



# Verification of randomized consensus algorithms under round-rigid adversaries

Nathalie Bertrand<sup>1</sup> · Igor Konnov<sup>2</sup> · Marijana Lazić<sup>3</sup> · Josef Widder<sup>2</sup>

Accepted: 17 December 2020 / Published online: 2 February 2021  
© Springer-Verlag GmbH Germany, part of Springer Nature 2021, corrected publication 2021

## Abstract

Randomized fault-tolerant distributed algorithms pose a number of challenges for automated verification: (i) parameterization in the number of processes and faults, (ii) randomized choices and probabilistic properties, and (iii) an unbounded number of asynchronous rounds. This combination makes verification hard. Challenge (i) was recently addressed in the framework of threshold automata. We extend threshold automata to model randomized consensus algorithms that perform an unbounded number of asynchronous rounds. For non-probabilistic properties, we show that it is necessary and sufficient to verify these properties under round-rigid schedules, that is, schedules where processes enter round  $r$  only after all processes finished round  $r - 1$ . For almost-sure termination, we analyze these algorithms under round-rigid adversaries, that is, fair adversaries that only generate round-rigid schedules. This allows us to do compositional and inductive reasoning that reduces verification of the asynchronous multi-round algorithms to model checking of a one-round threshold automaton. We apply this framework and automatically verify the following classic algorithms: Ben-Or's and Bracha's seminal consensus algorithms for crashes and Byzantine faults, 2-set agreement for crash faults, and RS-Bosco for the Byzantine case.

**Keywords** Verification · Distributed algorithms · Fault tolerance · Probabilistic · Parameterized

---

Supported by Interchain Foundation, Switzerland; by the Austrian Science Fund (FWF) via the National Research Network RiSE (S11403, S11405), project PRAVDA (P27722), and Doctoral College LogICS (W1255-N23); by the Vienna Science and Technology Fund (WWTF) via project APALACHE (ICT15-103); and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Under Grant Agreement No. 787367 (PaVeS). Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and others, see [www.grid5000.fr](http://www.grid5000.fr).

---

This is an extended version of Bertrand et al. [7], which appeared in the proceedings of CONCUR 2019. In the conference version, we did not provide proofs. As a result, also the definitions were just sketched and we omitted preliminary lemmas. In this paper, we completely develop our theory. Moreover, we give more detailed discussions and explanation with new figures and diagrams, we add and extend examples.

---

✉ Marijana Lazić  
lazic@in.tum.de

Igor Konnov  
igor@informal.systems

## 1 Introduction

Fault-tolerant distributed systems such as Blockchain or Paxos recently received much attention. Still, these systems are out of reach with current automated verification techniques. One problem comes from the scale: These systems should be verified for a very large (ideally even an unbounded) number of participants. In addition, many systems (including Blockchain) provide probabilistic guarantees. To check their correctness, one has to reason about their behavior in a probabilistic setting. We take a step toward this direction and consider the verification of randomized distributed algorithms in the parameterized setting.

In this paper, we make first steps toward parameterized verification of fault-tolerant randomized distributed algorithms. We consider consensus algorithms that follow the ideas of Ben-Or [4]. Interestingly, these algorithms were

<sup>1</sup> University Rennes, Inria, CNRS, IRISA, Rennes, France

<sup>2</sup> Informal Systems, Vienna, Austria

<sup>3</sup> TU München, Munich, Germany

analyzed in [29,31] where probabilistic reasoning was done using the probabilistic model checker PRISM [30] for systems consisting of 10–20 processes, while only safety was verified in the parameterized setting using Cadence SMV. From a different perspective, these algorithms extend asynchronous threshold-guarded distributed algorithms from Konnov et al. [24,25] with two features (i) a random choice (coin toss), and (ii) repeated executions of the same algorithm until it converges (with probability 1).

A prominent example is Ben-Or’s fault-tolerant consensus algorithm [4] given in Fig. 1. It circumvents the impossibility of asynchronous consensus [19] by relaxing the termination requirement to almost-sure termination, i.e., termination with probability 1. Here processes execute an infinite sequence of asynchronous loop iterations, which are called rounds  $r$ . Each round consists of two stages where they first exchange messages tagged  $R$ , wait until the number of received messages reaches a certain threshold (given as expression over parameters in line 5) and then exchange messages tagged  $P$ . In the code,  $n$  is the number of processes, among which at most  $t$  are Byzantine faulty (which may send conflicting information).

Figure 2 shows an example execution of Ben-Or’s algorithm in the distributed environment of six processes, one of them being Byzantine. We only depict the time line of the first couple of steps for the five correct processes. Time moved downwards, and each line roughly corresponds to a time step. In each line, the statements of that line are executed. When we write, e.g., “ $\geq n - t$  messages ( $R, 1, *$ )”, we mean that the process has received at least  $n - t$  messages that match the expression (possibly including a message from itself), and the corresponding guard in the code evaluates to true. As the processes are executed asynchronously, they may receive messages in different orders. In our example, processes 1 and 2 receive two messages of type  $(P, 1, 0, D)$ , whereas processes 3 and 4 receive only one message of type  $(P, 1, 0, D)$ . As a result, the processes follow different control flows of the algorithm. Nevertheless, the correct processes decide in the end of the second round. Also observe that due to asynchrony, the processes may take steps at different times. As a result, process 1 sets  $r$  to 2 and thus enters the second round before process 5 has started its first round. Thus, at the same time, processes may be in different rounds.

The algorithm is designed to satisfy the following three properties:

*Agreement:* no two correct processes decide on different values.

*Validity:* if all correct processes have  $v$  as the initial value, then no process decides  $1 - v$ .

*Probabilistic wait-free termination:* with probability 1, every correct process eventually decides.

The correctness of the algorithm should be verified for all values of the parameters  $n$  and  $t$  that meet a so-called resilience condition, e.g.,  $n > 5t$ . Carefully chosen thresholds (namely  $n - t$ ,  $(n + t)/2$ , and  $t + 1$ ) on the number of received messages of a given type ensure agreement. At the end of a round, if there is no “strong majority” for a value, i.e., less than  $(n + t)/2$  messages were received (cf. line 13), a process picks a new value randomly in line 16. Observe that if a process decides in line 14, it nevertheless continues to execute the algorithm for the rounds to follow.

While these non-trivial threshold expressions can be dealt with using the methods in [24], several challenges remain. The technique in [24] can be used to verify one iteration of the round from Fig. 1 only. However, consensus algorithms should prevent that there are no two rounds  $r$  and  $r'$  such that a process decides 0 in  $r$  and another decides 1 in  $r'$ . This calls for a compositional approach that allows one to compose verification results for individual rounds. A challenge in the composition is that distributed algorithms implement “asynchronous rounds”, i.e., during a run, processes may be in different rounds at the same time.

The combination of distributed aspects and probabilities makes reasoning difficult. Quoting Lehmann and Rabin [33], “proofs of correctness for probabilistic distributed systems are extremely slippery”. This advocates the development of automated verification techniques for probabilistic properties of randomized distributed algorithms in the parameterized setting.

**Contributions.** We extend the framework of threshold automata [24] to round-based algorithms with coin-toss transitions. For the new framework, we achieve the following:

1. For safety verification, we introduce a method for compositional round-based reasoning. This allows us to invoke a reduction similar to the one in [12,15,17]. We highlight necessary fairness conditions on individual rounds. This provides us with specifications to be checked on a one-round automaton.
2. We reduce probabilistic liveness verification to proving termination with positive probability within a fixed number of rounds. To do so, we restrict ourselves to round-rigid adversaries, that is, adversaries that respect the round ordering. In contrast to existing work that proves almost-sure termination for fixed number of participants [29,31], these are the first parameterized model checking results for probabilistic properties.
3. Using the tool ByMC [24,26], we automatically check the specifications that we derive in Points 1. and 2. and thus verify challenging benchmarks in the parameterized setting. We verify Ben-Or’s [4] and Bracha’s [11] classic algorithms, and more recent algorithms such as 2-set agreement [38], and RS-Bosco [42].

```

1  bool v := input_value({0, 1});
2  int r := 1;
3  while (true) do
4    send (R,r,v) to all;
5    wait for n - t messages (R,r,*);
6    if received at least (n + t) / 2 messages (R,r,w)
7    then send (P,r,w,D) to all;
8    else send (P,r,?) to all;
9    wait for n - t messages (P,r,*);
10   if received at least t + 1 messages (P,r,w,D)
11   then {
12     v := w;
13     if received at least (n + t) / 2 messages (P,r,w,D)
14     then decide w;
15   }
16   else v := random({0, 1});
17   r := r + 1;
18 od
    
```

Fig. 1 Pseudocode of Ben-Or’s algorithm for Byzantine faults

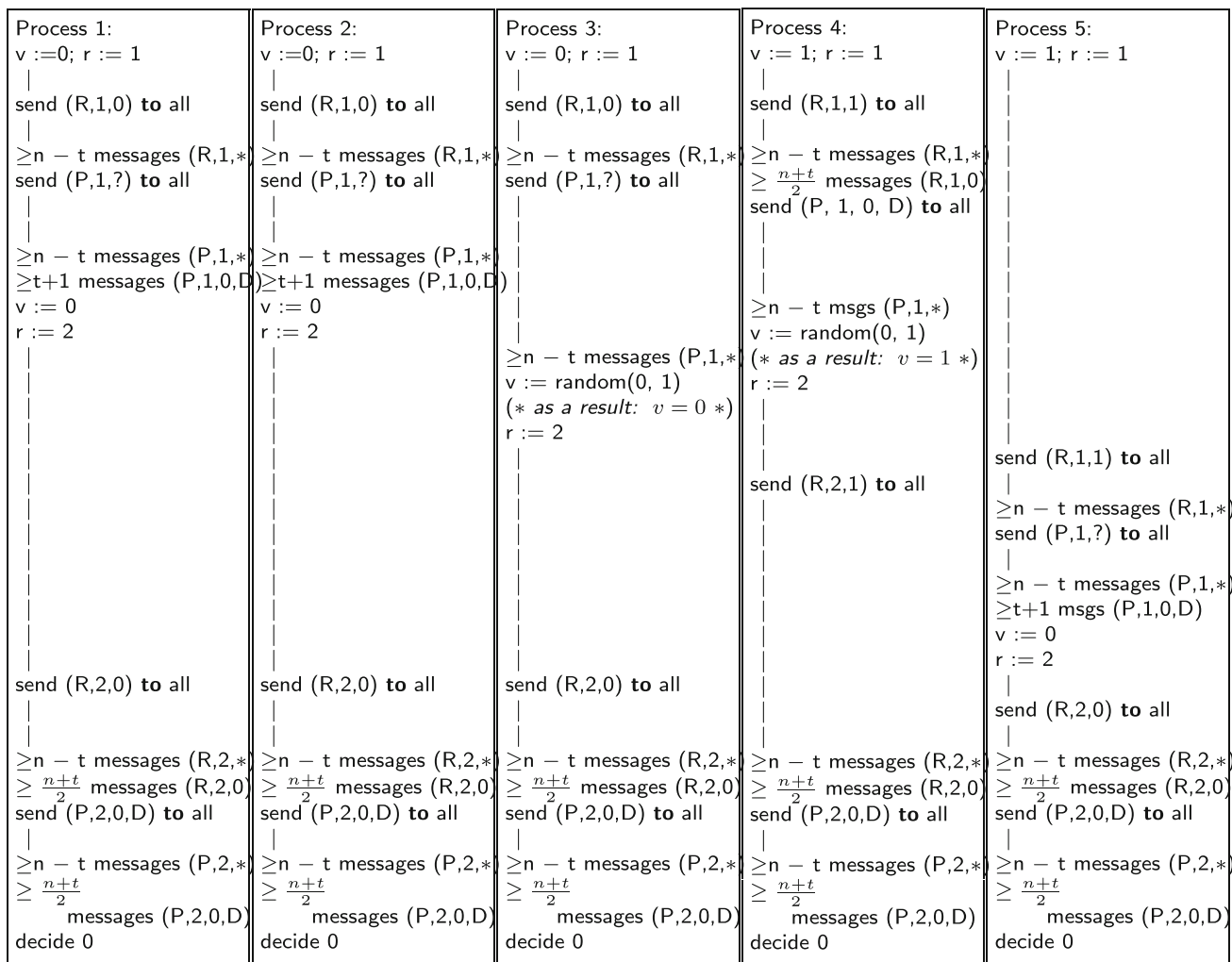


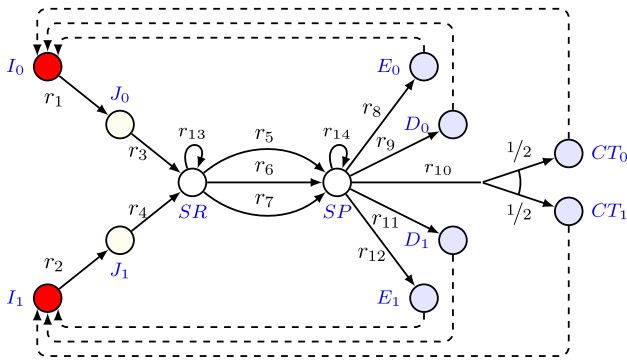
Fig. 2 Distributed execution of Ben-Or’s algorithm for 5 correct and 1 faulty process (n = 6, t = 1, f = 1). The figure shows the code that is executed by the correct processes

## 2 Overview

### 2.1 Modeling randomized threshold-based algorithms

We introduce probabilistic threshold automata for the modeling of randomized threshold-based algorithms. An example of such an automaton is given in Fig. 3. Nodes represent local

states of processes, which move along the labeled edges or forks. Local states are called locations, while edges and forks are called rules. The automaton rules are given in Table 1. When a rule is annotated with a guard  $\varphi$  and an update  $u$ , a process can move along the edge only if  $\varphi$  evaluates to true, and this is followed by the update  $u$  of shared variables. Additionally, each tine of a fork is labeled with a number in the [0, 1] interval, representing the probability of a process



**Fig. 3** Ben-Or’s algorithm as a probabilistic threshold automaton with resilience condition  $n > 3t \wedge t \geq f \geq 0 \wedge t > 0$

moving along the fork to end up at the target location of the tine. If we ignore the dashed arrows in Fig. 3, a threshold automaton captures the behavior of a process in one round, that is, a loop iteration in Fig. 1.

While most rules are derived directly from the pseudocode, some have to be added for modeling purposes: The self-loops of rules  $r_{13}$  and  $r_{14}$  model the “wait” statements in lines 5 and 9. In the standard asynchronous distributed computing model [19], a process repeatedly performs steps that include possible reception of messages until the condition of the “wait” is satisfied. Formally, this results in local stutter steps (modulo possibly received messages) in the control locations of lines 5 and 9 which are modeled with the self-loops. The rules  $r_1$  and  $r_2$  are the result of introducing so-called border locations  $I_0$  and  $I_1$  which, intuitively, inserts control locations between two loop iterations that do not belong to any iteration. This is required in our proofs for a reduction argument that reasons about steps from different iterations.

The algorithm is parameterized:  $n$  is the number of processes,  $t$  is the assumed number of faults, and  $f$  is the actual number of faults. It should be demonstrated to work under the resilience condition  $n > 5t \wedge t \geq f \wedge t > 0$ . Observe that the parameters  $n$  and  $t$  show up in the code of Fig. 1, while  $f$  does not. That is, for a concrete system, the values of  $n$  and  $t$  must be fixed a priori and compiled into the executable. The value  $f$  is outside of the control of a designer as it captures the number of faults in a run, which is determined by an unreliable environment (e.g., physical faults in components). In that, the correctness of fault-tolerant distributed algorithms is only restricted to runs where  $f \leq t$ , which is captured by the resilience condition. However, at the level of the threshold automata model, we do not distinguish between fixed (known) and unknown parameters. From a model checking perspective, by setting  $f > t$ , we can generate executions that violate certain specification in runs where there are more faults than expected, which is interesting when analyzing and comparing distributed algorithms.

*One round.* The code in Fig. 1 refers to numbers of received messages, and as is typical for distributed algorithms, their relation to sent messages (that is, the semantics of send and receive) is not explicit in the pseudo code. To formalize the behavior, the encoding in the threshold automaton directly refers to the numbers of sent messages, and they are encoded in the shared variables  $x_i$  and  $y_i$ . For instance, the locations  $J_0$  and  $J_1$  capture that a loop is entered with  $v$  being 0 and 1, respectively. Sending an  $(R, r, 0)$  and  $(R, r, 1)$  message is captured by the increments on the shared variables  $x_0$  and  $x_1$  in the rules  $r_3$  and  $r_4$ , respectively, e.g., a process that is in location  $J_0$  uses rule  $r_3$  to go to location  $SR$  (“sent  $R$  message”), and increments  $x_0$  in doing so. Waiting for  $R$  and  $P$  messages in the lines 5 and 9 is captured by looping in the locations  $SR$  and  $SP$ . In line 7, a process sends, e.g.,

**Table 1** The rules of the probabilistic threshold automaton for the Ben-Or’s algorithm

Rule	Guard	Update
$r_1$	$true$	–
$r_2$	$true$	–
$r_3$	$true$	$x_0++$
$r_4$	$true$	$x_1++$
$r_5$	$x_0+x_1 \geq n-t-f \wedge x_0 \geq (n+t)/2-f$	$y_0++$
$r_6$	$x_0+x_1 \geq n-t-f \wedge x_1 \geq (n+t)/2-f$	$y_1++$
$r_7$	$x_0+x_1 \geq n-t-f \wedge x_0 \geq (n-3t)/2-f \wedge x_1 \geq (n-3t)/2-f$	$y_2++$
$r_8$	$y_0+y_1+y_2 \geq n-t-f \wedge y_2 \geq (n-3t)/2-f \wedge y_0 \geq t+1-f$	–
$r_9$	$y_0+y_1+y_2 \geq n-t-f \wedge y_0 > (n+t)/2-f$	–
$r_{10}$	$y_0+y_1+y_2 \geq n-t-f \wedge y_2 \geq (n-3t)/2-f \wedge y_2 > n-2t-f-1$	–
$r_{11}$	$y_0+y_1+y_2 \geq n-t-f \wedge y_1 > (n+t)/2-f$	–
$r_{12}$	$y_0+y_1+y_2 \geq n-t-f \wedge y_2 \geq (n-3t)/2-f \wedge y_1 \geq t+1-f$	–
$r_{13}$	$true$	–
$r_{14}$	$true$	–

a  $(P, r, 0, D)$  message if it has received  $n-t$  messages out of which  $(n+t)/2$  are  $(R, r, 0)$  messages. This is captured in the guard of rule  $r_5$  where  $x_0+x_1 \geq n-t-f$  checks the number of messages in total, and  $x_0 \geq (n+t)/2 - f$  checks for the specific messages containing 0.

The “ $-f$ ” term models that in the message passing semantics underlying Fig. 1,  $f$  messages from Byzantine faults may be received *in addition* to the messages sent by correct processes (modeled by shared variables in Fig. 3). The branching at the end of the loop from lines 10 to 18 is captured by the rules outgoing of  $SP$ . In particular rule,  $r_{10}$  captures the coin toss in line 16. The non-determinism due to faults and asynchrony is captured by multiple rules being enabled at the same time.

Recall that behavior of a process in a single round is modeled by the solid edges in Fig. 3. Note that in this case, threshold guards should be evaluated according to the values of shared variables, e.g.,  $x_0$  and  $x_1$ , in the observed round.

*Round switches.* The dashed edges, called round-switch rules, encode how a process, after finishing a round, starts the next one. The round number  $r$  serves as the loop iterator in Fig. 1, and in each iteration, processes send messages that carry  $r$ . *To capture this, each round  $r$  maintains independent copies of the variables  $x_0, x_1, y_0, y_1$ , which are initialized with 0.* Because there are infinitely many rounds, this means a priori we have infinitely many variables.

**Example 1** Recall the distributed execution of Ben-Or’s algorithm in Fig. 2. We show how to model the pseudocode as a threshold automaton and the distributed execution as an execution of a counter system. Consider the threshold automaton in Fig. 3, and let us fix a system based on this automaton; for instance, let there be  $n = 6$  processes where  $f = t = 1$  process is Byzantine faulty. Note that in this case, we explicitly model only the five correct processes. In this example, accompanied in Fig. 4, we show a run in such a system; that is, we only show a prefix as every run is infinitely long.

Assume three correct processes start with value 0 and two with value 1. This initial configuration is denoted by  $\sigma_0$  and depicted in the upper left corner of Fig. 4, where three red circles in location  $I_0$  represent three correct processes with initial value 0, and similarly, two circles in  $I_1$  represent two correct processes with initial value 1.

After applying  $\tau_1 = (r_1, 1)^3(r_2, 1)(r_3, 1)^3(r_4, 1)$  to  $\sigma_0$ , we reach configuration  $\sigma_1$ , where 4 processes are in location  $SR$  and one is still in its initial location. We use a short notation  $(r_1, 1)^3$  for  $(r_1, 1)(r_1, 1)(r_1, 1)$ , where 3 processes execute  $r_1$  in the first round. After applying  $\tau_2 = (r_5, 1)^1(r_7, 1)^3(r_2, 1)(r_8, 1)^2(r_{10}, 1)^2$  to  $\sigma_1$ , we reach  $\sigma_2$  where the four processes from  $SR$  reach final location of the first round, which is depicted in the lower left corner of Fig. 4.

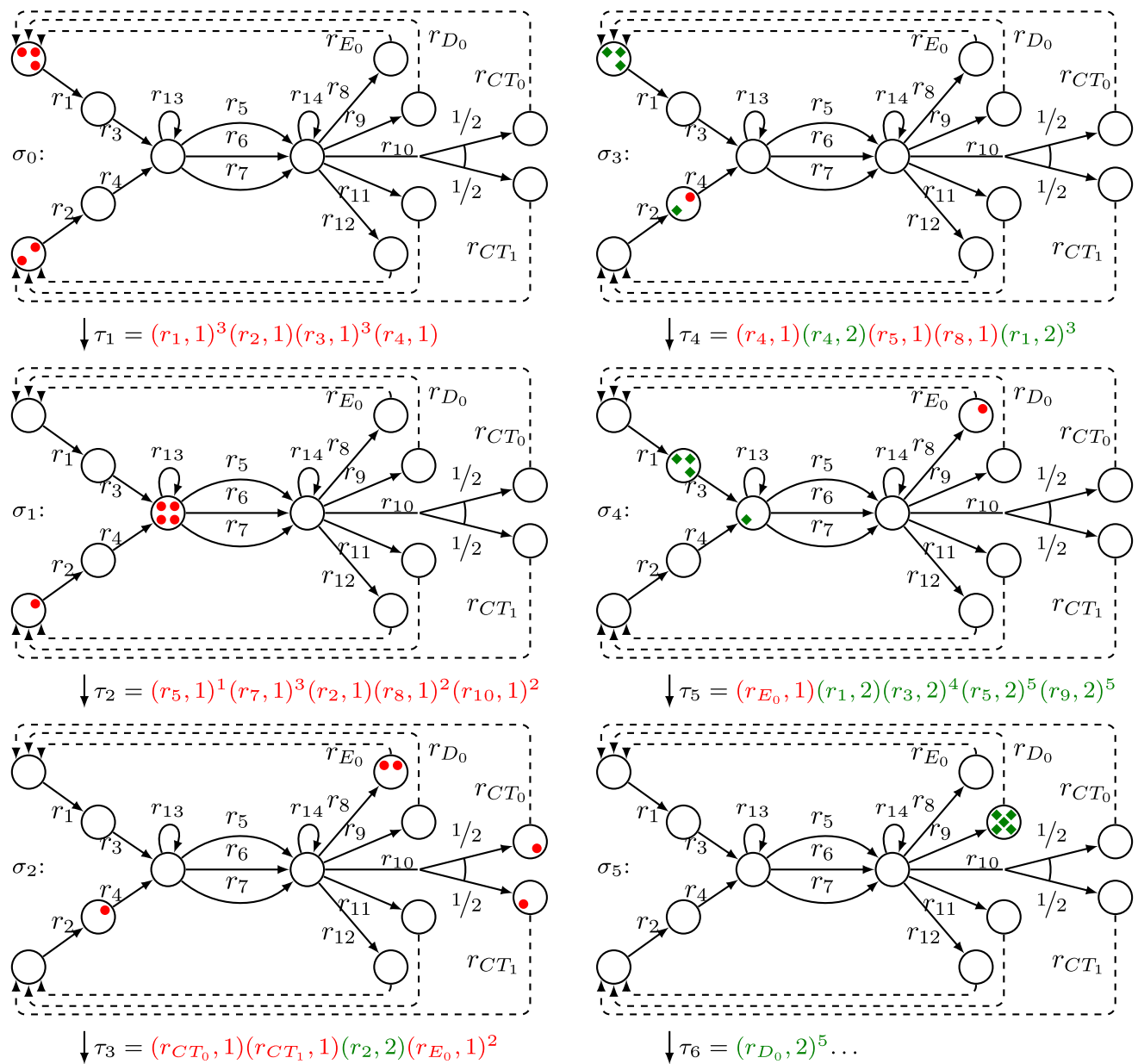
Next, the four “fast” processes move and start the second round by executing  $\tau_3 = (r_{CT_0}, 1)(r_{CT_1}, 1)(r_2, 2)(r_{E_0}, 1)^2$  and reaching  $\sigma_3$ . In order to distinguish processes from the first and the second round, we depict those in the second round as green diamonds. Applying  $\tau_4 = (r_4, 1)(r_4, 2)(r_5, 1)(r_8, 1)(r_1, 2)^3$  to  $\sigma_3$  leads to  $\sigma_4$ . Here, we can see that processes move in their own relative speeds, and at the same time, they might be in different rounds.

Finally, by executing  $\tau_5 = (r_{E_0}, 1)(r_1, 2)(r_3, 2)^4(r_5, 2)^5(r_9, 2)^5$  all correct processes decide value 0; that is, they all reach location  $D_0$ , depicted in the lower right corner of Fig. 4. Note that processes do not stop the execution here, but continue to the following round. It is important to notice that in the rest of the run, no matter how we extend it, every correct process will finish every following round in  $D_0$ ; that is, it will eventually decide 0.  $\square$

*Liveness and fairness.* Liveness properties of distributed algorithms typically require fairness constraints, e.g., every message sent by a correct process to a correct process is eventually received. For instance, this implies in Fig. 1 that if  $n-t$  correct processes have sent messages of the form  $(R, 1, *)$  and  $(n+t)/2$  correct processes have sent messages of the form  $(R, 1, 0)$ , then every correct process should eventually execute line 7 and proceed to line 9. We capture this by the following fairness constraint: If  $x_0+x_1 \geq n-t \wedge x_0 \geq (n+t)/2$ —that is, rule  $r_5$  is enabled without the help of the  $f$  faulty processes but by “correct processes alone”—then the source location of rule  $r_5$ ; namely,  $SR$  should eventually be evacuated; that is, its corresponding counter should eventually be 0.

*Restrictions.* The definition of threshold automata in its general form allows two disturbing features. First, the updates allow increments and decrements in shared variables. As was shown in [28], this feature allows us to use threshold automata to encode two counter machines, for which the halting problem is undecidable. As a result, without this restriction, parameterized verification of threshold automata is undecidable. Second, the most general definition also allows loops (closed paths) that contain rules that increase shared variables. In [28], it is shown that this leads to counter systems whose diameter is not bounded. As our model checker ByMC does bounded model checking, such threshold automata also cannot be handled. Luckily, none of these features is needed to encode fault-tolerant distributed algorithms: First, as increments in shared variables are used to model the sending of messages, we only need increments, as one cannot make a message unsent (which would correspond to a decrement). Second, increasing within a loop would correspond to a process iteratively sending the same message over and over again. As we use threshold automata to count messages from *distinct* processes, this increments would violate the intended semantics we require to cap-





**Fig. 4** A part of a run in the system based on the threshold automaton from Fig. 3, accompanying Example 1. Red circles represent correct processes in the first round, and green diamonds represent correct pro-

cesses in the second round. Similarly, red and green transitions are executed in the first and the second round, respectively

ture for distributed algorithms. Thus, it is convenient to consider standard—so-called “canonic”—restrictions here, *i.e.*, increments only of shared variables, and no updates of shared variables within loops. These restrictions still allow us to model threshold-based fault-tolerant distributed algorithms [24]. As a result, threshold automata without probabilistic forks and round switching rules can be automatically checked for safety and liveness [23,24]. Adding forks and round switches is required to adequately model randomized distributed algorithms. Here, we introduce the

restriction<sup>1</sup> (met by all our benchmarks) that coin-toss transitions only appear at the end of a round, *e.g.*, line 16 of Fig. 1. Intuitively, as discussed in Sect. 1, a coin toss is only necessary if there is no strong majority. Thus, all our benchmarks have this feature, and we exploit it in Sect. 7.

<sup>1</sup> This restriction is needed in Sect. 7.1 and in particular to establish Lemma 11.

## 2.2 Our approach at a glance

To sum up the above from a verification viewpoint, these algorithms have two sources of unboundedness: (i) They are parameterized by the number of participating processes, and (ii) they run for an infinite number of rounds. This paper is based on the idea to reduce the analysis of the iterative part (the rounds) to a few verification tasks for one-round systems, thus solving the verification challenge posed by (ii). Then, we can invoke existing model checking techniques from Konnov et al. [23,24] that address (i).

To reduce to the verification of one-round systems, we need to take several steps. First, we introduce the framework of probabilistic threshold automata in Sect. 3 that gives a precise semantics to the distributed algorithms (that are typically only described in pseudocode). This allows us in Sect. 4 to formalize the folklore consensus properties in the precise semantics provided by threshold automata. We arrive at temporal logic specifications that speak about multiple rounds. At this point, we have a precise formal understanding of the verification task: We have an infinite-state model of the computation of the distributed algorithm, and temporal logic specifications that contain multiple quantified round variables that range over an infinite sequence of rounds.

After having formalized all the objects of study, we are in the position to develop the reduction arguments. We start by reducing the problem statement, namely the temporal logic formulas. We show how to transform consensus specifications into one-round temporal formulas in Sect. 5 by analyzing the formulas: Consensus specifications often talk about at least two different rounds. In this case, we need to use round invariants that imply the specifications. For example, if we want to verify agreement, we have to check that no two processes decide different values, possibly in different rounds. We do this in two steps: (i) We check the round invariant that no process changes its decision from round to round, and (ii) we check that within a round, no two processes disagree. It remains the challenge of infinitely many rounds, which we address in the non-probabilistic setting in Sect. 6. Here, the main challenge is, as discussed above, that at the same time, different processes may be in different rounds. We simplify the verification by exploiting a reduction based on communication-closed rounds [12,15,17]. We prove that every execution in which steps are arbitrarily interleaved can be reduced to an “equivalent” execution where, roughly speaking, at all times all processes are on the same round. To do so, we prove that one can reorder transitions of any fair execution such that in the resulting (reordered) execution, the round numbers of the transitions are in a non-decreasing order. The mentioned equivalence is with respect to temporal logic properties. More precisely, the obtained ordered execution is stutter equivalent with the original one, and thus, they satisfy the same  $LTL_{\neg X}$  properties over the atomic

propositions describing only one round. In other words, any interleaved multi-round system that poses the verification challenge (ii) can be transformed to a sequential composition of one-round systems, which reduces the verification to one-round systems, which can be automatically checked by the model checker ByMC [26].

Verifying almost-sure termination under round-rigid adversaries calls for distinct arguments. Our methodology follows the lines of the manual proof of Ben-Or’s consensus algorithm by Aguilera and Toueg [1]. However, our arguments are not specific to Ben-Or’s algorithm and apply to other randomized distributed algorithms (see Sect. 8). Compared to their paper-and-pencil proof, the threshold automata framework required us to provide a more formal setting and a more informative proof, also pinpointing the needed hypotheses that we discuss in Sect. 7. As in the non-probabilistic case, the crucial parts of our proof are automatically checked by the model checker ByMC. Hence, the correctness we establish stands on less slippery ground, addressing the above-mentioned concerns of Lehmann and Rabin.

## 3 The framework of probabilistic threshold automata

To start with, we introduce our model of probabilistic threshold automata.

**Definition 1** A *probabilistic threshold automaton* PTA is a tuple  $(\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$ , where

- $\mathcal{L}$  is a finite set of locations that contains the following disjoint subsets:
  - *initial locations*  $\mathcal{I}$ ,
  - *final locations*  $\mathcal{F}$ , and
  - *border locations*  $\mathcal{B}$ ,
 with  $|\mathcal{B}| = |\mathcal{I}|$ ;
- $\mathcal{V}$  is a set of variables. It is partitioned in two sets:
  - $\Pi$  contains *parameter variables*, and
  - $\Gamma$  contains *shared variables*;
- $\mathcal{R}$  is a finite set of *rules*; and
- $RC$ , the *resilience condition*, is a formula in linear integer arithmetic over parameter variables.

In the following, we introduce rules in detail and give syntactic restrictions on rules that model the local transitions of a distributed algorithm from/to particular locations. The resilience condition  $RC$  only appears in the definition of the semantics in Sect. 3.1.

A *simple guard* is an expression of the form

$$b \cdot x \geq \bar{a} \cdot \mathbf{p}^\top + a_0 \text{ or } b \cdot x < \bar{a} \cdot \mathbf{p}^\top + a_0,$$

where  $x \in \Gamma$  is a shared variable,  $\bar{a} \in \mathbb{Z}^{|\Gamma|}$  is a vector of integers,  $a_0, b \in \mathbb{Z}$ , and  $\mathbf{p}$  is the vector of all parameters. The set of all simple guards is denoted by  $\mathcal{G}$ . A *threshold guard* (or just a *guard*) is a conjunction of simple guards.

A *rule*  $r$  is a tuple  $(\text{from}, \delta_{to}, \varphi, \mathbf{u})$  where  $\text{from} \in \mathcal{L}$  is the *source* location,  $\delta_{to} \in \text{Dist}(\mathcal{L})$  is a probability distribution over the *destination* locations,  $\varphi$  is a conjunction of guards, and  $\mathbf{u} \in \mathbb{N}_0^{|\Gamma|}$  is the *update vector*.

If  $r.\delta_{to}$  is a Dirac distribution, i.e., there exists  $\ell \in \mathcal{L}$  such that  $r.\delta_{to}(\ell) = 1$ , we call  $r$  a *Dirac rule* and write it as  $(\text{from}, \ell, \varphi, \mathbf{u})$ . Destination locations of non-Dirac rules are in  $\mathcal{F}$ . (Coin-toss transitions only happen at the end of a round.) If all rules of PTA are Dirac, then this automaton is also a threshold automaton [23].

As in [23], we only consider so-called *canonic* threshold automata, that is, every rule  $r$  that lies on a cycle ensures that  $r.\mathbf{u} = \mathbf{0}$ . Moreover, to simplify formalization of fairness constraints (to model reliable communication between processes of a distributed algorithm), we will exploit a characteristic of all our benchmarks, namely that there are no cycles within a round, except possibly self-loops.

**Remark 1** The above condition  $r.\mathbf{u} = \mathbf{0}$  for a rule  $r$  on a cycle may seem to be prohibitively restrictive. Note, however, that we use a shared variable  $x \in \Gamma$  to encode the number of the messages of type  $x$  that are sent by all correct processes. Hence, when constructing a threshold automaton, it is important to preserve the following invariant: *the automaton may increase every variable at most once*. This invariant allows us to model sending of a message in the environment with reliable communication [19] (which still allows for process failures). In an implementation of a distributed algorithm, a node would maintain the set of messages that it has received from other peers, and the node would discard duplicate messages. If a rule  $r$  increased the variable  $x$ , and the rule  $r$  lied on a cycle, then this would model the situation, in which a single process broadcasts a message by using several identities. This is forbidden in the classical fault-tolerant distributed algorithms.

We have investigated extensions of (non-probabilistic) threshold automata, including the automata that allow all rules to increase shared variables [28]. For such automata, the parameterized model checking problem is still decidable. However, the reduction-based techniques of ByMC [26] are not applicable to counter systems of non-canonical threshold automata.  $\square$

Probabilistic threshold automata model algorithms with multiple rounds that follow the same code. Informally, a round happens between border locations and final locations.

The round-switch rules let processes move from final locations of a given round to border locations of the next round. From each border location, there is exactly one Dirac rule to an initial location, and it has a form  $(\ell, \ell', \text{true}, \mathbf{0})$  where  $\ell \in \mathcal{B}$  and  $\ell' \in \mathcal{I}$ . As  $|\mathcal{B}| = |\mathcal{I}|$ , one can think of border locations as copies of initial locations. It remains to model from which final locations to which border location (that is, initial for the next round) processes move. This is done by *round-switch rules*. They can be described as Dirac rules  $(\ell, \ell', \text{true}, \mathbf{0})$  with  $\ell \in \mathcal{F}$  and  $\ell' \in \mathcal{B}$ . The set of round-switch rules is denoted by  $\mathcal{S} \subseteq \mathcal{R}$ .

A location belongs to  $\mathcal{B}$  if and only if all the incoming edges are in  $\mathcal{S}$ . Similarly, a location is in  $\mathcal{F}$  if and only if there is only one outgoing edge and it is in  $\mathcal{S}$ .

Figure 3 depicts a PTA with border locations  $\mathcal{B} = \{I_0, I_1\}$ , initial locations  $\mathcal{I} = \{J_0, J_1\}$ , and final locations  $\mathcal{F} = \{E_0, E_1, D_0, D_1, CT_0, CT_1\}$ . The only rule that is not a Dirac rule is  $r_{10}$ , and round-switch rules are represented by dashed arrows.

### 3.1 Probabilistic counter systems

The semantics of a probabilistic threshold automaton is an infinite-state Markov decision process (MDP), which we formally define below. First, we must define admissible parameters with respect to a given resilience condition.

A resilience condition  $RC$  defines the set of *admissible parameters*  $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Gamma|} : \mathbf{p} \models RC\}$ . We introduce a function  $N : \mathbf{P}_{RC} \rightarrow \mathbb{N}_0$  that maps a vector of admissible parameters to a number of modeled processes in the system. For instance, for the automaton in Fig. 3,  $N$  is the function  $(n, t, f) \mapsto n - f$ , as we model only the  $n - f$  correct processes explicitly, while the effect of faulty processes is captured in non-deterministic choices between different guards as discussed in Sect. 2. Given a PTA and a function  $N$ , we define the semantics, called *probabilistic counter system*  $\text{Sys}(\text{PTA})$ , to be the infinite-state MDP  $(\Sigma, I, \text{Act}, \Delta)$ , where  $\Sigma$  is the set of configurations for PTA among which  $I \subseteq \Sigma$  are initial, the set of actions is  $\text{Act} = \mathcal{R} \times \mathbb{N}_0$ , and  $\Delta : \Sigma \times \text{Act} \rightarrow \text{Dist}(\Sigma)$  is the probabilistic transition function.

**Configurations.** In a configuration  $\sigma = (\kappa, \mathbf{g}, \mathbf{p})$ , the function  $\sigma.\kappa : \mathcal{L} \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$  describes values of location counters *per round*, the function  $\sigma.\mathbf{g} : \Gamma \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$  defines shared variable values *per round*, and the vector  $\sigma.\mathbf{p} \in \mathbb{N}_0^{|\Gamma|}$  sets parameter values. We denote the vector  $(\mathbf{g}[x, k])_{x \in \Gamma}$  of shared variables in a round  $k$  by  $\mathbf{g}[k]$ , and by  $\kappa[k]$ , we denote the vector  $(\kappa[\ell, k])_{\ell \in \mathcal{L}}$  of location counters in a round  $k$ .

A configuration is *initial* if all processes are in initial locations of round 0, and all global variables evaluate to 0. Formally,  $\sigma = (\kappa, \mathbf{g}, \mathbf{p})$  is initial, if for every  $x \in \Gamma$  and  $k \in \mathbb{N}_0$ , we have  $\sigma.\mathbf{g}[x, k] = 0$ , if  $\sum_{\ell \in \mathcal{B}} \sigma.\kappa[\ell, 0] = N(\mathbf{p})$ ,



and finally if for every  $(\ell, k) \in ((\mathcal{L} \setminus \mathcal{B}) \times \{0\}) \cup (\mathcal{L} \times \mathbb{N})$ , it holds that  $\sigma.\kappa[\ell, k] = 0$ .

A threshold guard evaluates to true in a configuration  $\sigma$  for a round  $k$ , written  $\sigma, k \models \varphi$ , if for all its conjuncts  $b \cdot x \geq \bar{a} \cdot \mathbf{p}^\top + a_0$ , it holds that  $b \cdot \sigma.\mathbf{g}[x, k] \geq \bar{a} \cdot (\sigma.\mathbf{p}^\top) + a_0$  (and similarly for conjuncts of the other form, i.e.,  $b \cdot x < \bar{a} \cdot \mathbf{p}^\top + a_0$ ).

**Actions.** An action  $\alpha = (r, k) \in \text{Act}$  stands for the execution of a rule  $r$  in round  $k$  (by a single process). We write  $\alpha.\text{from}$  for  $r.\text{from}$ ,  $\alpha.\delta_{io}$  for  $r.\delta_{io}$ , etc.

An action  $\alpha = (r, k)$  is *unlocked* in a configuration  $\sigma$ , if the guard of its rule evaluates to true in round  $k$ , that is,  $\sigma, k \models r.\varphi$ . An action  $\alpha = (r, k)$  is *applicable* to a configuration  $\sigma$  if  $\alpha$  is unlocked in  $\sigma$ , and there is at least one process in the source location  $r.\text{from}$  at round  $k$ , formally,  $\sigma.\kappa[r.\text{from}, k] \geq 1$ . When an action  $\alpha$  is applicable to  $\sigma$ , and when  $\ell$  is a potential destination location for the probabilistic action  $\alpha$ , we write  $\text{apply}(\sigma, \alpha, \ell)$  for the resulting configuration: Parameters are unchanged, shared variables are updated according to the update vector  $r.\mathbf{u}$ , and the values of counters are modified in a natural way: As a process moves from  $r.\text{from}$  to  $\ell$  in round  $k$ , the counter  $\kappa[r.\text{from}, k]$  is decreased by 1 and counter  $\kappa[\ell, k]$  is increased by 1. Formally, we have that  $\text{apply}(\alpha, \ell, \sigma) = \sigma'$  if and only if  $\text{apply}(\alpha, \ell, \sigma)$  is defined and the following holds:

- The update vector changes the shared variables at round  $k$ , that is,  $\sigma'.\mathbf{g}[k] = \sigma.\mathbf{g}[k] + \alpha.\mathbf{u}$ , and  $\sigma'.\mathbf{g}[k'] = \sigma.\mathbf{g}[k']$ , for every round  $k' \neq k$ ,
- The parameter values do not change:  $\sigma'.\mathbf{p} = \sigma.\mathbf{p}$ ,
- A self-loop within a round does not change the variables: If  $r \in \mathcal{R} \setminus \mathcal{S}$  and  $\alpha.\text{from} = \ell$ , then  $\sigma'.\kappa = \sigma.\kappa$ ,
- An edge within a round (different from a self-loop) updates the round variables: If  $r \in \mathcal{R} \setminus \mathcal{S}$  and  $\alpha.\text{from} \neq \ell$ , then
  - $\sigma'.\kappa[\alpha.\text{from}, k] = \sigma.\kappa[\alpha.\text{from}, k] - 1$ ,
  - $\sigma'.\kappa[\ell, k] = \sigma.\kappa[\ell, k] + 1$ ,
  - $\forall \ell \in \mathcal{L} \setminus \{\alpha.\text{from}, \ell\}, \sigma'.\kappa[\ell, k] = \sigma.\kappa[\ell, k]$ , and
  - $\sigma'.\kappa[k'] = \sigma.\kappa[k']$ , for all rounds  $k' \neq k$
- A round-switch edge updates the counters of the rounds  $k$  and  $k + 1$ : If  $r \in \mathcal{S}$ , then
  - $\sigma'.\kappa[\alpha.\text{from}, k] = \sigma.\kappa[\alpha.\text{from}, k] - 1$ ,
  - $\sigma'.\kappa[\ell, k + 1] = \sigma.\kappa[\ell, k + 1] + 1$ , and
  - $\sigma'.\kappa[\ell', k'] = \sigma.\kappa[\ell', k']$ , for all  $(\ell', k') \in \mathcal{L} \times \mathbb{N}_0 \setminus \{(\alpha.\text{from}, k), (\ell, k + 1)\}$ .

**Probabilistic transition function.** The probabilistic transition function  $\Delta$  is defined such that for every two configurations  $\sigma$  and  $\sigma'$  and for every action  $\alpha$  applicable to  $\sigma$ , we have

$$\Delta(\sigma, \alpha)(\sigma') = \begin{cases} \alpha.\delta_{io}(\ell) & \text{if } \text{apply}(\sigma, \alpha, \ell) = \sigma', \\ 0 & \text{otherwise.} \end{cases}$$

A (finite or infinite) *path* in  $\text{Sys}(\text{PTA})$  is a sequence of configurations  $\sigma_0, \sigma_1, \dots$ , such that for every  $\sigma_i, i > 0$ , there exist an action  $\alpha_i$  and a location  $\ell_i$  such that  $\text{apply}(\sigma_{i-1}, \alpha_i, \ell_i) = \sigma_i$ .

### 3.2 Non-probabilistic counter systems

Non-probabilistic threshold automata were introduced in [25], and they can be seen as a special case of probabilistic threshold automata where all rules are Dirac rules. The definition in [25] did not capture multi-round algorithms; that is, there are no border and final locations and thus no restrictions on rules from/to these locations. In this section, we discuss non-probabilistic counterparts of probabilistic threshold automata and probabilistic counter systems. Doing so, our objective is twofold: On the one hand, it is natural to compare PTA with the formalism they extend; on the other hand, we will see that the two natural ways to assign a derandomized semantics to a PTA coincide (see commutative diagram in Fig. 5.)

With a PTA, one can naturally associate a non-probabilistic threshold automaton, by replacing probabilities with non-determinism: Every probabilistic rule  $r = (\text{from}, \delta_{io}, \varphi, \mathbf{u})$  is replaced by non-deterministic rules of the form  $r_\ell = (\text{from}, \ell, \varphi, \mathbf{u})$ , for every location  $\ell$  with  $\delta_{io}(\ell) > 0$ .

**Definition 2** Given a PTA  $= (\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$ , its induced (non-probabilistic) threshold automaton is

$$\text{TA}_{\text{PTA}} = (\mathcal{L}, \mathcal{V}, \mathcal{R}_{np}, RC),$$

where the set of rules  $\mathcal{R}_{np}$  is defined as

$$\{(from, \ell, \varphi, \mathbf{u}) : (from, \delta_{io}, \varphi, \mathbf{u}) \in \mathcal{R} \wedge \ell \in \mathcal{L} \wedge \delta_{io}(\ell) > 0\}.$$

If the rule  $(from, \delta_{io}, \varphi, \mathbf{u})$  is denoted by  $r$ , and a location  $\ell \in \mathcal{L}$  has  $\delta_{io}(\ell) > 0$ , then the obtained rule  $(from, \ell, \varphi, \mathbf{u})$  is denoted by  $r_\ell$ .

We write TA instead of  $\text{TA}_{\text{PTA}}$  when the automaton PTA is clear from the context. Every rule from  $\mathcal{R}_{np}$  corresponds to exactly one rule in  $\mathcal{R}$ , and for every rule in  $\mathcal{R}$ , there is at least one corresponding rule in  $\mathcal{R}_{np}$  (and exactly one for Dirac rules).

If we understand a TA as a PTA where all rules are Dirac rules, we can define transitions using the partial function  $\text{apply}$  in order to obtain an infinite (non-probabilistic) counter system, which we denote by  $\text{Sys}_\infty(\text{TA})$ . Moreover, since in this case  $\mathcal{R} = \mathcal{R}_{np}$ , actions of the PTA exactly match transitions of its TA. We obtain  $\sigma'$  by applying  $t = (r, k)$  to  $\sigma$

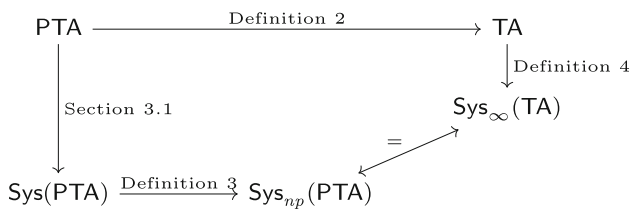


Fig. 5 Diagram following Proposition 1

and write this as  $\sigma' = t(\sigma)$ , if and only if for the destination location  $\ell$  of  $r$ , it holds that  $apply(\sigma, t, \ell) = \sigma'$ .

Also, starting from a PTA, one can define the probabilistic counter system  $Sys(PTA)$  and consequently its non-probabilistic counterpart  $Sys_{np}(PTA)$ . As the definitions of  $Sys_{np}(PTA)$  and  $Sys_{\infty}(TA)$  are equivalent for a given PTA, we are free to choose one and always use  $Sys_{\infty}(TA)$ . We formalize this intuition below.

**Definition 3** Given an arbitrary probabilistic counter system  $Sys(PTA) = (\Sigma, I, Act, \Delta)$ , we define its non-probabilistic version  $Sys_{np}(PTA)$  to be the tuple  $(\Sigma, I, R)$ , where  $R$  is a transition relation defined below.

If  $Act = \mathcal{R} \times \mathbb{N}_0$  and if  $\mathcal{R}_{np}$  is defined from  $\mathcal{R}$  as in Definition 2, then transitions are tuples  $t = (r_{\ell}, k) \in \mathcal{R}_{np} \times \mathbb{N}_0$  such that  $\alpha = (r, k)$  is an action from  $Act$  and for  $\ell \in \mathcal{L}$  holds that  $\alpha.\delta_{r_{\ell}}(\ell) > 0$ . Transition  $t$  is unlocked in a configuration  $\sigma$  from  $Sys_{np}(PTA)$  if  $\alpha$  is unlocked in  $\sigma$  in  $Sys(PTA)$ . Similarly, we define when  $t$  is applicable to  $\sigma$ . We obtain  $\sigma'$  by applying an applicable transition  $t$  to  $\sigma$ , written  $t(\sigma) = \sigma'$ , if and only if there exists a location  $\ell \in \mathcal{L}$  such that  $apply(\sigma, \alpha, \ell) = \sigma'$ .

Two configurations  $\sigma$  and  $\sigma'$  are in the transition relation  $R$ , i.e.,  $(\sigma, \sigma') \in R$ , if and only if there exists a transition  $t$  such that  $\sigma' = t(\sigma)$ .

**Definition 4** Given an arbitrary threshold automaton  $TA = (\mathcal{L}, \mathcal{V}, \mathcal{R}_{np}, RC)$ , with border, initial, and final location sets  $\mathcal{B}, \mathcal{I}$ , and  $\mathcal{F}$ , respectively, we define its infinite counter system  $Sys_{\infty}(TA)$  to be the tuple  $(\Sigma, I, R)$ . Configurations from  $\Sigma$  and  $I$  are defined as in Sect. 3.1. A transition  $t$  is a tuple  $(r_{\ell}, k) \in \mathcal{R}_{np} \times \mathbb{N}_0$ . Since it coincides with Dirac actions, we define when a transition is unlocked in a configuration and when it is applicable to a configuration, in the same way as for a Dirac action in Sect. 3.1. A configuration  $\sigma'$  is obtained by applying an applicable transition  $t = (r_{\ell}, k)$  to  $\sigma$ , written  $\sigma' = t(\sigma)$ , if and only if  $apply(\alpha, \ell, \sigma) = \sigma'$ , for a Dirac action  $\alpha = (r_{\ell}, k)$  and the destination location  $\ell$  of  $r$ .

Now, we have  $(\sigma, \sigma') \in R$ , if and only if there exists a transition  $t$  such that  $\sigma' = t(\sigma)$ .

**Proposition 1** Given a PTA, the non-probabilistic version  $Sys_{np}(PTA)$  of its counter system coincides with the infinite counter system  $Sys_{\infty}(TA)$  of its threshold automaton.

It is easy to see that the diagram from Fig. 5 commutes, and thus, every PTA yields the unique non-probabilistic counter system. The two constructions give us possibility to remove probabilistic reasoning either on the level of a PTA (using Definition 2) or on the level of a counter system  $Sys(PTA)$  (using Definition 3).

**Schedules and paths.** A (finite or infinite) sequence of transitions is called *schedule*, and it is often denoted by  $\tau$ . A schedule  $\tau = t_1, t_2, \dots, t_{|\tau|}$  is applicable to a configuration  $\sigma$  if there is a sequence of configurations  $\sigma = \sigma_0, \sigma_1, \dots, \sigma_{|\tau|}$  such that for every  $1 \leq i \leq |\tau|$ , we have that  $t_i$  is applicable to  $\sigma_{i-1}$  and  $\sigma_i = t_i(\sigma_{i-1})$ . A *path* in  $Sys_{np}(PTA)$  is an alternating sequence of configurations and transitions, for example,  $\sigma_0, t_1, \sigma_1, \dots, t_{|\tau|}, \sigma_{|\tau|}$ , such that for every  $t_i, 1 \leq i \leq |\tau|$ , in the sequence, we have that  $t_i$  is applicable to  $\sigma_{i-1}$  and  $\sigma_i = t_i(\sigma_{i-1})$ . Given a configuration  $\sigma_0$  and a schedule  $\tau = t_1, t_2, \dots, t_{|\tau|}$ , we denote by  $path(\sigma_0, \tau)$  a path  $\sigma_0, t_1, \sigma_1, \dots, t_{|\tau|}, \sigma_{|\tau|}$  where  $t_i(\sigma_{i-1}) = \sigma_i, 1 \leq i \leq |\tau|$ . Similarly, we define an infinite schedule  $\tau = t_1, t_2, \dots$ , and an infinite path  $\sigma_0, t_1, \sigma_1, \dots$ , also denoted by  $path(\sigma_0, \tau)$ .

**Observation 1** Since every transition in  $Sys_{\infty}(TA)$  comes from an action in  $Sys(PTA)$ , note that every path in  $Sys_{\infty}(TA)$  corresponds to a path in  $Sys(PTA)$ .

An infinite path is *fair* if no transition is applicable forever from some point on. Equivalently, when a transition is applicable, eventually either its guard becomes false, or all processes leave its source location.

**Remark 2** We use the above fairness constraint as it is convenient for our proofs. At the same time, it captures the standard weak fairness constraint of reliable communication for distributed algorithms: The requirement is that it is always the case that if there is a message to be received, then a message reception event will eventually happen [32, Chap. 8.4]. For the threshold guards that means that if a guard of a rule evaluates to true, and a process is at the source location of that rule, the process should eventually take the transition of the rule. We consider (i) a finite number of processes in each run and (ii) acyclic threshold automata, which implies that if a guard is enabled from some point on forever, its source location must eventually be empty forever, which is our fairness constraint.  $\square$

### 3.3 Adversaries

The non-determinism in Markov decision processes is traditionally resolved by a so-called adversary [3, Chap. 10]. Let  $Paths$  be the set of all finite paths in  $Sys(PTA)$ . An *adversary* is a function  $a : Paths \rightarrow Act$ , that given a finite path  $\pi$  of  $Sys(PTA)$  selects an action applicable to the last configuration of  $\pi$ . Given an initial configuration  $\sigma_0$ , an adversary  $a$  generates a set  $paths(\sigma_0, a)$  of infinite paths

$\sigma_0, \sigma_1, \dots$  with the following property: For every  $i > 0$ , there exists a location  $\ell_i$  such that  $\sigma_i = \text{apply}(\sigma_{i-1}, \alpha_i, \ell_i)$ , where  $\alpha_i = a(\sigma_0, \sigma_1, \dots, \sigma_{i-1})$ .

As usual, the MDP  $\text{Sys(PTA)}$  together with an initial configuration  $\sigma_0$  and an adversary  $a$  induce a Markov chain, written  $\mathcal{M}_a^{\sigma_0}$ . Precisely, the state space of  $\mathcal{M}_a^{\sigma_0}$  is  $\text{Paths}$ , its initial state is  $\sigma_0$  (the initial configuration, which is also a path of length 0), and the probabilistic transition function  $\delta_a : \text{Paths} \rightarrow \text{Dist}(\text{Paths})$  is defined for every  $ht \in \text{Paths}$  starting in  $\sigma_0$  and ending with a transition, and every configurations  $\sigma, \sigma' \in \Sigma$  and every transition  $t$  by:

$$(\delta_a(ht\sigma))(ht\sigma't') = \Delta(\sigma, a(ht\sigma))(\sigma'),$$

where  $t'$  is the transition  $(a(ht\sigma), \ell)$  with  $\text{apply}(\sigma, a(ht\sigma), \ell) = \sigma'$ . In words, the probability in  $\mathcal{M}_a^{\sigma_0}$  to move from state  $ht\sigma$  to state  $ht\sigma't'$  is nonzero as soon as there exists a transition  $t' = (r_\ell, k)$  such that  $\sigma' = \text{apply}(r, k, \ell, \sigma)$ . This equals to the probability that the corresponding process moves to  $\ell$  if the scheduler  $a$  picks action  $(r, k)$ . We write  $\mathbb{P}_a^{\sigma_0}$  for the probability measure over infinite paths starting at  $\sigma_0$  in  $\mathcal{M}_a^{\sigma_0}$ . An adversary  $a$  is *fair* if all paths in  $\text{paths}(\sigma_0, a)$  are fair.

We call an adversary  $a$  *round-rigid* if it is fair, and if every sequence of actions it produces can be decomposed to a concatenation of sequences of actions of the form  $s_1 \cdot s_1^p \cdot s_2 \cdot s_2^p \dots$ , where for all  $k \in \mathbb{N}$ , we have that the sequence  $s_k$  contains only Dirac actions of round  $k$ , and  $s_k^p$  contains only non-Dirac actions of round  $k$  (one per process). We denote the set of all round-rigid adversaries by  $\mathcal{A}^R$ .

### 3.4 Atomic propositions and stutter equivalence

The atomic propositions we consider describe the non-emptiness of a location in a given round, i.e., whether there is at least one process in location  $\ell \in \mathcal{L} \setminus \mathcal{B}$  in round  $k$ . The set of all such propositions for a round  $k \in \mathbb{N}_0$  is denoted by

$$\text{AP}_k = \{p(\ell, k) : \ell \in \mathcal{L} \setminus \mathcal{B}\} \cup \{g(\varphi, k) : \varphi \in \mathcal{G}\}.$$

For every  $k$ , we define a labeling function  $\lambda_k : \Sigma \rightarrow 2^{\text{AP}_k}$  such that  $p(\ell, k) \in \lambda_k(\sigma)$  iff  $\sigma.\kappa[\ell, k] > 0$ , and  $g(\varphi, k) \in \lambda_k(\sigma)$  iff  $\sigma, k \models \varphi$ . By abusing notation, we write “ $\kappa[\ell, k] > 0$ ” and “ $\kappa[\ell, k] = 0$ ” instead of  $p(\ell, k)$  and  $\neg p(\ell, k)$ , resp.

For a path  $\pi = \sigma_0, t_1, \sigma_1, \dots, t_n, \sigma_n, n \in \mathbb{N}$ , and a round  $k$ , a trace  $\text{trace}_k(\pi)$  w.r.t. the labeling function  $\lambda_k$  is the sequence  $\lambda_k(\sigma_0)\lambda_k(\sigma_1) \dots \lambda_k(\sigma_n)$ . Similarly, if a path is infinite  $\pi = \sigma_0, t_1, \sigma_1, t_2, \sigma_2, \dots$ , then  $\text{trace}_k(\pi) = \lambda_k(\sigma_0)\lambda_k(\sigma_1) \dots$ .

We say that two finite traces are stutter equivalent w.r.t.  $\text{AP}_k$ , denoted  $\text{trace}_k(\pi_1) \stackrel{\Delta}{=} \text{trace}_k(\pi_2)$ , if there is a finite sequence  $A_0A_1 \dots A_n \in (2^{\text{AP}_k})^+, n \in \mathbb{N}_0$ , such that both  $\text{trace}_k(\pi_1)$  and  $\text{trace}_k(\pi_2)$  are contained in the language given by the regular expression  $A_0^+A_1^+ \dots A_n^+$ . If traces of  $\pi_1$

and  $\pi_2$  are infinite, then stutter equivalence  $\text{trace}_k(\pi_1) \stackrel{\Delta}{=} \text{trace}_k(\pi_2)$  is defined in the standard way [3]: If traces of  $\pi_1$  and  $\pi_2$  are infinite, then we have  $\text{trace}_k(\pi_1) \stackrel{\Delta}{=} \text{trace}_k(\pi_2)$ , if there is an infinite sequence  $A_0A_1 \dots$  with  $A_i \subseteq \text{AP}_k$ , and natural numbers  $n_0, n_1, n_2, \dots, m_0, m_1, m_2 \dots \geq 1$  such that

$$\begin{aligned} \text{trace}_k(\pi_1) &= \underbrace{A_0 \dots A_0}_{n_0\text{-times}} \underbrace{A_1 \dots A_1}_{n_1\text{-times}} \underbrace{A_2 \dots A_2}_{n_2\text{-times}} \dots \\ \text{trace}_k(\pi_2) &= \underbrace{A_0 \dots A_0}_{m_0\text{-times}} \underbrace{A_1 \dots A_1}_{m_1\text{-times}} \underbrace{A_2 \dots A_2}_{m_2\text{-times}} \dots \end{aligned}$$

To simplify notation, we say that paths  $\pi_1$  and  $\pi_2$  are stutter equivalent w.r.t.  $\text{AP}_k$  and write  $\pi_1 \stackrel{\Delta}{=} \pi_2$ , instead of referring to specific path traces.

We denote by  $\pi_1 \stackrel{\Delta}{=} \pi_2$  that the paths  $\pi_1$  and  $\pi_2$  are stutter equivalent [3] w.r.t.  $\text{AP}_k$ . Two counter systems  $C_0$  and  $C_1$  are stutter equivalent w.r.t.  $\text{AP}_k$ , written  $C_0 \stackrel{\Delta}{=} C_1$ , if for every  $i \in \{0, 1\}$  and every path  $\pi$  from  $C_i$ , there is a path  $\pi'$  from  $C_{1-i}$  such that  $\pi \stackrel{\Delta}{=} \pi'$ .

**Remark 3** We emphasize that atomic propositions cannot check emptiness of border locations from the set  $\mathcal{B}$ . The specifications cannot observe the moment of transition from one round to another. An example illustrating why we have this restriction is given in Remark 4. This allows us to swap transitions of adjacent rounds in Sect. 6.  $\square$

## 4 Consensus properties and their verification

In probabilistic (binary) consensus, every correct process has an initial value from  $\{0, 1\}$ . It consists of safety specifications and an almost-sure termination requirement, which we consider in its round-rigid variant:

- Agreement:* no two correct processes decide differently.
- Validity:* if all correct processes have  $v$  as the initial value, then no process decides  $1 - v$ .
- Round-rigid probabilistic termination:* for every round-rigid adversary, with probability 1 every correct process eventually decides.

We now discuss the formalization of these specifications in the context of Ben-Or’s algorithm whose threshold automaton is given in Fig. 3.

**Formalization.** In order to formulate and analyze the specifications, we partition every set  $\mathcal{I}, \mathcal{B}$ , and  $\mathcal{F}$ , into two subsets  $\mathcal{I}_0 \uplus \mathcal{I}_1, \mathcal{B}_0 \uplus \mathcal{B}_1$ , and  $\mathcal{F}_0 \uplus \mathcal{F}_1$ , respectively. For every  $v \in \{0, 1\}$ , the partitions satisfy the following:

- (R1) The processes that are initially in a location  $\ell \in \mathcal{I}_v$  have the initial value  $v$ .

**Table 2** The syntax of round-based specifications: *pform* defines probabilistic formulas, *qform* defines multi-round temporal formulas, *tform* defines temporal path formulas, and *sform* defines state formulas

$$\begin{aligned}
 pform &::= \mathbb{P}_a^\sigma (\exists r \in \mathbb{N}_0. tform) = 1 \\
 qform &::= \mathbf{A} tform \mid \forall r \in \mathbb{N}_0. qform \\
 tform &::= sform \mid \mathbf{G} tform \mid \mathbf{F} tform \mid tform \vee tform \\
 sform &::= cform \mid gform \wedge cform \\
 cform &::= \bigvee_{\ell \in Locs} \kappa[\ell, r] \neq 0 \mid \bigwedge_{\ell \in Locs} \kappa[\ell, r] = 0 \mid cform \vee cform \\
 gform &::= guard \mid \neg gform \mid gform \wedge gform \\
 r &::= \text{name of a round variable, e.g., } k \text{ or } k'
 \end{aligned}$$

We assume that  $Locs \subseteq \mathcal{L}$  is a set of locations, *guard* is a threshold guard, and  $k \in \mathbb{N}_0$  is a round number

(R2) Rules connecting locations from  $\mathcal{B}$  and  $\mathcal{I}$  respect the partitioning, *i.e.*, they connect  $\mathcal{B}_v$  and  $\mathcal{I}_v$ . Similarly, rules connecting locations from  $\mathcal{F}$  and  $\mathcal{B}$  respect the partitioning.

We introduce two subsets  $\mathcal{D}_v \subseteq \mathcal{F}_v$ , for  $v \in \{0, 1\}$ . Intuitively, a process is in  $\mathcal{D}_v$  in a round  $k$  if and only if it decides  $v$  in that round.

The syntax of the specification language is given in Table 2. We use the universal counterpart of ELTL<sub>FT</sub> introduced in [24] and extend it with round quantifiers (starting with *qform*) and probabilities (starting with *pform*). While ELTL<sub>FT</sub> was initially introduced in [24] as an existential fragment of LTL in order to check whether there exists an execution that violates specification, here we directly check whether all executions satisfy specifications, and therefore, we use its universal counterpart.

In the following, we give an informal meaning of the formulas in Table 2. The formal semantics of temporal operators  $\mathbf{A}$ ,  $\mathbf{F}$ , and  $\mathbf{G}$  can be found in a textbook on model checking, *e.g.*, [14]. The rules *gform*, *cform*, and *sform* produce formulas over the atomic propositions. The rule *gform* produces formulas about threshold guards. It allows one to write a threshold guard, *e.g.*, the guard  $x_0 + x_1 \geq n - t - f$ , the guard  $x_0 \geq (n + t)/2 - f$ , and Boolean combinations thereof. The rule *cform* produces formulas about counters. It allows one to write that all processes are outside of given locations, that is,  $\bigwedge_{\ell \in Locs} \kappa[\ell, r] = 0$ ; or that at least one process resides in a location in a given set, that is,  $\bigvee_{\ell \in Locs} \kappa[\ell, r] \neq 0$ . Note that the formulas produced by *cform* are referring to the round number  $r$ , which is a free variable in these formulas. The round number  $r$  is bound by either a universal, or existential quantifier in the rules *pform* and *qform*.

The formulas produced by the rule *cform* can be conjoined only by *disjunction*, whereas the formulas produced by *cform* and *gform* can be conjoined only by *conjunction*. These combinations give us formulas that are produced by the rule *sform*. Note that these syntactic restrictions on the propositions were carefully chosen in [24], to ensure decidability of parameterized model checking.

The rule *tform* allows us to write certain temporal formulas: (1) that a proposition holds in the current state, that is, *sform*; (2) that a formula holds in the current state and all successor states along an execution, that is,  $\mathbf{G} tform$ ; (3) that a formula holds in the current state or in at least one successor state along an execution, that is,  $\mathbf{F} tform$ ; (4) that at least one of the temporal formulas holds true in the current state, that is,  $tform \vee tform$ .

Finally, the rule *qform* produces temporal formulas over all executions, that is,  $\mathbf{A} tform$ , and over all round numbers  $\forall r \in \mathbb{N}_0. qform$ . The rule *pform* produces quantitative formulas, that is, that the probability of a temporal formula being true for some round is equal to one. In our specifications, we consider only closed formulas produced by *pform* and *qform*, that is, the formulas, in which all round numbers are bound with the quantifier  $\forall r \in \mathbb{N}_0$  or with the quantifier  $\exists r \in \mathbb{N}_0$ .

Similar to atomic propositions, not every LTL formula can be turned into the form in Table 2. We have introduced this specific fragment of LTL to express specifications of round-based fault-tolerant distributed algorithms. The imposed constraints allow us to use the model checker ByMC.

Now, we can formalize the consensus specifications as follows:

*Agreement*: for both  $v \in \{0, 1\}$ , the following holds:

$$\begin{aligned}
 \forall k \in \mathbb{N}_0, \forall k' \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{F} \bigvee_{\ell \in \mathcal{D}_v} \kappa[\ell, k] > 0 \right. \\
 \left. \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{D}_{1-v}} \kappa[\ell', k'] = 0 \right) \quad (1)
 \end{aligned}$$

*Validity*: for both  $v \in \{0, 1\}$ , the following holds:

$$\begin{aligned}
 \forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{G} \bigwedge_{\ell \in \mathcal{I}_v} \kappa[\ell, 0] = 0 \right. \\
 \left. \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{D}_v} \kappa[\ell', k] = 0 \right) \quad (2)
 \end{aligned}$$

*Round-rigid probabilistic termination*: for every initial configuration  $\sigma$  and every round-rigid adversary  $\mathfrak{a}$ , the following holds:



$$\mathbb{P}_a^\sigma \left[ \exists k \in \mathbb{N}_0. \bigvee_{v \in \{0,1\}} \mathbf{G} \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell, k] = 0 \right] = 1 \tag{3}$$

Agreement and validity are non-probabilistic properties and can be analyzed on the non-probabilistic counter system  $\text{Sys}_\infty(\text{TA})$ . For verifying round-rigid probabilistic termination, we make explicit the following assumption that is present in all our benchmarks: All non-Dirac transitions have nonzero probability to lead to an  $\mathcal{F}_v$  location, for both values  $v \in \{0, 1\}$ . Indeed, recall that coin tosses are only used when there is no strong majority and are then used to sample a new value.

In Sect. 5, we formalize safety specifications and reduce them to single-round specifications. In Sect. 6, we reduce verification of multi-round counter systems to verification of single-round systems. In Sect. 7, we explain our approach to prove probabilistic termination.

### 5 Reduction to specifications with one round quantifier

Let’s have another look at the properties that we formalized in the previous section. We observe that *Agreement* contains two round variables  $k$  and  $k'$ , and *Validity* considers rounds 0 and  $k$ . Thus, both involve two round numbers. As ByMC can only analyze systems with a few rounds [24], the properties are only allowed to use one round number. In this section, we show how to check formulas (1) and (2) by checking properties that refer to one round.

To do so, first we introduce two round invariants (4) and (5). The rest of the section is then devoted to proving that these round invariants imply the consensus properties agreement and validity. In more detail, Lemma 1 establishes a central property for inductive arguments and links properties over counters of final locations for some round  $k$  to properties of counters of initial locations for round  $k + 1$ . We apply this first to (5) in Lemma 2, which then eventually allows us to prove Proposition 2 that establishes that to prove agreement and validity, it is sufficient to check (4) and (5).

We start with the round invariants. The first round invariant claims that in every round and in every path, once a process decides  $v$  in a round, no process ever enters a location from  $\mathcal{F}_{1-v}$  in that round. Formally:

$$\forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{F} \bigvee_{\ell \in \mathcal{D}_v} \kappa[\ell, k] > 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{F}_{1-v}} \kappa[\ell', k] = 0 \right) \tag{4}$$

The second round invariant claims that in every round and in every path, if no process starts a round with a value  $v$ , then no process terminates that round with value  $v$ . Formally:

$$\forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{G} \bigwedge_{\ell \in \mathcal{I}_v} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{F}_v} \kappa[\ell', k] = 0 \right) \tag{5}$$

The benefit of analyzing these two formulas instead of (1) and (2) lies in the fact that formulas (4) and (5) describe properties of only one round in a path. We shall later show in Theorem 2 that one-round specifications can be checked in a one-round counter system, instead of an infinite counter system  $\text{Sys}_\infty(\text{TA})$ .

Next, we want to prove that formulas (4) and (5) indeed imply formulas (1) and (2).

Let us first give some useful properties of  $\text{Sys}_\infty(\text{TA})$ . The following lemma states that in every round and in every run, if no process ever enters a final location with value  $v$ , then in the next round, there will be no process in any initial location with that value  $v$ .

**Lemma 1** (Round switch) *For every  $\text{Sys}_\infty(\text{TA})$  and every  $v \in \{0, 1\}$ :*

$$\forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{G} \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{I}_v} \kappa[\ell', k + 1] = 0 \right). \tag{6}$$

**Proof** By definitions of  $\mathcal{F}_v$ ,  $\mathcal{B}_v$  and  $\mathcal{I}_v$ , that is, by restriction (R2), we have that

$$\forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{G} \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell'' \in \mathcal{B}_v} \kappa[\ell'', k + 1] = 0 \right),$$

and

$$\forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{G} \bigwedge_{\ell'' \in \mathcal{B}_v} \kappa[\ell'', k + 1] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{I}_v} \kappa[\ell', k + 1] = 0 \right).$$

The two formulas together yield the required one for both values of  $v$ . □

Using the lemma and formula (5), we can show that once we reach a round in which no process has initial value  $v$ , every future round will have the same property; that is, no process will ever have initial value  $v$ . The following lemma formalizes that claim.



**Lemma 2** For every  $\text{Sys}_\infty(\text{TA})$  such that  $\text{Sys}_\infty(\text{TA}) \models (5)$ , and for every  $v \in \{0, 1\}$ , the following holds:

$$\forall k \in \mathbb{N}_0, \forall k' \in \mathbb{N}_0. (k \leq k' \rightarrow \mathbf{A} (\mathbf{G} \bigwedge_{\ell \in \mathcal{I}_v} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{I}_v} \kappa[\ell', k'] = 0)), \quad (7)$$

$$\forall k \in \mathbb{N}_0, \forall k' \in \mathbb{N}_0. (k \leq k' \rightarrow \mathbf{A} (\mathbf{G} \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{F}_v} \kappa[\ell', k'] = 0)). \quad (8)$$

**Proof** Assume formula (5) holds for the runs of  $\text{Sys}_\infty(\text{TA})$ . By combining Lemma 1 together with Eq. (5)—by reasoning about the locations  $\mathcal{F}_v$  and  $\mathcal{I}_v$ —, we conclude that the runs of  $\text{Sys}_\infty(\text{TA})$  satisfy the following formula:

$$\forall k \in \mathbb{N}_0. \mathbf{A} (\mathbf{G} \bigwedge_{\ell \in \mathcal{I}_v} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{I}_v} \kappa[\ell', k + 1] = 0). \quad (9)$$

By induction, we obtain the required formula (7). Finally, by combining formulas (6)–(7) we obtain formula (8).  $\square$

Finally, we can prove our main claim that formulas (4) and (5) imply formulas (1) and (2).

**Proposition 2** If  $\text{Sys}_\infty(\text{TA}) \models (4) \wedge (5)$ , then  $\text{Sys}_\infty(\text{TA}) \models (1) \wedge (2)$ .

**Proof** Assume  $\text{Sys}_\infty(\text{TA}) \models (4) \wedge (5)$ .

Let us first focus on formula (1) and prove that  $\text{Sys}_\infty(\text{TA}) \models (1)$ . Assume by contradiction that the formula does not hold on  $\text{Sys}_\infty(\text{TA})$ , that is, there exist rounds  $k, k' \in \mathbb{N}_0$  and a path  $\pi$  such that:

$$\pi \models \mathbf{F} \bigvee_{\ell_0 \in \mathcal{D}_0} \kappa[\ell_0, k] > 0 \wedge \mathbf{F} \bigvee_{\ell_1 \in \mathcal{D}_1} \kappa[\ell_1, k'] > 0. \quad (10)$$

Since by formula (10) we have  $\pi \models \mathbf{F} \bigvee_{\ell_0 \in \mathcal{D}_0} \kappa[\ell_0, k] > 0$ , then from formula (4) with  $v = 0$ , we obtain that it also holds  $\pi \models \mathbf{G} \bigwedge_{\ell \in \mathcal{F}_1} \kappa[\ell, k] = 0$ . As  $\mathcal{D}_1 \subseteq \mathcal{F}_1$ , we know that no process decides 1 in round  $k$ . Now, formula (8) from Lemma 2 for  $v = 1$  yields that  $\pi \models \mathbf{G} \bigwedge_{\ell \in \mathcal{F}_1} \kappa[\ell, k_1] = 0$  for every  $k_1 \geq k$ , i.e., in any round greater than  $k$ , no process will ever decide 1. As by (10), we have that  $\pi \models \mathbf{F} \bigvee_{\ell_1 \in \mathcal{D}_1} \kappa[\ell_1, k'] > 0$ , i.e., a process decides 1 in a round  $k'$ , thus it must be that  $k' < k$ .

Now, we consider the other part of formula (10), i.e.,  $\pi \models \mathbf{F} \bigvee_{\ell_1 \in \mathcal{D}_1} \kappa[\ell_1, k'] > 0$ . By following the analogous analysis, we conclude that it must be that  $k < k'$ . This brings us to the contradiction with  $k' < k$ , which proves the first part of the statement that of (4) and (5) implies of (1).

Next, we focus on formula (2) and prove by contradiction that it must hold. We start by assuming that the formula does not hold; that is, there exist a round  $k$  and a path  $\pi$  such that no process (ever) has initial value  $v$  in the first round of  $\pi$  and eventually in a round  $k$  a process decides  $v$ . Formally,

$$\pi \models \mathbf{G} \bigwedge_{\ell \in \mathcal{I}_v} \kappa[\ell, 0] = 0 \wedge \mathbf{F} \bigvee_{\ell' \in \mathcal{D}_v} \kappa[\ell', k] > 0. \quad (11)$$

Since we have  $\pi \models \mathbf{G} \bigwedge_{\ell \in \mathcal{I}_v} \kappa[\ell, 0] = 0$  and also  $\text{Sys}_\infty(\text{TA}) \models (5)$ , implying that formula (5) holds on  $\pi$ , we conclude that  $\pi \models \mathbf{G} \bigwedge_{\ell' \in \mathcal{F}_v} \kappa[\ell', 0] = 0$ . Then, by formula (8) we have that for every  $k \in \mathbb{N}_0$  it holds  $\pi \models \mathbf{G} \bigwedge_{\ell' \in \mathcal{F}_v} \kappa[\ell', k] = 0$ . By  $\mathcal{D}_v \subseteq \mathcal{F}_v$ , we also have that  $\pi \models \mathbf{G} \bigwedge_{\ell' \in \mathcal{D}_v} \kappa[\ell', k] = 0$ . As this contradicts our assumption from (11) that  $\pi \models \mathbf{F} \bigvee_{\ell' \in \mathcal{D}_v} \kappa[\ell', k] > 0$ , it proves the second part of the statement that of (4) and (5) implies of (2).  $\square$

It is important to note that ByMC cannot check soundness of the arguments given in this section. This kind of compositional reasoning has to be done by the user for the specific temporal properties. For temporal properties that are different from (1) to (3), one has to find round invariants similar to the ones below. Formalizing the reduction arguments in a proof system such as TLAPS [47] is out of the scope of this paper. As complete temporal reasoning in TLAPS is still under development, it is hard to predict the effort that is required for mechanization of such proofs.

## 6 Reduction to single-round counter system

Given a property of one round, our goal is to prove that there is a counterexample to the property in the multi-round system iff there is a counterexample in a single-round system. This is stated in Theorem 2, which is the main result of this section on page 20. It allows us to use ByMC on a single-round system.

The proof idea contains two parts. First, in Sect. 6.1 we prove that one can replace an arbitrary finite schedule with a round-rigid one, while preserving atomic propositions of a fixed round. We show that swapping two adjacent transitions that do not respect the order over round numbers in an execution gives us a legal stutter equivalent execution, i.e., an execution satisfying the same  $\text{LTL}_{-X}$  properties.

Second, in Sect. 6.2 we extend this reasoning to infinite schedules and lift it from schedules to transition systems. The main idea is to do inductive and compositional reasoning over the rounds. To do so, we need well-defined round boundaries, which is the case if every round that is started is also finished, a property we can automatically check for fair schedules. In more detail, regarding propositions for one

round, we show that the multi-round transition system is stutter equivalent to a single-round transition system. This holds under the assumption that all fair executions of a single-round transition system terminate, and this can be checked with ByMC, using the technique from Konnov et al. [24].

We are interested in stutter equivalence of systems because of the fundamental result that stutter equivalent systems satisfy the same  $LTL_{\neg X}$  specifications [3, Thm. 7.92]:

**Proposition 3** Fix a  $k \in \mathbb{N}_0$ . If  $\pi_1$  and  $\pi_2$  are paths such that  $\pi_1 \stackrel{\Delta}{=} \pi_2$ , then for every formula  $\varphi$  of  $LTL_{\neg X}$  over  $AP_k$ , we have  $\pi_1 \models \varphi$  if and only if  $\pi_2 \models \varphi$ .

This allows us to check the properties of consensus on a single-round transition system.

### 6.1 Reduction from arbitrary schedules to round-rigid schedules

As discussed in Sect. 2, we want to show how an arbitrary schedule in which steps are arbitrarily interleaved can be reduced to an “equivalent” schedule where “at all times all processes are on the same round”. The following definitions are a formalization of the latter requirements within our framework. It defines a schedule as *round-rigid* if the round numbers of transitions are ordered: Intuitively, no process is allowed to perform its first round  $k$  step before all other processes have done their final round  $k - 1$  step.

**Definition 5** A schedule  $\tau = (r_1, k_1) \cdot (r_2, k_2) \cdot \dots \cdot (r_m, k_m)$ ,  $m \in \mathbb{N}_0$ , is called *round-rigid* if for every  $1 \leq i < j \leq m$ , we have  $k_i \leq k_j$ .

In the rest of the section, we will prove that from an arbitrary schedule, we can arrive at a round-rigid one that satisfies the same  $LTL_{\neg X}$  temporal properties. We start with the following technical lemma which gives us the most important transition invariants that we can use to reason about reordering transitions in the proof of Lemma 4.

**Lemma 3** Let  $\sigma$  be a configuration, and let  $t = (r, k)$  be a transition. If  $\sigma' = t(\sigma)$ , then the following holds:

- (a)  $\sigma'.g[k'] = \sigma.g[k']$ , for every round  $k' \neq k$ ,
- (b)  $\sigma'.\kappa[k'] = \sigma.\kappa[k']$ , for every  $k' \in \mathbb{N}_0 \setminus \{k, k+1\}$ ,
- (c)  $\sigma'.\kappa[\ell, k'] = \sigma.\kappa[\ell, k']$ , for every round  $k' \neq k$  and every location  $\ell \in \mathcal{L} \setminus \mathcal{B}$ ,
- (d)  $\sigma'.\kappa[k+1] \geq \sigma.\kappa[k+1]$ ,
- (e)  $\sigma', k' \models \varphi$  iff  $\sigma, k' \models \varphi$ , for every round  $k' \neq k$  and every guard  $\varphi \in \mathcal{G}$ .

**Proof** The first four statements follow directly from the definitions of transitions. Finally, for the point (e) note that the evaluation of the guard  $\varphi$  in round  $k'$  depends only on the values of parameters  $\mathbf{p}$  that do not change along an execution,

and shared variables  $\mathbf{g}[k']$  in round  $k'$  that are unchanged according to the point (a).  $\square$

The following lemma establishes a central argument for inductive round-based reasoning: A transition can always be moved before a transition of a later round. It is proved using arguments on the commutativity of transitions, similar to Elrad and Francez [17].

**Lemma 4** Let  $\sigma$  be a configuration, and  $t_1 = (r_1, k_1)$  and  $t_2 = (r_2, k_2)$  be transitions, such that  $k_1 > k_2$ . If  $t_1 \cdot t_2$  is applicable to  $\sigma$ , then  $t_2 \cdot t_1$  is also applicable to  $\sigma$ .

**Proof** Let us denote  $t_1(\sigma)$  by  $\sigma_1$ . As  $t_1 \cdot t_2$  is applicable to  $\sigma$ , this means that  $t_1$  is applicable to  $\sigma$  and  $t_2$  is applicable to  $\sigma_1$ . By definition of applicability, this means that

$$\sigma.\kappa[r_1.from, k_1] \geq 1 \quad \text{and} \quad \sigma_1.\kappa[r_2.from, k_2] \geq 1, \quad (12)$$

and additionally, we have that  $\sigma, k_1 \models t_1.\varphi$  and  $\sigma_1, k_2 \models t_2.\varphi$ .

We show that  $t_2 \cdot t_1$  is applicable to  $\sigma$  by showing that: (i)  $t_2$  is applicable to  $\sigma$ , and (ii)  $t_1$  is applicable to  $t_2(\sigma)$ .

(i) First, we need to show that  $\sigma.\kappa[r_2.from, k_2] \geq 1$  and  $\sigma, k_2 \models t_2.\varphi$ .

As  $\sigma_1 = t_1(\sigma)$  and  $k_2 < k_1$ , by Lemma 3(b) we have  $\sigma_1.\kappa[r_2.from, k_2] = \sigma.\kappa[r_2.from, k_2]$ . From this and (12) we get that  $\sigma.\kappa[r_2.from, k_2] \geq 1$ .

Recall that  $\sigma_1, k_2 \models t_2.\varphi$ . By Lemma 3(e), it must be the case that also  $\sigma, k_2 \models t_2.\varphi$ . This shows that  $t_2$  is applicable to  $\sigma$ .

(ii) Let  $\sigma_2 = t_2(\sigma)$ . Next, we show that  $t_1$  is applicable to  $\sigma_2$ . Using the same reasoning as in (i), we prove that  $\sigma_2.\kappa[r_1.from, k_1] \geq 1$  and that  $\sigma_2, k_1 \models t_1.\varphi$ .

Because  $\sigma_2 = t_2(\sigma)$  and  $k_2 < k_1$ , Lemmas 3(b) and (d) yield  $\sigma_2.\kappa[r_1.from, k_1] \geq \sigma.\kappa[r_1.from, k_1]$ . Together with (12), we obtain  $\sigma_2.\kappa[r_1.from, k_1] \geq 1$ .

To this end, we show that  $\sigma_2, k_1 \models t_1.\varphi$ . Because  $\sigma_2 = t_2(\sigma)$  and  $k_1 > k_2$ , by Lemma 3(a), we know that  $\sigma.g[k_1] = \sigma_2.g[k_1]$ . Since by the initial assumption we have  $\sigma, k_1 \models t_1.\varphi$ , Lemma 3(e) yields  $\sigma_2, k_1 \models t_1.\varphi$ .  $\square$

We have thus seen that “out of order” transitions can be swapped such that the resulting sequence of transitions again is a valid schedule. However, when swapping transitions, intermediate configurations change: Intuitively, if a process  $p$  moves out a locations before another process  $q$  moves into the same location, they are never in the location at the same time, while if  $q$  moves first, they are. The following lemma shows that despite of this, the swapping does not interfere with our temporal formulas; the original schedule and the reordered schedule are stutter equivalent with respect to our atomic propositions. The reason is that we only swap transitions of *different* rounds, while our temporal logic fragment talks only about one round.

**Lemma 5** *Let  $\sigma$  be a configuration, and let  $t_1 = (r_1, k_1)$  and  $t_2 = (r_2, k_2)$  be transitions such that  $k_1 > k_2$ . If  $t_1 \cdot t_2$  is applicable to  $\sigma$ , then the following holds:*

- (a) Both  $t_1 \cdot t_2$  and  $t_2 \cdot t_1$  reach the same configuration, i.e.,  $t_1 \cdot t_2(\sigma) = t_2 \cdot t_1(\sigma)$ .
- (b) For all  $k \in \mathbb{N}_0$ , we have  $\text{path}(\sigma, t_1 \cdot t_2) \stackrel{\Delta}{=}_k \text{path}(\sigma, t_2 \cdot t_1)$ .

**Proof** Note that since  $t_1 \cdot t_2$  is applicable to  $\sigma$ , we also have that  $t_2 \cdot t_1$  is applicable to  $\sigma$  by Lemma 4, since  $k_1 > k_2$ .

(a) When a transition is applied to a configuration, the obtained configuration has the same parameter values, and counters and global variables are increased or decreased depending on the transition (and independently of the initial configuration). For any configuration  $(\kappa, \mathbf{g}, \mathbf{p})$ , we can write  $t_i(\kappa, \mathbf{g}, \mathbf{p}) = (\kappa + \mathbf{u}_i, \mathbf{g} + \mathbf{v}_i, \mathbf{p})$  for  $i \in \{1, 2\}$ , and some vectors  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{v}_1, \mathbf{v}_2$  of integers. By only using commutativity of addition and subtraction, we obtain  $t_1 \cdot t_2(\sigma) = (\kappa + \mathbf{u}_1 + \mathbf{u}_2, \mathbf{g} + \mathbf{v}_1 + \mathbf{v}_2, \mathbf{p}) = (\kappa + \mathbf{u}_2 + \mathbf{u}_1, \mathbf{g} + \mathbf{v}_2 + \mathbf{v}_1, \mathbf{p}) = t_2 \cdot t_1(\sigma)$ .

(b) Let  $\sigma_1 = t_1(\sigma)$ ,  $\sigma_2 = t_2(\sigma)$ , and  $\sigma_3 = t_1 \cdot t_2(\sigma)$ . Then,  $\text{trace}_k(\text{path}(\sigma, t_1 \cdot t_2)) = \lambda_k(\sigma)\lambda_k(\sigma_1)\lambda_k(\sigma_3)$ , and  $\text{trace}_k(\text{path}(\sigma, t_2 \cdot t_1)) = \lambda_k(\sigma)\lambda_k(\sigma_2)\lambda_k(\sigma_3)$ . We consider three cases: (i)  $k \neq k_1$  and  $k \neq k_2$ , (ii)  $k = k_1$ , and (iii)  $k = k_2$ .

(i) In this case, due to Lemmas 3(c) and (e), we have  $\lambda_k(\sigma) = \lambda_k(\sigma_1) = \lambda_k(\sigma_2) = \lambda_k(\sigma_3)$ . Therefore, both traces are  $\lambda_k(\sigma)\lambda_k(\sigma)\lambda_k(\sigma)$ , and they are clearly stutter equivalent.

(ii) Since  $k = k_1 > k_2$ , then again by Lemmas 3(c) and 3(e) we have  $\lambda_k(\sigma_1) = \lambda_k(\sigma_3)$  and  $\lambda_k(\sigma) = \lambda_k(\sigma_2)$ . Thus,  $\text{trace}_k(\text{path}(\sigma, t_1 \cdot t_2)) = \lambda_k(\sigma)\lambda_k(\sigma_3)\lambda_k(\sigma_3)$ , and  $\text{trace}_k(\text{path}(\sigma, t_2 \cdot t_1)) = \lambda_k(\sigma)\lambda_k(\sigma)\lambda_k(\sigma_3)$ , and the traces are stutter equivalent.

(iii) The last case is analogous to the previous one. □

**Remark 4** Let us briefly discuss why it is crucial to introduce border locations as buffers between two adjacent rounds, but not to reason about them in specifications, that is, why it is crucial to exclude atomic propositions checking their emptiness.

Note that the proof of Lemma 5 heavily relies on Lemma 3(c) that holds only when  $\ell \notin \mathcal{B}$ . If we were to include atomic propositions that refer to the emptiness of border locations (or if we did not have border locations at all), Lemma 5 would not hold. Namely, assume the following scenario: Let  $\sigma$  be a configuration with one process in the final location  $\ell_f$  of round 3, one process in any non-final location  $\ell$  of round 4, and let all other processes be in round 5 or higher. Let  $\ell_b \in \mathcal{B}$  be a border location such that  $r_1 \in \mathcal{S}$  is the round-switch rule  $(\ell_f, \ell_b, \text{true}, \mathbf{0})$  and  $t_1 = (r_1, 3)$ . Let  $r_2 \in \mathcal{R} \setminus \mathcal{S}$  be the rule  $(\ell, \ell', \varphi, \mathbf{u})$ , for some  $\ell' \in \mathcal{L}$ , with  $\varphi$  true in  $\sigma$ , and  $t_2 = (r_2, 4)$ . Then, the traces corresponding to  $\text{path}(\sigma, t_1 \cdot t_2)$  and  $\text{path}(\sigma, t_2 \cdot t_1)$  w.r.t. the round 4 are,

respectively (for simplicity, we omit guards),

$$\{p(\ell, 4)\} \{p(\ell_b, 4), p(\ell, 4)\} \{p(\ell_b, 4), p(\ell', 4)\} \\ \text{and } \{p(\ell, 4)\} \{p(\ell', 4)\} \{p(\ell_b, 4), p(\ell', 4)\}.$$

Note that these are not stutter equivalent traces, and thus, for example, formula  $\mathbf{F}(\kappa[\ell_b, 4] = 0 \wedge \kappa[\ell, 4] = 0)$  is satisfied only in  $\text{path}(\sigma, t_2 \cdot t_1)$ , but not in  $\text{path}(\sigma, t_1 \cdot t_2)$ .

If there were no border locations, that is, if round-switch rules connected final locations with initial ones, we could use the same counterexample (with  $\ell_b \in \mathcal{I}$  instead of  $\ell_b \in \mathcal{B}$ ) to show that it is not possible to maintain one-round properties while swapping transitions of different rounds.

Since Lemma 5 is the main building block of our technique, it is necessary to (i) introduce border locations and (ii) exclude atomic proposition checking their emptiness. □

Before proving our central result in Proposition 4 below, we need one more technical lemma. The following lemma tells us that adding or removing transitions of a round different from  $k$  results in a  $k$ -stutter equivalent path.

**Lemma 6** *Let  $\sigma$  be a configuration and let  $t_1 = (r_1, k_1)$  and  $t_2 = (r_2, k_2)$  be transitions such that  $t_1 t_2$  is applicable to  $\sigma$ . Then, the following holds:*

- (a)  $\text{path}(\sigma, t_1 t_2) \stackrel{\Delta}{=}_k \text{path}(\sigma, t_1)$ , for every  $k \neq k_2$ , and
- (b)  $\text{path}(\sigma, t_1 t_2) \stackrel{\Delta}{=}_k \text{path}(t_1(\sigma), t_2)$ , for every  $k \neq k_1$ .

**Proof** It follows directly from Lemma 3(c) and (e). □

The following proposition shows that every finite schedule can be reordered into a round-rigid one that is stutter equivalent regarding  $\text{LTL}_{\neg\chi}$  formulas over proposition from  $\text{AP}_k$ , for all rounds  $k$ .

**Proposition 4** *For every configuration  $\sigma$  and every finite schedule  $\tau$  applicable to  $\sigma$ , there is a round-rigid schedule  $\tau'$  such that the following holds:*

- (a) Schedule  $\tau'$  is applicable to configuration  $\sigma$ .
- (b)  $\tau'$  and  $\tau$  reach the same configuration when applied to  $\sigma$ , i.e.,  $\tau'(\sigma) = \tau(\sigma)$ .
- (c) For every  $k \in \mathbb{N}_0$ , we have  $\text{path}(\sigma, \tau) \stackrel{\Delta}{=}_k \text{path}(\sigma, \tau')$ .

**Proof** Since  $\tau$  is finite, claim (a) follows from Lemma 4, the second claim follows from Lemma 5(a), and the last one from Lemma 5(b). □

Thus, instead of reasoning about all finite schedules of  $\text{Sys}_{\infty}(\text{TA})$ , it is sufficient to reason about its round-rigid schedules. In the following section, we use this to simplify the verification further, namely to a single-round counter system.

### 6.2 From round-rigid schedules to single-round counter system

The previous section established that every arbitrarily interleaved schedule can be reduced to a sequence of one-round schedules. But these schedules are still defined with respect to the threshold automate framework for multiple rounds from Sect. 3. However, we would like to use the model checker ByMC [24,25] that works on *single-round threshold automata*. As a first step, we define in Definition 6 a specific single-round threshold automaton  $TA^{rd}$  as a function of a model from Sect. 3. Roughly speaking, we focus on one round, but also keep the border locations of the next round, where we add self-loops. Figure 6 represents the single-round threshold automaton associated with the PTA from Fig. 3. For such a threshold automaton, we then define a counter system  $Sys^k(TA^{rd})$ , which can be analyzed with ByMC. After some technical lemmas, we eventually prove Theorem 1 which established stutter equivalence of  $Sys^k(TA^{rd})$  to the system of Sect. 3 with respect to propositions talking about round  $k$ . As final step, Theorem 2 eliminates the round number  $k$  which finally allows us to check specifications for the multi-round system from Sect. 3 using single-round systems.

On a more technical note, we can prove these theorems for specific fairness constraints. We restrict ourselves to fair schedules, that is, those where no transition is applicable forever. We also assume that every fair schedule of a single-round system terminates, i.e., eventually every process reaches a location from  $\mathcal{B}'$ . Under the fairness assumption, we check the latter assumption with ByMC. Moreover, we restrict ourselves to non-blocking threshold automata; that is, we require that in each configuration, each location has at least one outgoing rule unlocked. As we use TAs to model distributed algorithms, this is no restriction: Locations in which no progress should be made unless certain thresholds are reached typically have self-loops that are guarded with `true` (e.g.,  $SR$  and  $SP$ ). Thus, for our benchmarks one can easily check whether they are non-blocking using SMT. (We have to check that there is no evaluation of the variables such that all outgoing rules are disabled.)

We start with the central definition of a single-round threshold automaton that constitutes the link between our theory and the model checker ByMC.

**Definition 6** Given a PTA  $(\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$  or its TA  $(\mathcal{L}, \mathcal{V}, \mathcal{R}_{np}, RC)$ , we define a *single-round threshold automaton*  $TA^{rd} = (\mathcal{L} \cup \mathcal{B}', \mathcal{V}, \mathcal{R}^{rd}, RC)$ , where  $\mathcal{B}' = \{\ell' : \ell \in \mathcal{B}\}$  are copies of border locations, and  $\mathcal{R}^{rd} = (\mathcal{R}_{np} \setminus \mathcal{S}) \cup \mathcal{S}' \cup \mathcal{R}^{loop}$ , where  $\mathcal{R}^{loop} = \{(\ell', \ell', true, \mathbf{0}) : \ell' \in \mathcal{B}'\}$  are self-loop rules at locations from  $\mathcal{B}'$  and

$$\mathcal{S}' = \{(from, \ell', true, \mathbf{0}) : (from, \ell, true, \mathbf{0}) \in \mathcal{S} \text{ with } \ell' \in \mathcal{B}'\}$$

consists of modifications of round-switch rules. Initial locations of  $TA^{rd}$  are the locations from  $\mathcal{B} \subseteq \mathcal{L}$ .

For a  $TA^{rd}$  and a  $k \in \mathbb{N}_0$  we define a counter system  $Sys^k(TA^{rd})$  as the tuple  $(\Sigma^k, I^k, R^k)$ . A configuration is a tuple  $\sigma = (\kappa, \mathbf{g}, \mathbf{p}) \in \Sigma^k$ , where  $\sigma.\kappa : \mathcal{D} \rightarrow \mathbb{N}_0$  defines values of the counters, for  $\mathcal{D} = (\mathcal{L} \times \{k\}) \cup (\mathcal{B}' \times \{k+1\})$ ; and  $\sigma.\mathbf{g} : \Gamma \times \{k\} \rightarrow \mathbb{N}_0$  defines shared variable values; and  $\sigma.\mathbf{p} \in \mathbb{N}_0^{|\Pi|}$  is a vector of parameter values.

Note that by using  $\mathcal{D}$  in the definition of  $\sigma.\kappa$  above, every configuration  $\sigma \in Sys^k(TA^{rd})$  can be extended to a valid configuration of  $Sys_\infty(TA)$ , by assigning zero to all other counters and global variables. In the following, we identify a configuration in  $Sys^k(TA^{rd})$  with its extension in  $Sys_\infty(TA)$ , since they have the same labeling function  $\lambda_k$ , for every  $k \in \mathbb{N}_0$ .

We define  $\Sigma_{\mathcal{B}}^k \subseteq \Sigma^k$ , for a  $k \in \mathbb{N}_0$ , to be the set of all configurations  $\sigma$  where every process is in a location from  $\mathcal{B}$ , and all shared variables are set to zero in  $k$ , formally,  $\sigma.\mathbf{g}[x, k] = 0$  for all  $x \in \Gamma$ , and  $\sum_{\ell \in \mathcal{B}} \sigma.\kappa[\ell, k] = N(\mathbf{p})$ , and  $\sigma.\kappa[\ell, i] = 0$  for all  $(\ell, i) \in \mathcal{D} \setminus (\mathcal{B} \times \{k\})$ . We call these configurations *border configurations for  $k$* . The set of initial configurations  $I^k$  is a subset of  $\Sigma_{\mathcal{B}}^k$ .

We define the transition relation  $R$  as in  $Sys_\infty(TA)$ , i.e., two configurations are in the relation  $R^k$  if and only if they (or more precisely, their above described extensions) are in  $R$ .

If we do not restrict initial configurations, all these systems are identical up to renaming, and this is formalized in the following lemma.

**Lemma 7** *All systems  $Sys^k(TA^{rd})$ ,  $k \in \mathbb{N}_0$ , are isomorphic to each other w.r.t.  $\Sigma_{\mathcal{B}}^k$ , that is, for every  $k \in \mathbb{N}_0$ , if  $I^k = \Sigma_{\mathcal{B}}^k$ , then we have  $Sys^0(TA^{rd}) \cong Sys^k(TA^{rd})$ .*

*Additional assumptions.* Recall that we restrict our attention to fair schedules, and moreover, we assume that all such schedules in  $Sys^0(TA^{rd})$  terminate; that is, they reach a configuration with all processes in  $\mathcal{B}'$ . Formally, we assume that for every fair schedule  $\pi$  in the system  $Sys^0(TA^{rd})$ , it holds that  $\pi \models \mathbf{F} \bigwedge_{\ell \in \mathcal{L}} \kappa[\ell, 0] = 0$ .

We can easily check this with ByMC [26] for the first round and from the following lemma conclude that any other round also terminates.

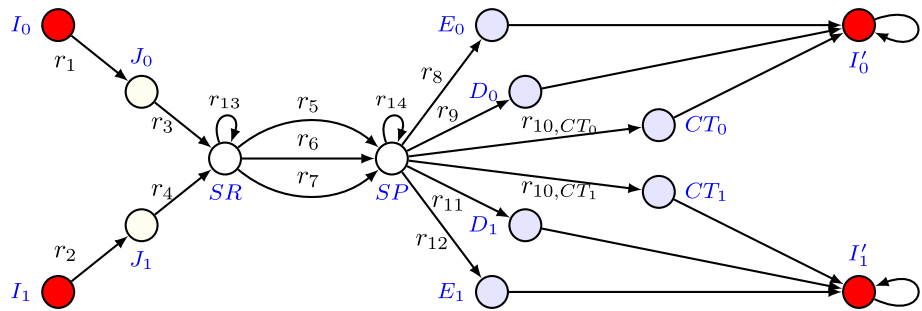
**Lemma 8** *If all fair executions in  $Sys^0(TA^{rd})$  terminate w.r.t.  $\Sigma_{\mathcal{B}}^0$ , then the same holds for  $Sys^k(TA^{rd})$  with respect to  $\Sigma_{\mathcal{B}}^k$ , for every  $k \in \mathbb{N}_0$ .*

**Proof** It follows directly from Lemma 7. □

In order to relate  $Sys_\infty(TA)$  and  $Sys^k(TA^{rd})$ ,  $k \in \mathbb{N}_0$ , we define the set of initial configurations  $I^k$  of  $Sys^k(TA^{rd})$  inductively. First, we define  $I^0$  to be equal to the set  $I$  of initial configurations of the system  $Sys_\infty(TA)$ . Next, for any  $k \geq 1$ , we define  $I^{k+1}$  to be the set of final configurations



**Fig. 6** The single-round threshold automaton  $TA^{rd}$  obtained from PTA in Fig. 3



of  $Sys^k(TA^{rd})$  if we restricted initial configurations of this system to  $I^k$ .

From now on, we fix a TA and a  $TA^{rd}$ , and if not specified differently, for every  $Sys^k(TA^{rd})$  we assume the above definition of  $I^k$ .

**Lemma 9** *If all fair executions of  $Sys^0(TA^{rd})$  w.r.t.  $\Sigma_B^0$  terminate, then for every  $k \in \mathbb{N}_0$ , we have that the set  $I^k$  is well defined and all fair executions of  $Sys^k(TA^{rd})$  terminate (w.r.t.  $I^k$ ).*

**Proof** We prove this claim by induction on  $k \in \mathbb{N}_0$ . The set  $I^0 = I$  is clearly well defined, and since  $I^0 \subseteq \Sigma_B^0$ , by our assumption we have that all fair executions of  $Sys^0(TA^{rd})$  terminate. Since for every  $k \in \mathbb{N}_0$ , we have  $I^k \subseteq \Sigma_B^k$ , by Lemma 8 we have that every fair execution of  $Sys^k(TA^{rd})$  terminates, and therefore,  $I^{k+1}$  is well defined.  $\square$

Let us make here a short digression by giving a property of every  $Sys_\infty(TA)$ , which is necessary for proving Theorem 1.

**Lemma 10** *Let TA be non-blocking, fix a  $k \in \mathbb{N}_0$ , and let  $\sigma$  be a configuration in  $Sys_\infty(TA)$  with a non-empty border location in round  $k+1$ , i.e.,  $\bigvee_{\ell \in B} \sigma.\kappa[\ell, k+1] \geq 1$ . Then, for every configuration  $\sigma'$  reachable from  $\sigma$ , there is a transition  $t = (r, f, k_1)$  with  $k_1 > k$  that is applicable to  $\sigma'$ .*

**Proof** Let  $\sigma$  be a configuration with a non-empty border location in round  $k+1$ , and let  $\sigma'$  be a configuration reachable from  $\sigma$ . Assume by contradiction that there is no transition  $t = (r, f, k_1)$  with  $k_1 > k$  that is applicable to  $\sigma'$ . Recall that by our assumption, every location has at least one unlocked outgoing rule. Thus, it must hold that for every location  $\ell$  we have that  $\sigma'.\kappa[\ell, k_1] = 0$ , for every  $k_1 > k$ . This is a contradiction with the assumption that  $\sigma'$  is reachable from  $\sigma$  and  $\bigvee_{\ell \in B} \sigma.\kappa[\ell, k+1] \geq 1$ .  $\square$

**Theorem 1** *If TA is non-blocking, and if all fair executions of  $Sys^0(TA^{rd})$  w.r.t.  $\Sigma_B^0$  terminate, then for every  $k \in \mathbb{N}_0$ , we have  $Sys^k(TA^{rd}) \triangleq_k Sys_\infty(TA)$ , i.e., the two systems are stutter equivalent w.r.t.  $AP_k$ .*

**Proof** We prove the statement by induction on  $k \in \mathbb{N}_0$ .  
**BASE CASE.** Let us first show that  $Sys^0(TA^{rd}) \triangleq_0 Sys_\infty(TA)$ .

( $\Rightarrow$ ) Let  $\pi = \text{path}(\sigma, \tau)$  be a path in  $Sys^0(TA^{rd})$ . We need to find a path  $\pi'$  from  $Sys_\infty(TA)$ , such that  $\pi \triangleq_k \pi'$ .

If  $\tau = t_1 t_2 \dots$ , then every transition  $t_i$  either exists also in TA, or it is a self-loop at the copy of a border location. Using this, we construct a schedule  $\tau' = t'_1 t'_2 \dots$  in the following way.

For every  $i \in \mathbb{N}$ , if  $t_i$  exists in TA, then we define  $t'_i$  to be exactly  $t_i$ , and if  $t'_i$  is a self-loop at an  $\ell' \in B'$ , then Lemma 10 gives us that there exists a transition  $\tilde{t}_i$  from a round greater than 0 that is applicable to the current configuration, and we define  $t'_i = \tilde{t}_i$ . Thus,  $\tau' = t'_1 t'_2 \dots$  is obtained from  $\tau$  by removing certain self-looping transitions and adding transitions of rounds greater than 0. By Lemma 6, we have  $\text{path}(\sigma, \tau') \triangleq_0 \text{path}(\sigma, \tau)$ .

Now, we have that  $\pi' = \text{path}(\sigma, \tau') \triangleq_0 \text{path}(\sigma, \tau) = \pi$ .

( $\Leftarrow$ ) Let now  $\pi = \text{path}(\sigma, \tau)$  be a path in  $Sys_\infty(TA)$ . We construct a path  $\pi' = \text{path}(\sigma', \tau')$  from  $Sys^k(TA^{rd})$  such that  $\pi \triangleq_k \pi'$ . Since  $I = I^0$ , we define  $\sigma' = \sigma$ .

Let  $\tau_0$  be the projection of  $\tau$  to round 0. There are two cases to consider. First, if  $\tau$  and  $\tau_0$  are either both infinite or both finite schedules, then by Lemma 6, they yield stutter equivalent paths starting in  $\sigma$ . Observe that by Lemma 3, counters  $\kappa[\ell, 0]$  only change due to transitions for round 0, so that the applicability of  $\tau_0$  to  $\sigma$  follows from the applicability of  $\tau$ . Thus, in these cases we define  $\tau'$  to be  $\tau_0$ .

Second, we show the construction of  $\tau'$  in the case when  $\tau$  is an infinite schedule and  $\tau_0$  is finite. In this case, we construct  $\tau'$  as infinite extension of  $\tau_0$  as follows: Note that since TA is non-blocking, there must exist at least one location  $\ell \in B_1$  that is non-empty after executing  $\tau_0$  from  $\sigma$ , i.e.,  $\tau_0(\sigma).\kappa[\ell, 1] \geq 1$ . This must also be the case in  $Sys^0(TA^{rd})$ , with a difference that the non-empty location belongs to  $B'$ , since  $B'$  plays the role of  $B_1$ . If  $r$  is the self-looping rule at  $\ell$ , then we obtain  $\tau'$  by concatenating infinitely many transitions  $(r, 1)$  to  $\tau_0$ , i.e.,  $\tau' = \tau_0(r, 1)^\omega$ . Transition  $(r, 1)$  does not affect atomic propositions of round 0, and thus, we have stutter equivalence by Lemma 6.

**INDUCTION STEP.** Let us assume that  $Sys^i(TA^{rd}) \triangleq_i Sys_\infty(TA)$  for every  $0 \leq i < k$ , and let us prove that the claim holds for  $k$ .

( $\Rightarrow$ ) Let  $\pi = \text{path}(\sigma, \tau)$  be a path in  $Sys^k(TA^{rd})$ . We need to find a path  $\pi'$  from  $Sys_\infty(TA)$ , such that  $\pi \triangleq_k \pi'$ .



Note that  $\sigma \in I^k$ . By definition of  $I^k$ , there exist a configuration  $\sigma_0 \in I^0$  and schedules  $\tau_1, \tau_2, \dots, \tau_{k-1}$ , such that every  $\tau_i$  contains only transitions of round  $i$ , and  $\tau_1 \tau_2 \dots \tau_{k-1}(\sigma_0) = \sigma$ . Since no transition here is of round  $k$ , it follows from Lemma 6 that we have that  $\text{path}(\sigma_0, \tau_1 \tau_2 \dots \tau_{k-1}) \stackrel{\triangle}{=} \text{path}(\sigma, \varepsilon)$ , where  $\varepsilon$  is the empty schedule. This path will be a prefix of  $\pi'$ .

If  $\tau = t_1 t_2 \dots$ , we use the same strategy as in the base case to define  $\tau' = t'_1 t'_2 \dots$  such that  $\text{path}(\sigma, \tau') \stackrel{\triangle}{=} \text{path}(\sigma, \tau)$ .

Now, we have that  $\pi' = \text{path}(\sigma_0, \tau_1 \tau_2 \dots \tau_{k-1} \tau') \stackrel{\triangle}{=} \text{path}(\sigma, \varepsilon \tau) = \pi$ .

( $\Leftarrow$ ) Let now  $\pi = \text{path}(\sigma, \tau)$  be a path in  $\text{Sys}_\infty(\text{TA})$ . We construct a path  $\pi'$  from  $\text{Sys}^k(\text{TA}^{\text{rd}})$  such that  $\pi \stackrel{\triangle}{=} \pi'$ .

As we assume that all fair executions of  $\text{Sys}^0(\text{TA}^{\text{rd}})$  terminate w.r.t.  $\Sigma_B^0$ , by Lemma 9, for  $0 \leq i < k$ , the set  $I^i$  is well defined and all the fair executions of  $\text{Sys}^i(\text{TA}^{\text{rd}})$  terminate. By the induction hypothesis, we know that  $\text{Sys}^i(\text{TA}^{\text{rd}}) \stackrel{\triangle}{=} \text{Sys}_\infty(\text{TA})$ . Together, this gives us that all rounds  $i$ , with  $0 \leq i < k$ , terminate in  $\text{Sys}_\infty(\text{TA})$ . Thus, every execution of  $\text{Sys}_\infty(\text{TA})$  has a finite prefix that contains all its transitions of rounds less than  $k$ .

Let  $\tau_{\text{pre}}$  be such a prefix of  $\tau = \tau_{\text{pre}} \tau_{\text{suf}}$ . Because  $\tau_{\text{pre}}$  is finite, we may invoke Proposition 4, from which follows that there exist schedules  $\tau_0, \tau_1, \dots, \tau_{k-1}, \tau_{\geq k}$  such that every  $\tau_i, 0 \leq i < k$  contains only round  $i$  transitions,  $\tau_{\geq k}$  contains transitions of rounds at least  $k$ , and the schedule  $\tau_0 \tau_1 \dots \tau_{k-1} \tau_{\geq k}$  is applicable to  $\sigma$ , leads to  $\tau_{\text{pre}}(\sigma)$  when applied to  $\sigma$ , and

$$\text{path}(\sigma, \tau_0 \tau_1 \dots \tau_{k-1} \tau_{\geq k} \tau_{\text{suf}}) \stackrel{\triangle}{=} \text{path}(\sigma, \tau_{\text{pre}} \tau_{\text{suf}}). \tag{13}$$

As  $\sigma \in I = I^0$ , the existence of schedules  $\tau_0, \tau_1, \dots, \tau_{k-1}$  confirms that  $\sigma' = \tau_0 \tau_1 \dots \tau_{k-1}(\sigma)$  is in  $I^k$ . Next, we apply the strategy from the base case to construct  $\tau'$  from  $\tau_{\geq k} \tau_{\text{suf}}$ , by projecting it to round  $k$ , such that

$$\text{path}(\sigma', \tau_{\geq k} \tau_{\text{suf}}) \stackrel{\triangle}{=} \text{path}(\sigma', \tau'). \tag{14}$$

By (13) and (14), we get  $\pi' = \text{path}(\sigma, \tau_0 \tau_1 \dots \tau_{k-1} \tau') \stackrel{\triangle}{=} \text{path}(\sigma, \tau_{\text{pre}} \tau_{\text{suf}}) = \pi$ .  $\square$

Note that different rounds might have different sets of initial configurations. Since our goal is to explore all rounds, we need to consider all possible initial configurations of all rounds. We do this by projecting them to the first round, creating their union, and checking the first round w.r.t. that union. Still, in our benchmarks all rounds have the same set of initial configurations, so the union coincides with the set of initial configurations of the first round.

By Lemma 7, for every  $k \in \mathbb{N}_0$  and every  $\sigma \in \Sigma_B^k$ , there is a corresponding configuration  $\sigma' \in \Sigma_B^0$  obtained from  $\sigma$  by renaming the round  $k$  to round 0. Let  $f_k$  be the renaming function, i.e.,  $\sigma' = f_k(\sigma)$ . Let us define  $\Sigma^u \subseteq \Sigma_B^0$  to be the

union of all renamed initial configurations from all rounds, i.e.,  $\{f_k(\sigma) : k \in \mathbb{N}_0, \sigma \in I^k\}$ .

The following theorem gives us a method for checking *qform*-formulas of Table 2 with one round quantifier, that is, formulas of the form  $\forall k \in \mathbb{N}_0. \mathbf{A} \varphi[k]$ , where  $\varphi[k]$  is a *tform*-formula.

**Theorem 2** *Let TA be non-blocking, and let all fair executions of  $\text{Sys}^0(\text{TA}^{\text{rd}})$  w.r.t.  $\Sigma_B^0$  terminate. If  $\varphi[k]$  is a tform-formula over  $\text{AP}_k$  for a round variable  $k \in \mathbb{N}_0$ , the following are equivalent:*

- (A)  $\text{Sys}_\infty(\text{TA}) \models \forall k \in \mathbb{N}_0. \mathbf{A} \varphi[k]$
- (B)  $\text{Sys}^0(\text{TA}^{\text{rd}}) \models \mathbf{A} \varphi[0]$  with respect to initial configurations  $\Sigma^u$ .

Let us first give an intuitive explanation. The theorem is proved using the following arguments. In statement (A), the universal quantification over  $k$  corresponds to the definition of  $\Sigma^u$  as union, over all rounds, of projections of all reachable initial configurations of that round.

For the implication (A)  $\rightarrow$  (B), note that an initial configuration in  $\Sigma^u$  is not necessarily initial in round 0, so that one cannot *a priori* take  $k = 0$ . Let us explain how to extend an execution of round  $k$  into an infinite execution in  $\text{Sys}_\infty(\text{TA})$ . By termination, all rounds up to  $k-1$  terminate, so that there is execution that reaches a configuration where all processes are in initial locations of round  $k$ . The executions of round  $k$  mimics the ones of round 0 (modulo the round number). Finally, the non-blocking assumption is required to be always able to extend to infinite executions after round  $k$  is terminated.

Implication (B)  $\rightarrow$  (A) exploits the fact that all rounds are equivalent up to renaming of round numbers (with the exception of possible initial configurations).

**Proof** Let us first formally prove that (A)  $\rightarrow$  (B). Assume by contradiction that (A) holds, but (B) does not, that is,  $\text{Sys}^0(\text{TA}^{\text{rd}}) \models \mathbf{E} \neg \varphi[0]$  w.r.t. initial configurations  $\Sigma^u$ . This means there is a path  $\pi = \text{path}(\sigma, \tau)$  such that  $\sigma \in \Sigma^u$  and  $\pi \models \neg \varphi[0]$ . Since  $\sigma \in \Sigma^u$ , there is a  $k \in \mathbb{N}_0$  and a  $\sigma_k \in I^k$  such that  $\sigma = f_k(\sigma_k)$ . From Lemma 7, we know that  $\text{Sys}^0(\text{TA}^{\text{rd}}) \cong \text{Sys}^k(\text{TA}^{\text{rd}})$ , and thus, there is a schedule  $\tau_k$  in  $\text{Sys}^k(\text{TA}^{\text{rd}})$  such that  $\text{path}(\sigma_k, \tau_k) \models \neg \varphi[k]$ . Now, by Theorem 1 there must be a path  $\pi'$  from  $\text{Sys}_\infty(\text{TA})$  such that  $\text{path}(\sigma_k, \tau_k) \stackrel{\triangle}{=} \pi'$ . By Proposition 3, we know that  $\pi' \models \neg \varphi[k]$ , and thus,  $\text{Sys}_\infty(\text{TA}) \models \exists k \in \mathbb{N}_0. \mathbf{E} \neg \varphi[k]$ . This is in contradiction with our assumption that (A) holds, which proves one direction of the statement.

Next, we prove the other direction, namely (B)  $\rightarrow$  (A). Assume again by contradiction that (B) holds, but (A) does not; that is, there is a  $k \in \mathbb{N}_0$  and a path  $\pi = \text{path}(\sigma, \tau)$  in  $\text{Sys}_\infty(\text{TA})$  such that  $\pi \models \neg \varphi[k]$ . By Theorem 1, we know that there exists a path  $\pi' = \text{path}(\sigma', \tau')$  in  $\text{Sys}^k(\text{TA}^{\text{rd}})$  with

$\pi \stackrel{\Delta}{=} \pi'$ , and then by Proposition 3 also  $\pi' \models \neg\varphi[k]$ . Finally, by Lemma 7 there is an equivalent path  $\pi_0$  in  $\text{Sys}^0(\text{TA}^{\text{rd}})$  starting in  $f_k(\sigma')$ . Then, we have that  $\pi_0 \models \neg\varphi[0]$ , and since  $f_k(\sigma') \in \Sigma^u$ , we know that  $\text{Sys}^0(\text{TA}^{\text{rd}}) \models \mathbf{E} \neg\varphi[0]$  w.r.t. initial configurations  $\Sigma^u$ . This contradicts the assumption that (B) holds and therefore concludes the other direction of the proof.  $\square$

In Sect. 4, we showed how to reduce our specifications to formulas of the form  $\forall k \in \mathbb{N}_0. \mathbf{A} \varphi[k]$ , where  $\varphi[k]$  is a *tform*-formula of Table 2. Theorem 2 deals with exactly this type of formulas, and therefore, it allows us to check specifications using single-round systems instead of  $\text{Sys}_\infty(\text{TA})$ .

### 7 Round-rigid probabilistic termination

We start by defining two conditions that are sufficient to establish round-rigid probabilistic termination (under round-rigid adversaries). Condition (C1) states the existence of a positive probability lower-bound for all processes ending round  $k$  with equal final values. Condition (C2) states that if all correct processes start round  $k$  with the same value, then they all will decide on that value in that round.

(C1) For every parameters  $\mathbf{p}$ , there is a bound  $p \in (0, 1]$ , such that for every round-rigid adversary  $\mathbf{a}$ , every  $k \in \mathbb{N}_0$ , and every configuration  $\sigma_k$  with parameters  $\mathbf{p}$  that is initial for round  $k$ , it holds that

$$\mathbb{P}_{\mathbf{a}}^{\sigma_k} \left( \bigvee_{v \in \{0,1\}} \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell, k] = 0 \right) \right) \geq p.$$

(C2) For all  $v \in \{0, 1\}$ ,  $\forall k \in \mathbb{N}_0. \mathbf{A} \left( \mathbf{G} \bigwedge_{\ell \in \mathcal{I}_{1-v}} \kappa[\ell, k] = 0 \rightarrow \mathbf{G} \bigwedge_{\ell' \in \mathcal{F} \setminus \mathcal{D}_v} \kappa[\ell', k] = 0 \right)$ .

Combining (C1) and (C2), under every round-rigid adversary, from any initial configuration of round  $k$ , the probability that all correct processes decide before end of round  $k+1$  is at least  $p$ . Thus, the probability not to decide within  $2n$  rounds is at most  $(1-p)^n$ , which tends to 0 when  $n$  tends to infinity. This reasoning follows the arguments of the hand-written proof [1]. More generally, such an analysis is standard and appears in many contexts to prove that an event is almost-sure (for instance, almost-sure termination of probabilistic programs [37]), thanks to the so-called zero-one law (see, e.g., [20]).

**Proposition 5** *If  $\text{Sys}_\infty(\text{PTA}) \models (C1)$  and  $\text{Sys}_\infty(\text{PTA}) \models (C2)$ , then  $\text{Sys}_\infty(\text{PTA}) \models (3)$ .*

**Proof** Fix a  $\mathbf{p} \in \mathbf{P}_{RC}$ , an initial configuration  $\sigma_0$ , and a round-rigid adversary  $\mathbf{a}$ .

Two possible options may occur along a path  $\pi \in \text{paths}(\sigma_0, \mathbf{a})$ : (i) Either round 0 ends with a final configuration in which all processes have the same value, say  $v$ , or (ii) round 0 ends with a final configuration with both values present.

(i) In this case, we have  $\pi \models \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_{1-v}} \kappa[\ell, 0] = 0 \right)$ , and by (C1), for  $k = 0$ , the probability that this case happens is at least  $p$ . Then, by Lemma 1 we also have  $\pi \models \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{I}_{1-v}} \kappa[\ell, 1] = 0 \right)$ . Using (C2), in this case all processes decide value  $v$  in round 1.

(ii) The probability that the second case happens is at most  $1-p$ . In this case, round 1 starts with an initial configuration  $\sigma_1$  with both initial values 0 and 1. From  $\sigma_1$  under  $\mathbf{a}$ , by the same reasoning as from  $\sigma_0$ , at the end on the round 1 we have the analogous two cases, and all processes decide in round 2 with probability at least  $p$ .

Iterating this reasoning, almost surely all processes eventually decide. Let us formally explain this iteration. Let  $\sigma_0$  be an initial configuration, and let  $\mathbf{a}$  be a round-rigid adversary. For a  $k \in \mathbb{N}$ , consider the event  $\mathcal{E}_k$ : From  $\sigma_0$  and under  $\mathbf{a}$ , not every process decides in the first  $k$  rounds. In particular, at the end of every round  $i < k$  it is not the case that everyone decides. By the reasoning above, namely case (ii) for round  $i$ , this happens with probability at most  $(1-p)$ . Therefore, for  $k$  rounds we have  $\mathbb{P}_{\mathbf{a}}^\sigma(\mathcal{E}_k) \leq (1-p)^k$ . The limit when  $k$  tends to infinity yields that the probability for not having round-rigid probabilistic termination is 0. This is equivalent to the required formula (3).  $\square$

Observe (C2) is a non-probabilistic property of the same form as (5), so that we can check (C2) using the method of Sect. 6.

In the rest of this section, we detail how to reduce the verification of (C1), to a verification task that can be handled by ByMC. First observe that (C1) contains a single-round variable, and recall that we restrict to round-rigid adversaries, so that it is sufficient to check them (omitting the round variables) on the single-round system. We introduce analogous objects as in the non-probabilistic case:  $\text{PTA}^{\text{rd}}$  (analogously to Definition 6), and its counter system  $\text{Sys}(\text{PTA}^{\text{rd}})$ .

#### 7.1 Reducing probabilistic to non-probabilistic specifications

Since probabilistic transitions end in final locations, they cannot appear on a cycle in  $\text{PTA}^{\text{rd}}$ . Thus, in each round, each process may take at most one coin toss. Recall that  $N(\mathbf{p})$  models the number of processes in the system. Then, for fixed parameter valuation  $\mathbf{p}$ , any path contains at most  $N(\mathbf{p})$  probabilistic transitions, and its probability is therefore uniformly lower-bounded. As a consequence, writing  $I_{\mathbf{p}}$  for the set of initial configurations with parameter valuation  $\mathbf{p}$ , we have:

**Lemma 11** *Let  $\mathbf{p} \in \mathbf{P}_{RC}$  be a parameter valuation. In  $\text{Sys}(\text{PTA}^{\text{rd}})$ , for every LTL formula  $\varphi$  over atomic proposition AP, the following two statements are equivalent:*

- (a)  $\exists p > 0, \forall \sigma \in I_{\mathbf{p}}, \forall a \in \mathcal{A}^R. \mathbb{P}_a^\sigma(\varphi) \geq p,$
- (b)  $\forall \sigma \in I_{\mathbf{p}}, \forall a \in \mathcal{A}^R, \exists \pi \in \text{paths}(\sigma, a). \pi \models \varphi.$

**Proof** Fix parameters  $\mathbf{p} \in \mathbf{P}_{RC}$ .

The implication from top to bottom is trivial: If a probability is lower bounded by a positive constant, then there must be at least a path satisfying that property. It is thus sufficient to prove the bottom to top implication.

Assume that from every initial configuration  $\sigma$  with parameter values  $\mathbf{p}$ , and for all round-rigid adversaries  $a$ , there exists a path  $\pi \in \text{paths}(\sigma, a)$  in  $\text{Sys}(\text{PTA}^{\text{rd}})$  such that  $\pi \models \varphi$ . Independently of  $\sigma$  and  $a$ , our assumption that non-Dirac transitions may only happen at the end of PTA yields that any path contains at most  $N(\mathbf{p})$  non-Dirac transitions. If  $\delta$  is the smallest probability value appearing on such transitions, the probability of any path in  $\text{Sys}(\text{PTA}^{\text{rd}})$  is therefore lower-bounded by  $\delta^{N(\mathbf{p})}$ . Therefore, we can set  $p = \delta^{N(\mathbf{p})}$ , which only depends on PTA and  $\mathbf{p}$ .  $\square$

### 7.2 Verifying (C1) on a non-probabilistic TA

Applying Lemma 11, proving (C1) is equivalent to proving the following property on  $\text{Sys}(\text{PTA}^{\text{rd}})$

$$\forall \sigma \in I_{\mathbf{p}}, \forall a \in \mathcal{A}^R, \exists \pi \in \text{paths}(\sigma, a). \pi \models \bigvee_{v \in \{0,1\}} \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell] = 0 \right). \tag{15}$$

In the sequel, we explain how to reduce the verification of (15) to checking the simpler formula

$$\mathbf{A} \bigvee_{v \in \{0,1\}} \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell] = 0 \right)$$

on a single-round non-probabilistic TA obtained from  $\text{PTA}^{\text{rd}}$ .

As in Sect. 6, it is possible to modify  $\text{PTA}^{\text{rd}}$  into a non-probabilistic TA, by replacing probabilistic choices by non-determinism. Still, the quantifier alternation of (15) (universal over initial configurations and adversaries vs. existential on paths) is not in the fragment handled by ByMC [26]. Once an initial configuration  $\sigma$  and an adversary  $a$  are fixed, the remaining branching is solely induced by non-Dirac transitions. By assumption, these transitions lead to final locations only, to both  $\mathcal{F}_0$  and  $\mathcal{F}_1$ , and under round-rigid adversaries, they are the last transitions to be fired. To prove (15), it is sufficient to prove that all processes that fire only Dirac transitions will reach final locations of the same type ( $\mathcal{F}_0$  or  $\mathcal{F}_1$ ). If this is the case, then the existence of a

path corresponds to all non-Dirac transitions being resolved in the same way. This allows us to remove the non-Dirac transitions from the model as follows.

Given a  $\text{PTA}^{\text{rd}}$ , we now define a threshold automaton  $\text{TA}^{\text{m}}$  with locations  $\mathcal{L}$  (without  $\mathcal{B}'$ ) such that for every non-Dirac rule  $r = (\text{from}, \delta_{to}, \varphi, \mathbf{u})$  in PTA, all locations  $\ell$  with  $\delta_{to}(\ell) > 0$  are merged into a new location  $\ell^{\text{mrg}}$  in  $\text{TA}^{\text{m}}$ . Note that this location must belong to  $\mathcal{F}$ . Naturally, instead of a non-Dirac rule  $r$  we obtain a Dirac rule  $(\text{from}, \ell^{\text{mrg}}, \varphi, \mathbf{u})$ . Also, we add self-loops at all final locations. Figure 7 illustrates the transformation on our running example from Fig. 3. The new final location  $\ell^{\text{mrg}}$  can be understood as an abstract state that abstracts the possible coin-toss outcomes; it belongs neither to  $\mathcal{F}_0$  nor  $\mathcal{F}_1$ .

Paths in  $\text{Sys}(\text{TA}^{\text{m}})$  correspond to prefixes of paths in  $\text{Sys}(\text{PTA}^{\text{rd}})$ . In  $\text{Sys}(\text{TA}^{\text{m}})$ , from a configuration  $\sigma$ , an adversary  $a$  yields a unique path; that is,  $\text{paths}(\sigma, a)$  is a singleton set. Thus, the existential quantifier from (15) can be replaced by the universal one.

**Lemma 12** *Let  $k \in \mathbb{N}_0$ , let  $\sigma$  be an initial configuration of  $\text{Sys}^k(\text{PTA}^{\text{rd}})$ , and let  $a$  be a round-rigid adversary. Then, the following statements are equivalent:*

- (a) *there exists  $\pi \in \text{paths}(\sigma, a)$  in  $\text{Sys}^k(\text{PTA}^{\text{rd}})$  such that  $\pi \models \bigvee_{v \in \{0,1\}} \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_v(\text{PTA}^{\text{rd}})} \kappa[\ell, k] = 0 \right);$*
- (b) *for every  $\pi \in \text{paths}(\sigma, a)$  in  $\text{Sys}^k(\text{TA}^{\text{m}})$ ,  $\pi \models \bigvee_{v \in \{0,1\}} \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_v(\text{TA}^{\text{m}})} \kappa[\ell, k] = 0 \right).$*

**Proof** Paths in  $\text{Sys}^k(\text{TA}^{\text{m}})$  are mapped uniquely to prefixes of paths in  $\text{Sys}^k(\text{PTA}^{\text{rd}})$ . Moreover, since every  $\text{paths}(\sigma, a)$  in  $\text{Sys}^k(\text{TA}^{\text{m}})$  is a singleton set, existential and universal quantifications coincide.  $\square$

By Lemma 12, property (15) on  $\text{PTA}^{\text{rd}}$  is equivalent to  $\mathbf{A} \bigvee_{v \in \{0,1\}} \mathbf{G} \left( \bigwedge_{\ell \in \mathcal{F}_v} \kappa[\ell] = 0 \right)$  on  $\text{Sys}(\text{TA}^{\text{m}})$ . The latter can be checked automatically by ByMC, allowing us to prove (C1).

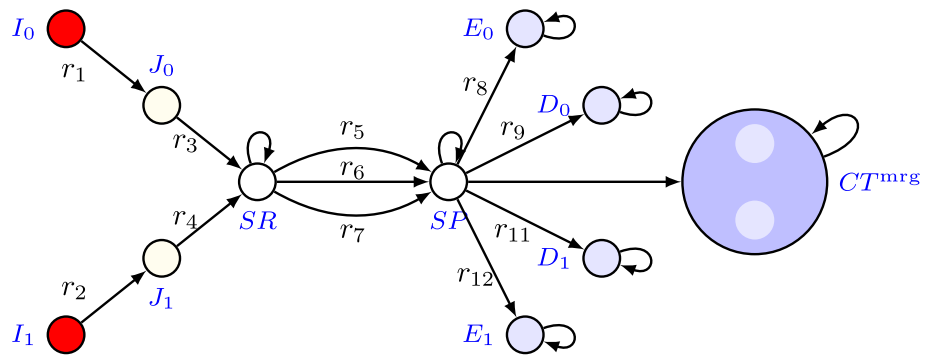
## 8 Experiments

We have applied the approach presented in Sects. 4–7 to five randomized fault-tolerant consensus algorithms.

(The benchmarks and the instructions on running the experiments are available from: <https://forsyte.at/software/bymc/artifact-rand-cons/>)

1. Randomized consensus by Ben-Or [4, Protocol 1], with two kinds of crashes: clean crashes (ben-or-cc), for which a process either sends to all processes or none, and dirty crashes (ben-or-dc), for which a process may send to a

**Fig. 7** A one-round non-probabilistic threshold automaton  $TA^m$  obtained from the PTA from Fig. 3



subset of processes. This algorithm works correctly when  $n > 2t$ .

2. Randomized Byzantine consensus by Ben-Or [4, Protocol 2] (ben-or-byz). This algorithm tolerates  $t$  Byzantine faults when  $n > 5t$ .
3. Randomized consensus by Bracha [11, Protocol 2] (rabc-c). It runs as a high-level algorithm together with a low-level broadcast algorithm that reduces the impact of Byzantine faults into “little more than fail-stop (faults)”. We check only the high-level algorithm for clean crashes.
4.  $k$ -set agreement for crash faults by Mostéfaoui et al. [38] (kset), for  $k = 2$ . This algorithm works in the presence of clean crashes when  $n > 3t$ .
5. Randomized Byzantine one-step consensus by Song and van Renesse [42] (rs-bosco). This algorithm tolerates Byzantine faults when  $n > 3t$ , and it terminates fast when  $n > 7t$  or  $n > 5t$  and  $f = 0$ .

Following the reduction approach of Sects. 4–7, for each benchmark, we have encoded two versions of one-round threshold automata: an N-automaton that models a coin toss by a non-deterministic choice in a coin-toss location (similar to  $TA_{PTA}$  in our framework) and is used for the non-probabilistic reasoning, and a P-automaton that never leaves the coin-toss location and which is used to prove round-rigid probabilistic termination (similar to  $TA^{td}$  in our framework). Both automata are given as the input to Byzantine Model Checker (ByMC) [26], which implements the parameterized model checking techniques for safety [23] and liveness [24] of counter systems of threshold automata.

Both automata follow the pattern shown in Fig. 3: Processes start in one of the initial locations (e.g.,  $J_0$  or  $J_1$ ), progress by switching locations and increasing shared variables and end up in a location that corresponds to a decision (e.g.,  $D_0$  or  $D_1$ ), an estimate of a decision (e.g.,  $E_0$  or  $E_1$ ), or a coin toss (CT).

Table 3 summarizes the properties that were verified in our experiments. Given the set of all possible locations  $\mathcal{L}$ , a subset  $Y = \{\ell_1, \dots, \ell_m\} \subseteq \mathcal{L}$  of locations, and the distinguished crashed location  $CR \in \mathcal{L}$ , we use the shorthand notation:

$EX\{\ell_1, \dots, \ell_m\}$  for  $\bigvee_{\ell \in Y} \kappa[\ell] \neq 0$  and  $ALL\{\ell_1, \dots, \ell_m\}$  for  $\bigwedge_{\ell \in \mathcal{L} \setminus Y} (\kappa[\ell] = 0 \vee \ell = CR)$ . For rs-bosco and kset, instead of checking  $S1$ , we check  $S1'$  and  $S1''$ .

Table 4 presents the computational results of our experiments: Column  $|\mathcal{L}|$  shows the number of automata locations, column  $|\mathcal{R}|$  shows the number of automata rules, column  $|\mathcal{S}|$  shows the number of SMT queries (which depends on the structure of the automaton and the specification), and column *time* shows the computation times—either in seconds or in the format HH:MM. As the N-automata have more rules than the P-automata, column  $|\mathcal{R}|$  shows the figures for N-automata. Benchmarks 1–5 need 30–170 MB, whereas rs-bosco needs up to 1.5 GB per CPU.

The benchmark rs-bosco is a challenge for the technique of Konnov et al. [24]: Its threshold automaton has 12 threshold guards that can change their values almost in any order. Additional combinations are produced by the temporal formulas. Although ByMC reduces the number of combinations by analyzing dependencies between the guards, it still produces between  $11!$  and  $14!$  SMT queries. Hence, we ran the experiments for rs-bosco on 1024 CPU cores of Grid5000 and gave the wall time results in Table 4. (To find the total computing time, multiply wall time by 1024.) ByMC timed out on the property  $S4$  after 1 day (shown as TO).

For all other benchmarks in Table 4, ByMC has reported that the specifications hold. By changing  $n > 3t$  to  $n > 2t$ , we found that rabc-cr can handle more faults. (The original  $n > 3t$  was needed to implement the underlying communication structure which we assume given in the experiments.) In other cases, whenever we changed the parameters, that is, increased the number of faults beyond the known bound, the tool reported an expected counterexample.

## 9 Related work

Initial research on computer-aided verification of non-randomized fault-tolerant distributed algorithms considered the verification of such systems in the concrete (that is, non-parameterized) case, where the number of participating



**Table 3** Properties verified in our experiments for value 0

Label	Name	Automaton	Formula
S1	agreement_0	N	$\mathbf{A G} (\neg \text{Ex}\{D0\}) \vee \mathbf{G} (\neg \text{Ex}\{D1, E1\})$
S2	validity_0	N	$\mathbf{A ALL}\{J0\} \rightarrow \mathbf{G} (\neg \text{Ex}\{D1, E1\})$
S3	completeness_0	N	$\mathbf{A ALL}\{J0\} \rightarrow \mathbf{G} (\neg \text{Ex}\{D1, E1\})$
S4	round-term	N	$\mathbf{A fair} \rightarrow \mathbf{F ALL}\{D0, D1, E0, E1, CT\}$
S5	decide-or-flip	P	$\mathbf{A fair} \rightarrow \mathbf{F} (\text{ALL}\{D0, E0, CT\} \vee \text{ALL}\{D1, E1, CT\})$
S1'	sim-agreement	N	$\mathbf{A G} (\neg \text{Ex}\{D0, E0\} \vee \neg \text{Ex}\{D1, E1\})$
S1''	2-agreement	N	$\mathbf{A G} (\neg \text{Ex}\{D0, E0\} \vee \neg \text{Ex}\{D1, E1\} \vee \neg \text{Ex}\{D2, E2\})$

**Table 4** The experiments for first 5 rows were run on a single computer (Apple MacBook Pro 2018, 16GB)

Automaton Name	L   R		S1/S1'/S1''		S2		S3		S4		S5	
	L	R	S	Time	S	Time	S	Time	S	Time	S	Time
Ben-or-cc	10	27	9	1	5	0	5	0	5	0	5	0
Ben-or-dc	10	32	9	1	5	1	5	0	5	0	5	1
Ben-or-byz	9	18	3	1	2	0	2	0	2	0	2	1
Rabc-cr	11	31	9	0	5	1	5	1	5	0	5	0
Kset	13	58	65	3	65	17	65	12	65	39	65	40
Rs-bosco	19	48	156M	03:21	156M	03:02	156M	03:21	TO	TO	156M	03:43

The experiments for last row (rs-bosco) were run in Grid5000 on 32 nodes (2 CPUs Intel Xeon Gold 6130, 16 cores/CPU, 192 GB). Wall times are given

processes is set to a small number (e.g., 4–10), and the correctness is automatically checked for these small instances, e.g., [12,22,43,49]. Recently, the parameterized case also gained much attention: The problem has been addressed by model checking [24,36,44], deductive verification [5,16,40], and interactive theorem proving [13,21,51]. These verification approaches address the parameterized setting, where the number of processes is a parameter. Verification for all values of the parameter is typically undecidable [2,9,18,45]. Distributed algorithm has also been verified for small systems (e.g., for 4–10 processes) in, e.g., [12,22,43,48].

For randomized distributed algorithms, the work in [29, 31] does probabilistic reasoning with the probabilistic model checker PRISM [30] for small systems (10–20 processes). Verification of safety for any number of processes was done using Cadence SMV.

Randomized distributed algorithms have also been addressed in a process algebra approach [46]. Similarly to our work, the authors exploit the communication-closure property of standard distributed algorithms, in order to design a purely syntactic partial-order state space reduction. The methodology is illustrated on a randomized mutual exclusion algorithm, but, in contrast to our contribution, no tool support is provided.

A few contributions address automated verification of probabilistic parameterized systems [6,34,35,41,52]. In contrast to these, our processes are not finite state, due to the round numbers and parameterized guards. The seminal work by Pnueli and Zuck [41] requires shared variables to be

bounded and cannot use arithmetic thresholds different from 1 and  $n$ . Algorithms for well-structured transition systems [6] do not directly apply to multi-parameter systems produced by probabilistic threshold automata. Approaches based on regular model checking [34,35] cannot handle arithmetic resilience conditions such as  $n > 3t$ , nor unbounded shared variables. Recently, in [52] an abstraction-based verification approach has been presented that exploits a reduction of almost sure properties for a parameterized MDP to model checking an abstract finite-state system with fairness.

The authors of Nestmann et al. [39] highlight problems on the notion of rounds in asynchronous distributed algorithms. The central problem is that the notion of a round provides some abstraction of time, which might not coincide with the notion of time that comes from the length of the prefix in asynchronous interleavings. In this paper, for algorithms that can be encoded in our iterated model, we show that a reduction argument ensures that for interesting specifications, we may focus on the rounds in reasoning about distributed algorithms in a sound way. We use reduction ideas similar to Elrad and Francez [17], Chaouch-Saad et al. [12], and Damian et al. [15]. Thus, similar to recent approaches that aim at connecting asynchrony to synchrony [10,15,27,50], we provide a precise relation between the asynchronous model and rounds asked for in [39].



## 10 Conclusions

We lifted the threshold automata framework to multi-round randomized consensus algorithms. We proved a reduction that allows us to check  $LTL_{\neg\chi}$  specifications over propositions for one round in a single-round automaton so that the verification results transfer directly to the multi-round counter system. Using round-based compositional reasoning, we have shown that this is sufficient to check specifications that span multiple rounds, e.g., agreement. Round-rigid probabilistic termination relies on a distinct reduction argument.

By experimental evaluation, we showed that the verification conditions that came out of our reduction can be automatically verified for several challenging randomized consensus algorithms in the parameterized setting. Since we do not directly check multi-round specifications, but rather only these one-round verification conditions, incorrect algorithms would lead to a counterexample to such a condition, which would then require manual inspection in order to understand the cause of the incorrectness.

Our proof methodology for round-rigid probabilistic termination applies to round-rigid adversaries only. As future work, we shall prove that verifying round-rigid probabilistic termination is sufficient to prove probabilistic termination for more general adversaries. Transforming an adversary into a round-rigid one while preserving the probabilistic properties over the induced paths comes up against the asynchrony in the system. Asynchrony typically leads to processes being in different rounds at the same point of the execution. For instance, a process may have reached round  $k$ , while others are still in round  $k' < k$ . Now, a process may perform a coin toss in some step at round  $k$ , before the other processes have left round  $k'$ . As a result, a priori, an adversary may schedule the remaining steps for round  $k'$  depending on the outcome of the earlier coin toss of the higher round  $k$ . Reconciling this with a reduction argument is challenging. As first step toward this objective, we showed the reduction argument for weak adversaries [8].

Concerning the probabilistic reasoning, our approach relies on a zero-one law and only allows one to prove almost-sure termination (and certainly general qualitative reachability properties). However, proving quantitative properties, for instance, on the expected number of rounds before termination, is currently out of reach of existing techniques. This long-term objective is definitely on our agenda.

## References

1. Aguilera, M., Toueg, S.: The correctness proof of Ben-Or's randomized consensus algorithm. *Distrib. Comput.* **25**(5), 1–11 (2012). Online first
2. Apt, K., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. *IPL* **15**, 307–309 (1986)
3. Baier, C., Katoen, J.-P.: *Principles of Model Checking*. MIT Press, Cambridge (2008)
4. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In: *PODC*, pp. 27–30 (1983)
5. Berkovits, I., Lazic, M., Losa, G., Padon, O., Shoham, S.: Verification of threshold-based distributed algorithms by decomposition to decidable logics. In: *CAV Part II*, pp. 245–266 (2019)
6. Bertrand, N., Fournier, P.: Parameterized verification of many identical probabilistic timed processes. In: *FSTTCS*, volume 24 of *LIPICs*, pp. 501–513 (2013)
7. Bertrand, N., Konnov, I., Lazic, M., Widder, J.: Verification of randomized consensus algorithms under round-rigid adversaries. In: *CONCUR*, volume 140 of *LIPICs*, pp. 33:1–33:15. Schloss Dagstuhl (2019)
8. Bertrand, N., Lazić, M., Widder, J.: A reduction theorem for randomized distributed algorithms under weak adversaries. In: *VMCAI* (2021) (to appear)
9. Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., Widder, J.: Decidability of Parameterized Verification Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, San Rafael (2015)
10. Bouajjani, A., Enea, C., Ji, K., Qadeer, S.: On the completeness of verifying message passing programs under bounded asynchrony. In: *CAV*, pp. 372–391 (2018)
11. Bracha, G.: Asynchronous Byzantine agreement protocols. *Inf. Comput.* **75**(2), 130–143 (1987)
12. Chaouch-Saad, M., Charron-Bost, B., Merz, S.: A reduction theorem for the verification of round-based distributed algorithms. In: *RP*, volume 5797 of *LNCS*, pp. 93–106 (2009)
13. Charron-Bost, B., Merz, S.: Formal verification of a consensus algorithm in the heard-of model. *IJSI* **3**(2–3), 273–303 (2009)
14. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): *Handbook of Model Checking*. Springer, Heidelberg (2018)
15. Damian, A., Drăgoi, C., Militaru, A., Widder, J.: Communication-closed asynchronous protocols. In: *CAV* (2), volume 11562 of *Lecture Notes in Computer Science*, pp. 344–363. Springer (2019)
16. Drăgoi, C., Henzinger, T.A., Veith, H., Widder, J., Zufferey, D.: A logic-based framework for verifying consensus algorithms. In: *VMCAI*, volume 8318 of *LNCS*, pp. 161–181 (2014)
17. Elrad, T., Francez, N.: Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program.* **2**(3), 155–173 (1982)
18. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: *POPL*, pp. 85–94 (1995)
19. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. *J. ACM* **32**(2), 374–382 (1985)
20. Grimmett, G.R., Strizaker, D.: *Probability and Random Processes*, 2nd edn. Oxford Science Publications, Oxford (1992)
21. Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: Ironfleet: proving safety and liveness of practical distributed systems. *Commun. ACM* **60**(7), 83–92 (2017)
22. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Towards modeling and model checking fault-tolerant distributed algorithms. In: *SPIN*, volume 7976 of *LNCS*, pp. 209–226 (2013)
23. Konnov, I., Lazic, M., Veith, H., Widder, J.: Para<sup>2</sup>: parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms. *Formal Methods Syst. Des.* **51**(2), 270–307 (2017)
24. Konnov, I., Lazić, M., Veith, H., Widder, J.: A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In: *POPL*, pp. 719–734 (2017)

25. Konnov, I., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. *Inf. Comput.* **252**, 95–109 (2017)
26. Konnov, I., Widder, J.: ByMC: byzantine model checker. In: *ISoLA* (3), volume 11246 of LNCS, pp. 327–342. Springer (2018)
27. Kragl, B., Qadeer, S., Henzinger, T.A.: Synchronizing the asynchronous. In: *CONCUR*, pp. 21:1–21:17 (2018)
28. Kukovec, J., Konnov, I., Widder, J.: Reachability in parameterized systems: all flavors of threshold automata. In: *CONCUR*, pp. 19:1–19:17 (2018)
29. Kwiatkowska, M.Z., Norman, G.: Verifying randomized byzantine agreement. In: *FORTE*, pp. 194–209 (2002)
30. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: *CAV*, pp. 585–591 (2011)
31. Kwiatkowska, M.Z., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In: *CAV*, pp. 194–206 (2001)
32. Lamport, L.: *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Addison-Wesley, Boston (2002)
33. Lehmann, Daniel J., Rabin, Michael O.: On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In: *POPL*, pp. 133–138 (1981)
34. Lengál, O., Lin, A.W., Majumdar, R., Rümmer, P.: Fair termination for parameterized probabilistic concurrent systems. In: *TACAS*, volume 10205 of LNCS, pp. 499–517 (2017). [https://doi.org/10.1007/978-3-662-54577-5\\_29](https://doi.org/10.1007/978-3-662-54577-5_29)
35. Lin, A.W., Rümmer, P.: Liveness of randomised parameterised systems under arbitrary schedulers. In *CAV*, volume 9780 of LNCS, pp. 112–133. Springer (2016). [https://doi.org/10.1007/978-3-319-41540-6\\_7](https://doi.org/10.1007/978-3-319-41540-6_7)
36. Maric, O., Sprenger, C., Basin, D.A.: Cutoff bounds for consensus algorithms. In: *CAV*, volume 10427 of LNCS, pp. 217–237 (2017)
37. McIver, A., Morgan, C.: *Abstraction, refinement and proof for probabilistic systems*. In: *Monographs in Computer Science*. Springer (2005). <https://doi.org/10.1007/b138392>
38. Mostéfaoui, A., Moumen, H., Raynal, M.: Randomized k-set agreement in crash-prone and Byzantine asynchronous systems. *Theor. Comput. Sci.* **709**, 80–97 (2018)
39. Nestmann, U., Fuzzati, R., Merro, M.: Modeling consensus in a process calculus. In: *CONCUR*, volume 2761 of LNCS, pp. 393–407 (2003)
40. Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: decidable reasoning about distributed protocols. *PACMPL* **1**(OOPSLA), 108:1–108:31 (2017)
41. Pnueli, A., Zuck, L.D.: Verification of multiprocess probabilistic protocols. *Distrib. Comput.* **1**(1), 53–72 (1986). <https://doi.org/10.1007/BF01843570>
42. Song, Y.J., van Renesse, R.: Bosco: one-step Byzantine asynchronous consensus. In: *DISC*, volume 5218 of LNCS, pp. 438–450 (2008)
43. Steiner, W., Rushby, J.M., Sorea, M., Pfeifer, H.: Model checking a fault-tolerant startup algorithm: from design exploration to exhaustive fault simulation. In: *DSN*, pp. 189–198 (2004)
44. Stoilkovska, I., Konnov, I., Widder, J., Zuleger, F.: Verifying safety of synchronous fault-tolerant algorithms bounded model checking. In: *TACAS*, Part II, volume 11428 of LNCS, pp. 357–374 (2019)
45. Suzuki, I.: Proving properties of a ring of finite-state machines. *Inf. Process. Lett.* **28**(4), 213–214 (1988)
46. Swaminathan, M., Katoen, J.-P., Olderog, E.-R.: Layered reasoning for randomized distributed algorithms. *Formal Asp. Comput.* **24**(4–6), 477–496 (2012). <https://doi.org/10.1007/s00165-012-0231-x>
47. TLA+ proof system. <https://tla.msr-inria.inria.fr/tlaps/content/Home.html>
48. Tsuchiya, T., Schiper, A.: Using bounded model checking to verify consensus algorithms. In: *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22–24, 2008, Proceedings*, pp. 466–480 (2008)
49. Tsuchiya, T., Schiper, A.: Verification of consensus algorithms using satisfiability solving. *Distrib. Comput.* **23**(5–6), 341–358 (2011)
50. Gleissenthall, K.V., Kici, R.G., Bakst, A.L., Stefan, D.E., Jhala, R.A.: Pretend synchrony. *PACMPL* **3**(POPL), 59:1–59:30 (2019)
51. Woos, D., Wilcox, J.R., Anton, S., Tatlock, Z., Ernst, M.D., Anderson, T.E.: Planning for change in a formal verification of the RAFT consensus protocol. In: *CPP*, pp. 154–165 (2016)
52. Zuck, L.D., McMillan, K.L., Torf, J.:  $P^5$ : planner-less proofs of probabilistic parameterized protocols. In: *VMCAI*, pp. 336–357 (2018)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.