A Reduction Theorem for Randomized Distributed Algorithms Under Weak Adversaries

Nathalie Bertrand, Marijana Lazic, Josef Widder

A Reduction Theorem for Randomized Distributed Algorithms under Weak Adversaries*

Nathalie Bertrand[1], Marijana Lazić[2], and Josef Widder[3]

[1] Univ Rennes, Inria, CNRS, IRISA, France
nathalie.bertrand@inria.fr
[2] Technical University of Munich, Germany
lazic@in.tum.de
[3] Informal Systems, Vienna, Austria
josef@informal.systems

Abstract. Weak adversaries are a way to model the uncertainty due to asynchrony in randomized distributed algorithms. They are a standard notion in correctness proofs for distributed algorithms, and express the property that the adversary (scheduler), which has to decide which messages to deliver to which process, has no means of inferring the outcome of random choices, and the content of the messages.
In this paper, we introduce a model for randomized distributed algorithms that allows us to formalize the notion of weak adversaries. It applies to randomized distributed algorithms that proceed in rounds and are tolerant to process failures. For this wide class of algorithms, we prove that for verification purposes, the class of weak adversaries can be restricted to simple ones, so-called round-rigid adversaries, that keep the processes tightly synchronized. As recently a verification method for round-rigid adversaries has been introduced, our new reduction theorem paves the way to the parameterized verification of randomized distributed algorithms under the more realistic weak adversaries.

1 Introduction

Automated verification of fault-tolerant distributed algorithms faces the combinatorial explosion problem. The asynchronous parallel composition of many processes leads to a huge number of executions. Recently, several verification methods [18,14,6,9,5] are based on the idea that for many distributed algorithms, instead of considering all these asynchronous executions, it is sufficient to consider only fewer (representative) synchronous executions. The central argument is similar to the reductions (also know as, mover analysis) by Lipton [15] and Elrad and Francez [10]: given an arbitrary execution, by repeatedly swapping

* This project has received funding from Interchain Foundation (Switzerland), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

neighboring transitions, one arrives at one of the representative (synchronous) executions. As this argument works on executions (traces), it works for reachability properties, and for specific stuttering-insensitive linear temporal properties.

In this paper, we extend this idea to randomized distributed algorithms and probabilistic properties [4,16,2]. Rather than arguing on traces, probabilistic guarantees require us to reason on Markov decision processes (MDPs). In MDPs the non-determinism is resolved by using adversaries, that is, by functions that map an execution prefix to the next action taken. In case the next action is a coin toss, we obtain a branching, where each branch is associated with a probability. As a result, an MDP together with an adversary induce a computation tree with probabilistic branching. As the adversary is a function on the prefix, it is not clear whether in the presence of this branching, it is possible to conduct a swapping argument on the computation tree that maintains probabilistic properties. The technical challenge we face is to characterize a family of adversaries that permits a swapping argument in order to arrive at a computation tree that corresponds to a synchronous execution. Restricting to synchronous executions considerably decreases the verification effort, by reducing the number of executions to check. For the analysis of distributed consensus algorithms, there are two well-researched classes of adversaries, namely strong and weak adversaries. Strong adversaries have full knowledge of the execution prefix, while weak adversaries are based on a projection (abstraction) of the execution prefix, in particular, they do not have access to the content of the exchanged messages and the outcomes of coin tosses. In this paper, we formalize weak adversaries, and make explicit that they inherently impose restrictions on the local code of a distributed algorithm, that is, they can only be defined for a class of distributed algorithms (which was not apparent from their mathematical definition in the literature).

Intuitively, these algorithms expose some form of symmetry regarding the local control flow. Consider a formalization of Ben-Or's consensus algorithm [4] in Figure 1. The subscript in the locations (nodes) encode the local estimate of the consensus value, for instance D_0 and D_1 are locations where processes decide 0 and 1, respectively. We observe that the control flows on the 0 side and the 1 side are symmetric: if we ignore the subscripts the paths through the graph are identical. In contrast, consider the (made-up) example in Figure 2. If at location J there would be a branching due to receiving messages with different consensus estimates, the two paths that lead to F differ in length. An adversary may observe whether a process has taken the left path or the right path which allows the adversary to infer knowledge on the consensus value that led to branching at location J. However, typical randomized consensus algorithms from the literature [4,7,16,17] have a structure similar to Figure 1. Almost sure termination of these algorithms have been automatically verified in [5] under synchronous executions formalized via round-rigid adversaries. In this paper we show that for these distributed algorithms the computation trees that are defined by weak adversaries can be reduced to round-rigid computation

trees by a swapping argument. As a result we show that *the verification results from [5] apply to a wider class of adversaries than originally claimed.*

More formally, our new reduction theorem says that for each weak adversary there exists a round-rigid adversary that maintains the original probabilities of properties. For strong adversaries, we were not able to derive such a reduction argument, which indicates that the verification problem for strong adversaries is harder. This would also explain why the mathematical proofs in the literature for strong adversaries are considerably more involved [1].

Contributions. We present in Section 2 a new formalization of randomized distributed algorithms that allows us to define the weak adversary model from the literature [2]. Our model is based on threshold automata [12] and their probabilistic extension [5]. To faithfully express the weak adversaries, we introduce a process-based semantics, i.e., rather than the counter system semantics from [12,5], we propose semantics based on processes that exchange messages. We then prove our reduction in two steps. First, in Section 3, we reduce adversaries to communication-closed [10] adversaries, that is, adversaries that to a process in round r only deliver messages of round $r' \leq r$. Then, in Section 4, we reduce weak communication-closed adversaries to round-rigid adversaries.

2 Modeling Randomized Threshold-based Algorithms

Probabilistic threshold automata with semantics based on counter systems were introduced in [5]. For a discussion on the operation of threshold-based distributed algorithms, and how they are captured by threshold automata we refer to [5]. Here we provide more concrete semantics based on processes and message buffers (modeled as sets). A *probabilistic threshold automaton with processes*, $\mathsf{PTA}_{\mathbb{P}}$, is a tuple $(\mathcal{L}, \mathcal{Z}, \mathcal{R}, RC)$, where

- $\mathcal{L}$ is a non-empty finite set of locations that contains the disjoint subsets: *initial locations* $\mathcal{I}$, *final locations* $\mathcal{F}$, and *border locations* $\mathcal{B}$, with $|\mathcal{B}| = |\mathcal{I}|$.
- $\mathcal{Z}$ is a disjoint union of the following five sets:
  - Π is a set of *parameter variables*;
  - $\mathbb{P} = \{p_1, \ldots, p_n\}$, for some $n \geq 1$, is a finite set of *processes*; It is the disjoint union of $\mathbb{C}$ and $\mathbb{F}$, representing sets of correct and faulty processes, respectively;
  - $\mathcal{T}$ is a finite set of *types* of messages
  - $\mathcal{V}$ is a finite set of *values* of messages, typically $\mathcal{V} = \{0, 1\}$;
  - $\Lambda \subseteq \{x_{t,v} \mid t \in \mathcal{T},\ v \in \mathcal{V}\}$ is a set of *local receive variables*;
- $\mathcal{R}$ is a finite set of *rules*; and
- RC, the *resilience condition*, is a constraint over parameter variables.

Example 1. Figure 1 depicts a $\mathsf{PTA}_{\mathbb{P}}$ that formalizes the seminal consensus algorithm by Ben-Or [4]. It has locations $\mathcal{L} = \mathcal{B} \cup \mathcal{I} \cup \mathcal{F} \cup \{SR, SP\}$, where $\mathcal{B} = \{I_0, I_1\}$ are border locations, $\mathcal{I} = \{J_0, J_1\}$ are initial locations, and $\mathcal{F} =$

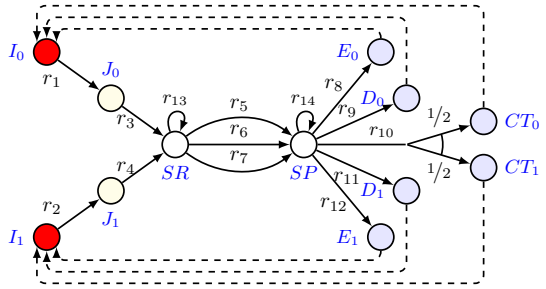

Fig. 1. Ben-Or's randomized consensus algorithm as a probabilistic threshold automaton with processes.

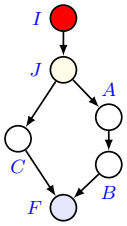

Fig. 2. Asymmetric threshold automaton used in Example 5.

Rule	Guard	Update
r_1	*true*	$\emptyset$
r_2	*true*	$\emptyset$
r_3	*true*	$\{x_0\}$
r_4	*true*	$\{x_1\}$
r_5	$x_0{+}x_1 \geq n{-}t \ \wedge \ x_0 \geq (n{+}t)/2$	$\{y_0\}$
r_6	$x_0{+}x_1 \geq n{-}t \ \wedge \ x_1 \geq (n{+}t)/2$	$\{y_1\}$
r_7	$x_0{+}x_1 \geq n{-}t \ \wedge$	
	$x_0 < (n{+}t)/2 \ \wedge \ x_1 < (n{+}t)/2$	$\{y_?\}$

Rule	Guard	Update
r_8	$y_0{+}y_1{+}y_? \geq n{-}t \ \wedge \ y_0 \geq t{+}1$	$\emptyset$
r_9	$y_0{+}y_1{+}y_? \geq n{-}t \ \wedge \ y_0 > (n{+}t)/2$	$\emptyset$
r_{10}	$y_0{+}y_1{+}y_? \geq n{-}t \ \wedge$	
	$y_0 < t{+}1 \ \wedge \ y_1 < t{+}1$	$\emptyset$
r_{11}	$y_0{+}y_1{+}y_? \geq n{-}t \ \wedge \ y_1 > (n{+}t)/2$	$\emptyset$
r_{12}	$y_0{+}y_1{+}y_? \geq n{-}t \ \wedge \ y_1 \geq t{+}1$	$\emptyset$
r_{13}	*true*	$\emptyset$
r_{14}	*true*	$\emptyset$

Table 1. The rules of the probabilistic threshold automaton for Ben-Or's algorithm from Figure 1, where $z_i \in \{x_0, x_1, y_0, y_1, y_?\}$ refers to messages of type z and value i.

$\{E_0, E_1, D_0, D_1, CT_0, CT_1\}$ are final locations. The set of parameters is $\Pi = \{n, t, f\}$, where n is the total set of processes, f is the number of faulty processes, and t is an upper bound on the number of faults. The 14 rules of the $\mathsf{PTA}_{\mathbb{P}}$ from Figure 1 are given in Table 1 (and detailed later). There are two message types, $\mathcal{T} = \{x, y\}$, and three values $\mathcal{V} = \{0, 1, ?\}$, where x-messages can only have values 0 and 1, and y-messages all three values. The local receive variables from Λ are thus written $x_0, x_1, y_0, y_1, y_?$ where we write shortly, e.g., type-value pair $(x, 0)$.

See [5] for an in-detail exposition of Ben-Or's algorithm and its formalization as threshold automaton. There, one can observe that the pseudo-code of this algorithm consists of a while loop, and one loop iteration is refered to as a *round*. In the threshold automaton in Figure 1, the solid arrows represent local transitions within a round, while dashed arrows represent local transitions to the next round. In each round, each process starts in I_0 or I_1. The subscript of the locations show what is the process' current estimate of the consensus value. A process informs its peers about its consensus estimate by firing rule r_3 or r_4 and sending a message of type x_0 or x_1, respectively. Then it waits in SR until sufficiently many — given by the guards — messages are received to fire r_5, r_6, and r_7, etc. If the thresholds are chosen properly, this shall ensure that if a process enters D_0 in some round, and thus decides 0, no process ever enters

D_1 in some round and decided 1 (agreement of consensus). The randomization is introduced in rule r_{10}: this is a coin toss where a process chooses its estimate for the next round if there was no clear majority around a value. The dashed arrows then show how a process transitions from a final location of round r to the beginning of round $r+1$. Performing an infinite number of rounds, and (if necessary) coin tosses, shall ensure that eventually every process decides.

Resilience condition. Let $\mathbb{N}_0$ denote the set of natural numbers including zero. A resilience condition RC defines the set of *admissible parameter values* $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Pi|} \colon \mathbf{p} \models RC\}$, for which the algorithm is designed to be correct. For example, Ben-Or's consensus algorithm is correct when $n/5 > t \geq f \geq 0$. We introduce a function $N \colon \mathbf{P}_{RC} \to \mathbb{N}_0$ that maps a vector of admissible parameters to a number of modeled processes in the system. For instance, for the automaton in Figure 1, N is the function $(n,t,f) \mapsto n-f$, as we model only the $n-f$ correct processes explicitly, while the effect of Byzantine faulty processes is captured in non-deterministic choices between different guards. For crash-resilient algorithms, where all processes are initially correct (until they crash), we model them all explicitly, that is, $N(n,t,f) = n$. The set of modeled processes is then $\mathbb{C} = \{p_1, \ldots, p_N\}$.

Messages. The set of all messages is $\mathcal{M} = (\mathbb{P} \times \mathcal{T} \times \mathcal{V} \times \mathbb{N}_0)$. A message m is a tuple $(\mathit{sen}, \mathit{type}, \mathit{val}, \mathit{rnd})$ where the process $\mathit{sen} \in \mathbb{P}$ is the sender, the message type is $\mathit{type} \in \mathcal{T}$, the value is $\mathit{val} \in \mathcal{V}$, and the message is sent in the round $\mathit{rnd} \in \mathbb{N}_0$. Note that we do not make explicit the process receiving the message, because we focus on broadcast communications, and thus messages are sent to every process.

Let $\mathcal{M}_{\mathbb{F}} = \mathbb{F} \times \mathcal{T} \times \mathcal{V} \times \mathbb{N}_0$ be the subset of all messages where the sender is a faulty process, and $\mathcal{M}_{\mathbb{C}} = \mathbb{C} \times \mathcal{T} \times \mathcal{V} \times \mathbb{N}_0$ the subset of messages sent by correct processes. In our example from Figure 1 we have $|\mathbb{F}| = f$ and $|\mathbb{C}| = n - f$.

In the sequel, we assume $\mathcal{M}$ is equipped with a total order $<_{\mathcal{M}}$. This total order can be naturally derived from the order on $\mathbb{N}_0$, and fixed orders on the processes, on the types, and on the values.

Rules. We introduce rules in detail, and give syntactic restrictions that model the local transitions of a distributed algorithm from/to particular locations. A rule r is a tuple $(\mathit{from}, \delta_{to}, \varphi, \boldsymbol{u})$ 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, $\boldsymbol{u} \subseteq \mathcal{T} \times \mathcal{V}$ is the *update set*, and φ is a guard, *i.e.*, a conjunction of expressions of the form $\sum_{v \in \mathcal{V}} (b_v \cdot x_{t,v}) \mathbin{\square} \bar{a} \cdot \mathbf{p}^{\intercal} + a_0$ where $t \in \mathcal{T}$ is a fixed message type; for a message value $v \in \mathcal{V}$, $b_v \in \mathbb{N}_0$ is a non-negative integer and $x_{t,v} \in \Lambda$ is a local receive variable; $\mathbin{\square} \in \{\geq, <\}$, $\bar{a} \in \mathbb{Z}^{|\Pi|}$ is a vector of integers, $a_0 \in \mathbb{Z}$, and $\mathbf{p}$ is the vector of all parameters. If a guard contains only one conjunct, we sometimes call it *a simple threshold guard* (or just *a simple guard*). The set of all simple guards that appear in a probabilistic threshold automaton $\mathsf{PTA}_{\mathbb{P}}$ is denoted by $\mathcal{G}(\mathsf{PTA}_{\mathbb{P}})$.

If $r.\delta_{to}$ is a Dirac distribution, *i.e.*, if there exists $\ell \in \mathcal{L}$ such that $r.\delta_{to}(\ell) = 1$, we call r a *Dirac rule*, and simply denote it $(\mathit{from}, \ell, \varphi, \boldsymbol{u})$.

Probabilistic threshold automata model algorithms with multiple rounds that follow the same code. They represent the behaviour each correct process follows within a round. 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 $(\ell, \ell', \mathtt{true}, \emptyset)$ 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 $(\ell, \ell', \mathtt{true}, \emptyset)$ 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 $\mathcal{B}$ iff all the incoming edges are in $\mathcal{S}$. Similarly, a location is in $\mathcal{F}$ iff there is only one outgoing edge and it is in $\mathcal{S}$.

Example 2. Back to our running example, the only rule that is not a Dirac rule is r_{10}, and round switch rules are represented by dashed arrows. Also the update sets are either empty sets or singletons, where we again write shortly, *e.g.*, x_0 instead of the type-value pair $(x, 0)$.

2.1 Symmetry in $\mathsf{PTA}_{\mathbb{P}}$

In the distributed algorithm community, weak adversaries are typically defined by not being able to observe message content and the outcome of coin tosses. In the $\mathsf{PTA}_{\mathbb{P}}$ model, one can often retrieve information about the outcome of a coin toss by the location a process ends up in, or about the message contents by the rule that is taken. For instance, in our example, depending on the outcome of a coin toss, a process goes either to location CT_0 or CT_1. Also, firing r_5 or r_6 reveals which messages of type x—with value 0 or 1—are in the majority. This motivates the introduction of two equivalence relations, one on locations and one on guards (and thus rules). In our example on the one hand, the locations CT_0 and CT_1 should be equivalent, and on the other hand the rules r_5 and r_6 should be equivalent. In the following, we formalize weak adversaries using such symmetries in threshold automata.

Equivalence relations on guards and rules. Let us first define a correspondence between threshold guards. Fix two simple threshold guards

$$\varphi_1 \colon \sum_{v \in \mathcal{V}} (b_{t_1,v} \cdot x_{t_1,v}) \,\square_1\, \bar{a} \cdot \mathbf{p}^\intercal + a_0 \quad \text{and} \quad \varphi_2 \colon \sum_{v \in \mathcal{V}} (d_{t_2,v} \cdot x_{t_2,v}) \,\square_2\, \bar{c} \cdot \mathbf{p}^\intercal + c_0.$$

We say that φ_1 and φ_2 *correspond to each other*, denoted by $\varphi_1 \equiv_\varphi \varphi_2$, if:

- $\square_1$ and $\square_2$ are the same relation, either $\geq$ or $<$,
- coefficients are the same, that is, $\bar{a} = \bar{c}$ and $a_0 = c_0$,
- message types are the same, that is, $t_1 = t_2$,
- there exists a permutation π on the set of values $\mathcal{V}$, such that $b_{t_1,v} = d_{t_2,\pi(v)}$.

We extend this definition to threshold guards. Let $\varphi_A = \varphi_1^A \wedge \ldots \wedge \varphi_k^A$ and $\varphi_B = \varphi_1^B \wedge \ldots \wedge \varphi_m^B$ be threshold guards, and each φ_i^A and φ_j^B be a simple guard. We say φ_A and φ_B correspond to each other, and write $\varphi_A \equiv_\varphi \varphi_B$, if $k = m$ and there is a permutation ρ on the set $\{1, \ldots, k\}$ such that $\varphi_i^A \equiv_\varphi \varphi_{\rho(i)}^B$ for every $1 \leq i \leq k$.

Example 3. In case $x, y \in \mathcal{T}$ and $0, 1, 2 \in \mathcal{V}$, we have $y_0 \geq n - 2t \equiv_\varphi y_1 \geq n - 2t$ and $y_0 + 3y_1 < t + 1 \equiv_\varphi y_1 + 3y_2 < t + 1$. In the example from Table 1 we have $r_5 \equiv_\varphi r_6$, $r_8 \equiv_\varphi r_{12}$, and $r_9 \equiv_\varphi r_{11}$.

The equivalence relation over guards, allows us to define an equivalence relation $\equiv_\mathcal{R} \subseteq \mathcal{R} \times \mathcal{R}$ on rules. Let r_1 and r_2 be two rules from $\mathcal{R}$. We have $r_1 \equiv_\mathcal{R} r_2$ if and only if it holds that: $r_1.\varphi \equiv_\varphi r_2.\varphi$, and there exists a permutation π on the set of values $\mathcal{V}$, such that $(t, v) \in r_1.\boldsymbol{u}$ if and only if $(t, \pi(v)) \in r_2.\boldsymbol{u}$.

Example 4. Consider again our example from Table 1. The rules r_1, r_2, r_3, r_4 have trivial guards, which are therefore all in the same equivalence class of $\equiv_\varphi$. In contrast, not all the rules are equivalent w.r.t. relation $\equiv_\mathcal{R}$, as their update sets are different. Thus, we have $r_1 \equiv_\mathcal{R} r_2$ and $r_3 \equiv_\mathcal{R} r_4$.

Equivalence relation on locations. We define equivalence relation $\equiv_\mathcal{L} \subseteq \mathcal{L} \times \mathcal{L}$ on locations inductively as follows:

- The set of border locations $\mathcal{B}$ is one equivalence class of $\equiv_\mathcal{L}$, that is, for every $\ell_1, \ell_2 \in \mathcal{B}$ and every $\ell_3 \notin \mathcal{B}$ it holds that $\ell_1 \equiv_\mathcal{L} \ell_2$, and $\ell_1 \not\equiv_\mathcal{L} \ell_3$.
- Let ℓ_1 and ℓ_2 be two locations from $\mathcal{L} \setminus \mathcal{B}$. We have $\ell_1 \equiv_\mathcal{L} \ell_2$ if and only if there exist rules r_1 and r_2 and locations ℓ_1^s and ℓ_2^s such that
  - ℓ_i^s is a source location of r_i, for $i = 1, 2$, that is, $r_i.\mathit{from} = \ell_i^s$,
  - ℓ_i is a destination location for r_i, formally, $r_i.\delta_{to}(\ell_i) > 0$, for $i = 1, 2$,
  - $\ell_1^s \equiv_\mathcal{L} \ell_2^s$, and $r_1 \equiv_\mathcal{R} r_2$.
- The set of final locations $\mathcal{F}$ is either one equivalence class of $\equiv_\mathcal{L}$ or a union of finitely many equivalence classes of $\equiv_\mathcal{L}$. As a consequence, there are no two locations $\ell_1 \in \mathcal{F}$ and $\ell_2 \notin \mathcal{F}$ such that $\ell_1 \equiv_\mathcal{L} \ell_2$.

Let $\mathsf{PTA}_\mathbb{P}$ be a probabilistic threshold automaton with processes, equipped with equivalence relations $\equiv_\mathcal{L}$ and $\equiv_\mathcal{R}$. Assume $\ell, \ell_0, \ell_1 \in \mathcal{L}$ are locations, and $r = (\mathit{from}, \delta_{to}, \varphi, \boldsymbol{u}) \in \mathcal{R}$ is a non-Dirac rule such that its source location is ℓ ($r.\mathit{from} = \ell$), and ℓ_0 and ℓ_1 are its destination locations. Then $\ell_0 \equiv_\mathcal{L} \ell_1$. In words, all destinations of a non-Dirac rule are equivalent locations.

Example 5. In Figure 1 we have 7 equivalence classes w.r.t. $\equiv_\mathcal{L}$, namely $\{I_0, I_1\}$, $\{J_0, J_1\}$, $\{SR\}$, $\{SP\}$, $\{E_0, E_1\}$, $\{D_0, D_1\}$, and $\{CT_0, CT_1\}$.

Such an equivalence relation does not always exist, due to the last requirement on final locations. For instance, on the automaton from Figure 2, where $I \in \mathcal{B}$, $J \in \mathcal{I}$, $F \in \mathcal{F}$, where all rules have guard *true* and empty update set, it is not possible to define $\equiv_\mathcal{L}$. Intuitively, an adversary is able to infer whether the left or the right branch is taken, and consequently in similar asymmetric automata it may infer information about message content or coin tosses. However, typical randomized consensus algorithms from the literature [4,7,16,17] have a structure similar to the one in Figure 1, and are thus symmetric.

2.2 Semantics of a $\mathsf{PTA}_{\mathbb{P}}$

The semantics of a probabilistic threshold automaton with processes is an infinite-state Markov decision process (MDP), which we formally define below.

Given a $\mathsf{PTA}_{\mathbb{P}}$ and a function N (defined earlier), we define the semantics, called *probabilistic system with processes* $\mathsf{Sys}(\mathsf{PTA}_{\mathbb{P}})$, to be infinite-state MDP $(\Sigma, I, \mathsf{Act}, \Delta)$, where Σ is the set of configurations for $\mathsf{PTA}_{\mathbb{P}}$ among which $I \subseteq \Sigma$ are initial, the set of actions is $\mathsf{Act} = \mathcal{P}(\mathcal{M}) \times \mathbb{C}$, and $\Delta \colon \Sigma \times \mathsf{Act} \to \mathsf{Dist}(\Sigma)$ is the probabilistic transition function.

Configurations. A configuration σ is a tuple $(\mathbf{s}, \mathit{Sent}, \mathit{Rcvd}, \mathbf{p})$, where the components are defined as follows:

- $\sigma.\mathbf{s} \colon \mathbb{C} \to \mathcal{L} \times \mathbb{N}_0$ is a function that describes the control states of processes, that is, the location and the round of each correct process,
- $\sigma.\mathit{Sent} \subseteq \mathcal{M}_{\mathbb{C}}$ is a set of messages sent by correct processes,
- $\sigma.\mathit{Rcvd} \colon \mathbb{C} \to \mathcal{P}(\mathcal{M})$ is a function that keeps track of the received messages for every correct process.
- $\sigma.\mathbf{p} \in \mathbb{N}_0^{|\Pi|}$ is a vector of parameter values.

We write $\sigma.\mathit{Rcvd}[p]_{|t,v,k}$ for the set of messages from $\sigma.\mathit{Rcvd}[p]$ of type t and value v that are sent in round k. Formally,

$$\sigma.\mathit{Rcvd}[p]_{|t,v,k} = \{m \in \sigma.\mathit{Rcvd}[p] \mid m.\mathit{type} = t \wedge m.\mathit{val} = v \wedge m.\mathit{rnd} = k\}.$$

We write $\sigma.\mathbf{s}_{\mathrm{loc}} \colon \mathbb{C} \to \mathcal{L}$ and $\sigma.\mathbf{s}_{\mathrm{rnd}} \colon \mathbb{C} \to \mathbb{N}_0$ for the projections to the first and the second component of $\sigma.\mathbf{s}$, respectively.

A configuration $\sigma = (\mathbf{s}, \mathit{Sent}, \mathit{Rcvd}, \mathbf{p})$ is *initial* if all processes are in border locations of round 0, and there are no sent nor received messages in any round:

- $\sigma.\mathit{Sent} = \emptyset$,
- for every $p \in \mathbb{C}$ we have $\sigma.\mathit{Rcvd}[p] = \emptyset$,
- for every $p \in \mathbb{C}$ there is a location $\ell \in \mathcal{B}$ such that $\sigma.\mathbf{s}[p] = (\ell, 0)$.

A threshold guard evaluates to true in a configuration σ for a process p and a round k, written $\sigma, p, k \models \varphi$, if for all its conjuncts $\sum_{v \in \mathcal{V}}(b_v \cdot x_{t,v}) \geq \bar{a} \cdot \mathbf{p}^{\intercal} + a_0$ we have $\sum_{v \in \mathcal{V}}(b_v \cdot |\sigma.\mathit{Rcvd}[p]_{|t,v,k}|) \geq \bar{a} \cdot (\sigma.\mathbf{p}^{\intercal}) + a_0$, and similarly for conjuncts of the other form, *i.e.*, $\sum_{v \in \mathcal{V}}(b_v \cdot x_{t,v}) < \bar{a} \cdot \mathbf{p}^{\intercal} + a_0$.

Actions. An action $\alpha = (M, p) \in \mathsf{Act}$ stands for the atomic execution of the following two steps: (i) process p receives the set of messages $M \subseteq \mathcal{M}$, and after that (ii) process p makes progress by executing a rule, if possible.

An action $\alpha = (M, p)$ is *applicable* to a configuration σ if each message from M has either been sent by a correct process or it comes from a faulty process, i.e., for every $m \in M$ we have: $m \in \sigma.\mathit{Sent}$ or $m \in \mathcal{M}_{\mathbb{F}}$.

A rule $r = (\mathit{from}, \delta_{to}, \varphi, \boldsymbol{u})$ is *executable* by a process p in a configuration σ with $\sigma.\mathbf{s}[p] = (\ell, k)$ if: (i) p is in the source location of the rule, that is, $\mathit{from} = \ell$,

and (ii) the guard evaluates to true in σ for p and k, that is, $\sigma, p, k \models \varphi$. In every configuration for every process there is at most one executable rule.

It is also important to note the role of the round number k in the definition of an executable rule. Whether a rule r is executable by p in σ depends only on the messages from the round in which this process is in σ. Thus, threshold automata are communication-closed [10,9] by construction and thus provide an effective model for many communication-closed fault-tolerant distributed algorithms in the literature. We consider this notion in more detail in Section 3.

Let σ be a configuration and let action $\alpha = (M, p)$ be applicable to σ. When α is applied to σ, process p receives messages from M, which results in configuration σ_{aux}, and then executes a rule r that is executable by p in σ_{aux} (if there is an executable rule), which finally results in σ'. Note that $M = \emptyset$ implies that $\sigma = \sigma_{aux}$, and if there is no executable rule in σ_{aux} then we have $\sigma_{aux} = \sigma'$.

Let us define a function $exec : \Sigma \times \mathsf{Act} \to \mathcal{R} \cup \{\bot\}$ that given a configuration σ and an action $\alpha = (M, p)$ applicable to σ, outputs (i) the unique rule r that is executable by p in configuration σ_{aux} obtained from σ by changing only $\sigma_{aux}.Rcvd[p] = \sigma.Rcvd[p] \cup M$, if such a rule exists, and (ii) it outputs $\bot$ if no such rule exists. We define $exec(\sigma, \alpha) = \bot$ if α is not applicable to σ.

Let $\alpha = (M, p)$ be an action applicable to σ, and let ℓ be either a potential destination location of $exec(\sigma, \alpha) \neq \bot$, or ℓ is the location of p in σ if $exec(\sigma, \alpha) = \bot$. We write $apply(\sigma, \alpha, \ell)$ for the resulting configuration: parameters are unchanged, all messages from M are added to $\sigma.Rcvd[p]$, and if $exec(\sigma, \alpha) = r \neq \bot$, then new messages from $\mathcal{M}$ are added to $\sigma.Sent$ according to the update set $r.\boldsymbol{u}$, and finally while the location and the round of all processes except p are unchanged, we have that location of p becomes ℓ and its round is unchanged (or increased by 1 if r is a round switch rule).

Formally, if $\alpha = (M, p)$, we have that $apply(\sigma, \alpha, \ell) = \sigma'$ if and only if $apply(\sigma, \alpha, \ell)$ is defined and the following holds:

- The parameter values do not change: $\sigma'.\mathbf{p} = \sigma.\mathbf{p}$.
- Process p receives all messages from M, formally, $\sigma'.Rcvd[p] = \sigma.Rcvd[p] \cup M$.
- The control states of processes, that is, their locations and rounds given by the function $\sigma'.\mathbf{s}$, are updated as follows:
  - After updating $\sigma.Rcvd[p]$, if there is no executable r for p, that is, if $exec(\sigma, \alpha) = \bot$, then the control states of all processes remain the same: $\sigma'.\mathbf{s} = \sigma.\mathbf{s}$.
  - Otherwise, if $exec(\sigma, \alpha) = r \neq \bot$, then the control states for all the processes except for p remain the same. Formally, $\sigma'.\mathbf{s}[q] = \sigma.\mathbf{s}[q]$ for every $q \neq p$.
    Process p moves to location ℓ and either (i) it stays in the same round if $r \notin \mathcal{S}$ is not a round switch rule, or (ii) it moves to the following round if $r \in \mathcal{S}$. Formally, if we denote the round of p in σ by $\sigma.\mathbf{s}_{\mathrm{rnd}}[p] = k$, then we have that $\sigma'.\mathbf{s}[p] = (\ell, k)$ if r is not a round switch, and $\sigma'.\mathbf{s}[p] = (\ell, k+1)$ if r is a round switch rule.
- The set of sent messages is updated as follows:
  - If $exec(\sigma, \alpha) = \bot$, then no rule is fired and thus no message is sent, that is, $\sigma'.Sent = \sigma.Sent$.

- If $exec(\sigma, \alpha) = r \neq \bot$ then the rule r is fired, and the update set $r.\boldsymbol{u}$ dictates the set of messages (their types and values) that process p sends to all in round $k = \sigma.\mathbf{s}_{\text{rnd}}[p]$, that is, $\sigma'.Sent = \sigma.Sent \cup \{(p, t, v, k) \mid (t, v) \in r.\boldsymbol{u}\}$.

Let $\alpha = (M, p)$ be applicable to σ, and let process p be in round k in configuration σ, that is, $\sigma.\mathbf{s}(p) = (\ell, k)$ for some location ℓ and round k. Then we define the *round of action* α *in configuration* σ to be k, and denote this by $rnd_\sigma(\alpha) = k$. When it is clear from the context which configuration we refer to, we only write $rnd(\alpha)$ instead of $rnd_\sigma(\alpha)$.

Probabilistic transition function. The probabilistic transition function Δ is defined such that for every two configurations σ and σ' and for every action α applicable to σ, with $exec(\sigma, \alpha) = r \in \mathcal{R} \cup \{\bot\}$, we have

$$\Delta(\sigma, \alpha)(\sigma') = \begin{cases} r.\delta_{to}(\ell) & \text{if } apply(\sigma, \alpha, \ell) = \sigma' \text{ for some } \ell \in \mathcal{L} \\ 0 & \text{otherwise.} \end{cases}$$

Note that if $r = \bot$ we define $r.\delta_{to}(\ell) = 1$, and if there exists a location ℓ with $apply(\sigma, \alpha, \ell) = \sigma'$, this location is uniquely defined.

Paths. A (finite or infinite) *path* in $\mathsf{Sys}(\mathsf{PTA})$ is an alternating sequence of configurations and actions $\sigma_0, \alpha_0, \sigma_1, \alpha_1 \ldots$, such that for $i > 0$, there exists a location ℓ_i such that $apply(\sigma_{i-1}, \alpha_{i-1}, \ell_i) = \sigma_i$. We denote the set of all paths by Paths and the set of all finite paths (ending with a configuration) by Paths_{fin}. The length of a finite path $\rho = \sigma_0, \alpha_0, \sigma_1, \alpha_1 \ldots, \sigma_k$ is the number of actions taken, that is, $|\rho| = k$. Wlog if ρ is an infinite path, we let $|\rho| = \infty$. We sometimes consider prefixes of a (finite or infinite) path ρ, and for $s < |\rho|$ write ρ_s for $\sigma_0, \alpha_0, \ldots, \sigma_s$. Also the last configuration σ_k of a finite path $\rho = \sigma_0, \alpha_0 \ldots, \alpha_{k-1}, \sigma_k$ is written $last(\rho)$. As sent messages cannot be unsent, the set of sent messages can only grow along a path. Thus, the set $last(\rho).Sent$ contains the set $\sigma_i.Sent$ for every $0 \leq i \leq k$. That is why we often write $\rho.Sent$ instead of $last(\rho).Sent$.

2.3 Message identities

An adversary formalizes which messages will be received next. When formalizing weak adversaries, we have to capture that the adversary can pick a message without being aware of the content of the message. For this we introduce message identities in the model. Note that every action may include sending a finite number of messages. Therefore, in a finite path there are finitely many sent messages, and we can assign them their identities (IDs for short). For a path $\rho \in \mathsf{Paths}_{fin}$, we define IDs of messages sent by correct processes along ρ by a function $ID[\rho] : \rho.Sent \to \mathbb{N}$ defined recursively on the length ρ:

Base case. If ρ is a degenerative path $\rho = \sigma_0$, then σ_0 is an initial configuration, and therefore, there are no sent messages in it. Formally, $\sigma_0.Sent = \emptyset$ and there is nothing to assign.

Recursion. Let $\rho = \tau\alpha\sigma \in \mathsf{Paths}_{fin}$ be a non-degenerative finite path. We distinguish two cases depending if new messages were sent while executing α.

- If no message is sent when applying α to $last(\tau)$, *i.e.* if either no rule is executed $exec(last(\tau), \alpha) = \bot$, or if there is an executed rule $exec(last(\tau), \alpha) = r \neq \bot$ but $r.\boldsymbol{u} = \emptyset$, then the function ID is unchanged: we set $ID[\rho] = ID[\tau]$.
- Otherwise, let $r = exec(last(\tau), \alpha)$ and let the set of messages sent when executing rule r from $last(\tau)$ be $r.\boldsymbol{u} = \{m_1, \ldots, m_k\}$ for some $k \geq 1$, with $m_1 <_{\mathcal{M}} m_2 \cdots <_{\mathcal{M}} m_k$. Then the function $ID[\rho]$ is defined as follows:

$$ID[\rho](m) = \begin{cases} ID[\tau](m) & \text{if } m \in \tau.Sent, \\ |\tau.Sent| + i & \text{if } m = m_i \in r.\boldsymbol{u} \text{ for } 1 \leq i \leq k \end{cases}$$

Revealing messages from their IDs. It is important to notice that for every natural number $n \leq |\tau.Sent|$ there is a unique message $m \in \tau.Sent$ with that identity, that is, with $ID(m) = n$. We define an inverse of msg when defined. Given a path $\rho \in \mathsf{Paths}_{fin}$ we define a function $rev\text{-}msg[\rho] : \{1, 2, \ldots, |\rho.Sent|\} \to \rho.Sent$, such that for every n with $1 \leq n \leq |\rho.Sent|$ we have $rev\text{-}msg[\rho](n) = m$ if and only if $ID[\rho](m) = n$.

We extend this definition to a set of IDs, and define $rev\text{-}msg[\rho]$ of a set of natural numbers $N \subset \mathbb{N}$ to be $rev\text{-}msg[\rho](N) = \{rev\text{-}msg[\rho](n) \mid n \in N\}$.

Faulty messages. Recall that $\mathcal{M}_{\mathbb{F}} = \mathbb{F} \times \mathcal{T} \times \mathcal{V} \times \mathbb{N}_0$ is the set of messages $m = (p, t, v, k) \in \mathcal{M}$ with the sender being a faulty process, that is, $p \in \mathbb{F}$. Note that this set has countably infinitely many elements, and therefore there exists a bijection between the set of natural numbers and $\mathcal{M}_{\mathbb{F}}$. We choose one such bijection $ID_f : \mathcal{M}_{\mathbb{F}} \to \mathbb{N}$ to be an enumerating function for the set of the faulty messages.

Similarly, we define a reveal function $rev\text{-}msg_f : \mathbb{N} \to \mathcal{M}_{\mathbb{F}}$ as the inverse of the identity function, that is, $rev\text{-}msg_f = ID_f^{-1}$. Moreover, for a set of natural numbers $N \subset \mathbb{N}$ we define $rev\text{-}msg_f(N) = \{rev\text{-}msg_f(n) \mid n \in N\}$.

2.4 Adversaries

The non-determinism in Markov decision processes is traditionally resolved by a so-called adversary, see *e.g.* [3, Chap. 10]. An *adversary* is a function $\mathtt{a} : \mathsf{Paths}_{fin} \to 2^{\mathbb{N} \times \{c,f\}} \times \mathbb{C}$ that given a finite path $\rho = \sigma_0, \alpha_0, \sigma_1, \ldots, \sigma_k$ of $\mathsf{Sys}(\mathsf{PTA}_{\mathbb{P}})$ selects a set of message IDs with the nature of their senders (a set of elements from $\mathbb{N} \times \{c\}$ or $\mathbb{N} \times \{f\}$) together with a correct process (thus from $\mathbb{C}$) to whom these messages are delivered.

As an adversary only gives message IDs, we need to understand which messages correspond to them. This is why we introduce the function $reveal[\mathtt{a}] : \mathsf{Paths}_{fin} \to \mathsf{Act}$ that reveals the next action in a path according to the choice of the adversary $\mathtt{a}$. Let $\mathtt{a}(\rho) = (N_1 \times \{c\} \times \{p\}) \cup (N_2 \times \{f\} \times \{p\})$, where N_1 and N_2 are finite sets of natural numbers, and $p \in \mathbb{C}$ is a correct process. Then we define $reveal[\mathtt{a}](\rho)$ to be the action $(M_1 \cup M_2, p)$, where $M_1 = rev\text{-}msg[\rho](N_1)$ and $M_2 = rev\text{-}msg_f(N_2)$.

Given a path ρ, we also define a function $\mathit{choice}[\rho] : 2^{\rho.\mathit{Sent}} \times \mathbb{C} \to 2^{\mathbb{N}\times\{c,f\}} \times \mathbb{C}$ that tells us which choice $\mathsf{a}(\rho)$ should the adversary take in order to obtain the expected action. Let $(M, p) \in 2^{\rho.\mathit{Sent}} \times \mathbb{C}$, with $M = M_1 \cup M_2$ where M_1 are messages sent by correct processes and M_2 are messages sent by faulty processes. Let $N_1 = \{\mathit{ID}[\rho](m) \mid m \in M_1\}$ and let $N_2 = \{\mathit{ID}_f(m) \mid m \in M_2\}$. Then we have $\mathit{choice}[\rho](M, p) = (N, p)$, where $N = (N_1 \times \{c\}) \cup (N_2 \times \{f\})$.

Given an initial configuration σ_0, an adversary a generates a set $\mathsf{paths}(\sigma_0, \mathsf{a})$ of infinite paths $\sigma_0, \alpha_0, \sigma_1, \ldots$ with the following property: for every $i > 0$, $\alpha_i = \mathit{reveal}[\mathsf{a}](\sigma_0, \sigma_1, \ldots, \sigma_{i-1})$ and there exists a location ℓ_i such that $\sigma_i = \mathit{apply}(\sigma_{i-1}, \alpha_i, \ell_i)$. Every infinite path $\pi \in \mathsf{paths}(\sigma_0, \mathsf{a})$, and every finite path ρ which is a prefix of an infinite path $\pi \in \mathsf{paths}(\sigma_0, \mathsf{a})$, are said to be *induced by* a.

The MDP $\mathsf{Sys}(\mathsf{PTA}_{\mathbb{P}})$ together with an initial configuration σ_0 and an adversary a induce a Markov chain, denoted by $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$. Precisely, the state space of $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ is $\mathsf{Paths}_{\mathit{fin}}$, its initial state is the initial configuration σ_0—which is also a path of length 0—and the probabilistic transition function $\delta_{\mathsf{a},\sigma_0} : \mathsf{Paths}_{\mathit{fin}} \to \mathsf{Dist}(\mathsf{Paths}_{\mathit{fin}})$ is defined for every $\tau \in \mathsf{Paths}_{\mathit{fin}}$ starting in σ_0 and ending in some configuration σ, for every action α, and every $\sigma' \in \Sigma$ by:

$$\big(\delta_{\mathsf{a}}(\tau)\big)(\tau\alpha\sigma') = \Delta(\sigma, \mathit{reveal}[\mathsf{a}](\tau))(\sigma').$$

In words, the probability in $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ to move from state τ to state $\tau\alpha\sigma'$ is non-zero as soon as there exists an action α' such that $\sigma' = \mathit{apply}(\sigma, \alpha', \ell)$ and a picks α'. This equals the probability that the corresponding process moves to ℓ if the adversary a picks action α'. Note that the Markov chain $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ is acyclic, and even has the shape of a tree, since its states are the finite paths in $\mathsf{Paths}_{\mathit{fin}}$. We write $\mathbf{P}_{\mathsf{a}}^{\sigma_0}$ for the probability measure over infinite paths starting at σ_0 in $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$.

Given σ_0, a and a finite path $\rho = \sigma_0, \alpha_0, \ldots, \sigma_k \in \mathsf{Paths}_{\mathit{fin}}$, we write $\mathcal{M}_{\mathsf{a}}^{\rho}$ for the Markov chain which corresponds to the part of $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ with initial state ρ. The probability measure $\mathbf{P}_{\mathsf{a}}^{\rho}$ in $\mathcal{M}_{\mathsf{a}}^{\rho}$ is inherited from the one in $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$.

Weak adversaries. In order to define weak adversaries, we introduce an equivalence relation on paths.

For two sets of messages M_1 and M_2 we say they are *equivalent up to message values* if there is a bijection $f : M_1 \to M_2$ such that for every $m \in M_1$ we have that m and $f(m)$ have the same sender, type and round. Formally, we have $m.\mathit{sen} = f(m).\mathit{sen}$, $m.\mathit{type} = f(m).\mathit{type}$, $m.\mathit{rnd} = f(m).\mathit{rnd}$.

The weak observation relation relates two configurations that differ only in message content and symmetric locations of processes. Formally:

Definition 1. *The* weak observation relation *is the equivalence relation* $\equiv_w \subseteq \Sigma^2$ *such that* $\sigma \equiv_w \sigma'$ *if and only if*

- *for every correct process* $p \in \mathbb{C}$*, if* $\sigma.\mathbf{s}(p) = (\ell_1, k_1)$ *and* $\sigma'.\mathbf{s}(p) = (\ell_2, k_2)$*, then* $k_1 = k_2$ *and* $\ell_1 \equiv_{\mathcal{L}} \ell_2$*.*
- $\sigma.\mathit{Sent}$ *and* $\sigma'.\mathit{Sent}$ *are equivalent up to message values*
- *for all* $p \in \mathbb{C}$*,* $\sigma.\mathit{Rcvd}(p)$ *and* $\sigma'.\mathit{Rcvd}(p)$ *are equivalent up to message values*

– $\sigma.\mathbf{p} = \sigma'.\mathbf{p}$

We extend the relation to finite paths of the same length: if $\pi = \sigma_1, \sigma_2, \ldots, \sigma_k$ *and* $\pi' = \sigma'_1, \sigma'_2, \ldots, \sigma'_k$*, then we write* $\pi \equiv_w \pi'$ *if* $\sigma_i \equiv_w \sigma'_i$ *for every* $1 \leq i \leq k$*.*

An adversary $\mathtt{a}$ is *weak* if for every two finite paths π and π' with $\pi \equiv_w \pi'$ we have that $\mathtt{a}(\pi) = \mathtt{a}(\pi')$. In words, a weak adversary does not distinguish two paths if they are equivalent, and thus makes the same choice for equivalent paths.

Note that for threshold automata on which $\equiv_{\mathcal{L}}$ cannot be defined, we can also not define $\equiv_w$. Therefore, weak adversaries are a property of only those distributed algorithms that can be modeled by symmetric automata.

Lemma 1. *Let* $\pi = \sigma_0, \ldots, \sigma_k, \alpha_k, \sigma_{k+1}$ *and* $\bar{\pi} = \sigma_0, \ldots, \sigma_k, \alpha_k, \bar{\sigma}_{k+1}$ *be two paths such that* $exec(\sigma_k, \alpha_k) = r$ *is a non-Dirac rule with two destination locations* $\ell \neq \bar{\ell}$*, with* $apply(\sigma_k, \alpha_k, \ell) = \sigma_{k+1}$ *and with* $apply(\sigma_k, \alpha_k, \bar{\ell}) = \bar{\sigma}_{k+1}$*. Then for every weak adversary* $\mathtt{a}$ *we have* $\mathtt{a}(\pi) = \mathtt{a}(\bar{\pi})$*.*

Proof. Fix an arbitrary weak adversary $\mathtt{a}$. By definition of weak adversaries, it is enough to show that $\pi \equiv_w \bar{\pi}$, and thus to prove that $\sigma_{k+1} \equiv_w \bar{\sigma}_{k+1}$.

We check all conditions for two configurations to be weakly-equivalent. Of course, $\sigma_{k+1}.\mathbf{p} = \bar{\sigma}_{k+1}.\mathbf{p}$. Let $\alpha_k = (M, p)$.

For every correct process $q \neq p$, $\sigma_{k+1}.\mathbf{s}(q) = \bar{\sigma}_{k+1}.\mathbf{s}(q)$. Writing $\sigma_{k+1}.\mathbf{s}(p) = (\ell, k)$ and $\bar{\sigma}_{k+1}.\mathbf{s}(p) = (\bar{\ell}, \bar{k})$, then the rounds k and $\bar{k}$ are trivially equal, and $\ell \equiv_{\mathcal{L}} \bar{\ell}$ holds by definition of the equivalence relation on locations.

Moreover, the update is defined by the rule r, and is independent of the destination location, so that the sent messages coincide: $\sigma_{k+1}.Sent = \sigma_k.Sent \cup r.\boldsymbol{u} = \bar{\sigma}_{k+1}.Sent$.

We now compare receive sets, and again they do not depend on the destination location, but only on r. For each correct process $q \neq p$ it trivially holds that $\sigma_{k+1}.Rcvd(q) = \bar{\sigma}_{k+1}.Rcvd(q)$. Also $\sigma_{k+1}.Rcvd(p) = \sigma_k.Rcvd(p) \cup M = \bar{\sigma}_{k+1}.Rcvd(p)$. □

We define two more notions for adversaries: An adversary $\mathtt{a}$ is *communication-closed* if for every finite path ρ the action $reveal[\mathtt{a}](\rho) = \alpha = (M, p)$ is such that each message $m \in M$ is sent before or in the same round in which process p is in $last(\rho)$. Formally, $m.rnd \leq rnd(\alpha)$. An adversary $\mathtt{a}$ is *round-rigid* if for every finite path ρ the action $reveal[\mathtt{a}](\rho) = \alpha = (M, p)$ has the smallest possible round, that is, there is no applicable action α' such that $rnd(\alpha') < rnd(\alpha)$. In the sequel, we show that weak round-rigid adversaries are as expressive as weak adversaries (see Theorem 1 and Theorem 2).

2.5 Atomic Propositions and Stutter Equivalence

Properties of threshold-based distributed algorithms are expressed in temporal logic. More precisely, we consider a stutter-insensitive fragment of LTL, namely, $\mathsf{LTL}_{-\mathsf{X}}$ [3, Chapter 7]. The atomic propositions describe the non-emptiness of a

location in a given round, *i.e.*, whether there is at least one correct process in location $\ell \in \mathcal{L} \setminus \mathcal{B}$ in round k [5]. The set of all such propositions for a round $k \in \mathbb{N}_0$ is denoted by $\mathrm{AP}_k = \{\mathrm{ap}(\ell, k) \colon \ell \in \mathcal{L} \setminus \mathcal{B}\}$. For every k we define a labeling function $\lambda_k \colon \Sigma \to 2^{\mathrm{AP}_k}$ such that $\mathrm{ap}(\ell, k) \in \lambda_k(\sigma)$ iff $\exists p \in \mathbb{C}.\ \sigma.\mathbf{s}(p) = (\ell, k)$.

For a path $\pi = \sigma_0, \alpha_1, \sigma_1, \ldots, \alpha_n, \sigma_n$, $n \in \mathbb{N}$, and a round k, a trace $\mathrm{trace}_k(\pi)$ w.r.t. the labeling function λ_k is the sequence $\lambda_k(\sigma_0)\lambda_k(\sigma_1)\ldots\lambda_k(\sigma_n)$. Similarly, if a path is infinite $\pi = \sigma_0, \alpha_1, \sigma_1, \alpha_2, \sigma_2, \ldots$, then $\mathrm{trace}_k(\pi) = \lambda_k(\sigma_0)\lambda_k(\sigma_1)\ldots$.

We say that two finite traces are stutter equivalent w.r.t. AP_k, denoted $\mathrm{trace}_k(\pi_1) \triangleq \mathrm{trace}_k(\pi_2)$, if there is a finite sequence $A_0 A_1 \ldots A_n \in (2^{\mathrm{AP}_k})^+$, $n \in \mathbb{N}_0$, such that both $\mathrm{trace}_k(\pi_1)$ and $\mathrm{trace}_k(\pi_2)$ are contained in the language given by the regular expression $A_0^+ A_1^+ \ldots A_n^+$. If traces of π_1 and π_2 are infinite, then stutter equivalence $\mathrm{trace}_k(\pi_1) \triangleq \mathrm{trace}_k(\pi_2)$ is defined in the standard way [3]. To simplify notation, we say that paths π_1 and π_2 are stutter equivalent w.r.t. AP_k, and write $\pi_1 \triangleq_k \pi_2$, instead of referring to specific path traces. Two stutter equivalent paths satisfy the same $\mathsf{LTL}_{\setminus \mathsf{X}}$ formulas [3, Theorem 7.92].

Remark. 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. This allows us to swap transitions of adjacent rounds below.

The following lemma expresses that an action may only change atomic propositions of its own round, as it only affects a process in that round.

Lemma 2. *Let $\pi = \sigma_0, \alpha_1, \sigma_1, \ldots, \sigma_{s-1}, \alpha_s, \sigma_s$ be a finite path. Then, for every round $k \neq rnd(\alpha_s)$, it holds that $\lambda_k(\sigma_{s-1}) = \lambda_k(\sigma_s)$.*

Proof. Let $\mathrm{ap}(\ell, k) \in \lambda_k(\sigma_{s-1})$, meaning that for some correct process $p \in \mathbb{C}$, $\sigma_{s-1}.\mathbf{s}(p) = (\ell, k)$. Since $rnd(\alpha_s) \neq k$, we have $\alpha_s = (M, q)$ for a set of messages M and some correct process $q \neq p$. Thus, application of α_s does not affect p, and $\sigma_s.\mathbf{s}(p) = (\ell, k)$. In other words, $\mathrm{ap}(\ell, k) \in \lambda_k(\sigma_s)$. This holds for every $\mathrm{ap}(\ell, k) \in \lambda_k(\sigma_{s-1})$, concluding the proof. □

Using Lemma 2, it is easy to prove that swapping two actions of different rounds in a path yield a stutter equivalent path w.r.t. AP_k for every $\in \mathbb{N}_0$:

Lemma 3. *Let $\pi = \sigma_0, \ldots, \sigma_s, \sigma_{s+1}, \sigma_{s+2}$ and $\pi' = \sigma_0, \ldots, \sigma_s, \sigma'_{s+1}, \sigma'_{s+2}$ be two paths with $\pi_s = \pi'_s = \sigma_0, \ldots, \sigma_s$, and such that there are two actions α and α' with $rnd(\alpha) \neq rnd(\alpha')$, and there are locations ℓ and ℓ' with*

$$\sigma_{s+1} = apply(\sigma_s, \alpha, \ell) \qquad \sigma_{s+2} = apply(\sigma_{s+1}, \alpha', \ell')$$
$$\sigma'_{s+1} = apply(\sigma_s, \alpha', \ell') \qquad \sigma'_{s+2} = apply(\sigma'_{s+1}, \alpha, \ell)$$

Then $\sigma_{s+2} = \sigma'_{s+2}$ and for every $k \in \mathbb{N}_0$, $\pi \triangleq_k \pi'$.

Proof. Let us first prove that $\sigma_{s+2} = \sigma'_{s+2}$. Both σ_{s+2} and σ'_{s+2} are obtained from σ_s by applying α with ℓ and α' with ℓ', just in different orders. By standard communication-closure arguments [10,8,9], if an action from a smaller round happens in an execution after an action of larger round, it is easy to prove that

these two actions do not affect each other. Thus, in any order they will lead to the same configuration.

Let us now fix an arbitrary $k \in \mathbb{N}_0$ and prove that $\pi \triangleq_k \pi'$. As these two paths have the same prefix of length s, it suffices to prove that $\sigma_s, \sigma_{s+1}, \sigma_{s+2} \triangleq_k \sigma_s, \sigma'_{s+1}, \sigma_{s+2}$. We distinguish three cases: (i) $k \neq rnd(\alpha)$ and $k \neq rnd(\alpha')$, (ii) $k = rnd(\alpha)$, and (iii) $k = rnd(\alpha')$. Cases (ii) and (iii) are symmetrical, so that we only prove (ii).

(i) By Lemma 2 we have $\lambda_k(\sigma_s) = \lambda_k(\sigma_{s+1}) = \lambda_k(\sigma'_{s+1}) = \lambda_k(\sigma_{s+2})$, which trivially yields the required statement.

(ii) As in this case $k \neq rnd(\alpha')$, we can apply Lemma 2 and obtain that $\lambda_k(\sigma_{s+1}) = \lambda_k(\sigma_{s+2})$ and $\lambda_k(\sigma_s) = \lambda_k(\sigma'_{s+1})$. Therefore, $\mathrm{trace}_k(\sigma_s, \sigma_{s+1}, \sigma_{s+2}) = \lambda_k(\sigma_s)\lambda_k(\sigma_{s+2})\lambda_k(\sigma_{s+2})$, and $\mathrm{trace}_k(\sigma_s, \sigma'_{s+1}, \sigma_{s+2}) = \lambda_k(\sigma_s)\lambda_k(\sigma_s)\lambda_k(\sigma_{s+2})$, and they are clearly stutter equivalent w.r.t. k. □

3 Reduction to Communication-closed Adversaries

In this section we show that in the threshold automata framework (which models communication-closed algorithms by construction, cf. Section 2.2), for every adversary, there exists an "equivalent" communication-closed adversary. This step is quite intuitive: if the adversary delivers a message m of round r to a process in round $r' \neq r$, this is "similar" to an adversary that instead does not deliver m now, but delivers m later when the process enters r. However, we need to formalize this, in order to set the stage of our central reduction in Section 4.

For a set of messages M and a round k we denote by $M|_k$ the set of messages from M sent in round k, that is, $M|_k = \{m \in M \mid m.rnd = k\}$. Similarly, we define $M|_{\leq k} = \cup_{i \leq k} M|_i$ the set of messages from M sent in any round $i \leq k$.

Communication-closed configurations and Markov chains. The definition of rules *executable* in σ in Section 2.2 yields that messages received from the "future", *i.e.,* from a round $k > \sigma.\mathbf{s}_{\mathrm{rnd}}(p)$, do not play a role for process p in σ. Namely, if two configurations σ and σ' differ only in the messages that processes have received from future rounds, then the same rules are executable in σ and σ'. That is why for every σ we define $\widetilde{\sigma}$ to be a configuration in which each process only has received messages from "past" and "present", i.e., from a round $k \leq \sigma.\mathbf{s}_{\mathrm{rnd}}(p)$. As border locations can be seen as borders between consecutive rounds, we decide to let processes from border locations in $\widetilde{\sigma}$ receive messages only from the "past", as if they were not yet in the next round. Formally, $\widetilde{\sigma}.\mathbf{s} = \sigma.\mathbf{s}$ and $\widetilde{\sigma}.Sent = \sigma.Sent$ and $\widetilde{\sigma}.\mathbf{p} = \sigma.\mathbf{p}$ and for all $p \in \mathbb{C}$, if $\mathbf{s}_{\mathrm{loc}}(p) \in \mathcal{B}$ then $\widetilde{\sigma}.Rcvd(p) = \sigma.Rcvd(p)|_{\leq \mathbf{s}_{\mathrm{rnd}}(p)-1}$ otherwise $\widetilde{\sigma}.Rcvd(p) = \sigma.Rcvd(p)|_{\leq \mathbf{s}_{\mathrm{rnd}}(p)}$. A configuration σ is communication-closed if $\sigma = \widetilde{\sigma}$.

Recall that a path has the form $\sigma_0, \alpha_0, \ldots, \alpha_{s-1}, \sigma_s$, but given a sequence of configurations $\sigma_0, \ldots, \sigma_s$ of a path generated by an adversary $\mathtt{a}$, we can easily recover the missing actions. This allows us to consider the states of Markov chains to be finite sequences of configurations rather than finite paths. Both representations are equivalent, and thus in this section we consider paths as sequences

of configurations. We can lift the notion of communication-closed configurations to communication-closed paths, such that given a path $\rho = \sigma_0, \ldots, \sigma_k$ we define $\mu(\rho) = \widetilde{\sigma}_0, \ldots, \widetilde{\sigma}_k$. Finally, we obtain a communication-closed Markov chain $\widetilde{\mathcal{M}}_{\mathtt{a}}^{\sigma_0}$ by replacing each state ρ in $\mathcal{M}_{\mathtt{a}}^{\sigma_0}$ by $\mu(\rho)$.

Communication-closed adversaries. Given an arbitrary adversary $\mathtt{a}$, we define its corresponding communication-closed adversary $\mathtt{cc}(\mathtt{a})$ as follows. If a process p is scheduled by $\mathtt{a}$ it is also scheduled by $\mathtt{cc}(\mathtt{a})$, but it receives different messages: if p is not at the beginning of a round, $\mathtt{cc}(\mathtt{a})$ should check which messages $\mathtt{a}$ would give to p, and among them $\mathtt{cc}(\mathtt{a})$ should choose only those messages that do not come from future rounds. Once a process reaches a round (when it is at a border location in $\mathcal{B}$), it receives all the messages from that round and all those that were previously sent to it, but the process could not receive it earlier (because at that time these were messages from the future.)

As $\mathtt{cc}(\mathtt{a})$ has to know the behavior of $\mathtt{a}$, we show how to recover a path generated by $\mathtt{a}$ (if it exists) if we are given a communication-closed path. Formally, given an adversary $\mathtt{a}$ and a communication-closed path ρ, we define $\nu_{\mathtt{a}}(\rho)$ to be a path τ generated by $\mathtt{a}$, such that $\mu(\tau) = \rho$, if such a path exists; otherwise, $\nu_{\mathtt{a}}(\rho)$ is undefined. Observe that if ρ is a path generated by $\mathtt{a}$, then $\rho = \nu_{\mathtt{a}}(\mu(\rho))$.

Finally, we define a communication-closed version of an adversary formally. Recall that τ_i denotes the prefix of τ of length i.

Definition 2. *Let* $\mathtt{a}$ *be an adversary. For a given finite path* ρ*, if* $\nu_{\mathtt{a}}(\mu(\rho))$ *is undefined then* $\mathtt{cc}(\mathtt{a})(\rho)$ *is an arbitrary action. Otherwise, if* $\nu_{\mathtt{a}}(\mu(\rho)) = \tau = \sigma_0, \ldots, \sigma_s$*, let* $\mathit{reveal}[\mathtt{a}](\tau_i) = \alpha_i = (M_i, p_{j_i})$*, for each* $0 \leq i \leq s$ *and some* $p_{j_i} \in \mathbb{C}$*. If* $\alpha_s = (M_s, p)$*, in order to define* $\mathtt{cc}(\mathtt{a})(\rho)$ *we distinguish two cases:*

- *If* $\sigma_s.\mathbf{s}[p] = (\ell, k)$ *with* $\ell \notin \mathcal{B}$*, then*

$$\mathtt{cc}(\mathtt{a})(\rho) = \mathit{choice}[\tau](M_s|_{\leq k}, p).$$

- *If* $\sigma_s.\mathbf{s}[p] = (\ell, k)$ *with* $\ell \in \mathcal{B}$*, then for* $S_p = \{i \mid 1 \leq i < s \wedge \alpha_i = (M_i, p)\}$ *being the set of indices of the actions involving process* p*:*

$$\mathtt{cc}(\mathtt{a})(\rho) = \mathit{choice}[\tau](M, p), \text{ for } M = M_s|_{\leq k} \cup \bigcup_{i \in S_p} M_i|_k.$$

Theorem 1. *For every adversary* $\mathtt{a}$*,* $\widetilde{\mathcal{M}}_{\mathtt{a}}^{\sigma_0} = \widetilde{\mathcal{M}}_{\mathtt{cc}(\mathtt{a})}^{\sigma_0}$.
Moreover, for every $\mathsf{LTL}_{-\mathsf{X}}$ *formula* ψ*,* $\mathbf{P}_{\mathtt{a}}^{\sigma_0}(\psi) = \mathbf{P}_{\mathtt{cc}(\mathtt{a})}^{\sigma_0}(\psi)$.

4 From Weak to Round-Rigid Adversaries

In this section we reduce a communication-closed weak adversary to a round-rigid adversary. More precisely, we show that we can transform the Markov chain defined by the weak adversary to a round-rigid Markov chain that satisfies specific temporal logic formulas with the same probabilities.

Swapping function for paths and swapped adversaries We first define a swapping function for a path $\rho = \sigma_0, \alpha_0, \sigma_1, \ldots, \sigma_s, \alpha_s, \sigma_{s+1}, \alpha_{s+1}, \sigma_{s+2} \ldots$ and $s \in \mathbb{N}$ a *swapping index* for ρ, such that $rnd(\alpha_s) > rnd(\alpha_{s+1})$. It applies to a path $\bar{\rho} = \bar{\sigma_0}, \ldots, \bar{\sigma_s}, \bar{\alpha}_s, \bar{\sigma}_{s+1}, \bar{\alpha}_{s+1}, \bar{\sigma}_{s+2}, \ldots$ such that $\bar{\rho}_s \equiv_w \rho_s$, and swaps its actions (and target locations) at steps s and $s{+}1$. Formally, if $\bar{\ell_s}$, $\bar{\ell}_{s+1}$ are the destination locations at step s and $s{+}1$, *i.e.* $apply(\bar{\sigma}_s, \bar{\alpha}_s, \bar{\ell_s}) = \bar{\sigma}_{s+1}$ and $apply(\bar{\sigma}_{s+1}, \bar{\alpha}_{s+1}, \bar{\ell}_{s+1}) = \bar{\sigma}_{s+2}$, then for $\sigma'_{s+1} = apply(\bar{\sigma}_s, \bar{\alpha}_{s+1}, \bar{\ell}_{s+1})$ we define

$$\mathsf{sw}[\rho, s](\bar{\rho}) = \begin{cases} \bar{\sigma_0}, \ldots, \bar{\sigma_s}, \bar{\alpha}_{s+1}, {\sigma'}_{s+1}, \bar{\alpha}_s, \bar{\sigma}_{s+2}, \ldots & \text{if } \bar{\rho}_s \equiv_w \rho_s \\ \bar{\rho} & \text{otherwise.} \end{cases}$$

Note that the configurations in $\bar{\rho}$ and $\mathsf{sw}[\rho, s](\bar{\rho})$ may only differ at position $s{+}1$, and that $\bar{\rho}$ and $\mathsf{sw}[\rho, s](\bar{\rho})$ are stutter-equivalent. Hence, for every $\mathsf{LTL}_{\setminus \mathsf{X}}$ formula ψ, $\bar{\rho} \models \psi$ iff $\mathsf{sw}[\rho, s](\bar{\rho}) \models \psi$. If $\rho' = \mathsf{sw}[\rho, s](\bar{\rho})$, we also write $\mathsf{sw}[\rho, s]^{-1}(\rho') = \bar{\rho}$.

Now, given a weak communication-closed adversary a, a path ρ, and a swapping index s for ρ, we define the *swapped adversary* $\mathsf{a}' = \mathsf{swap}[\rho, s](\mathsf{a})$ that, intuitively, will implement the swapping function $\mathsf{sw}[\rho, s]$ over a-induced paths. Formally, for any finite path ρ', the definition distinguishes whether $\rho'_s \equiv \rho_s$, and depends on the length of ρ':

(i) if $|\rho'| < s$, then $\mathsf{a}'(\rho') = \mathsf{a}(\rho')$;
(ii) if $\rho'_s \not\equiv_w \rho_s$ and $|\rho'| \geq s$, then $\mathsf{a}'(\rho') = \mathsf{a}(\rho')$;
(iii) if $\rho'_s \equiv_w \rho_s$ and $|\rho'| = s$, then

$$\mathsf{a}'(\rho') = choice[\rho']\big(reveal[\mathsf{a}](\rho_{s+1})\big);$$

(iv) if $\rho'_s \equiv_w \rho_s$ and $|\rho'| = s{+}1$, then

$$\mathsf{a}'(\rho') = choice[\rho']\big(reveal[\mathsf{a}](\rho_s)\big);$$

(v) if $\rho'_s \equiv_w \rho_s$ and $|\rho'| \geq s{+}2$, then

$$\mathsf{a}'(\rho') = choice[\rho']\big(reveal[\mathsf{a}](\mathsf{sw}[\rho, s]^{-1}(\rho'))\big).$$

Let us give some intuition on the definition of the swapped adversary $\mathsf{swap}[\rho, s](\mathsf{a})$. Cases (i) and (ii) concern paths that are not involved in the swapping: either they are shorter than the position s at which the swap occurs, or their prefix of length s is not weakly-equivalent to the one of ρ. In these easy cases, $\mathsf{a}' = \mathsf{swap}[\rho, s](\mathsf{a})$ is defined as a. Case (iii) applies to all paths of length s that are weakly-equivalent to ρ_s, and the goal is to define $\mathsf{swap}[\rho, s](\mathsf{a})$ so that it selects action α_{s+1}. However, under a and $\mathsf{swap}[\rho, s](\mathsf{a})$, the identities of messages may be different along paths (and their extensions) that are equivalent to ρ_s. We thus need to use the $reveal[\mathsf{a}]$ and $choice[\_]$ functions to define that the action prescribed by $\mathsf{swap}[\rho, s](\mathsf{a})$ is α_{s+1}. Case (iv) applies to paths of length $s{+}1$, whose prefix of length s is weakly-equivalent to ρ_s. For them, the decision $\mathsf{swap}[\rho, s](\mathsf{a})$ should results in action α_s. Finally, (v) deals with longer paths, for which a and $\mathsf{swap}[\rho, s](\mathsf{a})$ take the same decisions, up to the renaming of message identities, and the earlier swapping of α_s and α_{s+1}.

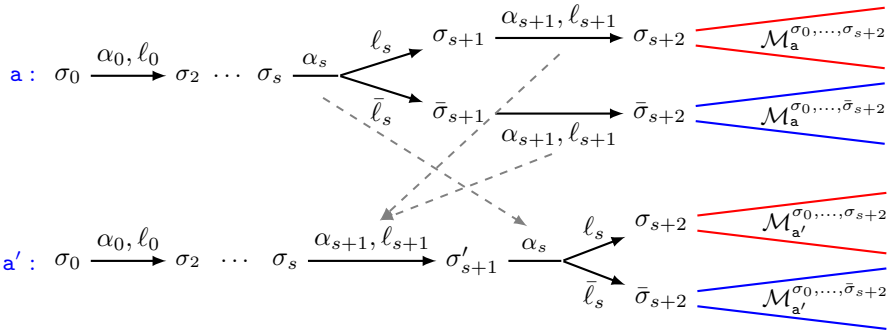

Fig. 3. Parts of Markov chains $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ (above) and $\mathcal{M}_{\mathsf{a}'}^{\sigma_0}$ (below) following Proposition 1. We assume here α_s is a non-Dirac action with two destination locations ℓ_s and $\bar{\ell}_s$, and α_{s+1} is a Dirac action with destination location ℓ_{s+1}. If some of actions $\alpha_0, \ldots, \alpha_{s-1}$ are non-Dirac, we omit drawing branches that are irrelevant for Proposition 1. Note that $\mathit{reveal}[\mathsf{a}](\sigma_0, \ldots, \sigma_{s+1})$ and $\mathit{reveal}[\mathsf{a}](\sigma_0, \ldots, \bar{\sigma}_{s+1})$ must be the same action α_{s+1}, as a is a weak adversary. This is the key insight that allows swapping, which would not be possible for a strong adversary with $\mathit{reveal}[\mathsf{a}](\sigma_0, \ldots, \sigma_{s+1}) \neq \mathit{reveal}[\mathsf{a}](\sigma_0, \ldots, \bar{\sigma}_{s+1})$.

Excerpts of the Markov chains $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ and $\mathcal{M}_{\mathsf{a}'}^{\sigma_0}$ in Figure 3 illustrate the transformation to the swapped adversary. Observe that swapping α_s and α_{s+1} relies on the fact that a is weak. Indeed, the same action α_{s+1} applies after α_s even if α_s induces a non-Dirac distribution. Observe that $\mathsf{swap}[\rho, s](\mathsf{a})$ is still a weak adversary, since it is defined uniformly over weakly-equivalent paths.

Adversaries a and $\mathsf{swap}[\rho, s](\mathsf{a})$ are tightly related. First, the successor configurations in two steps after a path $\rho' \equiv_w \rho_s$ are the same, and they have the same probabilities to happen. Second, the Markov chains from these points are identical. Finally the probabilities of $\mathsf{LTL}_{\text{-}\mathsf{X}}$ formulas are preserved.

Proposition 1. *Let* a *be a weak communication-closed adversary,* ρ *an* a*-induced path and* s *a swapping index. Then, the swapped adversary* $\mathsf{a}' = \mathsf{swap}[\rho, s](\mathsf{a})$ *is again a weak communication-closed adversary, and it satisfies:*

1. *for every* $\rho' \equiv_w \rho_s$ *and every* $\sigma \in \Sigma$*, we have*
$$\mathbf{P}_{\mathsf{a}}^{\rho'}(\mathsf{F}^{=2}\sigma) = \mathbf{P}_{\mathsf{a}'}^{\rho'}(\mathsf{F}^{=2}\sigma)$$
2. *for every path* $\bar{\rho}$ *with* $|\bar{\rho}| = s{+}2$*, we have*
$$\mathcal{M}_{\mathsf{a}}^{\bar{\rho}} = \mathcal{M}_{\mathsf{a}'}^{\mathsf{sw}[\rho,s](\bar{\rho})}$$
3. *for every* $\mathsf{LTL}_{\text{-}\mathsf{X}}$ *formula* ψ*, we have*
$$\mathbf{P}_{\mathsf{a}}^{\sigma_0}(\psi) = \mathbf{P}_{\mathsf{a}'}^{\sigma_0}(\psi)$$

Proof. Remark that indeed $\mathsf{swap}[\rho, s](\mathsf{a})$ is weak, because it is defined uniformely for weakly-equivalent paths. It is also communication-closed, as a is.

Let us now prove the three statements.

1. Let $\rho' \equiv_w \rho_s$, and let σ be a configuration with $\mathbf{P}_{\mathsf{a}}^{\rho'}(\mathbf{F}^{=2}\sigma) > 0$. Then there exists an a-induced path $\rho'\alpha_s\sigma_{s+1}\alpha_{s+1}\sigma$ and locations ℓ_s, ℓ_{s+1} such that

$$\begin{aligned}\sigma_{s+1} &= \mathit{apply}(\sigma_s, \alpha_s, \ell_s)\\ \sigma &= \mathit{apply}(\sigma_{s+1}, \alpha_{s+1}, \ell_{s+1})\end{aligned}$$

   Let $\bar{\rho} = \mathsf{sw}[\rho, s](\rho'\alpha_s\sigma_{s+1}\alpha_{s+1}\sigma)$. By definition of $\mathsf{swap}[\rho, s](\mathsf{a})$, $\bar{\rho}$ is a path induced by $\mathsf{swap}[\rho, s](\mathsf{a})$, and it ends in σ. More precisely, letting

$$\begin{aligned}\alpha'_{s+1} &= \mathit{choice}[\rho']\big(\mathit{reveal}[\mathsf{a}](\rho')\big)\\ \sigma'_{s+1} &= \mathit{apply}(\sigma_s, \alpha'_{s+1}, \ell_{s+1})\\ \alpha'_s &= \mathit{choice}[\rho'\alpha'_{s+1}\sigma'_{s+1}]\big(\mathit{reveal}[\mathsf{a}'](\rho'\alpha_s\sigma_{s+1})\big)\\ \sigma'_{s+2} &= \mathit{apply}(\sigma'_{s+1}, \alpha'_s, \ell_s)\end{aligned}$$

   then $\sigma'_{s+2} = \sigma$ and $\bar{\rho} = \rho'\alpha'_{s+1}\sigma'_{s+1}\alpha'_s\sigma$. Thus $\mathbf{P}^{\rho'}_{\mathsf{swap}[\rho,s](\mathsf{a})}(\mathbf{F}^{=2}\sigma) > 0$. Moreover, by commutativity of multiplication, the probabilities of reaching σ in two steps in $\mathcal{M}_{\mathsf{a}}^{\rho'}$ and $\mathcal{M}^{\rho'}_{\mathsf{swap}[\rho,s](\mathsf{a})}$ coincide: they are equal to $\alpha_s.\delta_{to}(\ell_s) \times \alpha_{s+1}.\delta_{to}(\ell_{s+1})$.
2. To prove that the Markov chains after $\bar{\rho}$ of length $s{+}2$ under a, and the one after $\mathsf{sw}[\rho, s](\bar{\rho})$ under $\mathsf{swap}[\rho, s](\mathsf{a})$ are equal, we observe that, for paths longer than $s{+}2$,

$$\mathsf{a}'(\rho') = \mathit{choice}[\rho']\big(\mathit{reveal}[\mathsf{a}](\mathsf{sw}[\rho, s]^{-1}(\rho'))\big) \ .$$

   This is (v) in the definition of $\mathsf{swap}[\rho, s](\mathsf{a})$, and also applies if $\rho' \not\equiv_w \rho_s$, in which case $\mathsf{sw}[\rho, s]^{-1}(\rho') = \rho'$. In words, $\mathsf{swap}[\rho, s](\mathsf{a})$ consists in applying a on the reverse swapped path. Therefore, the subsequent Markov chains are equal, as illustrated on Figure 3.
3. Finally, to prove that the probabilities of $\mathsf{LTL}_{\text{-}\mathsf{X}}$ formulas are preserved, we argue that $\mathcal{M}_{\mathsf{a}}^{\sigma_0}$ and $\mathcal{M}^{\sigma_0}_{\mathsf{swap}[\rho,s](\mathsf{a})}$ are essentially the same, up to the swapping of some paths at positions s and $s{+}1$. Remember that they both are tree-shaped. First, they are equal up to depth s. Then item (i) shows that the successors from depth s in two steps are the same, and they happen with same probabilities. Last, item (ii) shows that the subsequent Markov chains are identical. To conclude, we use Lemma 3 to justify that even if actions are swapped at positions s and $s{+}1$, they satisfy the same $\mathsf{LTL}_{\text{-}\mathsf{X}}$ formulas.

□

Theorem 2. *For every weak communication-closed adversary* a *under which every round terminates, there exists a weak round-rigid adversary* a' *such that for every* $\mathsf{LTL}_{\text{-}\mathsf{X}}$ *formula* ψ *we have* $\mathbf{P}_{\mathsf{a}}^{\sigma_0}(\psi) = \mathbf{P}_{\mathsf{a}'}^{\sigma_0}(\psi)$.

Proof (sketch). Theorem 2 is obtained by applying iteratively Proposition 1 to consecutive actions that are in reverse order. Since every round is assumed to terminate under a, one can start by moving towards the beginning all actions of round 1 that happen after actions of later rounds; then one swaps all actions of round 2, and so on, to obtain in the limit a weak adversary which is round-rigid.

5 Conclusions

Parameterized verification of safety and almost sure termination of a class of distributed consensus algorithms [4,7,16,17] has been recently considered in [5]. For almost sure termination, the authors limited themselves to so-called "round-rigid" adversaries, which were introduced by the authors for that purpose. Verification under these adversaries was reduced to verification of specifications in a linear temporal logic that can be checked (within a few minutes) with the ByMC model checker [13].

In this paper, we have shown that automated verification under weak adversaries can be reduced to verification of round-rigid adversaries. More precisely, in order to verify randomized distributed algorithms under weak adversaries, one only needs to verify their behavior under round-rigid adversaries, which has been done in [5] for various randomized consensus algorithms. In order to define weak adversaries, we were forced to reason within a system model with semantics that explicitly talks about processes with IDs and messages. In contrast, the standard semantics of threshold automata, namely, counter systems is used in [5] and in ByMC. For a complete chain of proof we would need to connect process-based semantics to counter systems. This is a rather standard technical argument so that we do not give it here. From a theoretical viewpoint we find our reduction from weak adversaries to round-rigid adversaries more interesting: reductions for concurrent distributed systems is typically done for reachability properties [15,10,11] or linear temporal properties [8,12,9]. As a result, the reduction argument is conducted on traces generated by the system: one shows that by swapping transitions in a trace we arrive at another, yet "simpler" trace of the system. In this paper, we lifted this reasoning from traces to computation trees and MDPs which shows that reductions are not only efficient in non-deterministic systems but also in probabilistic systems defined by distributed algorithms. This mirrors the recent "synchronizing" trend in the verification of non-deterministic fault-tolerant distributed algorithms [18,14,6,9,5], and opens this domain to automated parameterized verification of *randomized* distributed algorithms.

References

1. Marcos Aguilera and Sam Toueg. The correctness proof of Ben-Or's randomized consensus algorithm. *Distributed Computing*, 25(5):371–381, 2012.
2. James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.
3. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
4. Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.
5. Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. In *CONCUR*, volume 140 of *LIPIcs*, pages 33:1–33:15, 2019.

6. Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In *CAV*, pages 372–391, 2018.
7. Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.
8. 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.
9. Andrei Damian, Cezara Drăgoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *CAV*, volume 11562 of *LNCS*, pages 344–363. Springer, 2019.
10. Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2(3):155–173, 1982.
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, 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.
13. Igor Konnov and Josef Widder. ByMC: Byzantine model checker. In *ISoLA*, volume 11246 of *LNCS*, pages 327–342. Springer, 2018.
14. Bernhard Kragl, Shaz Qadeer, and Thomas A. Henzinger. Synchronizing the asynchronous. In *CONCUR*, volume 118 of *LIPIcs*, pages 21:1–21:17, 2018.
15. Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, 1975.
16. Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Randomized k-set agreement in crash-prone and Byzantine asynchronous systems. *Theoretical Computer Science*, 709:80–97, 2018.
17. Yee Jiun Song and Robbert van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *DISC*, volume 5218 of *LNCS*, pages 438–450, 2008.
18. Klaus v. Gleissenthall, Rami Gökhan Kici, Alexander Bakst, Deian Stefan, and Ranjit Jhala. Pretend synchrony. In *POPL*, pages 59:1–59:30, 2019.