j$, and where the subschedule is maximal as before.

Definition 4 (Sorted schedule). *Given a schedule τ, and the relation $\prec_C^{lin}$, we define $sort(\tau)$ as the schedule that satisfies:*

1. *$sort(\tau)$ is a permutation of schedule τ.*
2. *two transitions from the same equivalence class maintain their relative order, that is, if $t \to_\tau t'$ and $t \sim_P t'$, then $t \to_{sort(\tau)} t'$.*
3. *for each equivalence class defined by $\sim_P$ there is at most one batch in $sort(\tau)$.*
4. *if $t \to_{sort(\tau)} t'$, then $t \sim_P t'$ or $[t] \prec_C^{lin} [t']$.*

The crucial observation is that if we have a schedule $\tau_1 = t \cdot t'$ applicable to configuration σ with $t.rule = t'.rule$, we can replace it with another applicable (one-transition) schedule $\tau_2 = t''$, with $t''.rule = t.rule$ and $t''.factor = t.factor + t'.factor$, such that $\tau_1(\sigma) = \tau_2(\sigma)$. Thus, we can reach the same configuration with a shorter schedule. More generally, we may replace a batch of a rule by a single accelerated transition whose factor is the sum of all factors in the batch.

In this section we give a bound on the diameter, i.e., the length of the shortest path between any two configurations σ and σ' for which there is a schedule τ applicable to σ satisfying $\sigma' = \tau(\sigma)$. A simple case is if $sort(\tau)$ is applicable to σ and each equivalence class defined by the precedence relation consists of a single rule (e.g., the control flow is a directed acyclic graph). Then by Definition 4 we have at most $|\mathcal{R}|$ batches in $sort(\tau)$, that is, one per rule. By the reasoning of above we can replace each batch by a single accelerated transition.

In general $sort(\tau)$ may not be applicable to σ, or there are equivalence classes containing multiple rules, i.e., rules form cycles in the precedence relation. The first issue comes from locking and unlocking. We identify milestone transitions, and show that two neighboring non-milestone transitions can be swapped according to *sort* in Section 4.3. We also deal with the issue of cycles in the precedence relation. It is ensured by *sort* that within a segment, all transitions that belong to a cycle form a batch. In Section 4.2, we replace such a batch by a batch where the remaining rules do not form a cycle. Removing cycles requires the assumption that shared variables are not incremented in cycles.

4.2 Removing Cycles

We consider the distance between two configurations σ and σ' that satisfy $\sigma.\mathbf{g} = \sigma'.\mathbf{g}$, i.e., along any schedule connecting these configurations, the values of shared

variables are unchanged, and thus the evaluations of guards are also unchanged. By Definition 3, we can apply this section's result to batches of a class.

Definition 5. *Given a schedule $\tau = t_1, t_2, \ldots$, we denote by $|\tau|$ the length of the schedule. Further, we define the following vectors*

$$\mathbf{in}(\tau)[\ell] = \sum_{\substack{1 \le i \le |\tau| \\ t_i.to = \ell}} t_i.factor, \quad \mathbf{out}(\tau)[\ell] = \sum_{\substack{1 \le i \le |\tau| \\ t_i.from = \ell}} t_i.factor, \quad \mathbf{up}(\tau) = \sum_{1 \le i \le |\tau|} t_i.\mathbf{u}.$$

From the definition of a counter system, we directly obtain:

Proposition 2. *For all configurations σ, and all schedules τ applicable to σ, if $\sigma' = \tau(\sigma)$, then $\sigma'.\boldsymbol{\kappa} = \sigma.\boldsymbol{\kappa} + \mathbf{in}(\tau) - \mathbf{out}(\tau)$, and $\sigma'.\mathbf{g} = \sigma.\mathbf{g} + \mathbf{up}(\tau)$.*

Proposition 3. *For all configurations σ, and all schedules τ and τ' applicable to σ, if $\mathbf{in}(\tau) = \mathbf{in}(\tau')$, $\mathbf{out}(\tau) = \mathbf{out}(\tau')$, and $\mathbf{up}(\tau) = \mathbf{up}(\tau')$, then $\tau(\sigma) = \tau'(\sigma)$.*

Given a schedule $\tau = t_1, t_2, \ldots$ we say that the index set $I = \{i_1, \ldots, i_j\}$ forms a cycle in τ, if for all b, $1 \le b < j$, it holds that $t_{i_b}.to = t_{i_{b+1}}.from$, and $t_{i_j}.to = t_{i_1}.from$. Let $\mathcal{R}(\tau) = \{r \colon t_i \in \tau \wedge t_i.rule = r\}$.

Proposition 4. *For all schedules τ, if τ contains a cycle, then there is a schedule τ' satisfying $|\tau'| < |\tau|$, $\mathbf{in}(\tau) = \mathbf{in}(\tau')$, $\mathbf{out}(\tau) = \mathbf{out}(\tau')$, and $\mathcal{R}(\tau') \subseteq \mathcal{R}(\tau)$.*

Repeated application of the proposition leads to a cycle-free schedule (possibly the empty schedule), and we obtain:

Theorem 2. *For all schedules τ, there is a schedule τ' that contains no cycles, $\mathbf{in}(\tau) = \mathbf{in}(\tau')$, $\mathbf{out}(\tau) = \mathbf{out}(\tau')$, and $\mathcal{R}(\tau') \subseteq \mathcal{R}(\tau)$.*

The issue with this theorem is that τ' is not necessarily applicable to the same configurations as τ. In the following theorem, we prove that if a schedule satisfies a specific condition on the order of transitions, then it is applicable.

Theorem 3. *Let σ and σ' be two configurations with $\sigma.\mathbf{g} = \sigma'.\mathbf{g}$, and let τ be a schedule with $\mathbf{up}(\tau) = \mathbf{0}$, all transitions unlocked in σ, and where if $t_i \rightarrow_\tau t_j$, then $t_j \not\prec_P t_i$. If $\sigma'.\boldsymbol{\kappa} - \sigma.\boldsymbol{\kappa} = \mathbf{in}(\tau) - \mathbf{out}(\tau)$, then τ is applicable to σ.*

Corollary 1. *For all configurations σ, and all schedules τ applicable to σ, with $\mathbf{up}(\tau) = \mathbf{0}$, there is a schedule with at most one batch per rule applicable to σ satisfying that τ' contains no cycles, $\tau'(\sigma) = \tau(\sigma)$, and $\mathcal{R}(\tau') \subseteq \mathcal{R}(\tau)$.*

4.3 Identifying Milestones and Swapping Transitions

In this section we deal with locking and unlocking. To this end, we start by defining milestones. Then the central Theorem 4 establishes that two consequent non-milestone transitions can be swapped, if needed to sort the segment according to $\prec_C^{lin}$: the resulting schedule is still applicable, and leads to the same configuration as the original one.

Definition 6 (Left Milestone). *Given a configuration σ and a schedule $\tau = \tau' \cdot t \cdot \tau''$ applicable to σ, the transition t is a left milestone for σ and τ, if*

1. *there is a transition t' in τ' satisfying $t' \not\prec_P^+ t \ \wedge \ t' \prec_U t$,*
2. *$t.\varphi^{\leq}$ is locked in σ, and*
3. *for all t' in τ', $t'.\varphi^{\leq} \neq t.\varphi^{\leq}$.*

Definition 7 (Right Milestone). *Given a configuration σ and a schedule $\tau = \tau' \cdot t \cdot \tau''$ applicable to σ, the transition t is a right milestone for σ and τ, if*

1. *there is a transition t'' in τ'' satisfying $t \not\prec_P^+ t'' \ \wedge \ t'' \prec_L t$,*
2. *$t.\varphi^{>}$ is locked in $\tau(\sigma)$, and*
3. *for all t'' in τ'', $t''.\varphi^{>} \neq t.\varphi^{>}$.*

Definition 8 (Segment). *Given a schedule τ and configuration σ, τ' is a segment if it is a subschedule of τ, and does not contain a milestone for σ and τ.*

Having defined milestones and segments, we arrive at our central result.

Theorem 4. *Let σ be a configuration, τ a schedule applicable to σ, and $\tau = \tau_1 \cdot t_1 \cdot t_2 \cdot \tau_2$. If transitions t_1 and t_2 are not milestones for σ and τ, and satisfy $[t_2] \prec_C^{lin} [t_1]$, then*

i. schedule $\tau' = \tau_1 \cdot t_2 \cdot t_1 \cdot \tau_2$ is applicable to σ,
ii. $\tau'(\sigma) = \tau(\sigma)$, and

Repeated application of the theorem leads to a schedule where milestones and sorted schedules alternate. By the definition of a milestone, there is at most one milestone per condition. Thus, the number of milestones is bounded by $\mathcal{C}$ (Definition 2). Together with Corollary 1, this is used to establish Theorem 1.

5 Experimental Evaluation

We have implemented the techniques in our tool ByMC [1]. Technical details about our approach to abstraction and refinement can be found in [13]. The input are the descriptions of our benchmarks in parametric Promela [17], which describe parameterized processes. Hence, as preliminary step ByMC computes the PIA data abstraction [16] to obtain finite state processes. Based on this, ByMC does preprocessing to compute threshold automata, the locking and unlocking relations, and to generate the inputs for our model checking back-ends.

Preprocessing. First, we compute the set of rules $\mathcal{R}$: Recall that a rule is a tuple $(\mathit{from}, \mathit{to}, \varphi^{\leq}, \varphi^{>}, \mathbf{u})$. ByMC calls NuSMV to explore a single process system with unrestricted shared variables, in order to compute the $(\mathit{from}, \mathit{to})$ pairs. From this, ByMC computes the reachable local states. In the case of our benchmark CBC, e.g., that cuts the local states we have to consider from 2000 to 100, approximately. All our experiments — including the ones with FASTer [3] — are based on the reduced local state space. Then, for each pair $(\mathit{from}, \mathit{to})$, ByMC explores symbolic path to compute the guards and update vectors for the pair. This gives us the set of rules $\mathcal{R}$. Then, ByMC encodes Definition 1 in Yices,

to construct the lock $\prec_L$ and unlock $\prec_U$ relations. Then, ByMC computes the relations $\{(r, r') \colon r' \not\prec_P^+ r \ \wedge \ r' \prec_U r\}$ and $\{(r, r'') \colon r \not\prec_P^+ r'' \ \wedge \ r'' \prec_L r\}$ as required by Definition 2. This provides the bounds.

Back-ends. ByMC generates the PIA counter abstraction [16] to be used by the following back-end model checkers. We have also implemented an automatic abstraction refinement loop for the counterexamples provided by NuSMV.
BMC. NuSMV 2.5.4 [10] (using MiniSAT) performs incremental bounded model checking with the bound $\hat{d}$. If a counterexample is reported, ByMC refines the system as explained in [16], if the counterexample is spurious.
BMCL. We combine NuSMV with the multi-core SAT solver Plingeling [5]: NuSMV does bounded model checking for 30 steps. Spurious counterexample are refined by ByMC. If there is no counterexample, NuSMV produces a single CNF formula with the bound $\hat{d}$, whose satisfiability is then checked with Plingeling.
BDD. NuSMV 2.5.4 performs BDD-based symbolic checking.
FAST. FASTer 2.1 [3] performs reachability analysis using plugin Mona-1.3.

5.1 Benchmarks

We encoded several asynchronous FTDAs in our parametric Promela, following the technique in [17]; they can be obtained from [1]. All models contain transitions with lower threshold guards. The benchmarks CBC also contain upper threshold guards. If we ignore self-loops, the precedence relation of all but NBAC and NBACC, which have non-trivial cycles, are partial orders.
Folklore reliable broadcast (FRB) [9]. In this algorithm, n processes have to agree on whether a process has broadcast a message, in the presence of $f \leq n$ crashes. Our model of FRB has one shared variable and the abstract domain of two intervals $[0, 1)$ and $[1, \infty)$. In this paper, we are concerned with the safety property *unforgeability*: If no process is initialized with value 1 (message from the broadcaster), then no correct process ever accepts.
Consistent broadcast (STRB) [31]. Here, we have $n - f$ correct processes and $f \geq 0$ Byzantine faulty ones. The resilience condition is $n > 3t \wedge t \geq f$. There is one shared variable and the abstract domain of four intervals $[0, 1)$, $[1, t + 1)$, $[t + 1, n - t)$, and $[n - t, \infty)$. Here, we check only unforgeability (see FRB), whereas in [16] we checked also liveness properties.
Byzantine agreement (ABA) [8]. There are $n > 3t$ processes, $f \leq t$ of them Byzantine faulty. The model has two shared variables. We have to consider two different cases for the abstract domain, namely, case ABA0 with the domain $[0, 1)$, $[1, t + 1)$, $[t + 1, \lceil \frac{n+t}{2} \rceil)$, and $[\lceil \frac{n+t}{2} \rceil, \infty)$ and case ABA1 with the domain $[0, 1)$, $[1, t + 1)$, $[t + 1, 2t + 1)$, $[2t + 1, \lceil \frac{n+t}{2} \rceil)$, and $[\lceil \frac{n+t}{2} \rceil, \infty)$. As for FRB, we check unforgeability. This case study, and all below, run out of memory when using Spin for model checking the counter abstraction [16].
Condition-based consensus (CBC) [27]. This is a restricted form of consensus solvable in asynchronous systems. We consider binary condition-based consensus in the presence of clean crashes, which requires four shared variables.

Table 1. Summary of experiments on AMD Opteron®Processor 6272 with 192 GB RAM and 32 CPU cores. Plingeling used up to 16 cores. "TO" denotes timeout of 24 hours; "OOM" denotes memory overrun of 64 GB; "ERR" denotes runtime error; "RTO" denotes that the refinement loop timed out.

Input	Threshold A.				Bounds			Time, [HH:]MM:SS				Memory, GB			
FTDA	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{R}\rvert$	$\mathcal{C}^{\leq}$	$\mathcal{C}^{>}$	d	$d^\star$	$\hat{d}$	BMCL	BMC	BDD	FAST	BMCL	BMC	BDD	FAST
Fig. 1	5	5	1	0	11	9	27	00:00:03	00:00:04	00:01	00:00:08	0.01	0.02	0.02	0.06
FRB	6	8	1	0	17	10	10	00:00:13	00:00:13	00:06	00:00:08	0.01	0.02	0.02	0.01
STRB	7	15	3	0	63	30	90	00:00:09	00:00:06	00:04	00:00:07	0.02	0.03	0.02	0.07
ABA0	37	180	6	0	1266	586	1758	00:21:26	02:20:10	00:15	00:08:40	6.37	1.49	0.07	3.56
ABA1	61	392	8	0	3536	1655	6620	TO 25%	TO 12%	00:33	02:36:25	TO	TO	0.08	15.65
CBC0	43	204	0	0	204	204	612	01:38:54	TO 57%	OOM	ERR	1.28	TO	OOM	ERR
CBC1	115	896	1	1	2690	2180	8720	TO 05%	TO 11%	TO	TO	TO	TO	TO	TO
NBACC	109	1724	6	0	12074	5500	16500	RTO	RTO	TO	TO	RTO	RTO	TO	TO
NBAC	77	1356	6	0	9498	4340	13020	RTO	RTO	TO	TO	RTO	RTO	TO	TO
When a Bug is Introduced															
ABA0	32	139	6	0	979	469	1407	00:00:16	00:00:18	TO	00:05:57	0.04	0.04	TO	2.70
ABA1	54	299	8	0	2699	1305	5220	00:00:22	00:00:21	TO	ERR	0.06	0.06	TO	ERR

Under the resilience condition $n > 2t \wedge f \geq 0$, we have to consider two different cases depending on f: If $f = 0$ we have case CBC0 with the domain $[0, 1)$, $[1, \lceil \frac{n}{2} \rceil)$, $[\lceil \frac{n}{2} \rceil, n - t)$, and $[n - t, \infty)$. If $f \neq 0$, case CBC1 has the domain: $[0, 1)$, $[1, f)$, $[f, \lceil \frac{n}{2} \rceil)$, $[\lceil \frac{n}{2} \rceil, n - t)$, and $[n - t, \infty)$. We verified several properties, all of which resulted in experiments with similar characteristics. We only give $validity_0$ in the table, i.e., no process accepts value 0, if all processes initially have value 1.

Non-blocking atomic commitment (NBAC and NBACC) [30,15]. Here, n processes are initialized with Yes or No and decide on whether to commit a transaction. The transaction must be aborted if at least one process is initialized to No. We consider the cases NBACC and NBAC of clean crashes and crashes, respectively. Both models contain four shared variables, and the abstract domain is $[0, 1)$ and $[1, n)$ and $[n-1, n)$, and $[n, \infty)$. The algorithm uses a failure detector, which is modeled as local variable that changes its value non-deterministically.

5.2 Evaluation

Table 1 summarizes the experiments. For the threshold automata, we give the number of local states $|\mathcal{L}|$, rules $|\mathcal{R}|$, and conditions according to Definition 2, i.e., $\mathcal{C}^{\leq}$ and $\mathcal{C}^{>}$. The column $\boldsymbol{d}$ provides the bound on the diameter as in Theorem 1, whereas the column $\boldsymbol{d^\star}$ provides an improved diameter: In the proof of Theorem 1, we bound the length of all segments by $|\mathcal{R}|$. However, by Definition 6, segments to the left of a left milestone cannot contain transitions for rules with the same condition as the milestone. The same is true for segments to the right of right milestones. ByMC explores all orders of milestones, an uses this observation about milestones to compute a more precise bound $\boldsymbol{d^\star}$ for the diameter. Our encoding of the counter abstraction only increments and decrements counters. If $|\hat{D}|$ is the size of the abstract domain, a transition in a counter system is

simulated by at most $|\hat{D}| - 1$ steps in the counter abstraction; this leads to the diameter $\hat{\boldsymbol{d}}$ for counter abstractions, which we use in our experiments.

As the experiments show, all techniques rapidly verify FRB, STRB, and Fig. 1. FRB and STRB had already been verified before using Spin [16]. The more challenging examples are ABA0 and ABA1, where BDD clearly outperforms the other techniques. Bounded model checking is slower here, because the diameter bound does not exploit knowledge on the specification. FAST performs well on these benchmarks. We believe this is because many rules are always disabled, due to the initial states as given in the specification. To confirm this intuition, we introduced a bug into ABA0 and ABA1, which allows the processes to non-deterministically change their value to 1. This led to a dramatic slowdown of BDD and FAST, as reflected in the last two lines.

Using the bounds of this paper, BMCL verified CBC0, whereas all other techniques failed. BMCL did not reach the bounds for CBC1 with our experimental setup, but we believe that the bound is within the reach with a better hardware or an improved implementation. In this case, we report the percentages of the bounds we reached with bounded model checking.

In the experiments with NBAC and NBACC, the refinement loop timed out. We are convinced that we can address this issue by integrating the refinement loop with an incremental bounded model checker.

6 Related Work and Discussions

Specific forms of counter systems can be used to model parameterized systems of concurrent processes. Lubachevsky [25] discusses *compact* programs that reach each state in a bounded number of steps, where the bound is independent of the number of processes. In [25] he gives examples of compact programs, and in [24] he proves that specific semaphore programs are compact. We not only show compactness, but give a bound on the diameter. In our case, communication is not restricted to semaphores, but we have threshold guards. Counter abstraction [29] follows this line of research, but as discussed in [4], does not scale well for large numbers of local states.

Acceleration in infinite state systems (e.g., in flat counter automata [22]) is a technique that computes the transitive closure of a transition relation and applies it to the set of states. The tool FAST [2] uses acceleration to compute the set of reachable states in a symbolic procedure. This appears closely related to our acceleration factors. However, in [2] a transition is chosen and accelerated dynamically in the course of symbolic state space exploration, while we statically use acceleration factors and reordering of transitions.

One achieves completeness for reachability in bounded model checking by exploring all runs that are not longer than the diameter of the system [6]. The notion of *completeness threshold* [11] generalizes this idea to safety and liveness properties. As in general, computing the diameter is believed to be as hard as the model checking problem, one can use a coarser bound provided by the

reoccurrence diameter [19]. In practice, the reoccurrence diameter of counter abstraction is prohibitively large, so that we give bounds on the diameter.

Partial orders are a useful concept for reasoning about distributed systems [20]. In model checking, *partial order reduction* [14,32,28] is used to reduce the search space. It is based on the idea that changing the order of steps of concurrent processes leads to "equivalent" behavior with respect to the specification. Typically, partial order reduction is used on-the-fly to prune runs that are equivalent to representative ones. In contrast, we bound the length of representative runs offline in order to ensure completeness of bounded model checking. A partial order reduction for threshold-guarded FTDAs was introduced in [7]. It can be used for model checking small instances, while we focus on parameterized model checking.

Our technique of determining which transitions can be swapped in a run reminds of *movers* as discussed by Lipton [23], or more generally the idea to show that certain actions can be grouped into larger atomic blocks to simplify proofs [12,21]. Movers address the issue of grouping many local transitions of a process together. In contrast, we conceptually group transitions of different processes together into one accelerated transition. Moreover, the definition of a mover by Lipton is independent of a specific run: a left mover (e.g., a "release" operation) is a transition that in *all runs* can "move to the left" with respect to transitions of other processes. In our work, we look at individual runs and identify which transitions (milestones) must not move in this run.

As next steps we will focus on liveness of fault-tolerant distributed algorithms. In fact the liveness specifications are in the fragment of linear temporal logic for which it is proven [18] that a formula can be translated into a cliquey Büchi automaton. For such automata, [18] provides a completeness threshold. Still, there are open questions related to applying our results to the idea of [18].

References

1. ByMC: Byzantine model checker (2013), `http://forsyte.tuwien.ac.at/software/bymc/`, accessed: June, 2014
2. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: Fast: acceleration from theory to practice. STTT 10(5), 401–424 (2008)
3. Bardin, S., Leroux, J., Point, G.: Fast extended release. In: Computer Aided Verification. pp. 63–66. Springer (2006)
4. Basler, G., Mazzucchi, M., Wahl, T., Kroening, D.: Symbolic counter abstraction for concurrent software. In: CAV. LNCS, vol. 5643, pp. 64–78 (2009)
5. Biere, A.: Lingeling, Plingeling and Treengeling entering the SAT competition 2013. Proceedings of SAT Competition 2013; Solver and p. 51 (2013)
6. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: TACAS. LNCS, vol. 1579, pp. 193–207 (1999)
7. Bokor, P., Kinder, J., Serafini, M., Suri, N.: Efficient model checking of fault-tolerant distributed protocols. In: DSN. pp. 73–84 (2011)
8. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. J. ACM 32(4), 824–840 (1985)
9. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. JACM 43(2), 225–267 (March 1996)

10. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: CAV. LNCS, vol. 2404, pp. 359–364 (2002)
11. 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)
12. Doeppner, T.W.: Parallel program correctness through refinement. In: POPL. pp. 155–169 (1977)
13. Gmeiner, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Tutorial on parameterized model checking of fault-tolerant distributed algorithms. In: Formal Methods for Executable Software Models. pp. 122–171. LNCS, Springer (2014)
14. Godefroid, P.: Using partial orders to improve automatic verification methods. In: CAV. LNCS, vol. 531, pp. 176–185 (1990)
15. Guerraoui, R.: Non-blocking atomic commit in asynchronous distributed systems with failure detectors. Distributed Computing 15(1), 17–25 (2002)
16. 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)
17. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Towards modeling and model checking fault-tolerant distributed algorithms. In: SPIN. LNCS, vol. 7976, pp. 209–226 (2013)
18. Kroening, D., Ouaknine, J., Strichman, O., Wahl, T., Worrell, J.: Linear completeness thresholds for bounded model checking. In: CAV. LNCS, vol. 6806, pp. 557–572 (2011)
19. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: VMCAI. LNCS, vol. 2575, pp. 298–309 (2003)
20. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
21. Lamport, L., Schneider, F.B.: Pretending atomicity. Tech. Rep. 44, SRC (1989)
22. Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: ATVA. LNCS, vol. 3707, pp. 489–503 (2005)
23. Lipton, R.J.: Reduction: A method of proving properties of parallel programs. Commun. ACM 18(12), 717–721 (1975)
24. Lubachevsky, B.D.: An approach to automating the verification of compact parallel coordination programs. II. Tech. Rep. 64, New York University. Computer Science Department (1983)
25. Lubachevsky, B.D.: An approach to automating the verification of compact parallel coordination programs. I. Acta Informatica 21(2), 125–169 (1984)
26. Lynch, N.: Distributed Algorithms. Morgan Kaufman (1996)
27. Mostéfaoui, A., Mourgaya, E., Parvédy, P.R., Raynal, M.: Evaluating the condition-based approach to solve consensus. In: DSN. pp. 541–550 (2003)
28. Peled, D.: All from one, one for all: on model checking using representatives. In: CAV. LNCS, vol. 697, pp. 409–423 (1993)
29. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,∞)- counter abstraction. In: CAV, LNCS, vol. 2404, pp. 93–111. Springer (2002)
30. Raynal, M.: A case study of agreement problems in distributed systems: Non-blocking atomic commitment. In: HASE. pp. 209–214 (1997)
31. Srikanth, T., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Dist. Comp. 2, 80–94 (1987)
32. Valmari, A.: Stubborn sets for reduced state space generation. In: Advances in Petri Nets 1990, LNCS, vol. 483, pp. 491–515. Springer (1991)