Fair Termination for Parameterized Probabilistic Concurrent Systems (Technical Report)

Ondřej Lengál[1], Anthony W. Lin[2], Rupak Majumdar[3], and Philipp Rümmer[4]

[1] FIT, Brno University of Technology, Czech Republic
[2] Department of Computer Science, University of Oxford, UK
[3] MPI-SWS Kaiserslautern, Germany
[4] Uppsala University, Sweden

Abstract. We consider the problem of automatically verifying that a parameterized family of probabilistic concurrent systems terminates with probability one for all instances against adversarial schedulers. A parameterized family defines an infinite-state system: for each number n, the family consists of an instance with n finite-state processes. In contrast to safety, the parameterized verification of liveness is currently still considered extremely challenging especially in the presence of probabilities in the model. One major challenge is to provide a sufficiently powerful symbolic framework. One well-known symbolic framework for the parameterized verification of non-probabilistic concurrent systems is *regular model checking*. Although the framework was recently extended to probabilistic systems, incorporating fairness in the framework — often crucial for verifying termination — has been especially difficult due to the presence of an infinite number of fairness constraints (one for each process). Our main contribution is a systematic, regularity-preserving, encoding of *finitary fairness* (a realistic notion of fairness proposed by Alur & Henzinger) in the framework of regular model checking for probabilistic parameterized systems. Our encoding reduces termination with finitary fairness to verifying parameterized termination *without fairness* over probabilistic systems in regular model checking (for which a verification framework already exists). We show that our algorithm could verify termination for many interesting examples from distributed algorithms (Herman's protocol) and evolutionary biology (Moran process, cell cycle switch), which do not hold under the standard notion of fairness. To the best of our knowledge, our algorithm is the first fully-automatic method that can prove termination for these examples.

1 Introduction

In parameterized probabilistic concurrent systems, a population of *agents*, each typically modeled as a finite-state probabilistic program, run concurrently in discrete time and update their states based on probabilistic transition rules. The interaction is governed by an underlying *topology*, which determines which agents can interact in one step, and a *scheduler*, which picks the specific agents involved in the interaction. Concurrent probabilistic systems arise as models of distributed algorithms [1,2,3,4,5], where each agent is a processor, the interaction between processors is determined by a communication topology, and the processor can update its internal state based on the communication as well as randomization. In each step, the scheduler adversarially chooses a processor to run. Concurrent probabilistic populations also arise in agent-based population models in biology [6], wherein an agent can represent an allele, a cell, or a species, and the interaction between agents describes how these entities evolve over time. For a population of a fixed size, there is a rich theory of probabilistic verification [7,8,9,10] based on finite-state Markov decision processes (MDPs). Verification questions for population models, however, ask if a property holds for populations of *all* sizes: even if each agent is finite-state, the family of all processes (for each population size) is an infinite-state MDP. Indeed, for many simple population models, one can show that the verification question is undecidable, even for reachability or safety properties in the non-probabilistic setting [11,12,13]. Consequently, the verification question for populations requires techniques beyond finite-state probabilistic verification, and requires symbolic techniques to represent potentially infinite sets of states.

One well-known symbolic framework for verifying parameterized non-probabilistic concurrent systems is *regular model checking* [14,15,16,17,18,19], where states of a population are modeled using words over a suitable alphabet, sets of states are represented as regular languages, and the transition relation is defined as a regular transducer. From parameterized verification of non-probabilistic processes, it is known that regular languages provide a robust symbolic representation of infinite sets, and automata-theoretic algorithms provide the basis of checking safety or termination properties.

In this paper, we consider the problem of verifying that a given parameterized family of probabilistic concurrent systems *almost surely terminates*, i.e., reaches certain final states with probability 1 from each initial state regardless of the behaviour of the schedulers. Termination is a fundamental property when verifying parameterized probabilistic systems. Since termination typically, however, fails without imposing certain *fairness* conditions on the scheduler, it is crucial to be able to incorporate fairness assumptions into a termination analysis. Therefore, although the framework of regular model checking has recently been extended for proving termination (without fairness) over parameterized probabilistic concurrent systems [20], it still cannot be used to prove termination for many interesting parameterized probabilistic concurrent systems.

What notion of fairness should we consider for proving termination for parameterized probabilistic concurrent systems? To answer this question, one would naturally start by looking at standard notions of fairness in probabilistic model checking [8], which asserts that every process must be chosen infinitely often. However, this notion seems to be too weak to prove termination for many of our examples, notably Herman's self-stabilizing protocol [2] in an asynchronous setting, and population models from biology (e.g. Moran's process [6]). The standard notion of fairness gives rise to a rather unintuitive and unrealistic strategy for the scheduler, which could delay an enabled process for as long as it desires while still being fair (see [21, Example 8] and the Herman's protocol example in Section 3). For this reason, we propose to consider Alur & Henzinger's [22] *finitary fairness* — a stronger notion of fairness that allows the scheduler to delaying executing an enabled process in an infinite run for at most k steps, for some unknown but fixed bound $k \in \mathbb{N}$. Alur & Henzinger argued that this fairness notion is more realistic in practice, but it is not as restrictive as the notion of *k-fairness*, which fixes the bound k a priori. In addition, it should be noted that finitary fairness is strictly weaker than probabilistic fairness (scheduler chooses processes randomly) for almost-sure termination over finite MDPs and parameterized probabilistic systems (an infinite family of finite MDPs). We will show in this paper that there are many interesting examples of parameterized probabilistic concurrent systems for which termination is satisfied under finitary fairness, but *not* under the most general notion of fairness.

Contributions. Our main contribution is a systematic, regularity-preserving, encoding of finitary fairness in the framework of regular model checking for parameterized probabilistic concurrent systems. More precisely, our encoding reduces the problem of verifying almost sure termination under finitary fairness to almost sure termination *without fairness* in regular model checking, for which a verification framework exists [20].

In general, the difficulty with finding an encoding of fairness is how to deal with an infinite number of fairness requirements (one for each process) in a systematic and regularity-preserving manner. There are known encodings of general notions of fairness in regular model checking, e.g., by using a token that is passed to the next process (with respect to some ordering of the processes) when the current process is executed, and ensuring that the first process holds the token and passes it to the right infinitely many times (e.g. see [16,15]). However, these encodings do not work in in our case for several reasons. Firstly, they do not take into account the unknown upper bound (from finitary fairness) within which time a process has to be executed. Adapting these encodings to finitary fairness would require *the use of unbounded counters*, which do not preserve regularity. Secondly, such encodings would yield the problem of verifying an almost-sure Rabin property (of the form $\Box\Diamond A \wedge \Diamond B$ in LTL notation, where A and B are regular sets). Although we could reduce this to an almost-sure termination property by means of product automata construction (i.e. by first converting the formula to deterministic Rabin automaton), the target set B in the resulting termination property $\Diamond B$ (consisting of configurations in strongly connected components satisfying some properties) is *not* necessarily regular.

Instead, we revisit the well-known *abstract program transformation* in the setting of non-probabilistic concurrent systems [23] encoding fairness into the program by associating to each process an unbounded counter that acts as an "alarm clock", which will "set off" if an enabled process has not been chosen by the scheduler for "too long." This abstract program transformation has been adapted by Alur & Henzinger [22] in the case of finitary fairness by additionally incorporating an extra counter n that stores the unknown upper bound and resetting the value of a counter belonging to a chosen process to the "default value" n. Our contributions are as follows:

1. We show how Alur & Henzinger's program transformation could be adapted to the setting of probabilistic parameterized concurrent systems (infinite family of finite MDPs). This involves constructing a new parameterization of the system (using the idea of weakly finite systems) and a proof that the transformation preserves reachability probabilities.
2. We show how the resulting abstract program transformation could be made concrete in the setting of regular model checking *without using automata models beyond finite automata*.
3. We have implemented this transformation in FAIRYTAIL. Combined with the existing algorithm [20] for verifying almost sure termination (without fairness) in regular model checking, we have successfully verified a number of models obtained from distributed algorithms and biological systems including Herman's pro-

tocol [2], Moran processes in a linear array [24,6], and the cell cycle switch model [25] on ring and line topologies. To the best of our knowledge, our algorithm is the first fully-automatic method that can prove termination for these examples.

Related work. There are few techniques for automatic verification of liveness properties of parameterized probabilistic programs. Almost sure verification of probabilistic finite-state programs goes back to Pnueli and co-workers [26,27]. Esparza et al. [28] generalize the reasoning to weakly finite programs, and describe a heuristic to guess a *terminating pattern* by constructing a nondeterministic program from a given probabilistic program and a terminating pattern candidate. This allows them to exploit model checkers and termination provers for nondeterministic programs. More recently, Lin and Rümmer [20] consider unconditional termination for parameterized probabilistic programs. While our work builds on these techniques, our main contribution is the incorporation of fairness in regular model checking of probabilistic programs, which was not considered before.

Fairness for concurrent probabilistic systems was considered by Vardi [10] and by Hart, Sharir, and Pnueli [26], and generalized later [27,29,30]. The focus was, however, on a fixed number of processes. The notion of fairness through explicit scheduling was developed by Olderog and Apt [31]. More recently, notions of fairness for infinitary control (i.e., where an infinite number of processes can be created) was considered by Hoenicke, Olderog, and Podelski [32,33].

Martingale techniques have been used to prove termination of sequential, infinite-state, probabilistic programs [34,35,36,37,38]. These results are not comparable to our results, as they do not consider unbounded families of fairness constraints nor communication topologies.

2 Preliminaries

General notations: For any two given real numbers $i \leq j$, we use a standard notation (with an extra subscript) to denote real intervals, e.g., $[i,j]_{\mathbb{R}} = \{k \in \mathbb{R} : i \leq k \leq j\}$ and $(i,j]_{\mathbb{R}} = \{k \in \mathbb{R} : i < k \leq j\}$. We will denote intervals over integers by removing the subscript, i.e., $[i,j] = [i,j]_{\mathbb{R}} \cap \mathbb{Z}$. Given a set S, we use S^* to denote the set of all finite sequences of elements from S. The set S^* always includes the empty sequence, which we denote by ϵ. We use S^+ to denote the set $S^* \backslash \{\epsilon\}$. Given two sets of words S_1, S_2, we use $S_1 \cdot S_2$ to denote the set $\{v \cdot w : v \in S_1, w \in S_2\}$ of words formed by concatenating words from S_1 with words from S_2. Given two relations $R_1, R_2 \subseteq S \times S$, we define their composition as $R_1 \circ R_2 = \{(s_1, s_3) : \exists s_2((s_1, s_2) \in R_1 \wedge (s_2, s_3) \in R_2)\}$.

Transition systems: We fix the (countably infinite) set AP of *atomic propositions*. Let ACT be a finite set of *action symbols*. A *transition system* over ACT is a tuple $\mathfrak{S} = \langle S; \{\rightarrow_a\}_{a \in \mathsf{ACT}}, \ell \rangle$, where S is a set of *configurations*, $\rightarrow_a \subseteq S \times S$ is a binary relation over S, and $\ell : \mathsf{AP} \rightarrow 2^S$ maps atomic propositions to sets of configurations (we omit ℓ if it is not important). We use $\rightarrow$ to denote the relation $\left(\bigcup_{a \in \mathsf{ACT}} \rightarrow_a\right)$. The notation $\rightarrow^+$ (resp. $\rightarrow^*$) is used to denote the transitive (resp. transitive-reflexive) closure of $\rightarrow$. We say that a sequence $s_1 \rightarrow \cdots \rightarrow s_n$ is a *path* (or *run*) in $\mathfrak{S}$ (or in $\rightarrow$). Given two paths $\pi_1 : s_1 \rightarrow^* s_2$ and $\pi_2 : s_2 \rightarrow^* s_3$ in $\rightarrow$, we may concatenate them to obtain $\pi_1 \odot \pi_2$ (by gluing together s_2). We call π_1 a *prefix* of $\pi_1 \odot \pi_2$. For each $S' \subseteq S$, we use the notations $pre_{\rightarrow}(S')$ and $post_{\rightarrow}(S')$ to denote the pre/post image of S' under $\rightarrow$. That is, $pre_{\rightarrow}(S') = \{p \in S : \exists q \in S'(p \rightarrow q)\}$ and $post_{\rightarrow}(S') = \{q \in S : \exists p \in S'(p \rightarrow q)\}$.

Words and automata: We assume basic familiarity with finite word automata. Fix a finite alphabet Σ. For each finite word $w = w_1 \ldots w_n \in \Sigma^*$, we write $w[i, j]$, where $1 \leq i \leq j \leq n$, to denote the segment $w_i \ldots w_j$. Given an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, a run of $\mathcal{A}$ on w is a function $\rho : \{0, \ldots, n\} \rightarrow Q$ with $\rho(0) = q_0$ that obeys the transition relation δ. We may also denote the run ρ by the word $\rho(0) \cdots \rho(n)$ over the alphabet Q. The run ρ is said to be *accepting* if $\rho(n) \in F$, in which case we say that w is *accepted* by $\mathcal{A}$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is the set of words in Σ^* accepted by $\mathcal{A}$.

Reachability games: We recall some basic concepts on 2-player reachability games (see e.g. [39, Chapter 2] on games with 1-accepting conditions). An *arena* is a transition system $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2 \rangle$, where S (i.e. the set of "game configurations") is partitioned into two disjoint sets V_1 and V_2 such that $pre_{\rightarrow_i}(S) \subseteq V_i$ for each $i \in \{1, 2\}$. The transition relation $\rightarrow_i$ denotes the actions of Player i. Similarly, for each $i \in \{1, 2\}$, the configurations V_i are controlled by Player i. In the following, Player 1 will also be called "Scheduler," and Player 2 "Process". Given a set $I_0 \subseteq S$ of initial configurations and a set $F \subseteq S$ of final (a.k.a. target) configurations, the goal of Player 2 is to reach F from I_0, while the goal of Player 1 is to avoid it. More formally, a *strategy* for Player i is a partial function $f : S^* V_i \rightarrow S$ such that, for each $v \in S^*$ and $p \in V_i$, if vp is a path in $\mathfrak{S}$ and p is not a dead end (i.e., $p \rightarrow_i q$ for some q), then $f(vp)$ is defined in such a way that $p \rightarrow_i f(vp)$. Given a strategy f_i for Player $i \in \{1, 2\}$ and an initial configuration $s_0 \in S$, we can define a unique (finite or infinite) path in $\mathfrak{S}$ such that $\pi : s_0 \rightarrow_{j_1} s_1 \rightarrow_{j_2} \cdots$ where $s_{j_{k+1}} = f_i(s_0 s_1 \ldots s_{j_k})$ for $i \in \{1, 2\}$ is the (unique) configuration s.t. $s_{j_k} \in V_i$. Player 2 *wins* iff some configuration in F appears in π, or if the path is finite and the last configuration belongs to Player 1. Player 1 *wins* iff Player 2 does not win; we say Player 2 *loses*. A strategy f for Player i is *winning* from I_0 if for each strategy g of Player $3 - i$, the unique path in $\mathfrak{S}$ from each $s_0 \in I_0$ witnesses a win for Player i. Such games (a.k.a. *reachability games*) are *determined* (see e.g. [39, Proposition 2.21]): either Player 1 has a winning strategy or Player 2 has a winning strategy.

Convention. *For notational simplicity, w.l.o.g., we make the following assumptions on our reachability games. They suffice for the purpose of proving liveness for parameterised systems.*

(A0) *Arenas are strictly alternating, i.e., a move made by a player does not take the game back to her configuration ($post_{\rightarrow_i}(S) \cap V_i = \emptyset$, for each $i \in \{1, 2\}$).*
(A1) *Initial and final configurations belong to Player 1, i.e., $I_0, F \subseteq V_1$*
(A2) *Non-final configurations are not dead ends: $\forall x \in S \setminus F, \exists y : x \rightarrow_1 y \vee x \rightarrow_2 y$.*

Markov chains: A (discrete-time) *Markov chain* (a.k.a. *DTMC*) is a structure of the form $\mathfrak{S} = \langle S; \delta, \ell \rangle$ where S is a set of configurations, δ is a function that associates a configuration $s \in S$ with a probability distribution over a sample space $D \subseteq S$ (i.e. the probability of going to a certain configuration from s), and $\ell : \mathsf{AP} \rightarrow 2^S$ maps atomic propositions to subsets of S. In what follows, we will assume that each $\delta(s)$ is a discrete probability distribution with a finite sample space. This assumption allows us

to simplify our notation: a DTMC $\langle S; \delta, \ell \rangle$ can be seen as a transition system $\langle S; \rightarrow, \ell \rangle$ with a transition probability function δ mapping a transition $t = (s, s') \in \rightarrow$ to a value $\delta(t) \in (0, 1]$ such that $\sum_{s' \in post(s)} \delta((s, s')) = 1$. That is, transitions with zero probabilities are removed from $\rightarrow$. We will write $s \xrightarrow{p} s'$ to denote $s \rightarrow s'$ and that $\delta((s, s')) = p$. The *underlying transition graph* of a DTMC $\langle S; \delta, \ell \rangle$ is the transition system $\langle S; \rightarrow, \ell \rangle$ with δ omitted. Given a finite path $\pi = s_0 \rightarrow \cdots \rightarrow s_n$ from the initial configuration $s_0 \in S$, let Run_π be the set of all finite/infinite paths with π as a prefix, i.e., of the form $\pi \odot \pi'$ for some finite/infinite path π'. Given a set $F \subseteq S$ of target configurations, the probability $\text{Prob}_{\mathfrak{S}}(s_0 \models \Diamond F)$ (the subscript $\mathfrak{S}$ may be omitted when understood) of reaching F from s_0 in $\mathfrak{S}$ can be defined using a standard cylinder construction (see e.g [40]) as follows. For each finite path $\pi = s_0 \rightarrow \cdots \rightarrow s_n$ in $\mathfrak{S}$ from s_0, we set Run_π to be a basic cylinder, to which we associate the probability $\text{Prob}(Run_\pi) = \prod_{i=0}^{n-1} \delta((s_i, s_{i+1}))$. This gives rise to a unique probability measure for the σ-algebra over the set of all runs from s_0. The probability $\text{Prob}(s_0 \models \Diamond F)$ is then the probability of the event F containing all paths in $\mathfrak{S}$ with some "accepting" finite path as a prefix, i.e., a finite path from s_0 ending in some configuration in F. In general, given an LTL formula φ over AP, the event containing all paths from s_0 in $\mathfrak{S}$ satisfying φ is measurable [10] and its probability value $\text{Prob}(s_0 \models \varphi)$ is well-defined.
Notation: Whenever understood, we will omit mention of ℓ from $\langle S; \delta, \ell \rangle$.

3 Abstract Models of Probabilistic Concurrent Programs

In this section, we recall the notion of Markov Decision Processes (MDPs) and fair MDPs [8]. These serve as our abstract models of probabilistic concurrent programs. We then define the notion of finitary fairness [22] and discuss its basic properties in the setting of MDPs.

3.1 Markov Decision Processes

A *Markov decision process* (*MDP*) is a strictly alternating arena $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2 \rangle$ such that $\langle S; \rightarrow_2 \rangle$ is a DTMC, i.e., $\rightarrow_2$ is associated with some transition probability function, and that the atomic propositions are not important. Intuitively, the transition relation $\rightarrow_1$ is nondeterministic (controlled by a "demonic" scheduler), whereas the transition relation $\rightarrow_2$ is probabilistic. By definition of arenas, the configurations of the MDPs are partitioned into the set V_1 of *nondeterministic states* (controlled by Scheduler) and the set V_2 of *probabilistic states*. Formally, $pre_{\rightarrow_1}(S) \cap pre_{\rightarrow_2}(S) = \emptyset$. Each Scheduler's strategy[5] $f : S^* V_1 \rightarrow S$ gives rise to an infinite-state DTMC with the underlying transition system $\mathfrak{S}_f = \langle S'; \rightarrow_3, \ell \rangle$ and the transition probability function δ' defined as follows. Here, S' is the set of all finite/infinite paths π from s_0. For each state $s' \in S$ and each path π from s_0 ending in some state $s \in S$, we define $\pi \rightarrow_3 \pi s'$ iff: (1) if $s \in V_1$ is a nondeterministic state, then $f(\pi) = s'$, and (2) if $s \in V_2$ is a probabilistic state, then $s \rightarrow_2 s'$. Intuitively, $\mathfrak{S}_f$ is an unfolding of the game arena $\mathfrak{S}$ (i.e. a disjoint union of trees) where branching only occurs on probabilistic states. Transitions $\pi \rightarrow_3 \pi s'$ satisfying Case (1) have the probability $\delta'((\pi, \pi s')) = 1$;

[5] Also called "scheduler" or "adversary" for short.

otherwise, its probability is $\delta'((\pi, \pi s')) = \delta((s, s'))$. We let ℓ be a function mapping each subset $X \subseteq S$ (used as an atomic proposition) to the set of all finite paths in $\mathfrak{S}_f$ from s_0 to X. Since $\mathfrak{S}_f$ is a DTMC, given an LTL formula φ over subsets of S as atomic propositions, the probability $\text{Prob}_{\mathfrak{S}_f}(s_0 \models \varphi)$ of satisfying φ in $\mathfrak{S}$ from s_0 under the scheduler f is well-defined. In particular, $\text{Prob}_{\mathfrak{S}_f}(s_0 \models \Diamond F)$ is the probability of reaching F from s_0 in $\mathfrak{S}$ under the scheduler f. The probability $\text{Prob}_{\mathfrak{S},\mathcal{C}}(s_0 \models \varphi)$ of satisfying φ from s_0 in the MDP $\mathfrak{S}$ under a class $\mathcal{C}$ of schedulers is defined to be the infimum of the set of all probabilities $\text{Prob}_{\mathfrak{S}_f}(s_0 \models \varphi)$ over all $f \in \mathcal{C}$. We will omit mention of $\mathcal{C}$ when it denotes the class of all schedulers.

An MDP is *weakly-finite* [28] if from each configuration, the set of all configurations that are reachable from it (in the underlying transition system of the MDP) is finite. Note that the state space of weakly-finite MDPs can be infinite. The restriction of weak finiteness is another way of defining the notion of *parameterized systems*, which are an infinite family of finite-state systems. Weakly-finite MDPs capture many interesting probabilistic concurrent systems in which each process is finite-state; this is the case for many probabilistic distributed protocols.

3.2 Fair Markov Decision Processes

A *fair Markov decision process (FMDP)* is a structure of the form $\mathfrak{S} = \langle S = V_1 \cup V_2; \to_1, \to_2, \mathfrak{C}, \mathfrak{J}\rangle$, where $\langle S = V_1 \cup V_2; \to_1, \to_2\rangle$ is an MDP, $\mathfrak{J}$ is a weak fairness (a.k.a. *justice*) requirement, and $\mathfrak{C}$ is a strong fairness (a.k.a. *compassion*) requirement. More precisely, a *weak fairness requirement* is a set (at most countably infinite) of *atomic weak fairness requirements* of the form $\Diamond\Box A \Rightarrow \Box\Diamond B$, for some $A, B \subseteq S$. Here, the $\Box$ and $\Diamond$ modalities are the standard "always" and "eventually" LTL operators. The set A (resp. B) will be called the *premise* (resp. *consequence*). Intuitively, if A is interpreted as "Process 1 is waiting to move" and B as "Process 1 is chosen", then this fairness requirement may be read as: at no point can Process 1 be continuously waiting to move without being chosen. In addition, a *strong fairness requirement* is a set (again, at most countably infinite) of *atomic strong fairness requirements* of the form $\Box\Diamond A \Rightarrow \Box\Diamond B$, for some $A, B \subseteq S$. Using the above example, a strong fairness requirement reads: if Process 1 is waiting to move infinitely often, then it is chosen infinitely often. As before, the set A (resp. B) will be called the *premise* (resp. *consequence*). In the following, when it is clear whether a fairness requirement is a justice or a compassion, we will denote it by the pair (A, B) of premise and consequence.

Given an FMDP $\mathfrak{S} = \langle S = V_1 \cup V_2; \to_1, \to_2, \mathfrak{C}, \mathfrak{J}\rangle$, a configuration $s_0 \in S$, and a scheduler f, since each atomic fairness requirement is an LTL formula and there are at most countably many atomic fairness requirements, the set of paths from s_0 in the DTMC $\mathfrak{S}_f$ induced by f satisfying $\mathfrak{C}$ and $\mathfrak{J}$ is measurable. We say that a scheduler f is $\mathfrak{S}$*-fair* if $\text{Prob}_{\mathfrak{S}_f}(s_0 \models \mathfrak{C} \wedge \mathfrak{J}) = 1$ for every initial configuration s_0. The fairness conditions $(\mathfrak{C}, \mathfrak{J})$ are *realizable* in $\mathfrak{S}$ if there exists at least one $\mathfrak{S}$-fair scheduler.

A natural fairness notion we consider in this paper is *process fairness*, which asserts that each process is chosen infinitely often. For this notion of fairness, we can assume that the consequence B of each atomic fairness requirement asserts that a particular process is chosen. We make one simplifying assumption: *each process is always enabled* (i.e., can always be chosen by the scheduler). This assumption is reasonable since we can always introduce an idle transition for each process. Under this assumption, we have that *from each $v_1 \in V_1$, there exists a transition $v_1 \to_1 v_2$ for some $v_2 \in B$.* This implies that our fairness conditions are always realizable and that the probability $\mathrm{Prob}_{\mathfrak{S},\mathcal{C}}(E)$ of event E over the set of all $\mathfrak{S}$-fair schedulers is well-defined.

3.3 Finitary Fairness

Given an FMDP $\mathfrak{S} = \langle S = V_1 \cup V_2; \to_1, \to_2, \mathfrak{C}, \mathfrak{J}\rangle$, a configuration $s_0 \in S$, and a number $k \in \mathbb{N}$, we say that a scheduler f is *$\mathfrak{S}$-k-fair* (or *k-fair* whenever $\mathfrak{S}$ is understood) if for each atomic fairness requirement (A, B):

1. if (A, B) is justice, then (the underlying graph of) $\mathfrak{S}_f$ contains no path π of length k satisfying the LTL formula $\Box(A \wedge \neg B)$.
2. if (A, B) is compassion, then $\mathfrak{S}_f$ contains no path π satisfying the LTL formula $\psi_k \wedge \Box\neg B$, where $\psi_0 := \mathit{true}$ and $\psi_i := \Diamond(A \wedge \psi_{i-1})$ for each $i > 0$.

In other words, a premise in a justice requirement cannot be satisfied for k *consecutive* steps without satisfying a consequence, while a premise in a compassion requirement cannot be satisfied for k (not necessarily consecutive) steps without satisfying a consequence. A scheduler is said to be *finitary fair (fin-fair)* if it is k-fair for some k. The fairness conditions $(\mathfrak{C}, \mathfrak{J})$ are said to be *finitary-realizable (fin-realizable)* in $\mathfrak{S}$ if there exists at least one fin-fair scheduler. Under this assumption, the probability $\mathrm{Prob}_{\mathfrak{S},\mathcal{C}}(E)$ of an event E over the set $\mathcal{C}$ of all fin-fair schedulers is well-defined. In what follows, for an FMDP $\mathfrak{S}$, we will simply denote $\mathrm{Prob}_{\mathfrak{S},\mathcal{C}}(E)$ as $\mathrm{Prob}_{\mathfrak{S}}(E)$. In this paper, we propose to study *termination of probabilistic concurrent programs under finitary fairness*, i.e., to determine whether $\mathrm{Prob}_{\mathfrak{S},\mathcal{C}}(s_0 \models \Diamond F) = 1$, where $\mathcal{C}$ is the class of all fin-fair schedulers.

The following proposition states one special property of weakly-finite MDPs.

Proposition 1. *Let $\mathfrak{S}$ and $\mathfrak{S}'$ be two weakly-finite fair MDPs with identical underlying transition systems (but possibly different probability values). For each set F of final states, and each initial configuration s_0, it is the case that $\mathit{Prob}_{\mathfrak{S}}(s_0 \models \Diamond F) = 1$ iff $\mathit{Prob}_{\mathfrak{S}'}(s_0 \models \Diamond F) = 1$.*

Proof. This proposition can be proved using basic machineries from probabilistic model checking [8]. Consider the finite MDPs $\mathfrak{S}^1$ and $\mathfrak{S}^2$ that are obtained from $\mathfrak{S}$ and $\mathfrak{S}'$ by removing configurations in F and their fairness requirements φ. It suffices to prove the following: for all schedulers $\mathrm{Prob}_{\mathfrak{S}^1_\sigma}(s_0 \models \varphi) = 0$ iff for all schedulers σ $\mathrm{Prob}_{\mathfrak{S}^2_\sigma}(s_0 \models \varphi) = 0$. This follows from standard results from probabilistic model checking [8, Theorem 10.122] since φ is a limit linear-time property. □

By Proposition 1, when dealing with almost-sure finitary-fair termination of weakly-finite MDPs, we only care whether a transition has a zero or a non-zero probability, i.e., if it is non-zero, then the exact value is irrelevant. Incidentally, the same also holds for other properties including almost-sure termination without fairness and qualitative temporal specifications [26,27,20]. *For this reason, we may simply omit these probability values from our symbolic representation of weakly-finite MDPs, which we will do from the next section onwards.*

3.4 Herman's Protocol

Herman's protocol [2] is a distributed self-stabilization algorithm for a population of processes organized in a ring. The *correct* configurations are those where exactly one process holds a token. If, through some error, the ring enters an *erroneous* configuration (in which multiple processes hold tokens), Herman's protocol ensures that the system will *self-stabilize*: it will almost surely go back to a configuration with only one token.

Let us discuss how the protocol works in more detail. Fix $N \geq 3$ processors organized in a ring. If a chosen process does not hold a token, then it can perform an idle transition (i.e. do nothing). If a chosen process holds a token, then it can keep holding the token with probability $\frac{1}{2}$ or pass it on to its clockwise neighbor (the process $(i+1) \bmod N$, for processes numbered $0, \ldots, N-1$) with probability $\frac{1}{2}$. If a process currently holds a token and receives another token from its (counter-clockwise) neighbor, then the two tokens are merged[6] into one, leaving the process with one token.

Formally, Hermann's protocol can be modeled as a weakly-finite Markov decision process whose states are vectors in $\{\bot, \top\}^*$. For each N, the state of the protocol is described by a vector of N bits, with the i-th bit being 1 iff the i-th process holds a token. From a state $\mathbf{v}$, the scheduler picks a process $i \in \{0, \ldots, N-1\}$. Given a chosen process i, the new state remains $\mathbf{v}$ if the chosen process i did not hold a token ($\mathbf{v}(i) = \bot$). If $\mathbf{v}(i) = \top$, the new state is $\mathbf{v}$ with probability $\frac{1}{2}$ and $\mathbf{v} \oplus e_i \oplus e_{(i+1) \bmod N}$ with probability $\frac{1}{2}$. Here, e_i denotes a vector with $\top$ in the i-th position and $\bot$ everywhere else, and $\oplus$ is the XOR operation. We want to ensure that, starting from an arbitrary initial assignment of tokens, any population self-stabilizes with probability 1.

Process fairness for Herman's protocol is a set of N atomic fairness requirements, each asserting that the process i is executed infinitely often, for each $i \in \{1, \ldots, N\}$. Unfortunately, Herman's protocol does *not* terminate with probability 1 against some fair schedulers. To see this, consider the start state $s_0 = (\top, \bot, \top)$. Let us call the token held by Process 0 "the first token", and the token held by Process 2 "the second token". Define a *round* as the following sequence of moves by the scheduler: keep choosing the process that holds the first token until it passes the token to the right, and do the same to the second token. For example, the two configurations obtained after completing the first and second rounds from s_0 are, respectively, $(\top, \top, \bot)$ and $(\bot, \top, \top)$. To see that the scheduler is fair, for each integer $i > 0$, the probability that the i-th round is not completed is 0 since the probability that one of the tokens will be kept at the same process for an infinite amount of time is 0. Therefore, the probability that some round is not completed is also 0. Completing two rounds ensure that all the processes are picked. Therefore, every process will be chosen with probability 1. On the other hand, observe that correct configurations are not seen in the induced DTMC, showing that self-stabilization holds with probability 0 under this scheduler.

Herman's protocol can be shown to self-stabilize with probability 1 under all fin-fair schedulers, which can be proved by our fully-automatic verification algorithm (presented later in the paper).

[6] Herman [2] describes a more general protocol in which tokens can be merged/destroyed with some probability. We consider this restriction for simplicity of presentation.

4 Regular Model Checking: A Symbolic Framework

In this section, we recall *regular model checking* (see e.g. [14,15,41]), a symbolic framework for specifying infinite-state systems based on finite automata and regular transducers and developing automatic verification (semi-)algorithms.

A transition system $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2 \rangle$ is specified in the framework as a regular language S (e.g. as a regular expression over some alphabet Σ), and two "regular relations" $\rightarrow_1, \rightarrow_2 \subseteq \Sigma^* \times \Sigma^*$. For simplicity, in the following we will assume that $S = \Sigma^*$. How do we specify regular relations? One standard way is to restrict to length-preserving relations (i.e. the relation may only contain a pair of words of the same length) and specify such relations as regular languages over the alphabet $\Sigma \times \Sigma$. There is, then, a simple one-to-one correspondence between the set of words over $\Sigma \times \Sigma$ and the set of all pairs of words over Σ of the same length. This can be achieved by mapping a pair (v, w) of words Σ with $|v| = |w| = n$ to a word $v \otimes w$, defined as $(v_1, w_1)(v_2, w_2) \cdots (v_n, w_n)$ whenever $v = v_1 \cdots v_n$ and $w = w_1 \cdots w_n$.

Proving that a property φ holds over a transition system $\mathfrak{S}$ is done "in a regular way,", by finding a "regular proof" for the property. For example, if φ asserts that the set Bad of bad states can never be reached, then a regular proof amounts to finding an inductive invariant Inv in the form of a regular language [14,15] that does not intersect with Bad, i.e., $Bad \cap Inv = \emptyset$, $S_0 \subseteq Inv$ (S_0 is a regular set of initial states), and $post_{\rightarrow}(Inv) \subseteq Inv$, where $\rightarrow = \rightarrow_1 \cup \rightarrow_2$. Since regular languages are effectively closed under boolean operations and taking pre/post images w.r.t. regular transducers, an algorithm for verifying the correctness of a given regular proof can be obtained by using language inclusion algorithms for regular automata, e.g., [42,43]. The framework of regular proofs is incomplete in general since it could happen that there is a proof, but no regular proof. The pathological cases when only non-regular proofs exist do not, however, seem to frequently occur in practice, e.g., see [44,45,18,14,19,46,47,15,48].

The framework of regular proofs has been extended to deal with almost-sure termination for weakly-finite probabilistic concurrent programs in [20]. We briefly summarise the main idea, since we reduce the fair termination problem to their setting. By Proposition 1, the actual probability values do not matter in proving almost-sure termination. For this reason, we may specify a weakly-finite MDP $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2 \rangle$ as a regular specification in the same way as we specify a non-probabilistic transition system in our regular specification language. Given an MDP $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2 \rangle$, a set $I_0 \subseteq V_1$ of initial configurations, and a set $F \subseteq V_1$ of final configurations, a regular proof for $\text{Prob}(s_0 \models F) = 1$ for each $s_0 \in I_0$ is a pair $\langle Inv, \prec \rangle$ consisting of a regular inductive invariant $Inv \subseteq S$ and a regular relation $\prec \subseteq S \times S$ such that:

1. $I_0 \subseteq Inv$ and $post_{\rightarrow}(Inv) \subseteq Inv$.
2. $\prec$ is a strict preorder on S, i.e., it is irreflexive ($\forall s \in S : s \not\prec s$) and transitive ($\forall s, s', s'' \in S : s \prec s' \wedge s' \prec s'' \implies s \prec s''$).
3. irrespective of the nondeterministic transitions from any configuration in Inv, there is a probabilistic transition to a configuration in Inv that decreases its rank with respect to $\prec$:

$$\forall x \in Inv \setminus F, y \in S \setminus F : \quad \big((x \rightarrow_1 y) \Rightarrow (\exists z \in Inv : (y \rightarrow_2 z) \wedge x \succ z)\big) .$$

An automata-theoretic algorithm can then be devised for checking the above verification conditions with respect to a given regular proof [20].

Example 1. **[Herman's protocol, continued]** We provide a regular encoding of Herman's protocol. The configurations are words over the alphabet $\{\top, \bot, \overline{\top}, \overline{\bot}\}$, where $\top$ (resp. $\bot$) signifies that a process holds (resp. does not hold) a token, while overlining the character signifies that the process is chosen by the scheduler. We set $\Sigma = \{\top, \bot\}$. The set S_0 of initial configurations is $\Sigma^*\top\Sigma^*$, i.e., at least one process holds a token. The set of final configurations is $\bot^*\top\bot^*$, i.e., there is only a single token in the system. The actions of the scheduler is to choose a process; this can be expressed as the regular expression $I^*((\top, \overline{\top}) + (\bot, \overline{\bot}))I^*$, where I denotes the regular language $(\top, \top) + (\bot, \bot)$. The probabilistic actions can be expressed as a union of the following three regular expressions:

$$I^*((\overline{\top}, \top) + (\overline{\bot}, \bot))I^* \qquad \textbf{(idle)}$$

$$I^*(\overline{\top}, \bot)((\bot, \top)) + (\top, \top))I^*, \quad ((\bot, \top) + (\top, \top))I^*(\overline{\top}, \bot)) \qquad \textbf{(pass token right)}$$

5 Handling Fairness Requirements

We now describe the main result of the paper: a general method for embedding finitary fairness into regular model checking for probabilistic concurrent systems.

5.1 Regular Specifications of Fairness

When a complex system or a distributed protocol is being modelled in regular model checking, it is often necessary to add an *infinite* number of fairness requirements. This is because such a system admits a finite but arbitrary number of agents or processes, each with its own fairness requirement (e.g. that the process should be executed infinitely often). For this reason, it is not enough to simply express the fairness requirements as a finite set of pairs of regular languages (one for the premise, and one for the consequence). We describe a regular way of specifying infinitely many fairness constraints. Our presentation is a generalisation of the regular specification of fairness from [16,15].

The general idea is to define a "regular function" $\mathcal{T}$ that maps a configuration $s = s_1 \cdots s_n \in S$ to a word $w = w_1 \cdots w_n$ such that w_i contains: (1) a bit b_i indicating whether s is in the premise of the i-th fairness requirement, (2) a bit b'_i indicating whether s is in the consequence of the i-th fairness requirement, and (3) a bit t indicating whether the i-th fairness requirement is justice or compassion. Such a regular specification of fairness allows an infinite number of fairness constraints since S is potentially infinite (i.e., containing words of unbounded lengths), though only the first $|s|$ fairness requirements matter for a word $s \in S$. This is sufficient for weakly-finite MDPs since the set of reachable configurations from any given configuration s is finite and so, among the infinite number of fairness constraints, only finitely many are distinguishable. The regular function can be defined by a letter-to-letter transducer with input alphabet Σ and output alphabet $\Gamma := \{0, 1\} \times \{0, 1\} \times \{0, 1\}$. Without loss of generality, we assume that the i-th letter in the output of every input word of $\mathcal{T}$ agree on the third bit (i.e., whether the fairness requirement is justice or compassion is well-defined): for every $s, s' \in S$ and $i \in \mathbb{N}$, if $\mathcal{T}(s)[i] = (a, b, c)$ and $\mathcal{T}(s')[i] = (a', b', c')$, then $c = c'$. Observe this condition on $\mathcal{T}$ can be algorithmically checked by using a simple automata-theoretic method: find two accepted words in which in some position their third bits differ.

In this case, $\mathcal{T}$ gives rise to compassion requirements $\mathfrak{C}$ and justice requirements $\mathfrak{J}$ by associating the i-th position in all output words by a unique fairness constraint. More precisely, let

- $A_i = \{s : \mathcal{T}(s)[i] = (1, j, t),$ for some $j, t \in \{0, 1\}\}$ and
- $B_i = \{s : \mathcal{T}(s)[i] = (j, 1, t),$ for some $j, t \in \{0, 1\}\}$.

Define:

(i) $\mathfrak{J} = \{\Diamond\Box A_i \Rightarrow \Box\Diamond B_i : \mathcal{T}(s)[i] = (i, j, 0),$ for some $s \in S$ and $j \in \{0, 1\}\}$,
(ii) $\mathfrak{C} = \{\Box\Diamond A_i \Rightarrow \Box\Diamond B_i : \mathcal{T}(s)[i] = (i, j, 1),$ for some $s \in S$ and $j \in \{0, 1\}\}$.

Therefore, by Proposition 1, our regular fairness specification allows us to define weakly-finite fair MDPs $\langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2, \mathfrak{C}, \mathfrak{J}\rangle$. In the following, we shall call such fair MDPs *regular*.

Our main theorem is a regularity-preserving reduction from proving almost sure termination for regular FMDPs (under finitary fairness) to proving almost sure termination for regular MDPs (without fairness).

Theorem 1. *Let $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2, \mathfrak{C}, \mathfrak{J}\rangle$ be a regular representation of an FMDP, $I_0 \subseteq V_1$ be a regular set of initial configurations, and $F \subseteq V_1$ be a regular set of final configurations. Then one can compute a regular representation of MDP $\mathfrak{S}' = \langle S = V_1' \cup V_2'; \rightsquigarrow_1, \rightsquigarrow_2\rangle$ and two regular sets $I_0', F' \subseteq V_1'$ such that it holds that if $\mathfrak{C}$ and $\mathfrak{J}$ are realizable, then $Prob_{\mathfrak{S}'}(I_0' \models \Diamond F') = 1$ iff $Prob_{\mathfrak{S}}(I_0 \models \Diamond F) = 1$.*

5.2 Abstract Program Transformation

Before proving Theorem 1, let us first recall an abstract program transformation *à la* Alur & Henzinger [22], which encodes finitary fairness into a program using integer counter variables. Intuitively, we reserve one variable for each atomic fairness condition as an "alarm clock" that will set off if its corresponding process has not been executed for a long time, and one global variable n that acts as a *default* value to reset a clock to as soon as the corresponding process is executed. Although Alur & Henzinger [22] did not discuss about probabilistic programs, their transformation can be easily adapted to the setting of MDPs, though correctness still has to be proven.

We now elaborate on the details of the transformation. Given an FMDP $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2, \mathfrak{C}, \mathfrak{J}\rangle$ with a probability distribution δ, the transformation will produce an MDP $\mathfrak{S}' = \langle S = V_1' \cup V_2'; \rightsquigarrow_1, \rightsquigarrow_2\rangle$ with a probability distribution δ' as follows. Introduce a set $\mathcal{V}$ of "counter" variables that range over natural numbers: x_j (for each $j \in \mathfrak{J}$), y_c (for each $c \in \mathfrak{C}$), and n. Let $\mathfrak{F}$ be the set of all valuations f mapping each variable in $\mathcal{V}$ to a natural number such that $f(x_j), f(y_c) \leq f(n)$ for each $j \in \mathfrak{J}$ and $c \in \mathfrak{C}$. We define $V_1' = V_1 \times \mathfrak{F}$ and $V_2' = V_2 \times \mathfrak{F}$. We now define the transition relation $\rightsquigarrow_i$ such that $(s, f) \rightsquigarrow_i (s', f')$ if $s \rightarrow_i s'$ and

- for each $z \in \mathcal{V}$, $f(z) > 0$,

- $f'(n) := f(n)$,
- x_j (for $j = (A, B) \in \mathfrak{J}$) and y_c (for $c = (A, B) \in \mathfrak{C}$) change as follows:

$$f'(x_j) = \begin{cases} f(x_j) - 1 & \text{if } s \in A \cap \overline{B} \\ f(n) & \text{if } s \in \overline{A} \cup B \end{cases} \qquad f'(y_c) = \begin{cases} f(n) & \text{if } s \in \overline{A} \cap \overline{B} \\ f(y_c) - 1 & \text{if } s \in A \cap \overline{B} \\ f(n) & \text{if } s \in B \end{cases}$$

($\overline{A}$ denotes the set-complement of A). Finally, we define the probability distribution δ' underlying $\rightsquigarrow_2$ as $\delta'((s, f), (s', f')) = \delta(s, s')$ whenever $s \in V_2$.

Intuitively, the variables x_j's and y_c's keep track of how long the scheduler has delayed choosing an enabled process, while the variable n (unchanged once the initial configuration of the MDP is fixed) aims to ensure that the scheduler is n-fair. Since n is a variable (not a constant), the resulting MDP $\mathfrak{S}'$ captures precisely the behaviour of $\mathfrak{S}$ under fin-fair schedulers.

Lemma 1. *If $\mathfrak{S}$ is a weakly-finite FMDP, then $\mathfrak{S}'$ is weakly-finite.*

Proof. Since $\mathfrak{S}$ is weakly-finite, once a configuration (s, f) of $\mathfrak{S}'$ is chosen, there are only finitely many different valuation $s' \in S$ such that (s', f') (for some $f' \in \mathfrak{F}$) is reachable from (s, f). In the following, we show that there are also only finitely many different valuations $f' \in \mathfrak{F}$ such that (s', f') (for some $s' \in S$) is reachable from (s, f). Let X be the (finite) set $X = post_{\rightarrow^*}(s)$. Define two equivalence relations $\sim_{(s,f),c}$ and $\sim_{(s,f),j}$ on the set $2^S \times 2^S$ of pairs of subsets of S as follows:

- $(A, B) \sim_{(s,f),c} (A', B')$ iff (a) $A \cap X = A' \cap X$ and $B \cap X = B' \cap X$, and (b) $(A, B) \in \mathfrak{C}$ iff $(A', B') \in \mathfrak{C}$.
- $(A, B) \sim_{(s,f),j} (A', B')$ iff (a) $A \cap X = A' \cap X$ and $B \cap X = B' \cap X$, and (b) $(A, B) \in \mathfrak{J}$ iff $(A', B') \in \mathfrak{J}$.

Observe that, since X is finite, both equivalence relations are of finite index (i.e. have only finitely many equivalence classes). This implies that we need not distinguish two variables in $\mathcal{V} \setminus \{n\}$ if they are both for the justice or both for the compassion requirements, in the same equivalence class in the appropriate relation $\sim_{(s,f),c}$ or $\sim_{(s,f),j}$, and they both have the *same* initial f-values. To see this, let $\rightsquigarrow \; := \; \rightsquigarrow_1 \cup \rightsquigarrow_2$. Observe that, for each $(s', f') \in post_{\rightsquigarrow^*}((s, f))$ and $c = (A, B), c' = (A', B') \in \mathfrak{C}$, it is the case that $f'(y_c) = f'(y_{c'})$ iff $f(y_c) = f(y_{c'})$. Similarly, for each $(s', f') \in post_{\rightsquigarrow^*}((s, f))$ and $j = (A, B), j' = (A', B') \in \mathfrak{J}$, it is the case that $f'(x_j) = f'(x_{j'})$ iff $f(x_j) = f(x_{j'})$. In other words, identical counter values across similar fairness constraints remain identical under an application of $\rightsquigarrow$. Since all counter values in all reachable configurations $(s', f') \in post_{\rightsquigarrow^*}((s, f))$ are in $\{0, \ldots, f(n)\}$, it immediately follows that $post_{\rightsquigarrow^*}((s, f))$ is finite. □

We next state a correctness lemma for the transformation. To this end, given a set $S_0 \subseteq S$ of initial configurations in $\mathfrak{S}$, we define:

- $S_0' := S_0 \times \mathfrak{F}_=$, where $\mathfrak{F}_=$ contains functions $f \in \mathfrak{F}$ such that $f(x_j) = f(y_c) = f(n)$ for each $j \in \mathfrak{J}$ and $c \in \mathfrak{C}$.

- $F' = (F \times \mathfrak{F}_{>0}) \cup (S \times \mathfrak{F}_0)$, where $\mathfrak{F}_0$ contains all $f \in \mathfrak{F}$ such that $f(x_j) = 0$ for some $j \in \mathfrak{J}$ or $f(y_c) = 0$ for some $c \in \mathfrak{C}$ (i.e. one of the alarms has been triggered), and $\mathfrak{F}_{>0} := \mathfrak{F} \setminus \mathfrak{F}_0$.

Lemma 2 (Correctness). *If $\mathfrak{S}$ is a weakly-finite FMDP, it is the case that*

$$\mathit{Prob}_{\mathfrak{S}}(S_0 \models \Diamond F) = \mathit{Prob}_{\mathfrak{S}'}(S_0' \models \Diamond F').$$

Proof. In this proof, we make use of the following notation. For a sequence π of pairs $(x_1, y_1), (x_2, y_2), \ldots$, we use the notation $\mathit{proj}_1(\pi)$ (resp. $\mathit{proj}_2(\pi)$) to denote the sequence π projected to the first (resp. second) arguments, i.e., $x_1, x_2, \ldots$ (resp. $y_1, y_2, \ldots$). Moreover, for each $k > 0$, we define $\mathfrak{F}_k$ to be the set of all functions $f \in \mathfrak{F}_=$ such that $f(n) = k$.

We first prove that $\mathrm{Prob}_{\mathfrak{S}}(S_0 \models \Diamond F) \geq \mathrm{Prob}_{\mathfrak{S}'}(S_0' \models \Diamond F')$, i.e., the transformation does not increase the probability of reaching final states. For each $k \in \mathbb{N}$, consider a k-fair scheduler σ for $\mathfrak{S}$. It suffices to prove that given any $s_0 \in S_0$, $\mathrm{Prob}_{\mathfrak{S}_\sigma}(s_0 \models \Diamond F) = \mathrm{Prob}_{\mathfrak{S}'_{\sigma'}}((s_0, f) \models \Diamond F')$, where $f \in \mathfrak{F}_k$ and $\sigma'(\pi) := \sigma(\mathit{proj}_1(\pi))$ (note that $\mathfrak{F}_k$ contains exactly one f compatible with s_0). In turn, to prove this, it suffices to prove that the DTMC $\mathfrak{S}_\sigma$ restricted to configurations reachable from s_0 is isomorphic to $\mathfrak{S}'_{\sigma'}$ restricted to configurations reachable from (s_0, f). This can be seen from the fact that configurations of the form $(S \setminus F) \times \mathfrak{F}_0$ are not reachable from (s_0, f); if they were reachable, since the counter encoding precisely emulates the definition of finitary fairness [22], the witnessing path π would give rise to a path $\mathit{proj}_1(\pi)$ that would witness that σ is not k-fair, contradicting our original assumption.

We next prove that $\mathrm{Prob}_{\mathfrak{S}}(S_0 \models \Diamond F) \leq \mathrm{Prob}_{\mathfrak{S}'}(S_0' \models \Diamond F')$, i.e., our transformation does not decrease the probability of reaching final states. Consider any $(s_0, f) \in S_0'$ and any scheduler σ' on $\mathfrak{S}'$. Consider the scheduler σ on $\mathfrak{S}$ that simulates the behaviour of σ', but as soon as one of the alarm clocks has set off the scheduler goes through all consequence sets (say, $X_1, \ldots, X_m$ for some $m \in \mathbb{N}$; the sequence is finite since $\mathfrak{S}$ and $\mathfrak{S}'$ are weakly finite) in some order and chooses actions that satisfy them in a round robin manner (which can be done since we consider process constraints). More precisely, for each path π in $\mathfrak{S}'_{\sigma'}$, define $\sigma(\mathit{proj}_1(\pi)) := \sigma'(\pi)$. For each path π ending in a configuration in $(S \setminus F) \times \mathfrak{F}_0$, the action of the scheduler on any path with π as a prefix is to loop through $X_1, \ldots, X_m$ and pick actions that satisfy them. Therefore, the scheduler σ is K-fair for $K := 2m$. Furthermore, consider the two tree-shaped DTMCs $\mathfrak{S}_1$ and $\mathfrak{S}_2$, where $\mathfrak{S}_1$ is obtained from $\mathfrak{S}_\sigma$ by restricting the sets of configurations to those that are reachable from s_0, and $\mathfrak{S}_2$ is obtained from $\mathfrak{S}_{\sigma'}$ by restricting the sets of configurations to those that are reachable from (s_0, f). $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are isomorphic except for subtrees Run_π where π is a path in $\mathfrak{S}_2$ from (s, f) ending in $(S \setminus F) \times \mathfrak{F}_0$ without visiting a configuration in $F \times \mathfrak{F}_{>0}$. The probability p of visiting a configuration in F' in $\mathfrak{S}_2$ from (s, f') on the condition that π is taken is 1. Thus, on the condition that the prefix $\mathit{proj}_1(\pi)$ is taken, the probability of visiting a configuration in F' in $\mathfrak{S}_1$ cannot exceed p. This shows us that $\mathrm{Prob}_{\mathfrak{S}_\sigma}(s_0 \models \Diamond F) \leq \mathrm{Prob}_{\mathfrak{S}'_{\sigma'}}((s_0, f) \models \Diamond F')$. Consequently, since the choice of $(s_0, f) \in S_0'$ and σ' was arbitrary, we can conclude that $\mathrm{Prob}_{\mathfrak{S}}(S_0 \models \Diamond F) \leq \mathrm{Prob}_{\mathfrak{S}'}(S_0' \models \Diamond F')$. □

These two lemmas immediately imply Theorem 1.

5.3 Finitary Fairness in Regular Model Checking

We now show how to implement the aforementioned abstract program transformation in our regular model checking framework. Fix a regular representation of an FMDP $\mathfrak{S} = \langle S = V_1 \cup V_2; \rightarrow_1, \rightarrow_2, \mathfrak{C}, \mathfrak{J}\rangle$, which includes two automata over the alphabet $\Sigma \times \Sigma$ representing $\rightarrow_1$ and $\rightarrow_2$, and an automaton over the alphabet $\Sigma \times \Gamma$ representing the regular specification of the fairness conditions $\mathfrak{C}$ and $\mathfrak{J}$. [Recall that $\Gamma := \{0,1\} \times \{0,1\} \times \{0,1\}$.] We describe the construction of $\rightsquigarrow_1$ (the construction for $\rightsquigarrow_2$ is similar). Let $\mathcal{A} = (\Sigma \times \Sigma, Q, \Delta, q_0, F)$ be an automaton representing $\rightarrow_1$ and $\mathcal{A}^f = (\Sigma \times \Gamma, Q^f, \Delta^f, q_0^f, F^f)$ be an automaton representing the regular specification of fairness. The construction of the automaton for $\rightsquigarrow_1$ has two stages.

Stage 1: compute an intermediate automaton. The intermediate automaton $\mathcal{B}$ will have the alphabet $\Sigma' := (\Sigma \times \Sigma) \cup \Gamma$ and recognize a subset of $((\Sigma \times \Sigma)\Gamma)^*$. Intuitively, on input $(a,b) \in \Sigma \times \Sigma$, the automaton $\mathcal{B}$ simultaneously takes a transition over (a,b) in $\mathcal{A}$ and any transition (a,c) in $\mathcal{A}^f$, proceeding into an intermediate state where it remembers the value of c, which it outputs in the next step. This process is repeated until both $\mathcal{A}$ and $\mathcal{A}^f$ accept. More precisely, the automaton is defined as $\mathcal{B} := (\Sigma', Q^B, \Delta^B, q_0^B, F^B)$ where:

- $Q^B = Q \times Q^f \times (\Gamma \cup \{?\})$, $q_0^B = (q_0, q_0^f, ?)$, and $F^B = F \times F^f \times \{?\}$
- Δ_B has the following transitions:
  - $((p_1, q_1^f, ?), (a,b), (p_2, q_2^f, c))$ if $(p_1, (a,b), p_2) \in \Delta$ and $(q_1^f, (a,c), q_2^f) \in \Delta^f$.
  - $((p, q^f, c), c, (p, q^f, ?))$ for each $c \in \Gamma$.

Stage 2: regular substitution of letters in Γ. Our encoding of counters is unary using symbols $\bullet$ and $\circ$, where $\bullet$ represents a pebble, and $\circ$ represents empty space. For instance, a number $n \in \mathbb{N}$ is encoded as $\bullet^n \circ^*$ (the number of $\circ$'s is arbitrary, though the length of the whole encoding is constant due to our use of length-preserving transducers). We define the following regular languages for manipulating the counters:

- (Identity) $\texttt{ID} := (\bullet,\bullet)^+(\circ,\circ)^*$,
- (Decrement) $\texttt{DEC} := (\bullet,\bullet)^*(\bullet,\circ)(\circ,\circ)^*$, and
- (Reset) $\texttt{RESET} := (\bullet,\bullet)^+(\circ,\bullet)^*$.

Define the regular substitution σ mapping letters in Γ to regular languages:

- if (x,y,z) is $(i,1,j)$ or $(0,i,0)$ (for $i,j \in \{0,1\}$), then $\sigma((x,y,z)) = \texttt{RESET}$.
- if (x,y,z) is of the form $(1,0,i)$ (for some $i \in \{0,1\}$), then $\sigma((x,y,z)) = \texttt{DEC}$.
- define $\sigma((0,0,1)) = \texttt{ID}$.

We then apply the regular substitution σ to the letters Γ appearing in our intermediate automaton $\mathcal{B}$. The resulting automaton implements the desired automaton for $\rightsquigarrow_1$.

Finishing off the rest of the construction. Computing S_0' and F' is easy. Define S_0' to be the set of all words $a_1 w_1 a_2 w_2 \cdots a_m w_m$ such that $a_1 \cdots a_m \in S_0$ and $w_i \in \bullet^+$ for each $i \in \{1, \ldots, m\}$. Similarly, define F' to be the set of all words $a_1 w_1 a_2 w_2 \cdots a_m w_m$ such that

- either $a_1 \cdots a_m \in F$ and $w_i \in (\bullet^+ \circ^*) \cup \circ^+$ for each $i \in \{1, \ldots, m\}$, or
- $w_i \in \circ^+$ for some $i \in \{1, \ldots, m\}$.

Finite automata for these sets could be easily constructed given automata for S_0 and F.

Example 2. **[Herman's protocol]** We encode process fairness in the following way. The counters use the unary encoding, their values represented as the lengths of sequences of $\bullet$'s padded on the right by the symbol $\circ$ (crucial to keep the transducers length-preserving). For example, the number 3 is represented by any word of the form $\bullet \bullet \bullet \circ^*$. Define $\mathcal{X} = \bullet^* \circ^*$, i.e., the set of all valid counters. The set of initial configurations can be expressed using the regular expression $(\Sigma \cdot \mathcal{X})^* (\top \cdot \mathcal{X})(\Sigma \cdot \mathcal{X})^*$, i.e., counters for all processes are initialized to an arbitrary value. The set of final configurations is now $(\bot \cdot \mathcal{X})^* (\top \cdot \mathcal{X})(\bot \cdot \mathcal{X})^* \cup (\Sigma \cdot \mathcal{X})^* (\Sigma \cdot \circ^*)(\Sigma \cdot \mathcal{X})^*$, i.e., either there is exactly one token in the system, or (at least) one counter has reached 0. Scheduler now also performs operations on the counters for processes: for a chosen process, the counter is reset, for other processes, the counter is decremented. This can be expressed as the language $(I \cdot \mathtt{DEC})^* \big(((\bot, \overline{\bot}) + (\top, \overline{\top})) \cdot \mathtt{RESET}\big)(I \cdot \mathtt{DEC})^*$. Actions of the protocol are the same as in the original encoding and the values of counters are left unmodified:

$$(I \cdot \mathtt{ID})^* \big(((\overline{\bot}, \bot) + (\overline{\top}, \top)) \cdot \mathtt{ID}\big)(I \cdot \mathtt{ID})^* \quad \textbf{(idle)}$$

$$(I \cdot \mathtt{ID})^* \big((\overline{\top}, \bot) \cdot \mathtt{ID}\big)\big(((\bot, \top) + (\top, \top)) \cdot \mathtt{ID}\big)(I \cdot \mathtt{ID})^* \quad \textbf{(pass token right}_1\textbf{)}$$

$$\big(((\bot, \top) + (\top, \top)) \cdot \mathtt{ID}\big)(I \cdot \mathtt{ID})^* \big((\overline{\top}, \bot) \cdot \mathtt{ID}\big) \quad \textbf{(pass token right}_2\textbf{)}$$

At this point, we can use existing tools for checking termination (without fairness constraints), e.g. [20]. Indeed, we can automatically check that the system after reduction terminates with probability one, thus proving that Herman's protocol fairly terminates with probability one (under finitary process-fair schedulers).

6 Implementation and Experiments

The approach presented in this paper has been implemented in the tool FAIRYTAIL.[7] For evaluation, we extracted models of a number of probabilistic parameterized systems. The tool receives a system with fairness conditions and transforms it into a system without fairness conditions, where fairness of the original system is encoded using counters. For solving liveness in the output transformed system, we use SLRP [20] (in the *incremental liveness proofs* setting) as the underlying liveness checker for parameterized systems.

Table 6 shows the results of our experiments. The times given are the wall clock times for the individual benchmarks on a PC with 4 Quad-Core AMD Opteron 8389 processors with Java heap memory limited to 64 GiB. The time included translation of the system into a system without fairness (always less than 1s) and the runtime of SLRP.

[7] `https://github.com/uuverifiers/autosat/tree/master/Fairness`

Table 1. Times of analyses of probabilistic paremeterised systems. The timeout was set to 10 hours (timeout is denoted as T/O).

Case study	Time
Herman's protocol (merge, line)	3.64 s
Herman's protocol (annih., line)	4.33 s
Herman's protocol (merge, ring)	4.31 s
Herman's protocol (annih., ring)	4.61 s
Moran process (2 types, line)	2 m 48 s
Moran process (3 types, line)	56 m 14 s
Cell cycle switch (1 types, line)	43.94 s
Cell cycle switch (2 types, line)	9 h 46 m
Clustering (2 types, line)	10 m 30 s
Clustering (3 types, line)	T/O
Coin game ($k = 3$, clique)	1 m 0 s

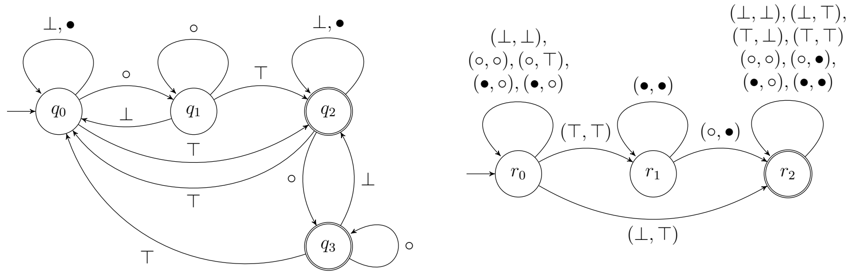
Herman's protocol. We consider two versions of *Herman's protocol* (described in more detail in Section 3.4) that differ in the way how they handle the situation when a process already holding a token receives another token—either the two tokens are merged into one, or they are both annihilated—and two topologies: a line and a ring. In our version of Herman's protocol, all processes are always enabled, and in case they do not hold a token, they can only stay in their state[8]. Fig. 1 shows an example of a synthesized solution for the model of Herman's protocol on a ring (the *annihilating* variant).

Moran process. Our second example uses variants of *Moran process*, a model of genetic drift, on a linear array [24], where the order in which individuals move is determined by the scheduler. Each node of the array is an allele, and there are two types of alleles: A and B (we can generalise this to any number $N \geq 2$ of alleles). At each round, the scheduler picks an allele. The allele will randomly either copy its type to itself or "infect" one of the neighbours (copy its type to a random neighbour). We check the fair termination property that the system eventually reaches a "drift" state, where all alleles are of the same type, under every process-fair scheduler, with probability one.

Note that the termination property is not true if we do not impose fairness: for example, when in $AABB$, an unfair scheduler can simply choose the first A all the time. Additionally, small variants of the model may not satisfy the property. Consider the variation where the chosen allele must infect one of its neighbors. For a linear array of size at least 4, a fair scheduler can play in such a way that guarantees that the system never terminates in a drift state.

Cell cycle switch. The model of *cell cycle switch* is a simplification of the model of [25] to reach a common decision of all members of a population between two possible outcomes that approximately matches the initial relative majority. We assume cells are of three types according to their decision: X, Y, and *undecided* (in the case of two types,

[8] Note that even for the line topology, this model requires fairness to verify that a configuration with a single token is reachable with probability 1. The model used in [20] did not require the fairness assumption since it modelled processes without tokens as disabled.



a) *Inv*

b) $\prec$

Fig. 1. Synthesized advice bits for Herman's protocol on a ring

we consider X and *undecided*). A decided cell (X or Y) can change the type of an undecided neighbour to its own. In addition, a decided cell can change a neighbour with an opposite decision to *undecided*. We verify that from any initial configuration, with probability 1, the system stabilizes into a configuration where all cells share a common decision.

Clustering. The *clustering* example considers a population model of alleles of 2 (resp. 3) types, say $\{A, B\}$ (resp. $\{A, B, C\}$), on a line. The alleles can change position with their neighbours of a different type, e.g. $AB \to BA$. We verify that from any configuration, the system will reach a state where the alleles form 2 (resp. 3) clusters of the same type.

Coin game. In the *coin game* use case, we consider a population protocol where every agent has one of two types of coins: *gold* or *silver*. In each step, an agent chosen by the scheduler will either keep its currency, or switch to the currency held by the majority from k randomly selected neighbours. We verify that we eventually get to a configuration where all agents have the same type of coins.

The experiments show that our encoding of fairness into systems is viable and can be used for verification of parameterized systems with fairness by their reduction to systems without fairness. On the other hand, when the size of the regular proof is large, we observe that the problem for the underlying solver gets significantly more difficult (as can be seen on the example of *clustering* for three types of alleles). We conjecture that the performance can be improved significantly by making the solver take into account the (not arbitrary) structure of the problem, which we leave for future work.

Future work. We leave the reader with several research challenges. A natural question is how to deal with non-finitary fairness for parameterized probabilistic concurrent systems in general and in the framework of regular model checking. Secondly, since there are numerous examples of population models over more complex topologies (e.g. grids), how do you deal with termination and fair termination over such models in the parameterized setting?

Acknowledgement. We thank anonymous reviewers and Dave Parker for their helpful feedback. This work was supported by the Czech Science Foundation (project 16-24707Y), the BUT FIT project FIT-S-17-4014, the IT4IXS: IT4Innovations Excellence in Science project (LQ1602), Yale-NUS Starting Grant, the European Research Council under ERC Grant Agreement no. 610150, and Swedish Research Council (2014-5484).

References

1. Lynch, N.A., Saias, I., Segala, R.: Proving time bounds for randomized distributed algorithms. In: PODC. (1994) 314–323
2. Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2) (1990) 63–67
3. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizing mutual exclusion. In: PODC. (1990) 119–131
4. Lehmann, D., Rabin, M.: On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract). In: POPL. (1981) 133–138
5. Fokkink, W.: Distributed Algorithms. MIT Press (2013)
6. Lieberman, E., Hauert, C., Nowak, M.A.: Evolutionary dynamics on graphs. Nature **433**(7023) (January 2005) 312–316
7. Courcoubetis, C., Yannakakis, M.: Minimum and maximum delay problems in real-time systems. Formal Methods in System Design **1**(4) (1992) 385–415
8. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
9. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., Qadeer, S., eds.: Proc. 23rd International Conference on Computer Aided Verification (CAV'11). Volume 6806 of LNCS., Springer (2011) 585–591
10. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS. (1985) 327–338
11. Apt, K.R., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. **22**(6) (1986) 307–309
12. Bertrand, N., Fournier, P.: Parameterized verification of many identical probabilistic timed processes. In: FSTTCS'13. Volume 24 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013) 501–513
13. Esparza, J.: Parameterized verification of crowds of anonymous processes. Dependable Software Systems Engineering **45** (2016) 59–71
14. Abdulla, P.A.: Regular model checking. STTT **14**(2) (2012) 109–118
15. Nilsson, M.: Regular Model Checking. PhD thesis, Uppsala Universitet (2005)
16. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J., Saksena, M.: Regular model checking for LTL(MSO). STTT **14**(2) (2012) 223–241
17. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large (extended abstract). In: CAV. (2003) 223–235
18. Neider, D., Jansen, N.: Regular model checking using solver technologies and automata learning. In: NFM. (2013) 16–31
19. To, A.W., Libkin, L.: Algorithmic metatheorems for decidable LTL model checking over infinite systems. In: FoSSaCS. (2010) 221–236
20. Lin, A.W., Rümmer, P.: Liveness of randomised parameterised systems under arbitrary schedulers. In: CAV'16 (2). Volume 9779 of LNCS., Springer (2016) 112–133
21. Bonnet, R., Kiefer, S., Lin, A.W.: Analysis of probabilistic basic parallel processes. In: FOSSACS. (2014) 43–57

22. Alur, R., Henzinger, T.A.: Finitary fairness. ACM Trans. Program. Lang. Syst. **20**(6) (1998) 1171–1194
23. Francez, N.: Fairness. Springer-Verlag New York, Inc., New York, NY, USA (1986)
24. Moran, P.A.: Random processes in genetics. Mathematical Proceedings of the Cambridge Philosophical Society **54**(1) (Jan 1958) 60–71
25. Cardelli, L., Csikász-Nagy, A.: The cell cycle switch computes approximate majority. Scientific Reports **2**(656) (2012)
26. Hart, S., Sharir, M., Pnueli, A.: Termination of probabilistic concurrent program. ACM Trans. Program. Lang. Syst. **5**(3) (1983) 356–380
27. Pnueli, A., Zuck, L.D.: Verification of multiprocess probabilistic protocols. Distributed Computing **1**(1) (1986) 53–72
28. Esparza, J., Gaiser, A., Kiefer, S.: Proving termination of probabilistic programs using patterns. In: CAV. (2012) 123–138
29. de Alfaro, L.: Temporal logics for the specification of performance and reliability. In: STACS 97, Symposium on Theoretical Aspects of Computer Science. Volume 1200 of Lecture Notes in Computer Science., Springer (1997) 165–176
30. Baier, C., Kwiatkowska, M.Z.: On the verification of qualitative properties of probabilistic processes under fairness constraints. Inf. Process. Lett. **66**(2) (1998) 71–79
31. Olderog, E., Apt, K.R.: Fairness in parallel programs: The transformational approach. ACM Trans. Program. Lang. Syst. **10**(3) (1988) 420–455
32. Olderog, E., Podelski, A.: Explicit fair scheduling for dynamic control. In: Concurrency, Compositionality, and Correctness, Essays in Honor of Willem-Paul de Roever. Volume 5930 of Lecture Notes in Computer Science., Springer (2010) 96–117
33. Hoenicke, J., Olderog, E., Podelski, A.: Fairness for dynamic control. In: TACAS'10. Volume 6015 of LNCS., Springer (2010) 251–265
34. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: CAV. (2013) 511–526
35. Monniaux, D.: An abstract analysis of the probabilistic termination of programs. In: SAS. Springer (2001) 111–126
36. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: POPL'15, ACM (2015) 489–501
37. Chakarov, A., Voronin, Y., Sankaranarayanan, S.: Deductive proofs of almost sure persistence and recurrence properties. In: TACAS. (2016) 260–279
38. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run-times of probabilistic programs. In: ESOP'16. Volume 9632 of LNCS., Springer (2016) 364–389
39. Grädel, E., Thomas, W., Wilke, T., eds.: Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]. Volume 2500 of Lecture Notes in Computer Science., Springer (2002)
40. Kwiatkowska, M.Z.: Model checking for probability and time: from theory to practice. In: LICS. (2003) 351
41. To, A.W.: Model Checking Infinite-State Systems: Generic and Specific Approaches. PhD thesis, LFCS, School of Informatics, University of Edinburgh (2010)
42. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: POPL'13, ACM (2013) 457–468
43. Abdulla, P.A., Chen, Y.F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: TACAS. (2010) 158–174
44. Boigelot, B., Herbreteau, F.: The power of hybrid acceleration. In: CAV. (2006) 438–451
45. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular (tree) model checking. STTT **14**(2) (2012) 167–191

46. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: acceleration from theory to practice. STTT **10**(5) (2008) 401–424
47. Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P.: Flat acceleration in symbolic model checking. In: ATVA. (2005) 474–488
48. Lin, A.W.: Accelerating tree-automatic relations. In: FSTTCS. (2012) 313–324