Form Methods Syst Des (2017) 51:270–307
DOI 10.1007/s10703-017-0297-4

Para^2: parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms

Igor Konnov[1] · **Marijana Lazić**[1] · **Helmut Veith**[1] · **Josef Widder**[1]

Published online: 20 September 2017

Abstract Automatic verification of threshold-based fault-tolerant distributed algorithms (FTDA) is challenging: FTDAs 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 for reachability properties: it always finds a counterexample, if there is an actual error. To verify state-of-the-art FTDAs, further improvement is needed. In contrast to encoding bounded executions of a counter system over an abstract finite domain in SAT, in this paper, we encode bounded executions over integer counters in SMT. In addition, we introduce a new form of reduction that exploits acceleration

Supported by: the Austrian Science Fund (FWF) through the National Research Network RiSE (S11403 and S11405), project PRAVDA (P27722), and Doctoral College LogiCS (W1255-N23); and by the Vienna Science and Technology Fund (WWTF) through project APALACHE (ICT15-103). This is an extended version of the paper "SMT and POR beat Counter Abstraction: Parameterized Model Checking of Threshold-Based Distributed Algorithms" that appeared in CAV (Part I), volume 9206 of LNCS, pages 85–102, 2015.

✉ Josef Widder
widder@forsyte.at
http://forsyte.at/widder

Igor Konnov
konnov@forsyte.at
http://forsyte.at/konnov

Marijana Lazić
lazic@forsyte.at
http://forsyte.at/lazic

Helmut Veith
veith@forsyte.at
http://forsyte.at/veith

[1] Institute of Information Systems E184/4, TU Wien (Vienna University of Technology), Favoritenstraße 9-11, 1040 Vienna, Austria

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.

Keywords Parameterized verification · Bounded model checking · Completeness · Partial orders in distributed systems · Reduction · Fault-tolerant distributed algorithms · Byzantine faults

1 Introduction

Replication is a classic approach to make computing systems more reliable. In order to avoid a single point of failure, one uses multiple processes in a distributed system. Then, if some of these processes fail (e.g., by crashing or deviating from their expected behavior) the distributed system as a whole should stay operational. For this purpose one uses fault-tolerant distributed algorithms (FTDAs). These algorithms have been extensively studied in distributed computing theory [1,50], and found application in safety critical systems (automotive or aeronautic industry). With the recent advent of data centers and cloud computing we observe growing interest in fault-tolerant distributed algorithms, and their correctness, also for more mainstream computer science applications [19,20,31,47,52,54,60].

We consider automatic verification techniques specifically for threshold-based fault-tolerant distributed algorithms. In these algorithms, processes collect messages from their peers, and check whether the number of received messages reaches a threshold, e.g., a threshold may ensure that acknowledgments from a majority of processes have been received. Waiting for majorities, or more generally waiting for quorums, is a key pattern of many fault-tolerant algorithms, e.g., consensus, replicated state machine, and atomic commit. In [34] we introduced an efficient encoding of these algorithms, which we used in [33] for abstraction-based parameterized model checking of safety and liveness of several case study algorithms, which are parameterized in the number of processes n and the fraction of faults t, e.g., $n > 3t$. In [41] we were able to verify reachability properties of more involved algorithms by applying bounded model checking. 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 the two affected counters by δ. The number δ is called *acceleration factor*, which can vary in a single run.

As we focus on threshold-based FTDAs, we consider counter systems 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.

Completeness of the method [41] 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 [43], our method uses a reduction similar to Lipton's [48]. Instead of pruning executions that are "similar" to ones explored before as in *partial order reduction* [28,53,59], 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 [41], we introduced the following automated method, which combines this idea with data abstraction [33]:

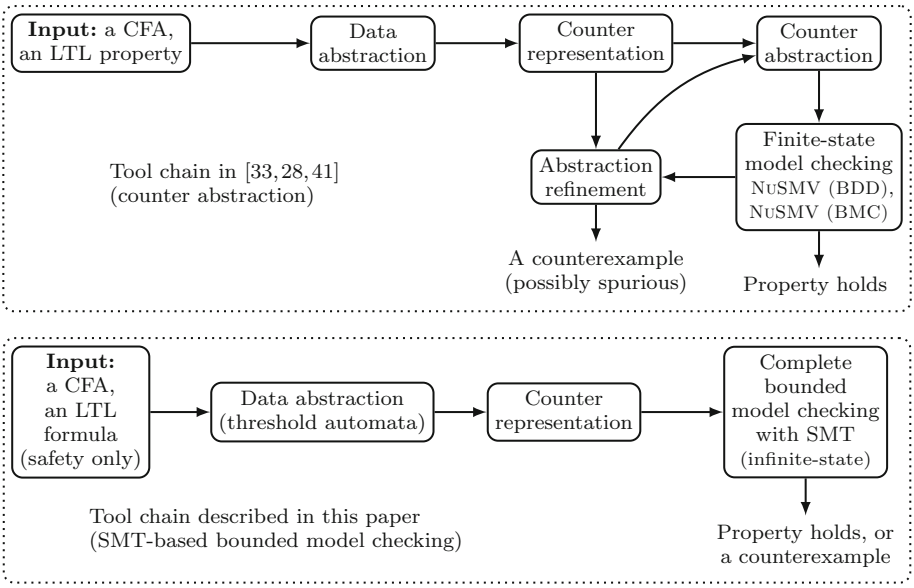

Fig. 1 Tool chain with counter abstraction [27,33,41] on top, and with SMT-based bounded model checking on bottom

1. Apply a parametric data abstraction to the process code to get a finite state process description, and construct the threshold automaton (TA) [33,36].
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 [33,55].
4. Perform SAT-based bounded model checking [6,16] 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 [13,33].

Figure 1 gives on top a diagram [40] that shows the technique based on counter abstraction. 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, counter abstraction can lead to spurious counterexamples. As counters range over a finite abstract domain, the semantics of abstract increment and decrements on the counters introduce nondeterminism. For instance, the value of a counter can remain unchanged after applying an increment. Intuitively, processes or messages can be "added" or "lost", which results in that, e.g., in the abstract model the number of messages sent is smaller than the number of processes that have sent a message, which obviously is spurious behavior. Second, counter abstraction works well in practice only for processes with a few dozens of local states. It has been observed in [4] that counter abstraction does not scale to hundreds of local states. We had similar experience with counter abstraction in our experiments in [41]. We conjecture that this is partly due to the many different interleavings, which result in a large search space.

To address these bottlenecks, we make two crucial *contributions* in this paper:

1. To eliminate one of the two sources of spurious counterexamples, namely, the nondeterminism 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.

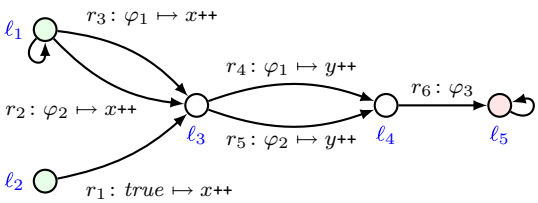

Fig. 2 An example threshold automaton with threshold guards "$\varphi_1 : x \geq \lceil (n+t)/2 \rceil - f$", "$\varphi_2 : y \geq (t+1) - f$", and "$\varphi_3 : y \geq (2t+1) - f$"

2. 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, which models that no process executes the rule), one obtains a run of the counter system. Hence, due to parameterization, 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.

The resulting method is depicted at the bottom of Fig. 1. Our construction can be seen as an aggressive form of 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 *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 $x < t + 1$ false (the parameters n and t are fixed in a run, and shared variables can only be increased). We fix those transitions that change the context, and apply the ideas of 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. We verified safety of FTDAs [10,18,29,51,56,57] that have not been automatically verified before. In addition we achieved dramatic speedup and reduced memory footprint for FTDAs [9,12,58] which previously were verified in [41].

In this article we focus on parameterized reachability properties. Recently, we extended this approach to safety and liveness, for which we used the reachability technique of this article as a black box [37].

2 Our approach at a glance

For modeling threshold-based FTDAs, we use threshold automata that were introduced in [38, 41] and are discussed in more detail in [40]. We use Fig. 2 to describe our contributions in this section. The figure presents a threshold automaton TA over two shared variables x and y and parameters n, t, and f, which is inspired by the distributed asynchronous broadcast protocol from [9]. There, $n - f$ correct processes concurrently follow the control flow of TA, and f processes are Byzantine faulty. As is typical for FTDAs, the parameters must satisfy a resilience condition, e.g., $n > 3t \wedge t \geq f \geq 0$, that is, less than a third of the processes are faulty. The circles depict the local states $\ell_1, \ldots, \ell_5$, two of them are the initial states ℓ_1 and 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 "$\varphi_1 : x \geq \lceil (n+t)/2 \rceil - f$", "$\varphi_2 : y \geq (t+1) - f$", and "$\varphi_3 : y \geq (2t+1) - f$", and an action act increases the shared variables (x and y) by one, or zero (as in rule r_6).

We associate with every local state ℓ_i 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 example, 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 (since the evaluation of the guard $y \geq t + 1 - f$ cannot change from true to false). This results in increasing x and $\boldsymbol{\kappa}[\ell_3]$ by five, and decreasing the counter $\boldsymbol{\kappa}[\ell_1]$ by five. When, as in our example, rule r_2 is conceptually executed by 5 processes, we denote this transition by $(r_2, 5)$, where 5 is the acceleration factor. A sequence of transitions forms a *schedule*, e.g., $(r_1, 2), (r_3, 1), (r_1, 1)$.

In this paper, we address a parameterized reachability problem, e.g., can at least one correct process reach the local state ℓ_5, when $n - f$ correct processes start in the local state ℓ_1? Or, in terms of counter systems, is a configuration with $\boldsymbol{\kappa}[\ell_5] \neq 0$ reachable from an initial configuration with $\boldsymbol{\kappa}[\ell_1] = n - f \wedge \boldsymbol{\kappa}[\ell_2] = 0$? As discussed in [41], acceleration does not affect reachability, and precise treatment of the resilience condition and threshold guards is crucial for solving this problem.

2.1 Schemas

When applied to a configuration, a schedule generates a *path*, that is, an alternating sequence of configurations and transitions. As initially x and y are zero, threshold guards φ_1, φ_2, and φ_3 evaluate to false. As rules may increase variables, these guards may eventually become true. In our example we do not consider guards like $x < t + 1$ that are initially true and become false, although we formally treat them in our technique. 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 application of r_4 (by different processes) first unlocks φ_2 and then φ_3. We introduce a notion of a *context* that is the set of threshold guards that evaluate to true in a configuration. For our example we observe that each path 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 a path is always monotonic, as the shared variables can only be increased.

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 call these rules unlocked in the context. This motivates our definition of a *schema*: a sequence of contexts and rules. We give an example of a schema below, where inside the curly brackets we give the contexts, and fixed sequences of rules in between. (We discuss the underlined rules below.)

$$\begin{aligned} S = \{\}\ & \underline{r_1}, \underline{r_1}\ \{\varphi_1\}\ \underline{r_1}, \underline{r_3}, r_4, \underline{r_4}\ \{\varphi_1, \varphi_2\} \\ & r_1, r_2, r_3, r_4, r_5, r_4, \underline{r_5}\ \{\varphi_1, \varphi_2, \varphi_3\}\ r_1, r_2, r_3, r_4, \underline{r_5}, \underline{r_6}\ \{\varphi_1, \varphi_2, \varphi_3\} \end{aligned} \tag{2.1}$$

Given a schema, we can generate a schedule by attaching to each rule an acceleration factor, which can possibly be 0. For instance, if we attach non-zero factors to the underlined rules in S, and a zero factor to the other rules, we generate the following schedule τ' (we omit the transitions with 0 factors here).

$$\tau' = \underbrace{(r_1, 1)}_{\tau'_1}, \underbrace{(r_1, 1)}_{t_1}, \underbrace{(r_1, 1), (r_3, 1)}_{\tau'_2}, \underbrace{(r_4, 1)}_{t_2}, \underbrace{\quad}_{\tau'_3}, \underbrace{(r_5, 1)}_{t_3}, \underbrace{(r_5, 2), (r_6, 4)}_{\tau'_4} \tag{2.2}$$

It can easily be checked that τ' is generated by schema S, because the sequence of the underlined rules in S matches the sequence of rules appearing in τ'.

In this paper, we show that the schedules generated by a few schemas — one per each monotonic sequence of contexts — span the set of all reachable configurations. To this end, we apply reduction and acceleration to relate arbitrary schedules to their representatives, which are generated by schemas.

2.2 Reduction and acceleration

In this section we show what we mean by a schedule being "related" to its representative. Consider, e.g., the following schedule τ from the initial state σ_0 with $n = 5$, $t = f = 1$, $\boldsymbol{\kappa}[\ell_1] = 1$, and $\boldsymbol{\kappa}[\ell_2] = 3$:

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

Observe that after $(r_1, 1), (r_1, 1)$, variable $x = 2$, and thus 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 (2.1), we apply partial order arguments — that is, a mover analysis [48] — 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 schedule τ' from Eq. (2.2), which we call the representative schedule of τ. As the representative schedule τ' is generated from the schema in (2.1), we say that the schema captures schedule τ. (It also captures τ'.) Importantly for reachability checking, if τ and τ' are applied to the same configuration, they end in the same configuration. These arguments are formalized in Sects. 5, 6 and 7.

2.3 Encoding a schema in SMT

One of the key insights in this paper is that reachability checking via schemas can be encoded efficiently as SMT queries in linear integer arithmetic. In more detail, finite paths of counter systems can be expressed with inequalities over counters such as $\boldsymbol{\kappa}[\ell_2]$ and $\boldsymbol{\kappa}[\ell_3]$, shared variables such as x and y, parameters such as n, t, and f, and acceleration factors. Also, threshold guards and resilience conditions are expressions in linear integer arithmetic.

We give an example of reachability checking with SMT using the *simple* schema $\{\}\ r_1, r_1\ \{\varphi_1\}$ which is contained in the schema S in Eq. (2.1). To obtain a complete encoding for S, one can similarly encode the other simple schemas and combine them.

To this end, we have to express constraints on three configurations σ_0, σ_1, and σ_2. For the initial configuration σ_0, we introduce integer variables: $\boldsymbol{\kappa}_1^0, \ldots, \boldsymbol{\kappa}_5^0$ for local state counters, x^0 and y^0 for shared variables, and n, t, and f for parameters. As is written in Eq. (2.3), the configuration σ_0 should satisfy the initial constraints, and its context should be empty (that is, all guards evaluate to false):

$$\boldsymbol{\kappa}_1^0 + \boldsymbol{\kappa}_2^0 = n - f \wedge \boldsymbol{\kappa}_3^0 = \boldsymbol{\kappa}_4^0 = \boldsymbol{\kappa}_5^0 = 0 \wedge x^0 = y^0 = 0$$
$$\wedge n \geq 3t \wedge t \geq f \geq 0 \wedge (\neg\varphi_1 \wedge \neg\varphi_2 \wedge \neg\varphi_3)[x^0/x, y^0/y] \tag{2.3}$$

The configuration σ_1 is reached from σ_0 by applying a transition with the rule r_1 and an acceleration factor δ^1, and the configuration σ_2 is reached from σ_1 by applying a transition with the rule r_1 and an acceleration factor δ^2. Applying transition with the rule r_1 to σ_0 just means to increase both $\boldsymbol{\kappa}[\ell_3]$ and x by δ^1 and decrease $\boldsymbol{\kappa}[\ell_2]$ by δ^1. Hence, we introduce four fresh variables per transition and add the arithmetic operations. According to the schema, configuration σ_2 has the context $\{\varphi_2\}$. The following equations express these constraints[1]:

$$\boldsymbol{\kappa}_3^1 = \boldsymbol{\kappa}_3^0 + \delta^1 \wedge \boldsymbol{\kappa}_2^1 = \boldsymbol{\kappa}_2^0 - \delta^1 \wedge x^1 = x^0 + \delta^1 \tag{2.4}$$
$$\boldsymbol{\kappa}_3^2 = \boldsymbol{\kappa}_3^1 + \delta^2 \wedge \boldsymbol{\kappa}_2^2 = \boldsymbol{\kappa}_2^1 - \delta^2 \wedge x^2 = x^1 + \delta^2$$
$$\wedge(\varphi_1 \wedge \neg\varphi_2 \wedge \neg\varphi_3)[x^2/x, y^0/y] \tag{2.5}$$

Finally, we express the reachability question for all paths generated by the simple schema $\{\}\ r_1, r_1\ \{\varphi_1\}$. Whether there is a configuration with $\boldsymbol{\kappa}[\ell_5] \neq 0$ reachable from an initial configuration with $\boldsymbol{\kappa}[\ell_1] = n - f$ and $\boldsymbol{\kappa}[\ell_2] = 0$ can then be encoded as:

$$\boldsymbol{\kappa}_1^0 = n - f \wedge \boldsymbol{\kappa}_2^0 = 0 \wedge \boldsymbol{\kappa}_5^0 \neq 0 \tag{2.6}$$

Note that we check only $\boldsymbol{\kappa}_5^0$ against zero, as the local state ℓ_5 is never updated by the rule r_1. It is easy to see that conjunction of Eqs. (2.3)–(2.6) does not have a solution, and thus all paths generated by the schema $\{\}\ r_1, r_1\ \{\varphi_1\}$ do not reach a configuration with $\boldsymbol{\kappa}[\ell_5] \neq 0$. By writing down constraints for the other three simple schemas in Eq. (2.1), we can check reachability for the paths generated by the whole schema as well. As discussed in Sect. 2.1, our results also imply reachability on all paths whose representatives are generated by the schema. More details on the SMT encoding can be found in Sect. 9.

3 Parameterized counter systems

We recall the framework of [41] to the extent necessary, and extend it with the notion of a context in Sect. 3.2. 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.

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 the natural numbers $\mathbb{N}_0$. 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 $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Pi|} : \mathbf{p} \models RC\}$.

A key ingredient of threshold automata are threshold guards (or, just guards):

Definition 3.1 A *threshold guard* is an inequality of one of the following two forms:

(R) $x \geq a_0 + a_1 \cdot p_1 + \cdots + a_{|\Pi|} \cdot p_{|\Pi|}$, or
(F) $x < a_0 + a_1 \cdot p_1 + \cdots + a_{|\Pi|} \cdot p_{|\Pi|}$,

where $x \in \Gamma$ is a shared variable, $a_0, \ldots, a_{|\Pi|} \in \mathbb{Z}$ are integer coefficients, and $p_1, \ldots, p_{|\Pi|} \in \Pi$ are parameters. We denote the set of all guards of the form (R) by Φ^{rise}, and the set of all guards of the form (F) by Φ^{fall}.

[1] Our model requires *all* variables to be non-negative integers. Although these constraints (e.g., $x^1 \geq 0$) have to be encoded in the SMT queries, we omit these constraints here for a more concise presentation.

A rule defines a conditional transition between local states that may update the shared variables. Formally, a *rule* is a tuple $(\mathit{from}, \mathit{to}, \varphi^{\text{rise}}, \varphi^{\text{fall}}, \mathbf{u})$: the local states *from* and *to* are from $\mathcal{L}$. (Intuitively, they capture from which local state to which a process moves.) A rule is only executed if the conditions φ^{rise} and φ^{fall} evaluate to true. Condition φ^{rise} is a conjunction of guards from Φ^{rise}, and φ^{fall} is a conjunction of guards from Φ^{fall} (cf. Definition 3.1). We denote the set of guards used in φ^{rise} by $\mathsf{guard}(\varphi^{\text{rise}})$, and $\mathsf{guard}(\varphi^{\text{fall}})$ is the set of guards used in φ^{fall}.

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. As $\mathbf{u} \in \mathbb{N}_0^{|\Gamma|}$, global variables can only be increased or left unchanged. As will be later formalized in Proposition 3.1, guards from Φ^{rise} can only change from false to true (rise), and guards from Φ^{fall} can change from true to false (fall). Finally, $\mathcal{R}$ is the finite set of rules. We use the dot notation to refer to components of rules, e.g., $r.\mathit{from}$ or $r.\mathbf{u}$.

Example 3.1 In Fig. 2, the rule $r_2 \colon \varphi_2 \mapsto x_{++}$ that describes a transition from ℓ_1 to ℓ_3, can formally be written as $(\ell_1, \ell_3, \varphi_2, \top, (1, 0))$. Its intuitive meaning is as follows. If the guard $\varphi_2 : y \geq (t+1) - f$ evaluates to true, a process can move from the local state ℓ_1 to the local state ℓ_3, and the global variable x is incremented, while y remains unchanged. We formalize the semantics as counter systems in Sect. 3.1.

Definition 3.2 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 $r_1 = r_2$.

Assumption 3.3 We limit ourselves to threshold automata relevant for FTDAs, i.e., those where $r.\mathbf{u} = \mathbf{0}$ for all rules $r \in \mathcal{R}$ that satisfy $r \prec_P^+ r$. Such automata were called *canonical* in [41].

Remark 3.1 We use threshold automata to model fault-tolerant distributed algorithms that count messages from distinct senders. These algorithms are based on an "idealistic" reliable communication assumption (no message loss); these assumptions are typically expected to be ensured by "lower level bookkeeping code", e.g., communication protocols. As a result, the algorithms we consider here do not gain from sending the same message (that is, increasing a variable) inside a loop, so that we can focus on threshold automata that do not increase shared variables in loops.

Example 3.2 In the threshold automaton from Fig. 3 we have that $r_2 \prec_P r_3 \prec_P r_4 \prec_P r_5 \prec_P r_6 \prec_P r_8 \prec_P r_2$. Thus, we have that $r_2 \prec_P^+ r_2$. In our case this implies that $r_2.\mathbf{u} = \mathbf{0}$ by definition. Similarly we can conclude that $r_3.\mathbf{u} = r_4.\mathbf{u} = r_5.\mathbf{u} = r_6.\mathbf{u} = r_7.\mathbf{u} = r_8.\mathbf{u} = \mathbf{0}$.

Looplets The relation $\sim_P$ defines equivalence classes of rules. An equivalence class corresponds to a loop or to 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 $\prec_C$.

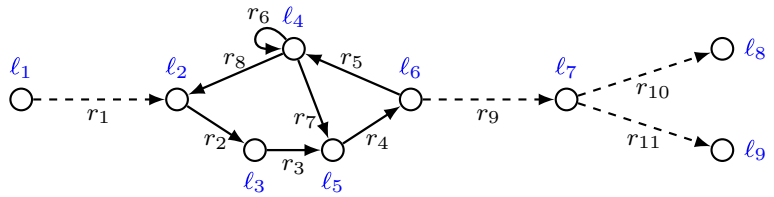

Fig. 3 A threshold automaton TA with local states $\mathcal{L} = \{\ell_i : 1 \leq i \leq 9\}$ and rules $\mathcal{R} = \{r_i : 1 \leq i \leq 11\}$. The rules drawn with solid arrows $\{r_2, \ldots, r_8\}$ constitute a single equivalence class, while all other rules are singleton equivalence classes

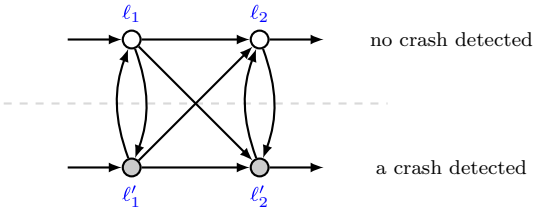

Fig. 4 A typical structure found in threshold automata that model fault-tolerant algorithms with a failure detector [12]. The gray circles depict those local states, where the failure detector reports a crash. The local states ℓ_i and ℓ'_i differ only in the output of the failure detector. As the failure detector output changes non-deterministically, the threshold automaton contains loops of size two

Example 3.3 Consider Fig. 3. The threshold automaton has five looplets: $c_1 = \{r_1\}$, $c_2 = \{r_2, \ldots, r_8\}$, $c_3 = \{r_9\}$, $c_4 = \{r_{10}\}$, and $c_5 = \{r_{11}\}$. From $r_9 \prec_P r_{10}$, it follows that $c_3 \prec_C c_4$, and from $r_4 \prec_P^+ r_{10}$, it follows that $c_2 \prec_C c_4$. We can pick two linear extensions of $\prec_C$, denoted by $\prec_1$ and $\prec_2$. We have $c_1 \prec_1 \cdots \prec_1 c_5$, and $c_1 \prec_2 c_2 \prec_2 c_3 \prec_2 c_5 \prec_2 c_4$. In this paper we always fix one linear extension.

Remark 3.2 It may seem natural to collapse such loops into singleton local states. In our case studies, e.g, [29], non-trivial loops are used to express non-deterministic choice due to failure detectors [12], as shown in Fig. 4. Importantly, some local states inside the loops appear in the specifications. Thus, one would have to use arguments from distributed computing to characterize when collapsing states is sound. In this paper, we present a technique that deals with the loops without need for additional modelling arguments.

3.1 Counter systems

We use a function $N : \mathbf{P}_{RC} \to \mathbb{N}_0$ to capture the number of processes for each combination of parameters. As we use SMT, we assume that N can be expressed in linear integer arithmetic. For instance, if only correct processes are explictly modeled we typically have $N(n, t, f) = n - f$, and the respective SMT expression is $n - f$. Given N, a threshold automaton TA, and admissible parameter values $\mathbf{p} \in \mathbf{P}_{RC}$, we define a counter system as a transition system (Σ, I, R). It consists of the set of configurations Σ, which contain evaluations of 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 $\sum_{i\notin\mathcal{I}} \sigma.\boldsymbol{\kappa}[i] = 0$. This means that in every initial configuration all global variables have zero values, and all $N(\mathbf{p})$ modeled processes are located only in the initial local states.

Example 3.4 Consider the threshold automaton from Fig. 2 with the initial states ℓ_1 and ℓ_2. Let us consider a system of five processes, one of them being Byzantine faulty. Thus, $n = 5$, $t = f = 1$, and we explicitely model $N(5, 1, 1) = n - f = 4$ correct processes. One of the initial configurations is $\sigma = (\boldsymbol{\kappa}, \mathbf{g}, \mathbf{p})$, where $\sigma.\boldsymbol{\kappa} = (1, 3, 0, 0, 0)$, $\sigma.\mathbf{g} = (0, 0)$, and $\sigma.\mathbf{p} = (5, 1, 1)$. In other words, there is one process in ℓ_1, three processes in ℓ_2, and global variables are initially $x = y = 0$. Note that $\sum_{i\in\mathcal{I}} \sigma.\boldsymbol{\kappa}[i] = \boldsymbol{\kappa}[\ell_1] + \boldsymbol{\kappa}[\ell_2] = 1 + 3 = 4 = N(5, 1, 1)$.

Transition relation R A *transition* is a pair $t = (rule, factor)$ of a rule of the TA and a non-negative integer called the *acceleration factor*, or just factor for short. (As already discussed in Sect. 2.1, we will use the zero factors when generating schedules from schemas.) For a transition $t = (rule, factor)$ we refer by $t.\mathbf{u}$ to $rule.\mathbf{u}$, and by $t.\varphi^{\text{fall}}$ to $rule.\varphi^{\text{fall}}$, etc. We say a transition t is *unlocked* in configuration σ if $(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + k \cdot t.\mathbf{u}, \sigma.\mathbf{p}) \models t.\varphi^{\text{rise}} \wedge t.\varphi^{\text{fall}}$, for $k \in \{0, \ldots, t.factor - 1\}$. Note that here we use a notation that a configuration satisfies a formula, which is considered true if and only if the formula becomes true when all free variables of the formulas are evaluated as in the configuration.

We say that transition t is *applicable (or enabled)* in configuration σ, if it is unlocked, and $\sigma.\boldsymbol{\kappa}[t.from] \geq t.factor$. (As all counters are non-negative, a transition with the zero *factor* is always applicable to all configurations provided that the guards are unlocked.) We say that σ' is the result of applying the enabled transition t to σ, and write $\sigma' = t(\sigma)$, if

- $\sigma'.\mathbf{g} = \sigma.\mathbf{g} + t.factor \cdot t.\mathbf{u}$ and $\sigma'.\mathbf{p} = \sigma.\mathbf{p}$
- if $t.from \neq t.to$ then
  - $\sigma'.\boldsymbol{\kappa}[t.from] = \sigma.\boldsymbol{\kappa}[t.from] - t.factor$,
  - $\sigma'.\boldsymbol{\kappa}[t.to] = \sigma.\boldsymbol{\kappa}[t.to] + t.factor$, and
  - $\forall \ell \in \mathcal{L}\backslash\{t.from, t.to\}$ it holds that $\sigma'.\boldsymbol{\kappa}[\ell] = \sigma.\boldsymbol{\kappa}[\ell]$
- if $t.from = t.to$ 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 rule $r \in \mathcal{R}$ and a factor $k \in \mathbb{N}_0$ such that $\sigma' = t(\sigma)$ for $t = (r, k)$. Updates do not decrease the values of shared variables, and thus the following proposition was introduced in [41]:

Proposition 3.1 [41] *For all configurations σ, all rules r, and all transitions t applicable to σ, the following holds:*

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

Schedules and paths A *schedule* is a (finite) sequence of transitions. For a schedule τ and an index $i : 1 \leq i \leq |\tau|$, by $\tau[i]$ we denote the ith transition of τ, and by τ^i we denote the prefix $\tau[1], \ldots, \tau[i]$ 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 $t_i.factor > 1$, 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-1} 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_0, 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^{rise} and Φ^{fall} in a configuration solely defines whether certain transitions are unlocked (but not necessarily enabled). From Proposition 3.1, one can see that after a transition has been applied, more guards from Φ^{rise} may get unlocked and more guards from Φ^{fall} may get locked. In other words, more guards from Φ^{rise} may evaluate to true and more guards from Φ^{fall} may evaluate to false. To capture this intuition, we define:

Definition 3.4 A *context* Ω is a pair $(\Omega^{\text{rise}}, \Omega^{\text{fall}})$ with $\Omega^{\text{rise}} \subseteq \Phi^{\text{rise}}$ and $\Omega^{\text{fall}} \subseteq \Phi^{\text{fall}}$. We denote by $|\Omega| = |\Omega^{\text{rise}}| + |\Omega^{\text{fall}}|$.

For two contexts $(\Omega_1^{\text{rise}}, \Omega_1^{\text{fall}})$ and $(\Omega_2^{\text{rise}}, \Omega_2^{\text{fall}})$, we define that $(\Omega_1^{\text{rise}}, \Omega_1^{\text{fall}}) \sqsubset (\Omega_2^{\text{rise}}, \Omega_2^{\text{fall}})$ if and only if $\Omega_1^{\text{rise}} \cup \Omega_1^{\text{fall}} \subset \Omega_2^{\text{rise}} \cup \Omega_2^{\text{fall}}$. 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 $\Omega_m = (\Phi^{\text{rise}}, \Phi^{\text{fall}})$ and $|\Omega_{i+1}| = |\Omega_i| + 1$, for $1 \leq i < m$. We obtain:

Proposition 3.2 *Every maximal monotonically increasing sequence of contexts is of length* $|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}| + 1$. *There are at most* $(|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|)!$ *such sequences.*

Example 3.5 For the example in Fig. 2, we have $\Phi^{\text{rise}} = \{\varphi_1, \varphi_2, \varphi_3\}$, and $\Phi^{\text{fall}} = \emptyset$. Thus, there are $(|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|)! = 6$ maximal monotonically increasing sequences of contexts. Two of them are $(\emptyset, \emptyset) \sqsubset (\{\varphi_1\}, \emptyset) \sqsubset (\{\varphi_1, \varphi_2\}, \emptyset) \sqsubset (\{\varphi_1, \varphi_2, \varphi_3\}, \emptyset)$ and $(\emptyset, \emptyset) \sqsubset (\{\varphi_3\}, \emptyset) \sqsubset (\{\varphi_1, \varphi_3\}, \emptyset) \sqsubset (\{\varphi_1, \varphi_2, \varphi_3\}, \emptyset)$. All of them have length $|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}| + 1 = 4$.

To every configuration σ, we attach the context consisting of all guards in Φ^{rise} that evaluate to true in σ, and all guards in Φ^{fall} that evaluate to *false* in σ:

Definition 3.5 Given a threshold automaton, we define its *configuration context* as a function $\omega : \Sigma \to 2^{\Phi^{\text{rise}}} \times 2^{\Phi^{\text{fall}}}$ that for each configuration $\sigma \in \Sigma$ gives a context $(\Omega^{\text{rise}}, \Omega^{\text{fall}})$ with $\Omega^{\text{rise}} = \{\varphi \in \Phi^{\text{rise}} : \sigma \models \varphi\}$ and $\Omega^{\text{fall}} = \{\varphi \in \Phi^{\text{fall}} : \sigma \not\models \varphi\}$.

The following monotonicity result is a direct consequence of Proposition 3.1.

Proposition 3.3 *If a transition* t *is enabled in a configuration* σ, *then either* $\omega(\sigma) \sqsubset \omega(t(\sigma))$, *or* $\omega(\sigma) = \omega(t(\sigma))$.

Definition 3.6 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 3.4 *A schedule* τ *is steady for a configuration* σ *if and only if* $\omega(\sigma) = \omega(\tau(\sigma))$.

In the following definition, we associate a sequence of contexts with a path:

Definition 3.7 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 there exist indices $n_0, \ldots, n_m$, with $0 = n_0 \leq n_1 \leq \ldots \leq n_m = |\tau| + 1$, such that for every k, $1 \leq k \leq m$, and every i with $n_{k-1} \leq i < n_k$, it holds that $\omega(\tau^i(\sigma)) = \Omega_k$.

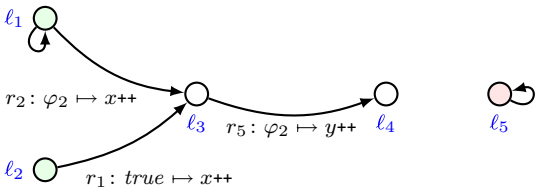

Fig. 5 The slice of the TA in Fig. 2 that is constructed for the context $(\{\varphi_2\}, \emptyset)$

Every path is consistent with a uniquely defined maximal monotonically increasing sequence of contexts. (Some of the indices $n_0, \ldots, n_m$ in Definition 3.7 may be equal.) In Sect. 4, we use such sequences of contexts to construct a schema recognizing many paths that are consistent with the same sequence of contexts.

A context defines which rules of the TA are unlocked. A schedule that is steady for a configuration visits only one context, and thus we can statically remove TA's rules that are locked in the context:

Definition 3.8 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 $\Omega = (\Omega^{\text{rise}}, \Omega^{\text{fall}})$ 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 $\big(\bigwedge_{\varphi \in \Omega^{\text{rise}}} \varphi\big) \to r.\varphi^{\text{rise}}$ and $\big(\bigwedge_{\psi \in \Phi^{\text{fall}} \setminus \Omega^{\text{fall}}} \psi\big) \to r.\varphi^{\text{fall}}$.

In other words, $\mathcal{R}|_\Omega$ contains those and only those rules r with guards that evaluate to true in all configurations σ with $\omega(\sigma) = \Omega$. These are exactly the guards from $\Omega^{\text{rise}} \cup (\Phi^{\text{fall}} \setminus \Omega^{\text{fall}})$. When $\omega(\sigma) = \Omega$, then all guards from Ω^{rise} evaluate to true, and then $r.\varphi^{\text{rise}}$ must also be true. As Ω^{fall} contains those guards from Φ^{fall} that evaluate to false in σ, then all other guards from Φ^{fall} must evaluate to true, and then $r.\varphi^{\text{fall}}$ must be true too. Figure 5 shows an example of a slice.

3.3 Model checking problem: 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, such 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

To address parameterized reachability, we introduce schemas, i.e., alternating sequences of contexts and rule sequences. A schema serves as a pattern for a set of paths, and is used to efficiently encode parameterized reachability in SMT. As parameters give rise to infinitely many initial states, a schema captures an *infinite* set of paths. We show how to construct a *finite* set of schemas $\mathcal{S}$ with the following property: for each schedule τ and each configuration σ there is a representative schedule $s(\tau)$ such that: (1) applying $s(\tau)$ to σ results in $\tau(\sigma)$, and (2) $\mathsf{path}(\sigma, s(\tau))$ is generated by a schema from $\mathcal{S}$.

Definition 4.1 A schema is a sequence $\Omega_0, \rho_1, \Omega_1, \ldots, \rho_m, \Omega_m$ of alternating contexts and rule sequences. We often 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 4.2 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 $f_1, \ldots, f_m \geq 0$, 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 $1 \leq i \leq k$.

Remark 4.1 Definition 4.2 allows schemas to generate paths that have transitions with zero acceleration factors. Applying a transition with a zero factor to a configuration σ results in the same configuration σ, which corresponds to a stuttering step. This does not affect reachability. In the following, we will apply Definition 4.2 to representative paths that may have transitions with zero factors.

Example 4.1 Let us go back to the example of a schema S and a schedule τ' introduced in Eqs. (2.1) and (2.2) in Sect. 2.1. It is easy to see that schema S can be decomposed into four simple schemas $S_1 \circ \cdots \circ S_4$, e.g., $S_1 = \{\}\ r_1, r_1\ \{\varphi_1\}$ and $S_2 = \{\varphi_1\}\ r_1, r_3, r_4, r_4\ \{\varphi_1, \varphi_2\}$. Consider an initial state σ_0 with $n = 5, t = f = 1, x = y = 0, \boldsymbol{\kappa}[\ell_1] = 1, \boldsymbol{\kappa}[\ell_2] = 3$, and $\boldsymbol{\kappa}[\ell_i] = 0$ for $i \in \{3, 4, 5\}$. To ensure that $\mathsf{path}(\sigma_0, \tau')$ is generated by schema S, one has to check Definition 4.2 for schemas $S_1, \ldots, S_4$ and schedules $(\tau'_1 \cdot t_1), (\tau'_2 \cdot t_2), (\tau'_3 \cdot t_3)$, and τ'_4, respectively. For instance, $\mathsf{path}(\sigma_0, \tau'_1 \cdot t_1)$ is generated by S_1. Indeed, take the sequence of indices 1 and 2 and the sequence of acceleration factors 1 and 1. The path $\mathsf{path}(\sigma_0, \tau'_1 \cdot t_1)$ ends in the configuration σ_1 that differs from σ_0 in that $\boldsymbol{\kappa}[\ell_2] = 1, \boldsymbol{\kappa}[\ell_3] = 2$, and $x = 2$. The contexts $\omega(\sigma_0) = (\{\}, \{\})$ and $\omega(\sigma_1) = (\{\varphi_1\}, \{\})$ match the contexts of schema S_1, as required by Definition 4.2.

Similarly, $\mathsf{path}(\sigma_1, \tau'_2 \cdot t_2)$ is generated by schema S_2. To see that, compare the contexts and use the index sequence 1, 2, 4, and acceleration factors 1.

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 for every set of configurations $C \subseteq \Sigma$ it is the case that the set of all states reachable from C via the paths generated by the schemas from $\mathcal{S}$, is exactly the set of all possible states reachable from C. Formally, $\forall C \subseteq \Sigma.\ \{\tau(\sigma) \mid \sigma \in C,\ \tau \text{ is applicable to } \sigma\} = \mathsf{Reach}(C, \mathcal{S})$.

In [41], a quantity $\mathcal{C}$ has been introduced that depends on the number of conditions in a TA. It has been 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 $\tau(\sigma)$ [41, Thm. 8]. Hence, by enumerating all sequences of rules of length up to d, one can construct a complete set of schemas:

Theorem 4.1 *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 $|\Phi^{\text{rise}}|$ and $|\Phi^{\text{fall}}|$ are in practice much smaller than the number of rules $|\mathcal{R}|$:

Theorem 4.2 *For all threshold automata, there exists a complete schema set of cardinality at most* $(|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|)!$. *In this set, the length of each schema does not exceed* $(3 \cdot |\Phi^{\text{rise}} \cup \Phi^{\text{fall}}| + 2) \cdot |\mathcal{R}|$.

In the following sections we prove the ingredients of the following argument for the theorem: construct the set Z of all maximal monotonically increasing sequences of contexts. From Proposition 3.2, we know that there are at most $(|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|)!$ maximal monotonically increasing sequences of contexts. Therefore, $|Z| \leq (|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|)!$. Then, for each sequence $z \in Z$, we do the following:

(1) We 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. We introduce and formally define this specific structure of representative schedules in Sects. 5, 6 and 7. We prove existence and properties of the representative schedule in Theorem 7.1 (Sect. 7). Before that we consider special cases: when all rules of a schedule belong to the same looplet (Theorem 5.1, Sect. 5), and when a schedule is steady (Theorem 6.1, Sect. 6).
(2) Next we construct a schema (for the sequence z) and show that it generates all paths of all schedules $s(\tau)$ found in (1). The length of the schema is at most $(3 \cdot (|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|) + 2) \cdot |\mathcal{R}|$. This is shown in Theorem 7.2 (Sect. 7).

Theorem 4.2 follows from the above theorems, which we prove in the following.

Remark 4.2 Let us stress the difference between [41] and this work. From [41], it follows that in order to check correctness of a TA it is sufficient to check only the schedules of bounded length $d(\mathsf{TA})$. The bound $d(\mathsf{TA})$ does not depend on the parameters, and can be computed for each TA. The proofs in [41] demonstrate that every schedule longer than $d(\mathsf{TA})$ can be transformed into an "equivalent" representative schedule, whose length is bounded by $d(\mathsf{TA})$. Consequently, one can treat every schedule of length up to $d(\mathsf{TA})$ as its own representative schedule. Similar reasoning does not apply to the schemas constructed in this paper: (i) we construct a complete set of schemas, whose cardinality is substantially smaller than $|\mathcal{S}_d|$, and (ii) the schemas constructed in this paper can be twice as long as the schemas in $\mathcal{S}_d$.

As discussed in Remark 3.2, the looplets in our case studies are typically either singleton looplets or looplets of size two. In fact, most of our benchmarks have singleton looplets only, and thus their threshold automata can be reduced to directed acyclic graphs. The theoretical constructs of Sect. 5.2 are presented for the more general case of looplets of any size. For most of the benchmarks — the ones not using failure detectors — we need only the simple construction laid out in Sect. 5.1.

5 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. The goal is to construct a single schema per looplet. The technical challenge is that this *single* schema must generate representative schedules for *all* possible schedules, where, intuitively, processes may move arbitrarily between all local states in the looplet. As a consequence, the rules that appear in the representative schedules can differ from the rules that appear in the arbitrary schedules visiting a looplet.

We fix a threshold automaton, a context Ω, a configuration σ with $\omega(\sigma) = \Omega$, a looplet c, and a schedule τ applicable to σ and using only rules from c. We then construct the representative schedule $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ and the schema $\mathsf{cschema}_c^{\Omega}$.

The technical details of the construction of $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ for the case when $|c| = 1$ is given in Sect. 5.1, and for the case when $|c| > 1$ in Sect. 5.2. We show in Sect. 5.3 that these constructions give us a schedule that has the desired properties: it reaches the same final state as the given schedule τ, and its length does not exceed $2 \cdot |c|$.

Note that in [41], the length of the representative schedule was bounded by $|c|$. However, all representative schedules of a looplet in this section can be generated by a single looplet schema.

5.1 Singleton looplet

Let us consider the case of the looplet c containing only one transition, that is, $|c| = 1$. There is a trivial representative schedule of a single transition:

Lemma 5.1 *Given a threshold automaton, a configuration σ, and a schedule $\tau = (r, f_1), \ldots, (r, f_m)$ applicable to σ, one of the two schedules is also applicable to σ and results in $\tau(\sigma)$: schedule $(r, f_1 + \cdots + f_m)$, or schedule $(r, 0)$.*

Proof We distinguish two cases:

Case $r.to = r.from$ Then, $r.\mathbf{u} = \mathbf{0}$, and $\tau^k(\sigma) = \sigma$ for $0 \leq k \leq |\tau|$. Consequently, the schedule $(r, 0)$ is applicable to σ, and it results in $\tau(\sigma) = \sigma$.

Case $r.to \neq r.from$ We prove by induction on the length $k : 1 \leq k \leq m$ of a prefix of τ, that the following constraints hold for all k:

$$(\tau^k(\sigma)).\boldsymbol{\kappa}[r.from] = \sigma.\boldsymbol{\kappa}[r.from] - (f_1 + \cdots + f_k) \tag{5.1}$$

$$(\tau^k(\sigma)).\mathbf{g} = \sigma.\mathbf{g} + (f_1 + \cdots + f_k) \cdot r.\mathbf{u} \tag{5.2}$$

$$(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + f \cdot r.\mathbf{u}, \sigma.\mathbf{p}) \models r.\varphi^{\text{fall}} \wedge r.\varphi^{\text{rise}} \text{ for all } f \in \{0, \ldots, f_1 + \cdots + f_k\} \tag{5.3}$$

Base case $k = 1$. As schedule τ is applicable to σ, its first transition is enabled in σ. Thus, by the definition of an enabled transition, the rule r is unlocked, i.e., for all $f \in \{0, \ldots, f_1\}$, it holds $(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + f_1 \cdot r.\mathbf{u}, \sigma.\mathbf{p}) \models r.\varphi^{\text{fall}} \wedge r.\varphi^{\text{rise}}$. By the definition, once the transition $\tau[1]$ is applied, it holds that $\tau^1(\sigma).\boldsymbol{\kappa}[from] = \sigma.\boldsymbol{\kappa}[from] - f_1$ and $(\tau^k(\sigma)).\mathbf{g} = \sigma.\mathbf{g} + f_1 \cdot r.\mathbf{u}$. Thus, Constraints (5.1)–(5.3) are satisfied for $k = 1$.

Inductive step $k > 1$. As schedule τ is applicable to σ, its prefix τ^k is applicable to σ. Hence, transition $\tau[k]$ is applicable to $\tau^{k-1}(\sigma)$.

By the definition of an enabled transition, for all $f \in \{0, \ldots, f_k\}$, it holds

$$((\tau^{k-1}(\sigma)).\boldsymbol{\kappa}, ((\tau^{k-1}(\sigma)).\mathbf{g} + f \cdot r.\mathbf{u}, \sigma.\mathbf{p}) \models r.\varphi^{\text{fall}} \wedge r.\varphi^{\text{rise}}.$$

By applying the Eq. (5.2) for $k - 1$ of the inductive hypothesis, we obtain that for all $f \in \{0, \ldots, f_k\}$, it holds that $(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + (f_1 + \cdots + f_{k-1} + f \cdot r.\mathbf{u}, \sigma.\mathbf{p}) \models r.\varphi^{\text{fall}} \wedge r.\varphi^{\text{rise}}$. By combining this constraint with the constraint (5.3) for $k - 1$, we arrive at the constraint (5.3) for k.

By applying $\tau[k]$, we get that $(\tau^k(\sigma)).\boldsymbol{\kappa}[r.from] = (\tau^{k-1}(\sigma)).\boldsymbol{\kappa}[r.from] - f_k$ and $(\tau^k(\sigma)).\mathbf{g} = (\tau^{k-1}(\sigma)).\mathbf{g} + f_k \cdot r.\mathbf{u}$. By applying (5.1) and (5.2) for $k-1$ to these equations, we arrive at the Eqs. (5.1) and (5.2) for k.

Based on (5.1) and (5.3) for all values of k, and in particular $k = m$, we can now show applicability. From Eq. (5.1), we immediately obtain that $\sigma.\boldsymbol{\kappa}[r.from] \geq f_1 + \cdots + f_m$. From constraint (5.3), we obtain that $(\sigma.\boldsymbol{\kappa}, \sigma.\mathbf{g} + f \cdot r.\mathbf{u}, \sigma.\mathbf{p}) \models r.\varphi^{\text{fall}} \wedge r.\varphi^{\text{rise}}$ for all $f \in \{0, \ldots, f_1 + \cdots + f_m\}$. These are the required conditions for the transition $(r, f_1 + \cdots + f_m)$ to be applicable to the configuration σ. □

Consequently, when c has a single rule r, for configuration σ and a schedule $\tau = (r, f_1), \ldots, (r, f_m)$, Lemma 5.1 allows us to take the singleton schedule (r, f) as $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ and to take the singleton schema $\{\Omega\}\, r\, \{\Omega\}$ as $\mathsf{cschema}_c^{\Omega}$. The factor f is either $f_1 + \ldots + f_m$ or zero.

5.2 Non-singleton looplet

Next we focus on non-singleton looplets. Thus, we assume that $|c| > 1$. Our construction is based on two directed trees, whose undirected versions are spanning trees, sharing the same root. In order to find a representative of a steady schedule τ which leads from σ to $\tau(\sigma)$, we determine for each local state how many processes have to move in or out of the state, and then we move them along the edges of the trees. First, we give the definitions of such trees, and then we show how to use them to construct the representative schedules and the schema.

Spanning out-trees and in-trees We construct the *underlying graph of looplet* c, that is, a directed graph G_c, whose vertices consist of local states that appear as components *from* or *to* of the rules from c, and the edges are the rules of c. More precisely, we construct a directed graph $G_c = (V_c, E_c, L_c)$, whose edges from E_c are labeled by function $L_c : E_c \to c$ with the rules of c as follows:

$$
\begin{aligned}
V_c &= \{\ell \mid \exists r \in c,\ r.to = \ell \vee r.from = \ell\},\\
E_c &= \{(\ell, \ell') \mid \exists r \in c,\ r.from = \ell,\ r.to = \ell'\},\\
L_c((\ell, \ell')) &= r, \text{ if } r.from = \ell,\ r.to = \ell' \text{ for } (\ell, \ell') \in E_c \text{ and } r \in c.
\end{aligned}
$$

Lemma 5.2 *Given a threshold automaton and a non-singleton looplet $c \in \mathcal{R}/\!\sim$, graph G_c is non-empty and strongly connected.*

Proof As, $|c| > 1$ and thus $E_c \geq 2$, graph G_c is non-empty. To prove that G_c is strongly connected, we consider a pair of rules $r_1, r_2 \in c$. By the definition of a looplet, it holds that $r_1 \prec_P^+ r_2$ and $r_2 \prec_P^+ r_1$. Thus, there is a path in G_c from $r_1.to$ to $r_2.from$, and there is a path in G_c from $r_2.to$ to $r_1.from$. As r_1 and r_2 correspond to some edges in G_c, there is a cycle that contains the vertices $r_1.from$, $r_1.to$, $r_2.from$, and $r_2.to$. Thus, graph G_c is strongly connected. □

As G_c is non-empty and strongly connected, we can fix an arbitrary node $h \in V_c$ — called a *hub* — and construct two directed trees, whose undirected versions are spanning trees of the undirected version of G_c. These are two subgraphs of G_c: a directed tree $T_{\text{out}} = (V_c, E_{\text{out}})$, whose edges $E_{\text{out}} \subseteq E_c$ are *pointing away from* h (out-tree); a directed tree $T_{\text{in}} = (V_c, E_{\text{in}})$, whose edges $E_{\text{in}} \subseteq E_c$ are *pointing to* h (in-tree). For every node $v \in V_c \backslash \{h\}$, it holds that $|\{u\colon (u, v) \in E_{\text{out}}\}| = 1$ and $|\{w\colon (v, w) \in E_{\text{in}}\}| = 1$.

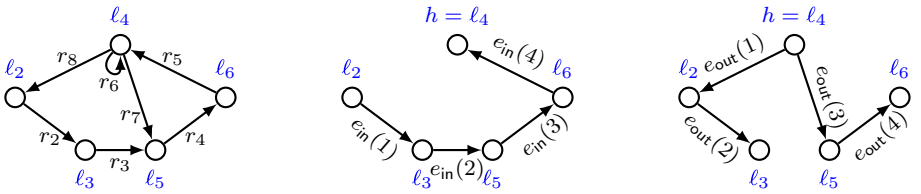

Fig. 6 The underlying graph of the looplet c_2 of the threshold automaton from Example 3.3 and Fig. 3 (left), together with trees T_{in} (middle) and T_{out} (right)

Further, we fix a topological order $\preceq_{\text{in}}$ on the edges of tree T_{in}. More precisely, $\preceq_{\text{in}}$ is such a partial order on E_{in} that for each pair of adjacent edges (ℓ, ℓ'), $(\ell', \ell'') \in E_{\text{in}}$, it holds that $(\ell, \ell') \preceq_{\text{in}} (\ell', \ell'')$. In the same way, we fix a topological order $\preceq_{\text{out}}$ on the edges of tree T_{out}.

Example 5.1 Consider again the threshold automaton from Example 3.3 and Fig. 3. We construct trees T_{in} and T_{out} for looplet c_2, shown in Fig. 6.

Note that $V_c = \{\ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$, and $E_c = \{(\ell_2, \ell_3), (\ell_3, \ell_5), (\ell_5, \ell_6), (\ell_6, \ell_4), (\ell_4, \ell_4), (\ell_4, \ell_5), (\ell_4, \ell_2)\}$. Fix ℓ_4 as a hub. We can fix a linear order $\preceq_{\text{in}}$ such that $(\ell_2, \ell_3) \preceq_{\text{in}} (\ell_3, \ell_5) \preceq_{\text{in}} (\ell_5, \ell_6) \preceq_{\text{in}} (\ell_6, \ell_4)$, and a linear order $\preceq_{\text{out}}$ such that $(\ell_4, \ell_2) \preceq_{\text{out}} (\ell_2, \ell_3) \preceq_{\text{out}} (\ell_4, \ell_5) \preceq_{\text{out}} (\ell_5, \ell_6)$.

Note that for the chosen hub l_4 and this specific example, T_{in} and $\preceq_{\text{in}}$ are uniquely defined, while an out-tree can be different from T_{out} from our Fig. 6 (the rules r_8, r_2, r_3, r_4 constitute a different tree from the same hub). Because out-tree T_{out} is not a chain, several linear orders different from $\preceq_{\text{out}}$ can be chosen, e.g., $(\ell_4, \ell_2) \preceq_{\text{out}} (\ell_4, \ell_5) \preceq_{\text{out}} (\ell_2, \ell_3) \preceq_{\text{out}} (\ell_5, \ell_6)$.

Representatives of non-singleton looplets Using these trees, we show how to construct a representative $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ of a schedule τ applicable to σ with $\sigma' = \tau(\sigma)$. For a configuration σ and a schedule τ applicable to σ, consider the trees T_{in} and T_{out}. We construct two sequences: the sequence $e_{\text{in}}(1), \ldots, e_{\text{in}}(|E_{\text{in}}|)$ of all edges of T_{in} following the order $\preceq_{\text{in}}$, i.e., if $e_{\text{in}}(i) \preceq_{\text{in}} e_{\text{in}}(j)$, then $i \leq j$; the sequence $e_{\text{out}}(1), \ldots, e_{\text{out}}(|E_{\text{out}}|)$ of all edges of T_{out} following the order $\preceq_{\text{out}}$. Further, we define the sequence of rules $r_{\text{in}}(1), \ldots, r_{\text{in}}(|E_{\text{in}}|)$ with $r_{\text{in}}(i) = L_c(e_{\text{in}}(i))$ for $1 \leq i \leq |E_{\text{in}}|$, and the sequence of rules $r_{\text{out}}(1), \ldots, r_{\text{out}}(|E_{\text{out}}|)$ with $r_{\text{out}}(i) = L_c(e_{\text{out}}(i))$ for $1 \leq i \leq |E_{\text{out}}|$. Using configurations σ and $\sigma' = \tau(\sigma)$, we define:

$$\delta_{\text{in}}(i) = \sigma.\boldsymbol{\kappa}[f] - \sigma'.\boldsymbol{\kappa}[f], \text{ for } f = r_{\text{in}}(i).\mathit{from} \text{ and } 1 \leq i \leq |E_{\text{in}}|,$$
$$\delta_{\text{out}}(j) = \sigma'.\boldsymbol{\kappa}[t] - \sigma.\boldsymbol{\kappa}[t], \text{ for } t = r_{\text{out}}(j).\mathit{to} \text{ and } 1 \leq j \leq |E_{\text{out}}|.$$

If $\delta_{\text{in}}(i) \geq 0$, then $\delta_{\text{in}}(i)$ processes should leave the local state $r_{\text{in}}(i).\mathit{from}$ towards the hub, and they do it exclusively using the edge $e_{\text{in}}(i)$. If $\delta_{\text{out}}(j) \geq 0$, then $\delta_{\text{out}}(j)$ processes should reach the state $r_{\text{out}}(j).\mathit{to}$ from the hub, and they do it exclusively using the edge $e_{\text{out}}(j)$. The negative values of $\delta_{\text{in}}(i)$ and $\delta_{\text{out}}(j)$ do not play any role in our construction, and thus, we use $\max(\delta_{\text{in}}(i), 0)$ and $\max(\delta_{\text{out}}(j), 0)$.

The main idea of the representative construction is as follows. First, we fire the sequence of rules $r_{\text{in}}(1), \ldots, r_{\text{in}}(k)$ to collect sufficiently many processes in the hub. Then, we fire the sequence of rules $r_{\text{out}}(1), \ldots, r_{\text{out}}(k)$ to distribute the required number of processes from the hub. As a result, for each location ℓ in the graph, the processes are transferred from ℓ

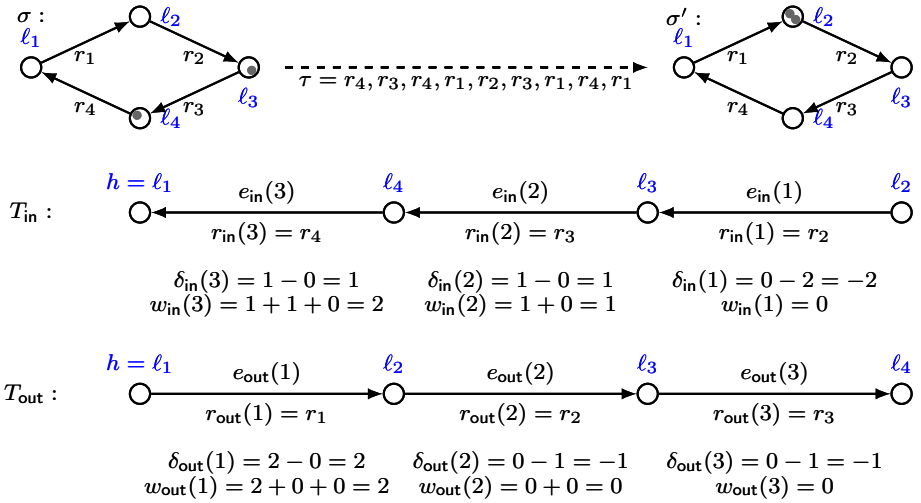

Fig. 7 Construction of the representative of a schedule using the rules in the four-element looplet, following Example 5.2

to the other locations, if $\sigma[\ell] > \sigma'[\ell]$, and additional processes arrive at ℓ, if $\sigma[\ell] < \sigma'[\ell]$. Using $\delta_{\mathsf{in}}(i)$ and $\delta_{\mathsf{out}}(i)$, we define the acceleration factors for each rule as follows:

$$w_{\mathsf{in}}(i) = \sum_{j\,:\,e_{\mathsf{in}}(j) \preceq_{\mathsf{in}} e_{\mathsf{in}}(i)} \max(\delta_{\mathsf{in}}(j), 0) \text{ and}$$

$$w_{\mathsf{out}}(i) = \sum_{j\,:\,e_{\mathsf{out}}(i) \preceq_{\mathsf{out}} e_{\mathsf{out}}(j)} \max(\delta_{\mathsf{out}}(j), 0).$$

Finally, we construct the schedule $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ as follows:

$$\begin{aligned}\mathsf{crep}_c^{\Omega}[\sigma, \tau] = {} & (r_{\mathsf{in}}(1), w_{\mathsf{in}}(1)), \ldots, (r_{\mathsf{in}}(|E_{\mathsf{in}}|), w_{\mathsf{in}}(|E_{\mathsf{in}}|)), \\ & (r_{\mathsf{out}}(1), w_{\mathsf{out}}(1)), \ldots, (r_{\mathsf{out}}(|E_{\mathsf{out}}|), w_{\mathsf{out}}(|E_{\mathsf{out}}|)).\end{aligned} \tag{5.4}$$

Example 5.2 Consider the TA shown in Fig. 7. Let c be the four-element looplet that contains the rules r_1, r_2, r_3, and r_4, and τ be the schedule $\tau = (r_4, 1), (r_3, 1), (r_4, 1), (r_1, 1), (r_2, 1), (r_3, 1), (r_1, 1), (r_4, 1), (r_1, 1)$ that uses the rules of the looplet c. Consider a configuration σ with $\sigma.\boldsymbol{\kappa}[\ell_3] = \sigma.\boldsymbol{\kappa}[\ell_4] = 1$, and $\sigma.\boldsymbol{\kappa}[\ell_1] = \sigma.\boldsymbol{\kappa}[\ell_2] = 0$. The final configuration $\sigma' = \tau(\sigma)$ has the following properties: $\sigma'.\boldsymbol{\kappa}[\ell_2] = 2$ and $\sigma'.\boldsymbol{\kappa}[\ell_1] = \sigma'.\boldsymbol{\kappa}[\ell_3] = \sigma'.\boldsymbol{\kappa}[\ell_4] = 0$. By comparing σ and σ', we notice that one process should move from ℓ_3 to ℓ_2, and one from ℓ_4 to ℓ_2. We will now show how this is achieved by our construction.

For constructing the representative schedule $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$, we first define trees T_{in} and T_{out}. If we chose ℓ_1 to be the hub, we get that $E_{\mathsf{in}} = \{(\ell_4, \ell_1), (\ell_3, \ell_4), (\ell_2, \ell_3)\}$, and thus the order is $(\ell_2, \ell_3) \preceq_{\mathsf{in}} (\ell_3, \ell_4) \preceq_{\mathsf{in}} (\ell_4, \ell_1)$. Therefore, we obtain $e_{\mathsf{in}}(1) = (\ell_2, \ell_3)$, $e_{\mathsf{in}}(2) = (\ell_3, \ell_4)$ and $e_{\mathsf{in}}(3) = (\ell_4, \ell_1)$. By calculating $\delta_{\mathsf{in}}(i)$ for every $i \in \{1, 2, 3\}$, we see that $\delta_{\mathsf{in}}(2) = 1$ and $\delta_{\mathsf{in}}(3) = 1$ are positive. Consequently, two processes go to the hub: one from $r_{\mathsf{in}}(2).\mathit{from} = \ell_3$ and one from $r_{\mathsf{in}}(3).\mathit{from} = \ell_4$. The coefficients w_{in} give us acceleration factors for all rules.

Similarly, we obtain $E_{\mathsf{out}} = \{(\ell_1, \ell_2), (\ell_2, \ell_3), (\ell_3, \ell_4)\}$, and the order must be $(\ell_1, \ell_2) \preceq_{\mathsf{out}} (\ell_2, \ell_3) \preceq_{\mathsf{out}} (\ell_3, \ell_4)$. Thus, $e_{\mathsf{out}}(1) = (\ell_1, \ell_2)$, $e_{\mathsf{in}}(2) = (\ell_2, \ell_3)$, and

$e_{\mathsf{out}}(3) = (\ell_3, \ell_4)$. Here only $\delta_{\mathsf{out}}(1) = 2$ has a positive value, and hence, two processes should move from hub to the local state $r_{\mathsf{out}}(1).to = \ell_2$. To achieve this, the acceleration factor of every rule $r_{\mathsf{out}}(i)$, $1 \le i \le 3$, must be $w_{\mathsf{out}}(i)$.

Therefore, by Eq. (5.4), the representative schedule is

$$\mathsf{crep}_c^{\Omega}[\sigma, \tau] = (r_2, 0), (r_3, 1), (r_4, 2), (r_1, 2), (r_2, 0), (r_3, 0).$$

Choosing another hub gives us another representative. For each hub, the representative is not longer than $2|c| = 8$, and leads to σ' when applied to σ.

In the following, we fix a threshold automaton TA, a context Ω, and a non-singleton looplet c of the slice $\mathsf{TA}|_{\Omega}$. We also fix a configuration σ of TA and a schedule τ that is contained in c and is applicable to σ. Our goal is to prove Lemma 5.8, which states that $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ is indeed applicable to σ and ends in $\tau(\sigma)$. To this end, we first prove auxiliary Lemmas 5.3–5.7.

Lemma 5.3 *For every $i : 1 \le i \le |E_{\mathsf{in}}|$, it holds that $\sigma.\boldsymbol{\kappa}[r_i.from] \ge \max(\delta_{\mathsf{in}}(i), 0)$, where $r_i = L_c(e_{\mathsf{in}}(i))$.*

Proof Recall that by the definition of a configuration, every counter $\sigma.\boldsymbol{\kappa}[\ell]$ is non-negative. If $\delta_{\mathsf{in}}(i) \ge 0$, then $\max(\delta_{\mathsf{in}}(i), 0) = \delta_{\mathsf{in}}(i) = \sigma.\boldsymbol{\kappa}[r_i.from] - \sigma'.\boldsymbol{\kappa}[r_i.from]$, which is bound from above by $\sigma.\boldsymbol{\kappa}[r_i.from]$. Otherwise, $\delta_{\mathsf{in}}(i) \le 0$, and we trivially have $\max(\delta_{\mathsf{in}}(i), 0) = 0$ and $0 \le \sigma.\boldsymbol{\kappa}[r_i.from]$. □

Lemma 5.4 *Schedule $\tau_{\mathsf{in}} = (r_{\mathsf{in}}(1), w_{\mathsf{in}}(1)), \ldots, (r_{\mathsf{in}}(|E_{\mathsf{in}}|), w_{\mathsf{in}}(|E_{\mathsf{in}}|))$ is applicable to configuration σ.*

Proof We denote by α^i the schedule $(r_{\mathsf{in}}(1), w_{\mathsf{in}}(1)), \ldots, (r_{\mathsf{in}}(i), w_{\mathsf{in}}(i))$, for $1 \le i \le |E_{\mathsf{in}}|$. Then $\tau_{\mathsf{in}} = \alpha^{|E_{\mathsf{in}}|}$.

All rules $r_{\mathsf{in}}(1), \ldots, r_{\mathsf{in}}(|E_{\mathsf{in}}|)$ are from $\mathcal{R}|_{\Omega}$, and thus are unlocked. Hence, it is sufficient to show that the values of the locations from the set V_c are large enough to enable each transition $(r_{\mathsf{in}}(i), w_{\mathsf{in}}(i))$ for $1 \le i \le |E_{\mathsf{in}}|$. To this end, we prove by induction that $(\alpha^{i-1}(\sigma)).\boldsymbol{\kappa}[r_i.from] \ge w_{\mathsf{in}}(i)$, for $1 \le i \le |E_{\mathsf{in}}|$ and $r_i = L_c(e_{\mathsf{in}}(i))$.

Base case $i = 1$. For $r_1 = L_c(e_{\mathsf{in}}(1))$, we want to show that $\sigma.\boldsymbol{\kappa}[r_1.from] \ge w_{\mathsf{in}}(1)$. As $e_{\mathsf{in}}(1)$ is the first element of the sequence $e_{\mathsf{in}}(1), \ldots, e_{\mathsf{in}}(E_{\mathsf{in}})$, which respects the order $\preceq_{\mathsf{in}}$, we conclude that $w_{\mathsf{in}}(1) = \max(\delta_{\mathsf{in}}(1), 0)$. From Lemma 5.3, it follows that $\sigma.\boldsymbol{\kappa}[r_1.from] \ge \max(\delta_{\mathsf{in}}(1), 0)$.

Inductive step k assume that for all $i : 1 \le i \le k - 1 < |E_{\mathsf{in}}|$, schedule α^i is applicable to σ and show that $(\alpha^{k-1}(\sigma)).\boldsymbol{\kappa}[r_k.from] \ge w_{\mathsf{in}}(k)$ with $r_k = L_c(e_{\mathsf{in}}(k))$.

To this end, we construct the set of edges P_k that precede the edge $e_{\mathsf{in}}(k)$ in the topological order $\preceq_{\mathsf{in}}$, that is, $P_k = \{e \mid e \in E_{\mathsf{in}},\ e \preceq_{\mathsf{in}} e_{\mathsf{in}}(k),\ e \ne e_{\mathsf{in}}(k)\}$. We show that the following equation holds:

$$\alpha^{k-1}(\sigma)).\boldsymbol{\kappa}[r_k.from] = \sigma.\boldsymbol{\kappa}[r_k.from] + \sum_{e_{\mathsf{in}}(j) \in P_k} \max(\delta_{\mathsf{in}}(j), 0). \tag{5.5}$$

Indeed, if one picks an edge $e_{\mathsf{in}}(j) \in P_k$, the edge $e_{\mathsf{in}}(j)$ adds $w_{\mathsf{in}}(j)$ to the counter $\boldsymbol{\kappa}[r_k.from]$. As the sequence $\{e_{\mathsf{in}}(i)\}_{i \le k}$ is topologically sorted, it follows that $j < k$. Moreover, as the tree T_{in} is oriented towards the root, $e_{\mathsf{in}}(k)$ is the only edge leaving the local state $r_k.from$. Thus, no edge $e_{\mathsf{in}}(i)$ with $i < k$ decrements the counter $\sigma.\boldsymbol{\kappa}[r_k.from]$.

From Eq. (5.5) and Lemma 5.3, we conclude that $(\alpha^{k-1}(\sigma)).\boldsymbol{\kappa}[r_k.from]$ is not less than $\max(\delta_{\mathsf{in}}(k), 0) + \sum_{e_{\mathsf{in}}(j):\ e_{\mathsf{in}}(j) \preceq_{\mathsf{in}} e_{\mathsf{in}}(k),\ j \ne k} \max(\delta_{\mathsf{in}}(j), 0)$, which equals to $w_{\mathsf{in}}(k)$. This proves the inductive step.

Therefore, we have shown that $\tau_{\mathsf{in}} = \alpha^{|E_{\mathsf{in}}|}$ is applicable to σ. □

The following lemma is easy to prove by induction on the length of a schedule. The base case for a single transition follows from the definition of a counter system.

Lemma 5.5 *Let σ and σ' be two configurations and τ be a schedule applicable to σ such that $\tau(\sigma) = \sigma'$. Then it holds that $\sum_{\ell \in \mathcal{L}} (\sigma'[\ell] - \sigma[\ell]) = 0$.*

Further, we show that the required number of processes is reaching (or leaving) the hub, when the transitions derived from the trees T_{in} and T_{out} are executed:

Lemma 5.6 *The following equality holds:*

$$\sigma'.\boldsymbol{\kappa}[h] - \sigma.\boldsymbol{\kappa}[h] = \sum_{1 \le i \le |E_{\mathsf{in}}|} \max(\delta_{\mathsf{in}}(i), 0) - \sum_{1 \le i \le |E_{\mathsf{out}}|} \max(\delta_{\mathsf{out}}(i), 0).$$

Proof Recall that T_{in} is a tree directed towards h, and the undirected version of T_{in} is a spanning tree of graph C. Hence, for each local state $\ell \in V_c \backslash \{h\}$, there is exactly one edge $e \in E_{\mathsf{in}}$ with $L_c(e).\mathit{from} = \ell$. Thus, the following equation holds:

$$\sum_{1 \le i \le |E_{\mathsf{in}}|} \max(\delta_{\mathsf{in}}(i), 0) = \sum_{\ell \in V_c \backslash \{h\}} \max(\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell], 0). \tag{5.6}$$

Similarly, T_{out} is a tree directed outwards h, and the undirected version of T_{out} is a spanning tree of graph C. Hence, for each local state $\ell \in V_c \backslash \{h\}$, there is exactly one edge $e \in E_{\mathsf{out}}$ with $L_c(e).\mathit{to} = \ell$. Thus, the following equation holds:

$$\sum_{1 \le i \le |E_{\mathsf{out}}|} \max(\delta_{\mathsf{out}}(i), 0) = \sum_{\ell \in V_c \backslash \{h\}} \max(\sigma'.\boldsymbol{\kappa}[\ell] - \sigma.\boldsymbol{\kappa}[\ell], 0). \tag{5.7}$$

By combining (5.6) and (5.7), we obtain the following:

$$\begin{aligned}
&\sum_{1 \le i \le |E_{\mathsf{in}}|} \max(\delta_{\mathsf{in}}(i), 0) - \sum_{1 \le i \le |E_{\mathsf{out}}|} \max(\delta_{\mathsf{out}}(i), 0) \\
&= \sum_{\ell \in V_c \backslash \{h\}} \big(\max(\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell], 0) - \max(\sigma'.\boldsymbol{\kappa}[\ell] - \sigma.\boldsymbol{\kappa}[\ell], 0) \big) \\
&= \sum_{\ell \in V_c \backslash \{h\}} \big(\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell] \big) = \left(\sum_{\ell \in V_c} \sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell] \right) - \big(\sigma.\boldsymbol{\kappa}[h] - \sigma'.\boldsymbol{\kappa}[h] \big).
\end{aligned} \tag{5.8}$$

As the initial schedule τ is applicable to σ, and $\tau(\sigma) = \sigma'$, by Lemma 5.5, $\sum_{\ell \in \mathcal{L}} (\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell]) = 0$. As all rules in $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ are from $\mathcal{R}|_{\Omega}$ and thus change only the counters of local states in V_c, for each local state $\ell \in \mathcal{L} \backslash V_c$, its respective counter does not change, that is, $\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell] = 0$. Hence, $\sum_{\ell \in V_c} (\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell]) = 0$. From this and Eq. (5.8), the statement of the lemma follows. □

Lemma 5.7 *If τ_{in} denotes the schedule $(r_{\mathsf{in}}(1), w_{\mathsf{in}}(1)), \ldots, (r_{\mathsf{in}}(|E_{\mathsf{in}}|), w_{\mathsf{in}}(|E_{\mathsf{in}}|))$, the following equation holds:*

$$\tau_{\mathsf{in}}(\sigma).\boldsymbol{\kappa}[\ell] = \begin{cases} \sigma'.\boldsymbol{\kappa}[h] + \sum_{1 \le i \le |E_{\mathsf{out}}|} \max(\delta_{\mathsf{out}}(i), 0), & \text{if } \ell = h \\ \min(\sigma.\boldsymbol{\kappa}[\ell], \sigma'.\boldsymbol{\kappa}[\ell]), & \text{if } \ell \in V_c \backslash \{h\}. \end{cases}$$

Proof We prove the lemma by case distinction:

Case $\ell = h$ We show that $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[h] = \sigma.\boldsymbol{\kappa}[h] + \sum_{1 \le i \le |E_{\mathsf{in}}|} \max(\delta_{\mathsf{in}}(i), 0)$. Indeed, let P be the indices of edges coming into h, i.e., $P = \{i \mid 1 \le i \le |E_{\mathsf{in}}|,\ L_c(e_{\mathsf{in}}(i)) =$

$r,\ h = r.to\}$. As all edges in T_{in} are oriented towards h, it holds that $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[h]$ equals to $\sigma.\boldsymbol{\kappa}[h] + \sum_{i \in P} w_{\mathsf{in}}(i)$. By unfolding the definition of w_{in}, we obtain that $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[h] = \sigma.\boldsymbol{\kappa}[h] + \sum_{1 \le i \le |E_{\mathsf{in}}|} \max(\delta_{\mathsf{in}}(i), 0)$. We observe that by Lemma 5.6, this sum equals to $\sigma'.\boldsymbol{\kappa}[h] + \sum_{1 \le i \le |E_{\mathsf{out}}|} \max(\delta_{\mathsf{out}}(i), 0)$. This proves the first case.

Case $\ell \in V_c \backslash \{h\}$ We show that $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[\ell] = \min(\sigma.\boldsymbol{\kappa}[\ell], \sigma'.\boldsymbol{\kappa}[\ell])$. Indeed, fix a node $\ell \in V_c \backslash \{h\}$ and construct two sets: the set of incoming edges $In = \{e_{\mathsf{in}}(i) \mid \exists \ell' \in V_c.\ e_{\mathsf{in}}(i) = (\ell', \ell)\}$ and the singleton set of outgoing edges $Out = \{e_{\mathsf{in}}(i) \mid \exists \ell' \in V_c.\ e_{\mathsf{in}}(i) = (\ell, \ell')\}$. By summing up the effect of all transitions in τ_{in}, we obtain $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[\ell] = \sigma.\boldsymbol{\kappa}[\ell] + \sum_{e_{\mathsf{in}}(i) \in In} w_{\mathsf{in}}(i) - \sum_{e_{\mathsf{out}}(i) \in Out} w_{\mathsf{out}}(i)$. By unfolding the definition of w_{in}, we obtain $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[\ell] = \sigma.\boldsymbol{\kappa}[\ell] - \sum_{e_{\mathsf{in}}(i) \in Out} \delta_{\mathsf{in}}(i)$, which can be rewritten as $\sigma.\boldsymbol{\kappa}[\ell] - \max(\sigma.\boldsymbol{\kappa}[\ell] - \sigma'.\boldsymbol{\kappa}[\ell], 0)$, which, in turn, equals to $\min(\sigma.\boldsymbol{\kappa}[\ell], \sigma'.\boldsymbol{\kappa}[\ell])$. This proves the second case. □

Now we are in a position to prove that schedule $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ is applicable to configuration σ and results in configuration $\tau(\sigma)$:

Lemma 5.8 *The schedule* $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ *has the following properties: (a)* $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ *is applicable to* σ*, and (b)* $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ *results in* $\tau(\sigma)$ *when applied to* σ*.*

Proof Denote with τ_{in} the prefix $(r_{\mathsf{in}}(1), w_{\mathsf{in}}(1)), \ldots, (r_{\mathsf{in}}(|E_{\mathsf{in}}|), w_{\mathsf{in}}(|E_{\mathsf{in}}|))$ of the schedule $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$. For each $j : 1 \le j \le |E_{\mathsf{out}}|$, denote with β^j the prefix of $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ that has length of $|E_{\mathsf{in}}| + j$. Note that $\beta^{|E_{\mathsf{out}}|} = \mathsf{crep}_c^{\Omega}[\sigma, \tau]$.

Proving applicability of $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ to σ We notice that all rules in $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ are from $\mathcal{R}|_{\Omega}$ and thus are unlocked, and that τ_{in} is applicable to σ by Lemma 5.4. Hence, we only have to check that the values of counters from V_c are large enough, so that transitions $(r_{\mathsf{out}}(j), w_{\mathsf{out}}(j))$ can fire.

We prove that each schedule β^j is applicable to σ, for $j : 1 \le j \le |E_{\mathsf{out}}|$. We do so by induction on the distance from the root h in the tree T_{out}.

Base case root node h. Denote with O_h the set $\{(\ell, \ell') \in E_{\mathsf{out}} \mid \ell = h\}$. Let $j_1, \ldots, j_m$ be the indices of all edges in O_h, and j_m be the maximum among them.

From Lemma 5.7, $(\tau_{\mathsf{in}}(\sigma)).\boldsymbol{\kappa}[h] = \sigma'.\boldsymbol{\kappa}[h] + \sum_{1 \le i \le |E_{\mathsf{out}}|} \max(\delta_{\mathsf{out}}(i), 0) = \sigma'.\boldsymbol{\kappa}[h] + \sum_{e_{\mathsf{out}}(j) \in O_h} w_{\mathsf{out}}(j)$. Thus, every transition $(e_{\mathsf{out}}(j), w_{\mathsf{out}}(j))$ with $e_{\mathsf{out}}(j) \in O_h$, is applicable to $\beta^{j-1}(\sigma)$. Also, $(\beta^{j_m}(\sigma)).\boldsymbol{\kappa}[h] = \sigma'.\boldsymbol{\kappa}[h]$.

Inductive step assume that for a node $\ell \in V_c$ and an edge $e_{\mathsf{out}}(k) = (\ell, \ell') \in E_{\mathsf{out}}$ outgoing from node ℓ, schedule β^k is applicable to configuration σ. Show that for each edge $e_{\mathsf{out}}(i)$ outgoing from node ℓ' the following hold: (i) schedule β^i is also applicable to σ; and (ii) $\beta^{|E_{\mathsf{out}}|}(\sigma).\boldsymbol{\kappa}[\ell'] = \sigma'.\boldsymbol{\kappa}[\ell']$.

(i) As the sequence $\{e_{\mathsf{out}}(j)\}_{j \le |E_{\mathsf{out}}|}$ is topologically sorted, for each edge $e_{\mathsf{out}}(i)$ outgoing from node ℓ', it holds that $k < i$.

From Lemma 5.7, we have that $\beta^k(\sigma).\boldsymbol{\kappa}[\ell'] = \min(\sigma.\boldsymbol{\kappa}[\ell'], \sigma'.\boldsymbol{\kappa}[\ell'])$. Because the transition $(e_{\mathsf{out}}(k), w_{\mathsf{out}}(k))$ adds $w_{\mathsf{out}}(k)$ to $\beta^{k-1}(\sigma).\boldsymbol{\kappa}[\ell']$, we have $\beta^k(\sigma).\boldsymbol{\kappa}[\ell'] = \min(\sigma.\boldsymbol{\kappa}[\ell'], \sigma'.\boldsymbol{\kappa}[\ell']) + w_{\mathsf{out}}(k)$. Let S be the set of all immediate successors of $e_{\mathsf{out}}(k)$, i.e., $S = \{i \mid \exists \ell''.\ (\ell', \ell'') = e_{\mathsf{out}}(i)\}$. From the definition of $w_{\mathsf{out}}(k)$, it follows that $w_{\mathsf{out}}(k) = \max(\delta_{\mathsf{out}}(k), 0) + \sum_{s \in S} w_{\mathsf{out}}(s)$. Thus, the transition $(e_{\mathsf{out}}(i), w_{\mathsf{out}}(i))$ for edge $e_{\mathsf{out}}(i)$ outgoing from node ℓ', can be executed.

(ii) Let $j_1, \ldots, j_m$ be the indices of all edges outgoing from ℓ', and j_m be the maximum among them. From (i), it follows that

$$(\beta^{j_m}(\sigma)).\boldsymbol{\kappa}[\ell'] = \min(\sigma.\boldsymbol{\kappa}[\ell'], \sigma'.\boldsymbol{\kappa}[\ell']) + \max(\delta_{\mathsf{out}}(k), 0),$$

which equals to $\sigma'.\boldsymbol{\kappa}[\ell']$.

This proves that the schedule $\beta^{|E_{\mathsf{out}}|} = \mathsf{crep}_c^{\Omega}[\sigma, \tau]$ is applicable to σ.

Proving that $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ *results in* $\tau(\sigma)$ From the induction above, we conclude that for each $\ell \in V_c$, it holds that $(\beta^{|E_{\mathsf{out}}|}(\sigma)).\boldsymbol{\kappa}[\ell] = \sigma'.\boldsymbol{\kappa}[\ell]$. Edges in the trees T_{in} and T_{out} change only local states from V_c. We conclude that for all $\ell \in \mathcal{L}$, it holds that $\mathsf{crep}_c^{\Omega}[\sigma, \tau](\sigma).\boldsymbol{\kappa}[\ell] = \sigma'.\boldsymbol{\kappa}[\ell]$. As the rules in non-singleton looplets do not change shared variables, $\mathsf{crep}_c^{\Omega}[\sigma, \tau](\sigma).\mathbf{g} = \sigma.\mathbf{g} = \sigma'.\mathbf{g}$. Therefore, $\mathsf{crep}_c^{\Omega}[\sigma, \tau](\sigma) = \sigma'$. □

5.3 Representatives for one context and one looplet

We now summarize results from Sects. 5.1 and 5.2, giving the representative of a schedule τ in the case when τ uses only the rules from one looplet, and does not change its context. If the given looplet consists of a single rule, the construction is given in Sect. 5.1, and otherwise in Sect. 5.2. We show that these constructions indeed give us a schedule of bounded length, that reaches the same state as τ.

In the following, given a threshold automaton TA and a looplet c, we will say that a *schedule* $\tau = t_1, \ldots, t_n$ *is contained in* c, if $[t_i.rule] = c$ for $1 \leq i \leq n$.

Theorem 5.1 *Fix a threshold automaton, and a context* Ω*, and a looplet* c *in the slice* $\mathsf{TA}|_{\Omega}$*. Let* σ *be a configuration and* τ *be a steady schedule contained in* c *and applicable to* σ*. 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 If $|c| = 1$, then we use a single accelerated transition or the empty schedule as representative, as described in Lemma 5.1.

If $|c| > 1$, we construct the representative as in Sect. 5.2, so that by Lemma 5.8 property (a) follows. For every edge $e \in E_c$, the rule $L_c(e)$ belongs to c, and thus $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ satisfies property (b). As $|E_{\mathsf{in}}| \leq |c|$ and $|E_{\mathsf{out}}| \leq |c|$, we conclude that $|\mathsf{crep}_c^{\Omega}[\sigma, \tau]| \leq 2 \cdot |c|$, and thus property c) is also satisfied. From this and Lemma 5.8, we conclude that $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ is the required representative schedule. □

Theorem 5.1 gives us a way to construct schemas that generate all representatives of the schedules contained in a looplet:

Theorem 5.2 *Fix a threshold automaton* TA*, a context* Ω*, and a looplet* c *in the slice* $\mathsf{TA}|_{\Omega}$*. There exists a schema* $\mathsf{cschema}_c^{\Omega}$ *with the following properties:*

Fix an arbitrary configuration σ *and a steady schedule* τ *that is contained in* c *and is applicable to* σ*. Let* $\tau' = \mathsf{crep}_c^{\Omega}[\sigma, \tau]$ *be the representative schedule of* τ*, from Theorem 5.1. Then,* $\mathit{path}(\sigma, \tau')$ *is generated by* $\mathsf{cschema}_c^{\Omega}$*. Moreover, the length of* $\mathsf{cschema}_c^{\Omega}$ *is at most* $2 \cdot |c|$.

Proof Note that $\tau' = \mathsf{crep}_c^{\Omega}[\sigma, \tau]$ can be constructed in two different ways depending on the looplet c.

If $|c| = 1$, then by Lemma 5.1 we have that $\tau' = (r, f)$ for a rule $r \in c$ and a factor $f \in \mathbb{N}_0$. In this case we construct $\mathsf{cschema}_c^{\Omega}$ to be

$$\mathsf{cschema}_c^{\Omega} = \{\Omega\}\, r\, \{\Omega\}.$$

It is easy to see that $\mathsf{path}(\sigma, \tau')$ is generated by $\mathsf{cschema}_c^{\Omega}$, as well as that the length of $\mathsf{cschema}_c^{\Omega}$ is exactly 1, that is less than $2 \cdot |c|$.

If $|c| > 1$, then we use the trees T_{in} and T_{out} to construct the schema $\mathsf{cschema}_c^{\Omega}$ as follows:

$$\mathsf{cschema}_c^{\Omega} = \{\Omega\}\, r_{\mathsf{in}}(1) \cdots r_{\mathsf{in}}(|E_{\mathsf{in}}|) \cdot r_{\mathsf{out}}(1) \cdots r_{\mathsf{out}}(|E_{\mathsf{out}}|)\, \{\Omega\}. \tag{5.9}$$

Since for an arbitrary configuration σ and a schedule τ, we use the same sequence of edges in Eqs. (5.4) and (5.9) to construct $\mathsf{crep}_c^{\Omega}[\sigma, \tau]$ and $\mathsf{cschema}_c^{\Omega}$, the schema $\mathsf{cschema}_c^{\Omega}$ generates all paths of the representative schedules, and its length is at most $2 \cdot |c|$. □

6 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.

Theorem 6.1 *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) $|\mathsf{srep}_{\Omega}[\sigma, \tau]| \leq 2 \cdot |(\mathcal{R}|_{\Omega})|$

To construct a representative schedule, we fix a context Ω of a 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 6.1 Let $\tau = t_1, \ldots, t_k$, for $k > 0$, be a schedule, and let C be a set of looplets. Given an increasing sequence of indices $i(1), \ldots, i(m) \in \{1, \ldots, k\}$, where $m \leq k$, 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 $\mathcal{R}|_{\Omega}$ 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:

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

Example 6.1 Consider the TA shown in Fig. 8. It has three looplets, namely $c_1 = \{r_1, r_2, r_3, r_4\}$, $c_2 = \{r_5\}$, $c_3 = \{r_6, r_7, r_8\}$, and the rules are depicted as solid, dotted, and dashed, respectively. These looplets are ordered such that $c_1 \prec_c^{lin} c_2 \prec_c^{lin} c_3$.

Let σ be the configuration represented in Fig. 8 left, i.e. $\boldsymbol{\kappa}[\ell_3] = \boldsymbol{\kappa}[\ell_4] = \boldsymbol{\kappa}[\ell_5] = 1$ and $\boldsymbol{\kappa}[\ell_3] = \boldsymbol{\kappa}[\ell_4] = \boldsymbol{\kappa}[\ell_5] = 0$. Let τ be the schedule $(r_4, 1), (r_6, 1), (r_3, 1), (r_4, 1), (r_1, 1), (r_2, 1), (r_7, 1), (r_3, 1), (r_1, 1), (r_5, 1), (r_7, 1), (r_4, 1), (r_8, 1), (r_1, 1), (r_6, 1), (r_7, 1), (r_5, 1), (r_8, 1), (r_7, 1)$. Note that τ is applicable to σ and that $\tau(\sigma)$ is the configuration σ' from Fig. 8 right, i.e. $\boldsymbol{\kappa}[\ell_5] = 1$, $\boldsymbol{\kappa}[\ell_6] = 2$ and $\boldsymbol{\kappa}[\ell_1] = \boldsymbol{\kappa}[\ell_2] = \boldsymbol{\kappa}[\ell_3] = \boldsymbol{\kappa}[\ell_4] = 0$. We construct the representative schedule $\mathsf{srep}_{\Omega}[\sigma, \tau]$.

Projection of τ on the looplets c_1, c_2, and c_3, gives us the following schedules:

$$\tau|_{c_1} = (r_4, 1), (r_3, 1), (r_4, 1), (r_1, 1), (r_2, 1), (r_3, 1), (r_1, 1), (r_4, 1), (r_1, 1),$$

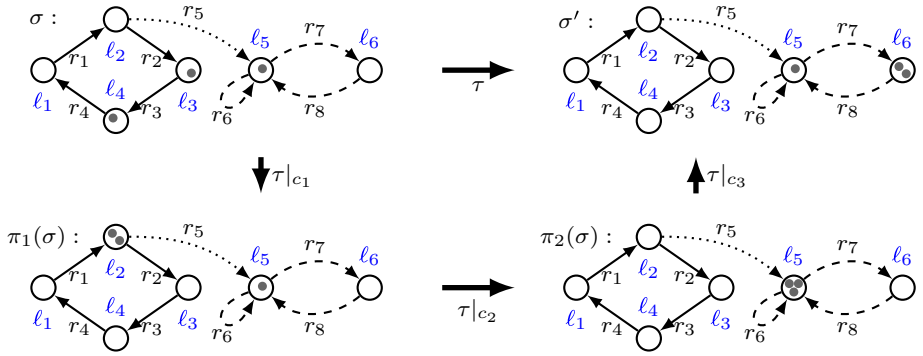

Fig. 8 Threshold automaton and configurations used in Example 6.1

$$\tau|_{c_2} = (r_5, 1), (r_5, 1),$$
$$\tau|_{c_3} = (r_6, 1), (r_7, 1), (r_7, 1), (r_8, 1), (r_6, 1), (r_7, 1), (r_8, 1), (r_7, 1).$$

Recall that

$$\mathsf{srep}_{\Omega}[\sigma, \tau] = \mathsf{crep}^{\Omega}_{c_1}[\pi_0(\sigma), \tau|_{c_1}] \cdot \mathsf{crep}^{\Omega}_{c_2}[\pi_1(\sigma), \tau|_{c_2}] \cdot \mathsf{crep}^{\Omega}_{c_3}[\pi_2(\sigma), \tau|_{c_3}].$$

In order to construct this schedule, we firstly construct the required configurations. Note that $\pi_0(\sigma) = \sigma$. Then $\pi_1(\sigma) = \tau|_{c_1}(\sigma)$, and this is the configuration from Fig. 8 lower left, i.e. $\boldsymbol{\kappa}[\ell_2] = 2$, $\boldsymbol{\kappa}[\ell_5] = 1$ and $\boldsymbol{\kappa}[\ell_1] = \boldsymbol{\kappa}[\ell_3] = \boldsymbol{\kappa}[\ell_4] = \boldsymbol{\kappa}[\ell_6] = 0$. Configuration $\pi_2(\sigma) = \tau|_{c_1} \cdot \tau|_{c_2}(\sigma) = \tau|_{c_2}(\pi_1(\sigma))$ is represented on Fig. 8 lower right, i.e. $\boldsymbol{\kappa}[\ell_5] = 3$ and all other counters are zero.

Section 5 deals with the construction of representatives of schedules that contain rules from only one looplet. Recall that construction of $\mathsf{crep}^{\Omega}_{c_1}[\pi_0(\sigma), \tau|_{c_1}]$ corresponds to the one from Example 5.2. Thus, we know that

$$\mathsf{crep}^{\Omega}_{c_1}[\pi_0(\sigma), \tau|_{c_1}] = (r_2, 0), (r_3, 1), (r_4, 2), (r_1, 2), (r_2, 0), (r_3, 0).$$

As c_2 is a singleton looplet, we use the result of Sect. 5.1. Thus,

$$\mathsf{crep}^{\Omega}_{c_2}[\pi_1(\sigma), \tau|_{c_2}] = (r_5, 2).$$

Using the result from Sect. 5.2 we obtain that

$$\mathsf{crep}^{\Omega}_{c_3}[\pi_2(\sigma), \tau|_{c_3}] = (r_8, 0), (r_7, 2),$$

and finaly we have the representative for τ that is

$$\mathsf{srep}_{\Omega}[\sigma, \tau] = (r_2, 0), (r_3, 1), (r_4, 2), (r_1, 2), (r_2, 0), (r_3, 0), (r_5, 2), (r_8, 0), (r_7, 2).$$

Lemma 6.1 (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 $\mathcal{R}|_{\Omega}$ with the property $c_i \prec^{lin}_{C} c_j$ 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 In the following, we show Points 1–3 one-by-one.

We need extra notation. For a local state ℓ we denote by $\mathbf{1}_\ell$ the $|\mathcal{L}|$-dimensional vector, where the ℓth component is 1, and all the other components are 0. Given a schedule $\rho = t_1 \cdots t_k$, we introduce a vector $\Delta_{\boldsymbol{\kappa}}(\rho) \in \mathbb{Z}^{|\mathcal{L}|}$ to keep counter difference and a vector $\Delta_{\mathbf{g}}(\rho) \in \mathbb{N}_0^{|\Gamma|}$ to keep difference on shared variables as follows:

$$\Delta_{\boldsymbol{\kappa}}(\rho) = \sum_{1 \le i \le |\rho|} t_i.\mathit{factor} \cdot (\mathbf{1}_{t_i.\mathit{to}} - \mathbf{1}_{t_i.\mathit{from}}) \quad \text{and} \quad \Delta_{\mathbf{g}}(\rho) = \sum_{1 \le i \le |\rho|} t_i.\mathbf{u}$$

Proof of (1) Assume by contradiction that schedule $\tau|_{c_1}$ is not applicable to configuration σ. Thus, there is a schedule τ' and a transition t^* that constitute a prefix of $\tau|_{c_1}$, with the following property: τ' is applicable to σ, whereas $\tau' \cdot t^*$ is not applicable to σ. Let $\ell = t^*.\mathit{from}$ and $\ell' = t^*.\mathit{to}$.

There are three cases of why t^* may be not applicable to $\tau'(\sigma)$:

(i) There is not enough processes to move: $(\sigma.\boldsymbol{\kappa} + \Delta_{\boldsymbol{\kappa}}(\tau' \cdot t^*))[\ell] < 0$. As τ is applicable to σ, there is a transition t of τ with $[t.\mathit{rule}] \neq c_1$ and $t.\mathit{to} = \ell$ as well as $t.\mathit{factor} > 0$. From this, by definition of $\prec_C^{\mathit{lin}}$, it follows that $[t.\mathit{rule}] \prec_C^{\mathit{lin}} c_1$. This contradicts the lemma's assumption on the order $c_1 \prec_C^{\mathit{lin}} \cdots \prec_C^{\mathit{lin}} c_m$.

(ii) The condition $t^*.\varphi^{\mathrm{rise}}$ is not satisfied, that is, $\tau'(\sigma) \not\models t^*.\varphi^{\mathrm{rise}}$. Then, there is a guard $\varphi \in \mathsf{guard}(t^*.\varphi^{\mathrm{rise}})$ with $\tau'(\sigma) \not\models \varphi$.

Since τ is applicable to σ, there is a prefix $\rho \cdot t$ of τ, for a schedule ρ and a transition t that unlocks φ in $\rho(\sigma)$, that is, $\rho(\sigma) \not\models \varphi$ and $t(\rho(\sigma)) \models \varphi$. Thus, transition t changes the context: $\omega(\rho(\sigma)) \neq \omega(t(\rho(\sigma)))$. This contradicts the assumption that schedule τ is steady.

(iii) The condition $t^*.\varphi^{\mathrm{fall}}$ is not satisfied: $\tau'(\sigma) \not\models t^*.\varphi^{\mathrm{fall}}$. Then, there is a guard $\varphi \in \mathsf{guard}(t^*.\varphi^{\mathrm{fall}})$ with $\tau'(\sigma) \not\models \varphi$.

Let ρ be the longest prefix of τ satisfying $\rho|_{c_1} = \tau'$. Note that $\rho \cdot t^*$ is also a prefix of τ. As $\rho|_{c_1} = \tau'$ and no transition decrements the shared variables, we conclude that $(\tau'(\sigma)).\mathbf{g} \le (\rho(\sigma)).\mathbf{g}$. From this and from the fact that $\tau'(\sigma) \not\models \varphi$, it follows that $\rho(\sigma) \not\models \varphi$. Thus transition t^* is not applicable to $\rho(\sigma)$. This contradicts the assumption that τ is applicable to σ.

From (i), (ii), and (iii), we conclude that *(1)* holds.

Proof of (2) We show that $\tau|_{c_2,\ldots,c_m}$ is applicable to $\tau|_{c_1}(\sigma)$.

To this end, we fix an arbitrary prefix τ' of τ, a transition t, and a suffix τ'', that constitute τ, that is, $\tau = \tau' \cdot t \cdot \tau''$. We show that if schedule $\tau'|_{c_2,\ldots,c_m}$ is applicable to $\tau|_{c_1}(\sigma)$, then so is $(\tau' \cdot t)|_{c_2,\ldots,c_m}$.

Let us assume that $\tau'|_{c_2,\ldots,c_m}$ is applicable to $\tau|_{c_1}(\sigma)$, and let σ'' denote the resulting state $(\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m})(\sigma)$. We consider two cases:

- $[t.\mathit{rule}] = c_1$. This case holds trivially, as $(\tau' \cdot t)|_{c_2,\ldots,c_m}$ equals to $\tau'|_{c_2,\ldots,c_m}$, which is applicable to $\tau|_{c_1}(\sigma)$ by assumption.
- $[t.\mathit{rule}] \neq c_1$. In order to prove that$(\tau' \cdot t)|_{c_2,\ldots,c_m}$ is applicable to $\tau|_{c_1}(\sigma)$, we show that counters $\sigma''.\boldsymbol{\kappa}$ and shared variables $\sigma''.\mathbf{g}$ are large enough, so that transition t is applicable to σ'':

(i) We start by showing that $\sigma''.\boldsymbol{\kappa}[t.\mathit{from}] \ge t.\mathit{factor}$. We distinguish between different cases on source and target states of transition t.

(i.A) We will show by contradiction that there is no rule $r \in c_1$ with $t.\mathit{to} = r.\mathit{from}$. Let's assume it exists. Then, on one hand, as $[t.\mathit{rule}] \neq c_1$, by definition of $\prec_C^{\mathit{lin}}$, it follows that $[t.\mathit{rule}] \prec_C^{\mathit{lin}} \ldots \prec_C^{\mathit{lin}} c_1$. On the other hand, as $[t.\mathit{rule}] \neq c_1$ and $c_1, \ldots, c_m$ are all classes of the rules used in τ, it holds that $[t.\mathit{rule}] \in \{c_2, \ldots, c_m\}$. By the lemma's

assumption, $c_1 \prec_C^{lin} \cdots \prec_C^{lin} c_m$, and thus, $c_1 \prec_C^{lin} \cdots \prec_C^{lin} [t.rule]$. We arrive at a contradiction.

(i.B) Let's consider the case of a rule $r \in c_1$ with $r.to = t.from$. Assume by contradiction that t is not applicable to σ'', that is, $\sigma''.\boldsymbol{\kappa}[t.from] < t.factor$. On one hand, transition t is not applicable to $\sigma'' = (\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m})(\sigma)$. Then by the definition of $\Delta_{\boldsymbol{\kappa}}$, it holds that $\sigma[t.from] + (\Delta_{\boldsymbol{\kappa}}(\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m}) + \Delta_{\boldsymbol{\kappa}}(t))[t.from] < 0$. By observing that $\tau|_{c_1} = \tau'|_{c_1} + \tau''|_{c_1}$, we derive the following inequality:

$$\sigma[t.from] + (\Delta_{\boldsymbol{\kappa}}(\tau'|_{c_1}) + \Delta_{\boldsymbol{\kappa}}(\tau''|_{c_1}) + \Delta_{\boldsymbol{\kappa}}(\tau'|_{c_2,\ldots,c_m}) + \Delta_{\boldsymbol{\kappa}}(t))[t.from] < 0 \quad (6.1)$$

On the other hand, schedule $\tau = \tau' \cdot t \cdot \tau''$ is applicable to configuration σ. Thus, $\sigma[t.from] + (\Delta_{\boldsymbol{\kappa}}(\tau') + \Delta_{\boldsymbol{\kappa}}(t) + \Delta_{\boldsymbol{\kappa}}(\tau''))[t.from] \geq 0$. By observing that $\tau|_{c_1} = \tau'|_{c_1} + \tau''|_{c_1}$ and $\tau|_{c_2,\ldots,c_m} = \tau'|_{c_2,\ldots,c_m} + \tau''|_{c_2,\ldots,c_m}$, we arrive at:

$$\sigma[t.from] + (\Delta_{\boldsymbol{\kappa}}(\tau'|_{c_1}) + \Delta_{\boldsymbol{\kappa}}(\tau'|_{c_2,\ldots,c_m}) + \Delta_{\boldsymbol{\kappa}}(t) + \Delta_{\boldsymbol{\kappa}}(\tau''|_{c_1}) + \Delta_{\boldsymbol{\kappa}}(\tau''|_{c_2,\ldots,c_m}))[t.from] \geq 0 \quad (6.2)$$

By subtracting (6.2) from (6.1), and by commutativity of vector addition, we arrive at $\Delta_{\boldsymbol{\kappa}}(\tau''|_{c_2,\ldots,c_m})[t.from] > 0$. Thus, there is a transition t' in $\tau''|_{c_2,\ldots,c_m}$ and a rule $r' \in c_1$ such that $t'.to = r'.from$. We again arrived at the contradictory Case (i.A). Hence, transition t must be applicable to configuration σ''.

(i.C) Otherwise, neither $t.from$ nor $t.to$ belong to the set of local states affected by the rules from c_1, i.e., $\{t.from, t.to\} \cap \{\ell \mid \exists r \in c_1.\ r.from = \ell \vee r.to = \ell\}$ is empty. Then, schedule $\tau|_{c_1}$ does not change the counter $\boldsymbol{\kappa}[t.from]$, and $\Delta_{\boldsymbol{\kappa}}(\tau')[t.from] = \Delta_{\boldsymbol{\kappa}}(\tau'|_{c_2,\ldots,c_m})[t.from]$. As t is applicable to $\tau'(\sigma)$, that is, $(\tau'(\sigma)).\boldsymbol{\kappa}[t.from] \geq t.factor$, we conclude that $\sigma''.\boldsymbol{\kappa}[t.from] \geq t.factor$.

(ii) We now show that $\sigma'' \models t.\varphi^{\text{rise}} \wedge t.\varphi^{\text{fall}}$. Assume by contradiction that $\sigma'' \not\models t.\varphi^{\text{rise}} \wedge t.\varphi^{\text{fall}}$. There are two cases to consider.

If $\sigma'' \not\models t.\varphi^{\text{rise}}$. By definition, the shared variables are never decremented in a non-singleton looplet. As τ' is a prefix of τ, schedule $\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m}$ includes all transitions of τ'. Thus, $\Delta_{\mathbf{g}}(\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m}) \geq \Delta_{\mathbf{g}}(\tau')$. From this and $\sigma'' \not\models t.\varphi^{\text{rise}}$, it follows that $\tau'(\sigma) \not\models t.\varphi^{\text{rise}}$. This contradicts applicability of τ to σ.

If $\sigma'' \not\models t.\varphi^{\text{fall}}$. Then, there is a guard $\varphi \in \mathsf{guard}(t.\varphi^{\text{fall}})$ with $\tau''(\sigma) \not\models \varphi$. On one hand, $\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m}$ is applicable to σ. On the other hand, τ is applicable to σ. We notice that $\Delta_{\mathbf{g}}(\tau) = \Delta_{\mathbf{g}}(\tau|_{c_1}) + \Delta_{\mathbf{g}}(\tau'|_{c_2,\ldots,c_m}) + \Delta_{\mathbf{g}}(\tau''|_{c_2,\ldots,c_m}) + \Delta_{\mathbf{g}}(t) \geq \Delta_{\mathbf{g}}(\tau|_{c_1}) + \Delta_{\mathbf{g}}(\tau'|_{c_2,\ldots,c_m})$. As shared variables are never decreased, it follows that $(\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m})(\sigma) \not\models \varphi$. Thus, $\omega(\sigma) \neq \omega(\tau(\sigma))$. This contradicts the assumption on that schedule τ is steady.

Having proved that, we conclude that transition t is applicable to configuration $(\tau|_{c_1} \cdot \tau'|_{c_2,\ldots,c_m})(\sigma)$. Thus, by induction $(\tau|_{c_1} \cdot \tau|_{c_2,\ldots,c_m})(\sigma)$ is applicable to σ. We conclude that Point *2* of the theorem holds.

Proof of (3) By the commutativity property of vector addition,

$$\Delta_{\boldsymbol{\kappa}}(\tau|_{c_1} \cdot \tau|_{c_2,\ldots,c_m}) = \Delta_{\boldsymbol{\kappa}}(\tau|_{c_1}) + \Delta_{\boldsymbol{\kappa}}(\tau|_{c_2,\ldots,c_m}) = \sum_{1 \leq i \leq |\tau|} \Delta_{\boldsymbol{\kappa}}(t_i) = \Delta_{\boldsymbol{\kappa}}(\tau).$$

Thus, $(\tau|_{c_1} \cdot \tau|_{c_2,\ldots,c_m})(\sigma) = \tau(\sigma)$, and Point (3) follows.

We have thus shown all three points of Lemma 6.1. □

Proof (*of Theorem* 6.1) By iteratively applying Lemma 6.1, we prove by induction that schedule $\tau|_{c_1} \cdot \ldots \cdot \tau|_{c_m}$ is applicable to σ and results in $\tau(\sigma)$. From Theorem 5.1, we conclude that each schedule $\tau|_{c_i}$ can be replaced by its representative $\mathsf{crep}^{\Omega}_{c_i}[\pi_{i-1}(\sigma), \tau|_{c_i}]$. Thus, $\mathsf{srep}_{\Omega}[\sigma, \tau]$ is applicable to σ and results in $\tau(\sigma)$. By Proposition 3.4, 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.

Theorem 6.2 *Fix a threshold automaton and a context Ω. Let $c_1, \ldots, c_m$ be the sorted sequence of all looplets of the slice $\mathcal{R}|_{\Omega}$, i.e., $c_1 \prec^{lin}_{C} \ldots \prec^{lin}_{C} c_m$. Schema $\mathsf{sschema}_{\Omega} = \mathsf{cschema}^{\Omega}_{c_1} \circ \cdots \circ \mathsf{cschema}^{\Omega}_{c_m}$ has two properties: (a) For a configuration σ with $\omega(\sigma) = \Omega$ and a steady schedule τ applicable to σ, $\mathsf{path}(\sigma, \tau')$ of the representative $\tau' = \mathsf{srep}_{\Omega}[\sigma, \tau]$ is generated by $\mathsf{sschema}_{\Omega}$; and (b) the length of $\mathsf{sschema}_{\Omega}$ is at most $2 \cdot |(\mathcal{R}|_{\Omega})|$.*

Proof Fix a configuration σ with $\omega(\sigma) = \Omega$ and a steady schedule τ applicable to σ. As $\mathsf{srep}_{\Omega}[\sigma, \tau]$ is a sorted sequence of the looplet representatives, all paths of $\mathsf{srep}_{\Omega}[\sigma, \tau]$ are generated by $\mathsf{sschema}_{\Omega}$, which is not longer than $2 \cdot |(\mathcal{R}|_{\Omega})|$. □

7 Proving the main result

Using the results from Sects. 5 and 6, for each configuration and each schedule (without restrictions) we construct a representative schedule.

Theorem 7.1 *Given a threshold automaton, a configuration σ, and a 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) $|\mathsf{rep}[\sigma, \tau]| \leq 2 \cdot |\mathcal{R}| \cdot (|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}| + 1) + |\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|$.

Proof Given a threshold automaton, fix a configuration σ and a schedule τ applicable to σ. Let $\Omega_1, \ldots, \Omega_{K+1}$ be the maximal monotonically increasing sequence of contexts such that $\mathsf{path}(\sigma, \tau)$ is consistent with the sequence by Definition 3.7. From Proposition 3.2, the length of the sequence is $K + 1 = |\Phi^{\text{rise}}| + |\Phi^{\text{fall}}| + 1$. Thus, there are at most K transitions $t^{\star}_1, \ldots, t^{\star}_K$ in τ that change their context, i.e., for $i \in \{1, \ldots, K\}$, it holds $\omega(\sigma_i) \sqsubset \omega(t^{\star}_i(\sigma_i))$ for $t^{\star}_i$'s respective state σ_i in τ. Therefore, we can divide τ into $K + 1$ steady schedules separated by the transitions $t^{\star}_1, \ldots, t^{\star}_K$:

$$\tau = \nu_1 \cdot t^{\star}_1 \cdot \nu_2 \cdots \nu_K \cdot t^{\star}_K \cdot \nu_{K+1}.$$

Now, the main idea is to replace the steady schedules with their representatives from Theorem 6.1. That is, using $t^{\star}_1, \ldots, t^{\star}_K$ and $\nu_1, \ldots, \nu_{K+1}$, we construct the schedules $\rho_1, \ldots, \rho_K$ (by convention, ρ_0 is the empty schedule):

$$\rho_i = \rho_{i-1} \cdot \nu_i \cdot t^{\star}_i \quad \text{for } 1 \leq i \leq K.$$

Finally, the representative schedule $\mathsf{rep}[\tau, \sigma]$ is constructed as follows:

$$\mathsf{rep}_{\Omega_1}[\sigma, \nu_1] \cdot t^{\star}_1 \cdot \mathsf{rep}_{\Omega_2}[\rho_1(\sigma), \nu_2] \cdots \mathsf{rep}_{\Omega_K}[\rho_{K-1}(\sigma), \nu_K] \cdot t^{\star}_K \cdot \mathsf{rep}_{\Omega_{K+1}}[\rho_K(\sigma), \nu_{K+1}]$$

From Theorem 6.1, it follows that $\mathsf{rep}[\tau, \sigma]$ is applicable to σ and it results in $\tau(\sigma)$. Moreover, the representative of a steady schedule is not longer than $2|\mathcal{R}|$, which together with K transitions gives us the bound $2|\mathcal{R}|(K+1)+K$. As we have that $K = |\Phi^{\text{rise}}|+|\Phi^{\text{fall}}|$, this gives us the required 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 7.2 *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* $3 \cdot |\mathcal{R}| \cdot (|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|) + 2 \cdot |\mathcal{R}|$.

Proof Given a threshold automaton, let ρ_{all} be the sequence $r_1, \ldots, r_{|\mathcal{R}|}$ of all rules from $\mathcal{R}$, and let $z = \Omega_0, \ldots, \Omega_m$ be a monotonically increasing sequence of contexts. By the construction in Theorem 7.1, 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 $\mathsf{sschema}_\Omega$ and $\{\Omega_i\}\, \rho_{\mathsf{all}}\, \{\Omega_{i+1}\}$. 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:

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

By inductively applying Theorem 6.2, 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 Theorem 7.1 and by noting that for every context, instead of one rule that is changing it, we add $|\mathcal{R}|$ extra rules. □

8 Complete set of schemas and optimizations

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 whether 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 s to compute the schemas in our longest experiment that runs for 4 h. In our tool we have implemented the following optimizations that lead to simpler and fewer SMT queries.

Entailment optimization We say that a guard $\varphi_1 \in \Phi^{\text{rise}}$ *entails* a guard $\varphi_2 \in \Phi^{\text{rise}}$, 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 $(\mathbf{g}, \mathbf{p}) \models \varphi_1 \rightarrow \varphi_2$. 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 $(\Omega^{\text{rise}}, \Omega^{\text{fall}})$ with $\varphi_1 \in \Omega^{\text{rise}}$ and $\varphi_2 \notin \Omega^{\text{rise}}$. 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^{fall}.

Control flow optimization Based on the proof of Lemma 6.1, we introduce the following optimization for TAs that are directed acyclic graphs (possibly with self loops). We say that a rule $r \in \mathcal{R}$ *may unlock* a guard $\varphi \in \Phi^{\text{rise}}$, if there is a $\mathbf{p} \in \mathbf{P}_{RC}$ and $\mathbf{g} \in \mathbb{N}_0^{|\Gamma|}$ satisfying: $(\mathbf{g}, \mathbf{p}) \models r.\varphi^{\text{rise}} \wedge r.\varphi^{\text{fall}}$ (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 from Fig. 2, the rule $r_1 \colon \mathit{true} \mapsto x_{++}$ may unlock the guard $\varphi_1 \colon x \geq \lceil (n+t)/2 \rceil - f$.

Let $\varphi \in \Phi^{\text{rise}}$ 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 $\psi_1, \ldots, \psi_n \in \Phi^{\text{rise}}$ 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 $\Phi^{\text{rise}} \setminus \{\varphi, \psi_1, \ldots, \psi_n\}$. 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 Sect. 9). 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^{fall}.

9 Checking a schema with SMT

We decompose a schema into a sequence of simple schemas, and encode the simple schemas. Given a simple schema $S = \{\Omega_1\}\, r_1, \ldots, r_m\, \{\Omega_2\}$, which contains m rules, we construct an SMT formula such that every model of the formula represents a path from $\mathcal{L}(S)$ — the language of paths generated by schema 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: Given the set of local states $\mathcal{L}$ and the set of shared variables Γ, a vector $\mathbf{k}^i = (k^i_1, \ldots, k^i_{|\mathcal{L}|})$ 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 $1 \leq i \leq m$.

Based on this we encode schemas, for which the sequence of rules $r_1, \ldots, r_m$ is fixed. We exploit this in two ways: First, we encode for each layer i the constraints of rule r_i. Second, as this constraint may update only two counters — the processes move from and move to according to the rule — we do not need $|\mathcal{L}|$ 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 $|\mathcal{L}| + |\Gamma| + m \cdot (2 + |\Gamma|)$ integer variables, $2m$ equations, and inequalities in linear integer arithmetic that represent threshold guards that evaluate to true (at most the number of threshold guards times m of these inequalities).

In the following, we use the notation $[k : m]$ to denote the set $\{k, \ldots, m\}$. In order to reuse the variables from the previous layers, we introduce a function $\upsilon : \mathcal{L} \times [0 : m] \to [0 : m]$ that for a layer $i \in [0 : m]$ and a local state $\ell \in \mathcal{L}$, gives the largest number $j \leq i$ of the layer, where the counter k^j_ℓ is updated:

$$\upsilon(\ell, i) = \begin{cases} i, & \text{if } i = 0 \vee \ell \in \{r_i.\mathit{from}, r_i.\mathit{to}\} \\ \upsilon(\ell, i-1), & \text{otherwise.} \end{cases}$$

Having defined layers, we encode: the effect of rules on counters and shared variables (in formulas M and U below), the effect of rules on the configuration (T), restrictions imposed by contexts (C), and, finally, the reachability question.

To represent m transitions, for each transition $i \in [1:m]$, we introduce a non-negative variable δ^i for the acceleration factor, and define two formulas: formula $M^\ell(i-1, i)$ to express the update of the counter of local state $\ell \in \mathcal{L}$, and formula $U^x(i-1, i)$ to represent the update of the shared variable $x \in \Gamma$:

$$M^\ell(i-1,i) \equiv \begin{cases} k_\ell^i = k_\ell^{\upsilon(\ell,i-1)} + \delta^i, & \text{for } \ell = r_i.\mathit{to} \text{ and } i \in [1:m] \\ k_\ell^i = k_\ell^{\upsilon(\ell,i-1)} - \delta^i, & \text{for } \ell = r_i.\mathit{from} \text{ and } i \in [1:m] \\ \mathit{true}, & \text{otherwise} \end{cases}$$

$$U^x(i-1,i) \equiv \begin{cases} x^i = x^{i-1} + \delta^i \cdot u, & \text{if } u = r_i.\mathbf{u}[j] > 0, \\ \mathit{true}, & \text{otherwise.} \end{cases}$$

The formula $T(i-1, i)$ collects all constraints by the rule r_i:

$$T(i-1,i) \equiv \bigwedge_{\ell \in \mathcal{L}} M^\ell(i-1,i) \wedge \bigwedge_{x \in \Gamma} U^x(i-1,i).$$

For a formula φ, we denote by $\varphi[\mathbf{x}^i]$ the formula, where each variable $x \in \Gamma$ is substituted with x^i. Then, given a context $\Omega = (\Omega^{\text{rise}}, \Omega^{\text{fall}})$, a formula $C^\Omega(i)$ adds the constraints of the context Ω on the layer i:

$$C_\Omega(i) \equiv \bigwedge_{\varphi \in \Omega^{\text{rise}}} \varphi[\mathbf{x}^i] \wedge \bigwedge_{\varphi \in \Phi^{\text{rise}} \setminus \Omega^{\text{rise}}} \neg\varphi[\mathbf{x}^i] \wedge \bigwedge_{\varphi \in \Omega^{\text{fall}}} \neg\varphi[\mathbf{x}^i] \wedge \bigwedge_{\varphi \in \Phi^{\text{fall}} \setminus \Omega^{\text{fall}}} \varphi[\mathbf{x}^i].$$

Finally, the formula $C_{\Omega_1}(0) \wedge T(0,1) \wedge \cdots \wedge T(m-1,m) \wedge C_{\Omega_2}(m)$ captures all the constraints of the schema $S = \{\Omega_1\}\, r_1, \ldots, r_m\, \{\Omega_2\}$, and thus, its models correspond to the paths of schedules that are generated by S.

Let $I(0)$ be the formula over the variables of layer i that captures the initial states of the threshold automaton, and $B(i)$ be a state property over the variables of layer i. Then, parameterized reachability for the schema S is encoded with the following formula in linear integer arithmetic:

$$I(0) \wedge C_{\Omega_1}(0) \wedge T(0,1) \wedge \cdots \wedge T(m-1,m) \wedge C_{\Omega_2}(m) \wedge \big(B(0) \vee \cdots \vee B(m)\big).$$

10 Experiments

We have extended our tool ByMC (Byzantine Model Checker [2]) with the technique discussed in this paper. All of our benchmark algorithms were originally published in pseudo-code, and we model them in a parametric extension of PROMELA, which was discussed in [27,34].

10.1 Benchmarks

We revisited several asynchronous FTDAs that were evaluated in [33,41]. In addition to these classic FTDAs, we considered asynchronous (Byzantine) consensus algorithms, namely,

BOSCO [57], C1CS [10], and CF1S [18], that are designed to work despite partial failure of the distributed system. In contrast to the conference version of this paper [39], we used a new version of the benchmarks from [37] that have been slightly updated for liveness properties. Hence, for some benchmarks, the running times of our tool may vary from [39]. The benchmarks, their source code in parametric of PROMELA, and the code of the threshold automata are freely available [30].

10.2 Implementation

ByMC supports several tool chains (shown in Fig. 1, p. 3), the first using counter abstraction (that is, process counters over an abstract domain), and the second using counter systems with counters over integers:

Data and counter abstractions In this chain, the message counters are first mapped to parametric intervals, e.g., counters range over the abstract domain $\hat{D} = \{[0, 1), [1, t + 1), [t + 1, n - t), [n - t, \infty)\}$. By doing so, we obtain a finite (data) abstraction of each process, and thus we can represent the system as a counter system: We maintain one counter $\kappa[\ell]$ per local state ℓ of a process, as well as the counters for the sent messages. Then, in the counter abstraction step, every process counter $\kappa[\ell]$ is mapped to the set of parametric intervals $\hat{D}$. As the abstractions may produce spurious counterexamples, we run them in an abstraction-refinement loop that incrementally prunes spurious transitions and unfair executions. More details on the data and counter abstractions and refinement can be found in [33]. In our experiments, we use two kinds of model checkers as backend:

1. *BDD* The counter abstraction is checked with nuXmv [11] using Binary Decision Diagrams (BDDs). For safety properties, the tool executes the command `check_invar`. In our experiments, we used the timeout of 3 days, as there was at least one benchmark that needed a bit more than a day to complete.
2. *BMC* The counter abstraction is checked with nuXmv using bounded model checking [6]. To ensure completeness (at the level of counter abstraction), we explore the computations of the length up to the diameter bounds that were obtained in [41]. To efficiently eliminate shallow spurious counterexamples, we first run the bounded model checker in the incremental mode up to length of 30. This is done by issuing the `nuXmv` command `check_ltlspec_sbmc_inc`, which uses the built-in SAT solver `MiniSAT`. Then, we run a single-shot SAT problem by issuing the `nuXmv` command `gen_ltlspec_sbmc` and checking the generated formula with the SAT solver `lingeling` [5]. In our experiments, we set the timeout to 1 day.

Reachability for threshold automata In this tool chain, to obtain a threshold automaton, our tool first applies data abstraction over the domain $\hat{D}$ to the PROMELA code, which abstracts the message counters that keep the number of messages received by every process, while the message counters for the sent messages are kept as integers. More details can be found in [40]. Having constructed a threshold automaton, we compare two verification approaches:

1. *PARA*^{2} *Bounded model checking with SMT* The approach of this article. BYMC enumerates the schemas (as explained in Sect. 4), encodes them in SMT (as explained in Sect. 9) and checks every schema with the SMT solver `Z3` [17].

2. *FAST Acceleration of counter automata* In this chain, our tool constructs a threshold automaton and checks the reachability properties with the existing tool FAST [3]. For comparison with our tool, we run FAST with the MONA plugin that produced the best results in our experiments.

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 [33] that adds to non-determinism.

Comparing to [39], in this paper, we have introduced an optimization to schema checking that dramatically reduced the running times for some of the benchmarks. In this optimization, we group schemas in a prefix tree, whose nodes are contexts and edges are simple schemas. In each node of the prefix tree, our tool checks, whether there are configurations that are reachable from the initial configurations by following the schemas in the prefix. If there are no such reachable configurations, we can safely prune the whole suffix and thus prove many schemas to be unsatisfiable at once.

10.3 Evaluation

Table 1 summarizes the features of threshold automata that are automatically constructed by ByMC from parametric PROMELA. The number of local states $|\mathcal{L}|$ varies from 7 (FRB and STRB) to hundreds (C1CS and CBC). Our threshold automata are obtained by applying interval abstraction to PROMELA code, which keeps track of the number of messages received by each process. Thus, the number $|\mathcal{L}|$ is proportional to the number of control states and $|\widehat{D}|^k$, where $\widehat{D}$ is the domain of parametric intervals (discussed above) and k is the number of message types. Sometimes, one can manually construct a more efficient threshold automaton that models the same fault-tolerant distributed algorithm and preserves the same safety properties. For instance, Fig. 2 shows a manual abstraction of ABA that has only 5 local states, in contrast to 61 local states in the automatic abstraction (cf. Table 1). We leave open the question of whether one can automatically construct a minimal threshold automaton with respect to given specifications.

Table 2 summarizes our experiments conducted with the techniques introduced in Sect. 10.2: BDD, BMC, PARA^2, and FAST. On large problems, our new technique works significantly better than BDD- and SAT-based model checking. BDD-based model checking works very well on top of counter abstraction. Importantly, our new technique does not use abstraction refinement. In comparison to our earlier experiments [39], we verified safety of a larger set of benchmarks with nuXmv. We believe that this is due to the improvements in nuXmv and, probably, slight modifications of the benchmarks from [37].

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

Table 1 The benchmarks used in our experiments. Some benchmarks, e.g., ABA, require us to consider several cases on the parameters, which are mentioned in the column "Case". The meaning of the other columns is as follows: $|\mathcal{L}|$ is the number of local states in TA, $|\mathcal{R}|$ is the number of rules in TA, $|\Phi^{\text{rise}}|$ and $|\Phi^{\text{fall}}|$ is the number of (R)- and (F)-guards respectively. Finally, $|\mathcal{S}|$ is the number of enumerated schemas, and Bound is the theoretical upper bound on $|\mathcal{S}|$, as given in Theorem 4.2

#	Input	Case	Threshold Automaton				Schemas	
	FTDA	(if more than one)	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\Phi^{\text{rise}}\rvert$	$\lvert\Phi^{\text{fall}}\rvert$	$\lvert\mathcal{S}\rvert$	Theor. Bound
1	**FRB**	—	7	10	1	0	1	1
2	**STRB**	—	7	15	3	0	4	6
3	**NBACC**	—	78	1356	0	0	1	1
4	**NBAC**	—	77	988	6	0	448	720
5	**NBACG**	—	24	44	4	0	14	24
6	**CF1S**	$f = 0$	41	266	4	0	14	24
7	**CF1S**	$f = 1$	41	266	4	1	60	120
8	**CF1S**	$f > 1$	68	672	6	1	3429	5040
9	**C1CS**	$f = 0$	101	1254	8	0	70	$4 \cdot 10^4$
10	**C1CS**	$f = 1$	70	629	6	1	140	5040
11	**C1CS**	$f > 1$	101	1298	8	1	630	$3.6 \cdot 10^5$
12	**BOSCO**	$\lfloor\frac{n+3t}{2}\rfloor + 1 = n - t$	28	126	6	0	20	720
13	**BOSCO**	$\lfloor\frac{n+3t}{2}\rfloor + 1 > n - t$	40	204	8	0	70	$4 \cdot 10^4$
14	**BOSCO**	$\lfloor\frac{n+3t}{2}\rfloor + 1 < n - t$	32	158	6	0	20	720
15	**BOSCO**	$n > 5t \wedge f = 0$	82	1292	12	0	924	$4.8 \cdot 10^8$
16	**BOSCO**	$n > 7t$	90	1656	12	0	924	$4.8 \cdot 10^8$
17	**ABA**	$\frac{n+t}{2} = 2t + 1$	37	180	6	0	448	720
18	**ABA**	$\frac{n+t}{2} > 2t + 1$	61	392	8	0	2100	$4 \cdot 10^4$
19	**CBC**	$\lfloor\frac{n}{2}\rfloor < n - t \wedge f = 0$	164	1996	22	12	2	$2.9 \cdot 10^{38}$
20	**CBC**	$\lfloor\frac{n}{2}\rfloor = n - t \wedge f = 0$	73	442	17	12	2	$8.8 \cdot 10^{30}$
21	**CBC**	$\lfloor\frac{n}{2}\rfloor < n - t \wedge f > 0$	304	6799	27	12	5	$2 \cdot 10^{46}$
22	**CBC**	$\lfloor\frac{n}{2}\rfloor = n - t \wedge f > 0$	161	2040	22	12	5	$2.9 \cdot 10^{38}$

us concrete values of the parameters and shows how many processes move at each step of the counterexample.

Our new method uses integer counters and thus does not introduce spurious behavior due to counter abstraction, but still has spurious behavior due to data abstraction on complex FTDAs such as BOSCO, C1CS, and NBAC. In these cases, we manually refine the interval domain by adding new symbolic interval borders, see [33]. We believe that these intervals can be obtained directly from threshold automata, and no refinement is necessary. We leave this question to future work.

Sets of schemas and time to check a single schema On one hand, Theorem 4.2 gives us a theoretical bound on the number of schemas to be explored. On the other hand, optimizations discussed in Sect. 8 introduce many ways of reducing the number of schemas. Two columns in Table 1 compare the theoretical bound and the practical number of schemas: the column "Theoretical bound" shows the bound of $(|\Phi^{\text{rise}}| + |\Phi^{\text{fall}}|)!$, while the column $|\mathcal{S}|$ shows the actual number of schemas. (For reachability, we are merging the schemas with the prefix tree, and thus the actual number of explored schemas is even smaller.) As one can see, the theoretical bound is quite pessimistic, and is only useful to show completeness of the set of

Table 2 Summary of our experiments on AMD Opteron® 6272, 32 cores, 192 GB. The symbols are: "🕒" for timeout (72 h. for BDD and 24 h. otherwise); "💣" for memory overrun of 32 GB; "⚠" for BDD nodes overrun; "🔄" for timeout in the refinement loop (72 h. for BDD and 24 h. otherwise); "☹" for spurious counterexamples due to counter abstraction

	Input	Time, seconds				Memory, GB			
#	FTDA	PARA^2	FAST	BMC	BDD	PARA^2	FAST	BMC	BDD
1	FRB	1	1	1	1	0.1	0.1	0.1	0.1
2	STRB	1	1	3	2	0.1	0.1	0.1	0.1
3	NBACC	13	💣	🕒	🕒	0.1	💣	🕒	🕒
4	NBAC	88	💣	🔄	🕒	0.1	💣	🔄	🕒
5	NBACG	1	⚠	☹	🔄	0.1	⚠	☹	🔄
6	CF1S	6	2227	723	122	0.1	10.7	1.5	0.2
7	CF1S	11	6510	2235	2643	0.1	22.1	2.0	0.4
8	CF1S	263	⚠	🕒	40451	0.3	⚠	🕒	1.9
9	C1CS	45	💣	💣	10071	0.1	💣	💣	2.5
10	C1CS	21	💣	94962	87141	0.1	💣		9.3
11	C1CS	171	💣	💣	🕒	0.3	💣	💣	🕒
12	BOSCO	3	⚠	17892	294	0.1	⚠	1.4	0.2
13	BOSCO	17	⚠	🔄	☹	0.1	⚠	🔄	☹
14	BOSCO	5	⚠	2424	4	0.1	⚠	1.9	0.1
15	BOSCO	1013	💣	🕒	405	0.2	💣	🕒	0.7
16	BOSCO	1459	⚠	🕒	847	0.4	⚠	🕒	1.3
17	ABA	16	767	☹	11	0.1	3.5	☹	0.1
18	ABA	294	5757	☹	41	0.3	12.4	☹	0.2
19	CBC	128	💣	💣	💣	0.6	💣	💣	💣
20	CBC	9	⚠	2671	41873	0.1	⚠	2.8	9.9
21	CBC	3351	3304	💣	💣	19.3	0.1	💣	💣
22	CBC	215	⚠	💣	💣	4.0	⚠	💣	💣

schemas. The much smaller numbers for the fault-tolerant distributed algorithms are due to a natural order on guards, e.g., as $x \geq t + 1$ becomes true earlier than $x \geq n - t$ under the resilience condition $n > 3t$. The drastic reduction in the case of CBC is due to the control flow optimization discussed in Sect. 8 and the fact that basically all guards are forward-unlocking.

When doing experiments, we noticed that the only kinds of guards that cannot be treated by our optimizations and blow up the number of schemas are the guards that use independent shared variables. For instance, consider the guards $x_0 \geq n - t$ and $x_1 \geq n - t$ that are counting the number of 0's and 1's sent by the correct processes. Even though they are mutually exlusive under the resilience condition $n > 3t$, our tool has to explore all possible orderings of these guards. We are not aware of a reduction that would prevent our method from exploding in the number of schemas for this example.

Since the schemas can be checked independently, one can check them in parallel. Figure 9 shows a distribution of schemas along with the time needed to check an individual schema. There are only a few divergent schemas that required more than 7 s to get checked, while the large portion of schemas require 1–3 s. Hence, a parallel implementation of the tool should verify the algorithms significantly faster. We leave such a parallel extension for future work.

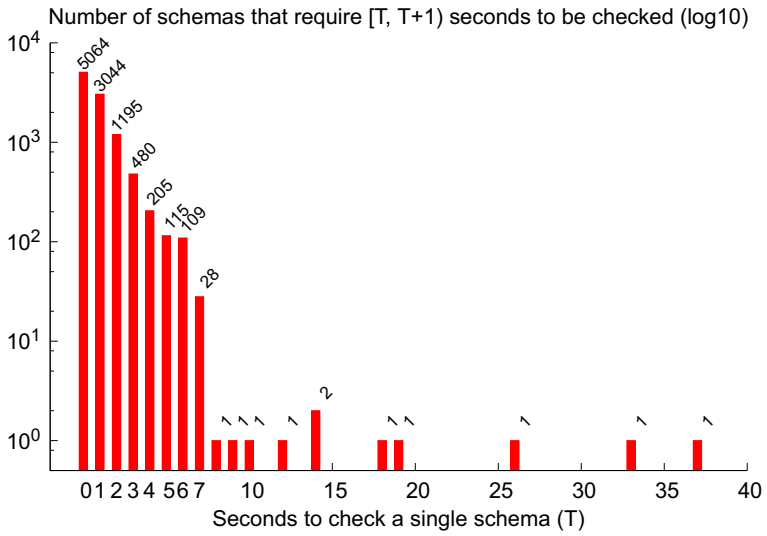

Fig. 9 The times required to check individual schemas and the distribution of schemas over these times (the value 0 refers to the running times of less than a second). The benchmarks containing the schemas that are verified in (a) $T \geq 8$ sec. and (b) $T \geq 18$ sec. are: (a) C1CS, CBC, CF1S, and (b) CBC and CF1S

11 Discussions and related work

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 [14,15,21–23,35]; cf. [7] for a survey. However, the FTDAs considered by us run under the different assumptions.

From a methodological viewpoint, our approach combines techniques from several areas including compact programs [49], counter abstraction [4,55], completeness thresholds for bounded model checking [6,16,42], partial order reduction [8,28,53,59], and Lipton's movers [48]. Regarding counter automata, our result entails *flattability* [46] of every counter system of threshold automata: a complete set of schemas immediately gives us a flat counter automaton. Hence, the acceleration-based semi-algorithms [3,46] should in principle terminate on the systems of TAs, though it did not always happen in our experiments. Similar to our SMT queries based on schemas, the *inductive data flow graphs* iDFG introduced in [24] are a succinct representations of schedules (they call them traces) for systems where the number of processes (or threads) is fixed. The work presented in [25] then considers parameterized verification. Further, our execution schemas are inspired by a general notion of *semi-linear path schemas* SLPS [45,46]. We construct a small complete set of schemas and thus a provably small SLPS. Besides, 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., $\{0, 1, \mathit{many}\}$ [55].

Many distributed algorithms can be represented with I/O Automata [50] or TLA+ [44]. 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 [32].

The results presented in this article can be used to check reachability properties of FTDAs. We can thus establish safety of FTDAs. However, for fault-tolerant distributed algorithms liveness is as important as safety: The seminal impossibility result by Fischer, Lynch, and Paterson [26] states that a fault-tolerant consensus algorithm cannot ensure both safety and liveness in asynchronous systems. In recent work [37] we also considered liveness verification, or more precisely, verification of temporal logic specification with the G and F temporal operators. In [37], we use the results of this article as a black box and show that combinations of schemas can be used to generate counterexamples to liveness properties, and that we can verify both safety and liveness by complete SMT-based bounded model checking.

Acknowledgements Open access funding provided by Austrian Science Fund (FWF). We are grateful to Azadeh Farzan for valuable discussions during her stay in Vienna and to the anonymous reviewers for their insightful comments regarding partial order reduction, and for suggestions that helped us in improving the presentation of the paper.

References

1. Attiya H, Welch J (2004) Distributed computing, 2nd edn. Wiley, New York
2. ByMC: Byzantine model checker (2013). http://forsyte.tuwien.ac.at/software/bymc/. Accessed Dec 2016
3. Bardin S, Finkel A, Leroux J, Petrucci L (2008) Fast: acceleration from theory to practice. STTT 10(5):401–424
4. Basler G, Mazzucchi M, Wahl T, Kroening D (2009) Symbolic counter abstraction for concurrent software. In: CAV. LNCS, vol 5643, pp 64–78
5. Biere A (2013) Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In: Proceedings of SAT competition 2013; Solver and p. 51
6. Biere A, Cimatti A, Clarke EM, Zhu Y (1999) Symbolic model checking without BDDs. In: TACAS. LNCS, vol 1579, pp 193–207
7. Bloem R, Jacobs S, Khalimov A, Konnov I, Rubin S, Veith H, Widder J (2015) Decidability of parameterized verification, synthesis lectures on distributed computing theory. Morgan & Claypool, San Rafael
8. Bokor P, Kinder J, Serafini M, Suri N (2011) Efficient model checking of fault-tolerant distributed protocols. In: DSN, pp 73–84
9. Bracha G, Toueg S (1985) Asynchronous consensus and broadcast protocols. J ACM 32(4):824–840
10. Brasileiro FV, Greve F, Mostéfaoui A, Raynal M (2001) Consensus in one communication step. In: PaCT. LNCS, vol 2127, pp 42–50
11. Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, Mover S, Roveri M, Tonetta S (2014) The nuXmv symbolic model checker. In: CAV. LNCS, vol 8559, pp 334–342
12. Chandra TD, Toueg S (1996) Unreliable failure detectors for reliable distributed systems. J ACM 43(2):225–267
13. Clarke E, Grumberg O, Jha S, Lu Y, Veith H (2003) Counterexample-guided abstraction refinement for symbolic model checking. J ACM 50(5):752–794
14. Clarke E, Talupur M, Touili T, Veith H (2004) Verification by network decomposition. In: CONCUR 2004, vol 3170, pp 276–291
15. Clarke E, Talupur M, Veith H (2008) Proving Ptolemy right: the environment abstraction framework for model checking concurrent systems. In: TACAS'08/ETAPS'08. Springer, Berlin, pp 33–47
16. Clarke EM, Kroening D, Ouaknine J, Strichman O (2004) Completeness and complexity of bounded model checking. In: VMCAI. LNCS, vol 2937, pp 85–96
17. De Moura L, Bjørner N (2008) Z3: an efficient SMT solver. In: Tools and algorithms for the construction and analysis of systems. LNCS, vol 1579, pp 337–340

18. Dobre D, Suri N (2006) One-step consensus with zero-degradation. In: DSN, pp 137–146
19. Drăgoi C, Henzinger TA, Zufferey D (2016) PSync: a partially synchronous language for fault-tolerant distributed algorithms. In: POPL, pp 400–415
20. Drăgoi C, Henzinger TA, Veith H, Widder J, Zufferey D (2014) A logic-based framework for verifying consensus algorithms. In: VMCAI. LNCS, vol 8318, pp 161–181
21. Emerson E, Namjoshi K (1995) Reasoning about rings. In: POPL, pp 85–94
22. Emerson EA, Kahlon V (2003) Model checking guarded protocols. In: LICS. IEEE, pp 361–370
23. Esparza J, Ganty P, Majumdar R (2013) Parameterized verification of asynchronous shared-memory systems. In: CAV, pp 124–140
24. Farzan A, Kincaid Z, Podelski A (2013) Inductive data flow graphs. In: POPL, pp 129–142
25. Farzan A, Kincaid Z, Podelski A (2015) Proof spaces for unbounded parallelism. In: POPL, pp 407–420
26. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382
27. Gmeiner A, Konnov I, Schmid U, Veith H, Widder J (2014) Tutorial on parameterized model checking of fault-tolerant distributed algorithms. In: SFM. LNCS, vol 8483. Springer, Berlin, pp 122–171
28. Godefroid P (1990) Using partial orders to improve automatic verification methods. In: CAV. LNCS, vol 531, pp 176–185
29. Guerraoui R (2002) Non-blocking atomic commit in asynchronous distributed systems with failure detectors. Distrib Comput 15(1):17–25
30. https://github.com/konnov/fault-tolerant-benchmarks/tree/master/fmsd17
31. Hawblitzel C, Howell J, Kapritsos M, Lorch JR, Parno B, Roberts ML, Setty STV, Zill B (2015) Ironfleet: proving practical distributed systems correct. In: SOSP, pp 1–17
32. Jensen H, Lynch N (1998) A proof of Burns n-process mutual exclusion algorithm using abstraction. In: Steffen B (ed) TACAS. LNCS, vol 1384. Springer, Berlin, pp 409–423
33. John A, Konnov I, Schmid U, Veith H, Widder J (2013) Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In: FMCAD, pp 201–209
34. John A, Konnov I, Schmid U, Veith H, Widder J (2013) Towards modeling and model checking fault-tolerant distributed algorithms. In: SPIN. LNCS, vol 7976, pp 209–226
35. Kaiser A, Kroening D, Wahl T (2012) Efficient coverability analysis by proof minimization. In: CONCUR, pp 500–515
36. Kesten Y, Pnueli A (2000) Control and data abstraction: the cornerstones of practical formal verification. STTT 2:328–342
37. Konnov I, Lazić M, Veith H, Widder J (2017) A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In: POPL, pp 719–734
38. Konnov I, Veith H, Widder J (2014) On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. In: CONCUR. LNCS, vol 8704, pp 125–140
39. Konnov I, Veith H, Widder J (2015) SMT and POR beat counter abstraction: parameterized model checking of threshold-based distributed algorithms. In: CAV (Part I). LNCS, vol 9206, pp 85–102
40. Konnov I, Veith H, Widder J (2016) What you always wanted to know about model checking of fault-tolerant distributed algorithms. In: PSI 2015, revised selected papers. LNCS, vol 9609. Springer, pp 6–21
41. Konnov I, Veith H, Widder J (2017) On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. Inf Comput 252:95–109
42. Kroening D, Strichman O (2003) Efficient computation of recurrence diameters. In: VMCAI. LNCS, vol 2575, pp 298–309
43. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565
44. Lamport L (2002) Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co. Inc, Boston
45. Leroux J, Sutre G (2004) On flatness for 2-dimensional vector addition systems with states. In: CONCUR 2004-concurrency theory. Springer, pp 402–416
46. Leroux J, Sutre G (2005) Flat counter automata almost everywhere! In: ATVA. LNCS, vol 3707, pp 489–503
47. Lesani M, Bell CJ, Chlipala A (2016) Chapar: certified causally consistent distributed key-value stores. In: POPL, pp 357–370
48. Lipton RJ (1975) Reduction: a method of proving properties of parallel programs. Commun ACM 18(12):717–721
49. Lubachevsky BD (1984) An approach to automating the verification of compact parallel coordination programs. I. Acta Inform 21(2):125–169
50. Lynch N (1996) Distributed algorithms. Morgan Kaufman, Burlington

51. Mostéfaoui A, Mourgaya E, Parvédy PR, Raynal M (2003) Evaluating the condition-based approach to solve consensus. In: DSN, pp 541–550
52. Padon O, McMillan KL, Panda A, Sagiv M, Shoham S (2016) Ivy: safety verification by interactive generalization. In: PLDI, pp 614–630
53. Peled D (1993) All from one, one for all: on model checking using representatives. In: CAV. LNCS, vol 697, pp 409–423
54. Peluso S, Turcu A, Palmieri R, Losa G, Ravindran B (2016) Making fast consensus generally faster. In: DSN, pp 156–167
55. Pnueli A, Xu J, Zuck L (2002) Liveness with (0,1,∞)-counter abstraction. In: CAV. LNCS, vol 2404, pp 93–111
56. Raynal M (1997) A case study of agreement problems in distributed systems: non-blocking atomic commitment. In: HASE, pp 209–214
57. Song YJ, van Renesse R (2008) Bosco: one-step Byzantine asynchronous consensus. In: DISC. LNCS, vol 5218, pp 438–450
58. Srikanth T, Toueg S (1987) Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib Comput 2:80–94
59. Valmari A (1991) Stubborn sets for reduced state space generation. In: Advances in Petri Nets 1990. LNCS, vol 483. Springer, pp 491–515
60. Wilcox JR, Woos D, Panchekha P, Tatlock Z, Wang X, Ernst MD, Anderson TE (2015) Verdi: a framework for implementing and formally verifying distributed systems. In: PLDI, pp 357–368