Verification of Randomized Consensus Algorithms Under Round-Rigid Adversaries

Nathalie Bertrand
Univ. Rennes, Inria, CNRS, IRISA, Rennes, France
nathalie.bertrand@inria.fr

Igor Konnov
University of Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
igor.konnov@inria.fr

Marijana Lazić
TU München, 85748 Garching bei München, Germany
lazic@in.tum.de

Josef Widder
TU Wien, 1040 Vienna, Austria
Interchain Foundation, 6340 Baar, Switzerland
widder@forsyte.at

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.

2012 ACM Subject Classification Software and its engineering → Software verification; Software and its engineering → Software fault tolerance; Theory of computation → Logic and verification

Keywords and phrases threshold automata, counter systems, parameterized verification, randomized distributed algorithms, Byzantine faults

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.33

Related Version Full version with proofs is available at `https://hal.inria.fr/hal-01925533`.

Funding Supported 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 `https://www.grid5000.fr`.

```
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 (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
11       messages (P,r,w,D) then {
12     v := w;
13     if received at least (n + t) / 2
14       messages (P,r,w,D)
15     then decide w;
16   } else v := random({0, 1});
17   r := r + 1;
18 od
```

Figure 1 Pseudo code of Ben-Or's algorithm for Byzantine faults.

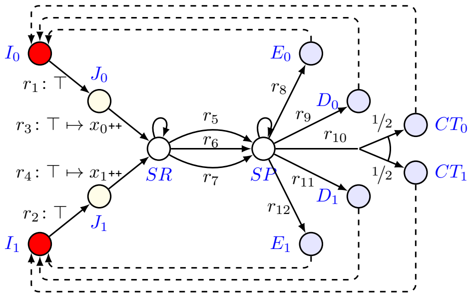
1 Introduction

Fault-tolerant distributed algorithms like Paxos and Blockchain recently receive 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 randomized distributed algorithms in the parameterized setting.

In this paper, we make first steps towards parameterized verification of fault-tolerant randomized distributed algorithms. We consider consensus algorithms that follow the ideas of Ben-Or [3]. Interestingly, these algorithms were analyzed in [17, 15] where probabilistic reasoning was done using the probabilistic model checker PRISM [16] for systems 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 [12, 10] 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 [3] given in Figure 1. It circumvents the impossibility of asynchronous consensus [9] 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). 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.*, $n > 3t$. Carefully chosen thresholds ($n-t$, $(n+t)/2$ and $t+1$) on the number of received messages of a given type, ensure *agreement*, *i.e.*, that two correct processes never decide on different values. 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.

While these non-trivial threshold expressions can be dealt with using the methods in [10], several challenges remain. The technique in [10] can be used to verify one iteration of the round from Figure 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.



$r_5: x_0 + x_1 \geq n - t - f \wedge x_0 \geq (n+t)/2 - f \mapsto y_0$++
$r_6: x_0 + x_1 \geq n - t - f \wedge x_1 \geq (n+t)/2 - f \mapsto y_1$++
$r_7: x_0 + x_1 \geq n - t - f \wedge x_0 \geq (n-3t)/2 - f$
$\wedge\ x_1 \geq (n-3t)/2 - f \mapsto y_?$++
$r_8: y_0 + y_1 + y_? \geq n - t - f \wedge y_? \geq (n-3t)/2 - f$
$\wedge\ y_0 \geq t + 1 - f$
$r_9: y_0 + y_1 + y_? \geq n - t - f \wedge y_0 \geq (n+t)/2 - f$
$r_{10}: y_0 + y_1 + y_? \geq n - t - f \wedge y_? \geq (n-3t)/2 - f$
$\wedge\ y_? \geq n - 2t - f - 1$

Figure 2 Ben-Or's algorithm as PTA with resilience condition $n > 3t \wedge t > 0 \wedge t \geq f \geq 0$.

In addition, the combination of distributed aspects and probabilities makes reasoning difficult. Quoting Lehmann and Rabin [18], "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 threshold automata framework from [10] 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 [8, 6, 7]. 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, these are the first parameterized model checking results for probabilistic properties.
3. We check the specifications that emerge from points 1. and 2. and thus verify challenging benchmarks in the parameterized setting. We verify Ben-Or's [3] and Bracha's [5] classic algorithms, and more recent algorithms such as 2-set agreement [21], and RS-Bosco [23].

2 Overview

We introduce probabilistic threshold automata to model randomized threshold-based algorithms. An example of such an automaton is given in Figure 2. Nodes represent local states (or locations) of processes, which move along the labeled edges or forks. Edges and forks are called rules. Labels have the form $\varphi \mapsto u$, meaning that a process can move along the edge only if φ 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 moving along the fork to end up at the target location of the tine. If we ignore the dashed arrows in Figure 2, a threshold automaton captures the behavior of a process in one round, that is, a loop iteration in Figure 1.

The code in Figure 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. 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 $n > 3t \wedge t \geq f \wedge t > 0$. 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., 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 received 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 Figure 1, f messages from Byzantine faults may be received *in addition* to the messages sent by correct processes (modeled by shared variables in Figure 2). 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 in the same configuration.

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 Figure 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.

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 Figure 1, and in each iteration, processes send messages that carry r. To capture this, our semantics will introduce fresh shared variables initialized with 0 for each round r. Because there are infinitely many rounds, this means a priori we have infinitely many variables.

As parameterized verification of threshold automata is in general undecidable [14], we consider the so-called "canonic" restrictions here, *i.e.*, only increments on shared variables, and no increments of the same variable within loops. These restrictions still allow us to model many threshold-based fault-tolerant distributed algorithms [10]. As a result, threshold automata without probabilistic forks and round switching rules can be automatically checked for safety and liveness [10]. Adding forks and round switches is required to adequately model randomized distributed algorithms. Here we will use a convenient restriction that requires that coin-toss transitions only appear at the end of a round, e.g., line 16 of Figure 1. Intuitively, as discussed in Section 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 Section 7.

In order to overcome the issue of infinitely many rounds, we prove in Section 6 that we can verify probabilistic threshold automata by analyzing a one-round automaton that fits in the framework of [10]. We prove that we can reorder transitions of any fair execution such that their round numbers are in an increasing order. The obtained ordered execution is stutter equivalent with the original one, and thus, they satisfy the same $\mathsf{LTL}_{\text{-}\mathsf{X}}$ properties over the atomic propositions describing only one round. In other words, our targeted concurrent systems can be transformed to a sequential composition of one-round systems.

The main problem with isolating a one-round system is that 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 whether 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.

Finally, 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 we apply it to other randomized distributed algorithms (see Section 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 hypothesis. The crucial parts of our proof are automatically checked by the model checker ByMC [13]. Hence the established correctness stands on less slippery ground, which addresses the mentioned concerns of Lehmann and Rabin.

3 The Probabilistic Threshold Automata Framework

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: Π contains *parameter variables*, and Γ 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 locations. The resilience condition RC only appears in the definition of the semantics in Section 3.1.

A rule r is a tuple $(\mathit{from}, \delta_{to}, \varphi, \vec{u})$ where $\mathit{from} \in \mathcal{L}$ is the *source* location, $\delta_{to} \in \mathsf{Dist}(\mathcal{L})$ is a probability distribution over the *destination* locations, $\vec{u} \in \mathbb{N}_0^{|\Gamma|}$ is the *update vector*, and φ is a guard, *i.e.*, a conjunction of expressions of the form $b \cdot x \geq \bar{a} \cdot \mathbf{p}^\intercal + a_0$ or $b \cdot x < \bar{a} \cdot \mathbf{p}^\intercal + a_0$, where $x \in \Gamma$ is a shared variable, $\bar{a} \in \mathbb{Z}^{|\Pi|}$ is a vector of integers, $a_0, b \in \mathbb{Z}$, and $\mathbf{p}$ is the vector of all parameters. 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 $(\mathit{from}, \ell, \varphi, \vec{u})$. Destination locations of non-Dirac rules are in $\mathcal{F}$ (coin-toss transitions only happen at the end of a round).

Probabilistic threshold automata model algorithms with successive identical rounds. Informally, a round happens between border locations and final locations. Then 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 $(\ell, \ell', \mathtt{true}, \vec{0})$ 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 $(\ell, \ell', \mathtt{true}, \vec{0})$ 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 is in $\mathcal{B}$ 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 2 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 Dirac rule is r_{10}, and round switch rules are represented by dashed arrows.

3.1 Probabilistic Counter Systems

A resilience condition RC defines the set of *admissible parameters* $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Pi|} : \mathbf{p} \models RC\}$. We introduce a function $N : \mathbf{P}_{RC} \to \mathbb{N}_0$ that maps a vector of admissible parameters to a number of modeled processes in the system. For instance, for the automaton in Figure 2, 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 Section 2. Given a PTA and a function N, we define the semantics, called *probabilistic counter system* Sys(PTA), to be the infinite-state MDP $(\Sigma, I, \mathsf{Act}, \Delta)$, where Σ is the set of configurations for PTA among which $I \subseteq \Sigma$ are initial, the set of actions is $\mathsf{Act} = \mathcal{R} \times \mathbb{N}_0$ and $\Delta : \Sigma \times \mathsf{Act} \to \mathsf{Dist}(\Sigma)$ is the probabilistic transition function.

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

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

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

Actions. An action $\alpha = (r,k) \in \mathsf{Act}$ stands for the execution of a rule r in round k (by a single process). We write $\alpha.\mathit{from}$ for $r.\mathit{from}$, $\alpha.\delta_{to}$ for $r.\delta_{to}$, etc. An action $\alpha = (r,k)$ is *unlocked* in a configuration σ, if its guard evaluates to true in its round, that is $\sigma, k \models r.\varphi$. An action $\alpha = (r,k)$ is *applicable* to a configuration σ if α is unlocked in σ, and there is at least one process in the source location $r.\mathit{from}$, formally, $\sigma.\vec{\boldsymbol{\kappa}}[r.\mathit{from},k] \geq 1$. When an action α is applicable to σ, and when ℓ is a potential destination location for the probabilistic action α, we write $\mathit{apply}(\sigma,\alpha,\ell)$ for the resulting configuration: parameters are unchanged, shared variables are updated according to the update vector $r.\vec{u}$, and the values of counters are modified in a natural way: as a process moves from $r.\mathit{from}$ to ℓ in round k, counter $\vec{\boldsymbol{\kappa}}[r.\mathit{from},k]$ is decreased by 1 and $\vec{\boldsymbol{\kappa}}[\ell,k]$ is increased by 1. The probabilistic transition function Δ is defined by: $\Delta(\sigma,\alpha)(\sigma') = \alpha.\delta_{to}(\ell)$ if $\mathit{apply}(\sigma,\alpha,\ell) = \sigma'$, and 0 otherwise.

3.2 Non-probabilistic Counter Systems

Non-probabilistic threshold automata are defined in [12], and they can be seen as a special case of probabilistic threshold automata where all rules are Dirac rules.

With a PTA, one can naturally associate a non-probabilistic threshold automaton, by replacing probabilities with non-determinism.

▶ **Definition 1.** *Given a PTA* $= (\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$*,* its (non-probabilistic) threshold automaton *is* $\mathsf{TA}_{\mathsf{PTA}} = (\mathcal{L}, \mathcal{V}, \mathcal{R}_{np}, RC)$ *where the set of rules* $\mathcal{R}_{np}$ *is defined as* $\{r_\ell = (\mathit{from}, \ell, \varphi, \vec{u}) : r = (\mathit{from}, \delta_{to}, \varphi, \vec{u}) \in \mathcal{R} \wedge \ell \in \mathcal{L} \wedge \delta_{to}(\ell) > 0\}$.

We write TA instead of $\mathsf{TA}_{\mathsf{PTA}}$ 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 *apply* in order to obtain an infinite (non-probabilistic) counter system, which we denote by $\mathsf{Sys}_\infty(\mathsf{TA})$. Moreover, since in this case $\mathcal{R} = \mathcal{R}_{np}$, actions of the PTA exactly match transitions of its TA. We obtain σ' by applying $t = (r, k)$ to σ, and write this as $\sigma' = t(\sigma)$, if and only if for the destination location ℓ of r holds that $apply(\sigma, t, \ell) = \sigma'$.

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

A (finite or infinite) sequence of transitions is called *schedule*, and it is often denoted by τ. A schedule $\tau = t_1, t_2, \ldots, t_{|\tau|}$ is applicable to a configuration σ if there is a sequence of configurations $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_{|\tau|}$ such that for every $1 \le i \le |\tau|$ we have that t_i is applicable to σ_{i-1} and $\sigma_i = t_i(\sigma_{i-1})$. A *path* is an alternating sequence of configurations and transitions, for example $\sigma_0, t_1, \sigma_1, \ldots, t_{|\tau|}, \sigma_{|\tau|}$, such that for every t_i, $1 \le i \le |\tau|$, in the sequence, we have that t_i is applicable to σ_{i-1} and $\sigma_i = t_i(\sigma_{i-1})$. Given a configuration σ_0 and a schedule $\tau = t_1, t_2, \ldots, t_{|\tau|}$, we denote by $\mathsf{path}(\sigma_0, \tau)$ a path $\sigma_0, t_1, \sigma_1, \ldots, t_{|\tau|}, \sigma_{|\tau|}$ where $t_i(\sigma_{i-1}) = \sigma_i$, $1 \le i \le |\tau|$. Similarly we define an infinite schedule $\tau = t_1, t_2, \ldots$, and an infinite path $\sigma_0, t_1, \sigma_1, \ldots$, also denoted by $\mathsf{path}(\sigma_0, \tau)$. 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.

Since every transition in $\mathsf{Sys}_\infty(\mathsf{TA})$ comes from an action in $\mathsf{Sys}(\mathsf{PTA})$, note that every path in $\mathsf{Sys}_\infty(\mathsf{TA})$ is a valid path in $\mathsf{Sys}(\mathsf{PTA})$.

3.3 Adversaries

As usual, the non-determinism is resolved by a so-called adversary. Let Paths be the set of all finite paths in $\mathsf{Sys}(\mathsf{PTA})$. An *adversary* is a function $\mathsf{a} : \mathsf{Paths} \to \mathsf{Act}$, that given a finite path π selects an action applicable to the last configuration of π. Given a configuration σ and an adversary a, we generate a family of paths, depending on the outcomes of non-Dirac transitions. We denote this set by $\mathsf{paths}(\sigma, \mathsf{a})$. An adversary a is *fair* if all paths in $\mathsf{paths}(\sigma, \mathsf{a})$ are fair. As usual, the Markov Decision Process (MDP) $\mathsf{Sys}(\mathsf{PTA})$ together with an initial configuration σ and an adversary a induce a Markov chain, written $\mathcal{M}^\sigma_{\mathsf{a}}$. We write $\mathbb{P}^\sigma_{\mathsf{a}}$ for the probability measure over infinite paths starting at σ in the latter Markov chain.

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 action of the form $s_1 \cdot s_1^p \cdot s_2 \cdot s_2^p \ldots$, where for all $k \in \mathbb{N}$, we have that s_k contains only Dirac actions of round k, and s_k^p contains only non-Dirac actions of round k. We denote the set of all round-rigid adversaries by $\mathcal{A}^{\mathrm{R}}$.

3.4 Atomic Propositions and Stutter Equivalence

The atomic propositions we consider describe non-emptiness of a location $\ell \in \mathcal{L} \setminus \mathcal{B}$ in a round k, *i.e.*, whether there is at least one process in location ℓ in round k. Formally, the set of all such propositions for a round $k \in \mathbb{N}_0$ is denoted by $\mathrm{AP}_k = \{\mathrm{p}(\ell, k) \colon \ell \in \mathcal{L} \setminus \mathcal{B}\}$. For every k we define a labeling function $\lambda_k \colon \Sigma \to 2^{\mathrm{AP}_k}$ such that $\mathrm{p}(\ell, k) \in \lambda_k(\sigma)$ iff $\sigma.\vec{\boldsymbol{\kappa}}[\ell, k] > 0$. By abusing notation, we write "$\vec{\boldsymbol{\kappa}}[\ell, k] > 0$" and "$\vec{\boldsymbol{\kappa}}[\ell, k] = 0$" instead of $\mathrm{p}(\ell, k)$ and $\neg\mathrm{p}(\ell, k)$, resp.

We denote by $\pi_1 \triangleq_k \pi_2$ that the paths π_1 and π_2 are stutter equivalent [2] w.r.t. AP_k. Two counter systems C_0 and C_1 are stutter equivalent w.r.t. AP_k, written $C_0 \triangleq_k C_1$, if for every path π from C_i there is a path π' from C_{1-i} such that $\pi \triangleq_k \pi'$, for $i \in \{0, 1\}$.

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 $1-v$.

Probabilistic wait-free termination: Under 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 Figure 2.

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.

(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. Now we can express the specifications as follows:

Agreement: For both values $v \in \{0,1\}$ the following holds:

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

Validity: For both $v \in \{0,1\}$ it holds

$$\forall k \in \mathbb{N}_0.\ \mathbf{A}\left(\bigwedge\nolimits_{\ell \in \mathcal{I}_v} \vec{\boldsymbol{\kappa}}[\ell,0] = 0 \ \rightarrow\ \mathbf{G} \bigwedge\nolimits_{\ell' \in \mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell',k] = 0\right) \tag{2}$$

Probabilistic wait-free termination: For every round-rigid adversary a

$$\mathbb{P}_{\mathsf{a}}\left(\bigvee\nolimits_{k \in \mathbb{N}_0} \bigvee\nolimits_{v \in \{0,1\}} \mathbf{G} \bigwedge\nolimits_{\ell \in \mathcal{F} \setminus \mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell,k] = 0\right) \quad = \quad 1 \tag{3}$$

Agreement and validity are non-probabilistic properties, and thus can be analyzed on the non-probabilistic counter system $\mathsf{Sys}_\infty(\mathsf{TA})$. For verifying probabilistic wait-free termination, we make explicit the following assumption that is present in all our benchmarks: all non-Dirac transitions have non-zero probability to lead to an $\mathcal{F}_v$ location, for both values $v \in \{0,1\}$.

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

5 Reduction to Specifications with one Round Quantifier

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 one round systems [10], 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. Namely, we introduce round invariants (4) and (5), and prove that they imply Agreement and Validity.

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:

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

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

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

The following proposition is proved using restriction (R2) and by an inductive argument over the round number.

▶ **Proposition 2.** *If $Sys_\infty(TA) \models (4) \wedge (5)$, then $Sys_\infty(TA) \models (1) \wedge (2)$.*

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 formulated in Theorem 6, and it allows us to use ByMC on a single-round system.

The proof idea contains two parts. First, in Section 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 $\mathsf{LTL}_{\text{-}\mathsf{X}}$ properties.

Second, in Section 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 using the technique from [10]. As stutter equivalence of systems implies preservation of $\mathsf{LTL}_{\text{-}\mathsf{X}}$ properties, this is sufficient to prove the main goal of the section.

6.1 Reduction from arbitrary schedules to round-rigid schedules

▶ **Definition 3.** *A schedule $\tau = (r_1, k_1) \cdot (r_2, k_2) \cdot \ldots \cdot (r_m, k_m)$, $m \in \mathbb{N}_0$, is called* round-rigid *if for every $1 \le i < j \le m$, we have $k_i \le k_j$.*

The following proposition shows that any finite schedule can be re-ordered into a round-rigid one that is stutter equivalent regarding $\mathsf{LTL}_{\text{-}\mathsf{X}}$ formulas over proposition from AP_k, for all rounds k. It is proved using arguments on the commutativity of transitions, similar to [8].

▶ **Proposition 4.** *For every configuration σ and every finite schedule τ applicable to σ, there is a round-rigid schedule τ' such that the following holds:*

(a) *Schedule τ' is applicable to configuration σ.*
(b) *τ' and τ reach the same configuration when applied to σ,* i.e., *$\tau'(\sigma) = \tau(\sigma)$.*
(c) *For every $k \in \mathbb{N}_0$ we have $\mathit{path}(\sigma, \tau) \triangleq_k \mathit{path}(\sigma, \tau')$.*

Thus, instead of reasoning about all finite schedules of $\mathsf{Sys}_\infty(\mathsf{TA})$, 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.

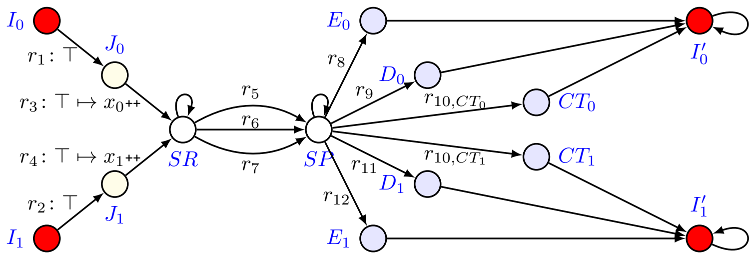


Figure 3 The single-round threshold automaton $\mathsf{TA}^{\mathrm{rd}}$ obtained from PTA in Figure 2.

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

For each PTA, we define a *single-round threshold automaton* that can be analyzed with the tools of [12] and [10]. Roughly speaking, we focus on one round, but also keep the border locations of the next round, where we add self-loops. Figure 3 represents the single-round threshold automaton associated with the PTA from Figure 2. We can prove that for specific fairness constraints, this automaton shows the same behavior as a round in $\mathsf{Sys}_\infty(\mathsf{TA})$.

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 state from $\mathcal{B}'$. Under the fairness assumption we check the latter assumption with ByMC [13]. 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).

▶ **Definition 5.** *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* $\mathcal{R}^{loop} = \{(\ell', \ell', \mathtt{true}, \vec{0}) : \ell' \in \mathcal{B}'\}$ *are self-loop rules at locations from* $\mathcal{B}'$ *and* $\mathcal{S}' = \{(from, \ell', \mathtt{true}, \vec{0}) : (from, \ell, \mathtt{true}, \vec{0}) \in \mathcal{S}$ *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 $\mathsf{TA}^{\mathrm{rd}}$ and a $k \in \mathbb{N}_0$ we define a counter system $\mathsf{Sys}^k(\mathsf{TA}^{\mathrm{rd}})$ as the tuple (Σ^k, I^k, R^k). A configuration is a tuple $\sigma = (\vec{\kappa}, \vec{g}, \mathbf{p}) \in \Sigma^k$, where $\sigma.\vec{\kappa} : \mathcal{D} \to \mathbb{N}_0$ defines values of the counters, for $\mathcal{D} = (\mathcal{L} \times \{k\}) \cup (\mathcal{B}' \times \{k+1\})$; and $\sigma.\vec{g} : \Gamma \times \{k\} \to \mathbb{N}_0$ 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 $\sigma.\vec{\kappa}$ above, every configuration $\sigma \in \mathsf{Sys}^k(\mathsf{TA}^{\mathrm{rd}})$ can be extended to a valid configuration of $\mathsf{Sys}_\infty(\mathsf{TA})$, by assigning zero to all other counters and global variables. In the following, we identify a configuration in $\mathsf{Sys}^k(\mathsf{TA}^{\mathrm{rd}})$ with its extension in $\mathsf{Sys}_\infty(\mathsf{TA})$, since they have the same labeling function λ_k, for every $k \in \mathbb{N}_0$.

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

We define the transition relation R as in $\mathsf{Sys}_\infty(\mathsf{TA})$, *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.

For every $k \in \mathbb{N}_0$ and every $\sigma \in \Sigma^k_{\mathcal{B}}$, there is a corresponding configuration $\sigma' \in \Sigma^0_{\mathcal{B}}$ obtained from σ by renaming the round k to round 0. Let f_k be the renaming function, *i.e.*, $\sigma' = f_k(\sigma)$. Let us define $\Sigma^u \subseteq \Sigma^0_{\mathcal{B}}$ to be the union of all renamed initial configurations from all rounds, that is, $\{f_k(\sigma)\colon k \in \mathbb{N}_0, \sigma \in I^k\}$.

▶ **Theorem 6.** *Let TA be non-blocking, and let all fair executions of $\mathsf{Sys}^0(\mathsf{TA}^{rd})$ w.r.t. $\Sigma^0_{\mathcal{B}}$ terminate. Given a formula $\varphi[i]$ from $\mathsf{LTL}_{\text{-}\mathsf{X}}$ over AP_i, for a round variable i, we have*
(A) *$\mathsf{Sys}^0(\mathsf{TA}^{rd}) \models \mathbf{E}\,\varphi[0]$ w.r.t. initial configurations Σ^u if and only if*
(B) *there exists $k \in \mathbb{N}_0$ such that $\mathsf{Sys}_\infty(\mathsf{TA}) \models \mathbf{E}\,\varphi[k]$.*

Proof sketch. The theorem is proved using the following arguments. In statement (B), the existential quantification over k corresponds to the defintion of Σ^u as union, over all rounds, of projections of all reachable initial configurations of that round.

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

Implication $(A) \to (B)$, note that an initial configuration in Σ^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 $\mathsf{Sys}_\infty(\mathsf{TA})$. 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 or round k mimick 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. ◀

In Section 4 we showed how to reduce our specifications to formulas of the form $\forall k \in \mathbb{N}_0.\, \mathbf{A}\,\psi[k]$. Theorem 6 deals with negations of such formulas, i.e., with existence of a round k such that formula $\mathbf{E}\,\varphi[k]$ holds. Thus, we can check specifications on the single-round system.

7 Probabilistic Wait-Free Termination

We start by defining two conditions that are sufficient to establish Probabilistic Wait-Free 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) There is a bound $p \in (0, 1]$, such that for every round-rigid adversary a, and every $k \in \mathbb{N}_0$, and every configuration σ_k with parameters $\mathbf{p}$ that is initial for round k, it holds that

$$\mathbb{P}^{\sigma_k}_{\mathsf{a}}\Big(\bigvee_{v\in\{0,1\}} \mathbf{G}\,\big(\bigwedge_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell,k]=0\big)\Big) > p.$$

(C2) For every $v \in \{0, 1\}$, $\forall k \in \mathbb{N}_0.\, \mathbf{A}\Big(\mathbf{G} \bigwedge_{\ell\in\mathcal{I}_{1-v}} \vec{\boldsymbol{\kappa}}[\ell,k] = 0 \;\to\; \mathbf{G} \bigwedge_{\ell'\in\mathcal{F}\setminus\mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell',k] = 0\Big)$.

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 [1].

▶ **Proposition 7.** *If $\mathsf{Sys}_\infty(\mathsf{PTA}) \models (C1) \wedge (C2)$, then $\mathsf{Sys}_\infty(\mathsf{PTA}) \models (3)$.*

Observe (C2) is a non-probabilistic property of the same form as (5), so that we can check (C2) using the method of Section 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: $\mathsf{PTA}^{\mathrm{rd}}$ (analogously to Definition 5), and its counter system $\mathsf{Sys}(\mathsf{PTA}^{\mathrm{rd}})$.

7.1 Reducing probabilistic to non-probabilistic specifications

Since probabilistic transitions end in final locations, they cannot appear on a cycle in $\mathsf{PTA}^{\mathrm{rd}}$. Therefore, 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:

▶ **Lemma 8.** *Let $\mathbf{p} \in \mathbf{P}_{RC}$ be a parameter valuation. In $\mathsf{Sys}(\mathsf{PTA}^{rd})$, for every LTL formula φ over atomic proposition* AP*, the following two statements are equivalent:*
(a) $\exists p > 0,\ \forall \sigma \in I_{\mathbf{p}},\ \forall \mathsf{a} \in \mathcal{A}^R.\quad \mathbb{P}^{\sigma}_{\mathsf{a}}(\varphi) > p,$
(b) $\forall \sigma \in I_{\mathbf{p}},\ \forall \mathsf{a} \in \mathcal{A}^R,\ \exists \pi \in \mathit{paths}(\sigma, \mathsf{a}).\quad \pi \models \varphi.$

7.2 Verifying (C1) on a non-probabilistic TA

Applying Lemma 8, proving (C1) is equivalent to proving the following property on $\mathsf{Sys}(\mathsf{PTA}^{\mathrm{rd}})$

$$\forall \sigma \in I_{\mathbf{p}},\ \forall \mathsf{a} \in \mathcal{A}^{\mathrm{R}},\ \exists \pi \in \mathsf{paths}(\sigma, \mathsf{a}).\quad \pi \models \bigvee\nolimits_{v\in\{0,1\}} \mathbf{G}\,(\bigwedge\nolimits_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell] = 0). \tag{6}$$

In the sequel, we explain how to reduce the verification of (6) to checking the simpler formula $\mathbf{A} \bigvee_{v\in\{0,1\}} \mathbf{G}\,(\bigwedge_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell] = 0)$ on a single-round non-probabilistic TA obtained from $\mathsf{PTA}^{\mathrm{rd}}$.

As in Section 6, it is possible to modify $\mathsf{PTA}^{\mathrm{rd}}$ into a non-probabilistic TA, by replacing probabilistic choices by non-determinism. Still, the quantifier alternation of (6) (universal over initial configurations and adversaries vs. existential on paths) is not in the fragment handled by ByMC [13]. Once an initial configuration σ and an adversary a are fixed, the remaining branching is 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 (6), 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 $\mathsf{PTA}^{\mathrm{rd}}$, we define a threshold automaton TA^{m} with locations $\mathcal{L}$ (without $\mathcal{B}'$) such that for every non-Dirac rule $r = (\mathit{from}, \delta_{to}, \varphi, \vec{u})$ in PTA, all locations ℓ with $\delta_{to}(\ell) > 0$ are merged into a new location ℓ^{mrg} in TA^{m}. Note that this location must belong to $\mathcal{F}$. Naturally, instead of a non-Dirac rule r we obtain a Dirac rule $(\mathit{from}, \ell^{\mathrm{mrg}}, \varphi, \vec{u})$. Also we add self-loops at all final locations. Paths in $\mathsf{Sys}(\mathsf{TA}^{\mathrm{m}})$ correspond to prefixes of paths in $\mathsf{Sys}(\mathsf{PTA}^{\mathrm{rd}})$. In $\mathsf{Sys}(\mathsf{TA}^{\mathrm{m}})$, from a configuration σ, an adversary a yields a unique path, that is, $\mathsf{paths}(\sigma, \mathsf{a})$ is a singleton set. Thus, the existential quantifier from (6) can be replaced by the universal one.

By construction, property (6) on $\mathsf{PTA}^{\mathrm{rd}}$ is equivalent to $\mathbf{A} \bigvee_{v\in\{0,1\}} \mathbf{G}\,(\bigwedge_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell] = 0)$ on $\mathsf{Sys}(\mathsf{TA}^{\mathrm{m}})$. The latter can be checked automatically by ByMC, allowing us to prove (C1).

8 Experiments

We have applied the approach presented in Sections 4–7 to five randomized fault-tolerant distributed algorithms. (The benchmarks and the instructions on running the experiments are available from: `https://forsyte.at/software/bymc/artifact42/`.)

Table 1 Temporal properties verified in our experiments for value 0.

Label	Name	Automaton	Formula
S1	`agreement_0`	N	$\mathbf{A}\,\mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D0}\}) \vee \mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S2	`validity_0`	N	$\mathbf{A}\,\textsc{All}\{\mathsf{V0}\} \rightarrow \mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S3	`completeness_0`	N	$\mathbf{A}\,\textsc{All}\{\mathsf{V0}\} \rightarrow \mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S4	`round-term`	N	$\mathbf{A}\,\mathit{fair} \rightarrow \mathbf{F}\,\textsc{All}\{\mathsf{D0},\mathsf{D1},\mathsf{E0},\mathsf{E1},\mathsf{CT}\}$
S5	`decide-or-flip`	P	$\mathbf{A}\,\mathit{fair} \rightarrow \mathbf{F}\,(\textsc{All}\{\mathsf{D0},\mathsf{E0},\mathsf{CT}\} \vee \textsc{All}\{\mathsf{D1},\mathsf{E1},\mathsf{CT}\})$
S1'	`sim-agreement`	N	$\mathbf{A}\,\mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D0},\mathsf{E0}\} \vee \neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S1"	`2-agreement`	N	$\mathbf{A}\,\mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D0},\mathsf{E0}\} \vee \neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\} \vee \neg\textsc{Ex}\{\mathsf{D2},\mathsf{E2}\})$

1. Protocol 1 for randomized consensus by Ben-Or [3], 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 subset of processes. This algorithm works correctly when $n > 2t$.
2. Protocol 2 for randomized Byzantine consensus (ben-or-byz) by Ben-Or [3]. This algorithm tolerates Byzantine faults when $n > 5t$.
3. Protocol 2 for randomized consensus (rabc-c) by Bracha [5]. It runs as a high-level algorithm together with a low-level broadcast that turns 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 (kset) by Raynal [21], for $k = 2$. This algorithm works in presense of clean crashes when $n > 3t$.
5. Randomized Byzantine one-step consensus (rs-bosco) by Song and van Renesse [23]. 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 Sections 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, 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 probabilistic wait-free termination. Both automata are given as the input to Byzantine Model Checker (ByMC) [13], which implements the parameterized model checking techniques for safety [11] and liveness [10] of counter systems of threshold automata.

Both automata follow the pattern shown in Figure 2: They start in one of the initial locations (e.g., V_0 or V_1), progress by switching locations and incrementing 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 1 summarizes the temporal properties that were verified in our experiments. Given the set of all possible locations $\mathcal{L}$, a set $Y = \{\ell_1, \dots, \ell_m\} \subseteq \mathcal{L}$ of locations, and the designated crashed location $\mathsf{CR} \in \mathcal{L}$, we use the shorthand notation: $\textsc{Ex}\{\ell_1, \dots, \ell_m\}$ for $\bigvee_{\ell \in Y} \vec{\boldsymbol{\kappa}}[\ell] \neq 0$ and $\textsc{All}\{\ell_1, \dots, \ell_m\}$ for $\bigwedge_{\ell \in \mathcal{L} \setminus Y} (\vec{\boldsymbol{\kappa}}[\ell] = 0 \vee \ell = \mathsf{CR})$. For rs-bosco and kset, instead of checking S1, we check S1' and S1".

Table 2 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), 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. To save space, we omit the figures for memory use from the table: Benchmarks 1–5 need 30–170 MB, whereas rs-bosco needs up to 1.5 GB per CPU.

Table 2 The experiments for first 5 rows were run on a single computer (Apple MacBook Pro 2018, 16GB). The experiments for last row (rs-bosco) were run in Grid5000 on 32 nodes (2 CPUs Intel Xeon Gold 6130, 16 cores/CPU, 192GB). Wall times are given.

Automaton			S1/S1'/S1"		S2		S3		S4		S5	
Name	$\vert\mathcal{L}\vert$	$\vert\mathcal{R}\vert$	$\vert\mathcal{S}\vert$	Time	$\vert\mathcal{S}\vert$	Time	$\vert\mathcal{S}\vert$	Time	$\vert\mathcal{S}\vert$	Time	$\vert\mathcal{S}\vert$	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	3:21	156M	3:02	156M	3:21	TO	TO	156M	3:43

The benchmark rs-bosco is a challenge for the technique of [10]: 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 2. (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 the benchmarks in Table 2, 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 a counterexample.

9 Conclusions

Our proof methodology for almost sure termination applies to round-rigid adversaries only. As future work we shall prove that verifying almost-sure termination under round-rigid adversaries is sufficient to prove it 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 fact that, depending on the outcome of a coin toss in some step at round k, different rules may be triggered later for processes in rounds less than k.

A few contributions address automated verification of probabilistic parameterized systems [22, 4, 20, 19]. 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 [22] requires shared variables to be bounded and cannot use arithmetic thresholds different from 1 and n. Algorithms for well-structured transition systems [4] do not directly apply to multi-parameter systems produced by probabilistic threshold automata. Regular model checking [20, 19] cannot handle arithmetic resilience conditions such as $n > 3t$, nor unbounded shared variables.

References

1 Marcos Aguilera and Sam Toueg. The correctness proof of Ben-Or's randomized consensus algorithm. *Distributed Computing*, pages 1–11, 2012. online first.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

3 Michael Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *PODC*, pages 27–30, 1983.

4 Nathalie Bertrand and Paulin Fournier. Parameterized Verification of Many Identical Probabilistic Timed Processes. In *FSTTCS*, volume 24 of *LIPIcs*, pages 501–513, 2013.

5 Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Inf. Comput.*, 75(2):130–143, 1987.

6 Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A Reduction Theorem for the Verification of Round-Based Distributed Algorithms. In *RP*, volume 5797 of *LNCS*, pages 93–106, 2009.

7 Andrei Damian, Cezara Drăgoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *CAV*, 2019. (to appear).

8 Tzilla Elrad and Nissim Francez. Decomposition of Distributed Programs into Communication-Closed Layers. *Sci. Comput. Program.*, 2(3):155–173, 1982.

9 Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with one Faulty Process. *J. ACM*, 32(2):374–382, 1985.

10 Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder. A Short Counterexample Property for Safety and Liveness Verification of Fault-tolerant Distributed Algorithms. In *POPL*, pages 719–734, 2017.

11 Igor Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. Para^2: Parameterized Path Reduction, Acceleration, and SMT for Reachability in Threshold-Guarded Distributed Algorithms. *Formal Methods in System Design*, 51(2):270–307, 2017.

12 Igor Konnov, Helmut Veith, and Josef Widder. On the Completeness of Bounded Model Checking for Threshold-Based Distributed Algorithms: Reachability. *Information and Computation*, 252:95–109, 2017.

13 Igor Konnov and Josef Widder. ByMC: Byzantine model checker. In *ISoLA (3)*, volume 11246 of *LNCS*, pages 327–342. Springer, 2018.

14 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in Parameterized Systems: All Flavors of Threshold Automata. In *CONCUR*, pages 19:1–19:17, 2018.

15 Marta Z. Kwiatkowska and Gethin Norman. Verifying Randomized Byzantine Agreement. In *FORTE*, pages 194–209, 2002.

16 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

17 Marta Z. Kwiatkowska, Gethin Norman, and Roberto Segala. Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM. In *CAV*, pages 194–206, 2001.

18 Daniel J. Lehmann and Michael O. Rabin. On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem. In *POPL*, pages 133–138, 1981.

19 Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. Fair Termination for Parameterized Probabilistic Concurrent Systems. In *TACAS*, volume 10205 of *LNCS*, pages 499–517, 2017. doi:10.1007/978-3-662-54577-5_29.

20 Anthony Widjaja Lin and Philipp Rümmer. Liveness of Randomised Parameterised Systems under Arbitrary Schedulers. In *CAV*, volume 9780 of *LNCS*, pages 112–133. Springer, 2016. doi:10.1007/978-3-319-41540-6_7.

21 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Randomized k-set agreement in crash-prone and Byzantine asynchronous systems. *Theor. Comput. Sci.*, 709:80–97, 2018.

22 Amir Pnueli and Lenore D. Zuck. Verification of Multiprocess Probabilistic Protocols. *Distributed Computing*, 1(1):53–72, 1986. doi:10.1007/BF01843570.

23 Yee Jiun Song and Robbert van Renesse. Bosco: One-Step Byzantine Asynchronous Consensus. In *DISC*, volume 5218 of *LNCS*, pages 438–450, 2008.