Monitoring Algorithmic Fairness

Thomas A. Henzinger, Mahyar Karimi, Konstantin Kueffner, and Kaushik Mallik(✉)

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria
{tah,mahyar.karimi,konstantin.kueffner,kaushik.mallik}@ist.ac.at

Abstract. Machine-learned systems are in widespread use for making decisions about humans, and it is important that they are *fair*, i.e., not biased against individuals based on sensitive attributes. We present runtime verification of algorithmic fairness for systems whose models are unknown, but are assumed to have a Markov chain structure. We introduce a specification language that can model many common algorithmic fairness properties, such as demographic parity, equal opportunity, and social burden. We build monitors that observe a long sequence of events as generated by a given system, and output, after each observation, a quantitative estimate of how fair or biased the system was on that run until that point in time. The estimate is proven to be correct modulo a variable error bound and a given confidence level, where the error bound gets tighter as the observed sequence gets longer. Our monitors are of two types, and use, respectively, frequentist and Bayesian statistical inference techniques. While the frequentist monitors compute estimates that are objectively correct with respect to the ground truth, the Bayesian monitors compute estimates that are correct subject to a given prior belief about the system's model. Using a prototype implementation, we show how we can monitor if a bank is fair in giving loans to applicants from different social backgrounds, and if a college is fair in admitting students while maintaining a reasonable financial burden on the society. Although they exhibit different theoretical complexities in certain cases, in our experiments, both frequentist and Bayesian monitors took less than a millisecond to update their verdicts after each observation.

1 Introduction

Runtime verification complements traditional static verification techniques, by offering lightweight solutions for checking properties based on a single, possibly long execution trace of a given system [8]. We present new runtime verification techniques for the problem of bias detection in decision-making software. The use of software for making critical decisions about humans is a growing trend; example areas include judiciary [13,20], policing [23,49], banking [48], etc. It is important that these software systems are unbiased towards the protected attributes

This work is supported by the European Research Council under Grant No.: ERC-2020-AdG101020093.

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13965, pp. 358–382, 2023.
https://doi.org/10.1007/978-3-031-37703-7_17

of humans, like gender, ethnicity, etc. However, they have often shown biases in their decisions in the past [20,47,55,57,58]. While there are many approaches for mitigating biases before deployment [20,47,55,57,58], recent runtime verification approaches [3,34] offer a new complementary tool to oversee *algorithmic fairness* in AI and machine-learned decision makers during deployment.

To verify algorithmic fairness at runtime, the given decision-maker is treated as a *generator* of events with an unknown model. The goal is to algorithmically design lightweight but rigorous *runtime monitors* against quantitative formal specifications. The monitors observe a long stream of events and, after each observation, output a quantitative, statistically sound estimate of how fair or biased the generator was until that point in time. While the existing approaches [3,34] considered only sequential decision making models and built monitors from the frequentist viewpoint in statistics, we allow the richer class of Markov chain models and present monitors from both the frequentist and the Bayesian statistical viewpoints.

Monitoring algorithmic fairness involves on-the-fly statistical estimations, a feature that has not been well-explored in the traditional runtime verification literature. As far as the algorithmic fairness literature is concerned, the existing works are mostly *model-based*, and either minimize decision biases of machine-learned systems at *design-time* (i.e., pre-processing) [11,41,65,66], or verify their absence at *inspection-time* (i.e., post-processing) [32]. In contrast, we verify algorithmic fairness at *runtime*, and do not require an explicit model of the generator. On one hand, the model-independence makes the monitors trustworthy, and on the other hand, it complements the existing model-based static analyses and design techniques, which are often insufficient due to partially unknown or imprecise models of systems in real-world environments.

We assume that the sequences of events generated by the generator can be modeled as sequences of states visited by a finite unknown Markov chain. This implies that the generator is well-behaved and the events follow each other according to some fixed probability distributions. Not only is this assumption satisfied by many machine-learned systems (see Sect. 1.1 for examples), it also provides just enough structure to lay the bare-bones foundations for runtime verification of algorithmic fairness properties. We emphasize that we do not require knowledge of the transition probabilities of the underlying Markov chain.

We propose a new specification language, called the Probabilistic Specification Expressions (PSEs), which can formalize a majority of the existing algorithmic fairness properties in the literature, including demographic parity [21], equal opportunity [32], disparate impact [25], etc. Let Q be the set of events. Syntactically, a PSE is a restricted arithmetic expression over the (unknown) transition probabilities of a Markov chain with the state space Q. Semantically, a PSE φ over Q is a function that maps every Markov chain M with the state space Q to a real number, and the value $\varphi(M)$ represents the degree of fairness or bias (with respect to φ) in the generator M. Our monitors observe a long sequence of events from Q, and after each observation, compute a statistically rigorous estimate of $\varphi(M)$ with a PAC-style error bound for a given confidence level. As the observed sequence gets longer, the error bound gets tighter.

Algorithmic fairness properties that are expressible using PSEs are quantitative refinements of the traditional qualitative fairness properties studied in formal methods. For example, a qualitative fairness property may require that if a certain event A occurs infinitely often, then another event B should follow infinitely often. In particular, a coin is qualitatively fair if infinitely many coin tosses contain both infinitely many heads and infinitely many tails. In contrast, the coin will be algorithmically fair (i.e., unbiased) if approximately half of the tosses come up heads. Technically, while qualitative weak and strong fairness properties are ω-regular, the algorithmic fairness properties are statistical and require counting. Moreover, for a qualitative fairness property, the satisfaction or violation cannot be established based on a finite prefix of the observed sequence. In contrast, for any given finite prefix of observations, the value of an algorithmic fairness property can be estimated using statistical techniques, assuming the future behaves statistically like the past (the Markov assumption).

As our main contribution, we present two different monitoring algorithms, using tools from frequentist and Bayesian statistics, respectively. The central idea of the *frequentist monitor* is that the probability of every transition of the monitored Markov chain M can be estimated using the fraction of times the transition is taken per visit to its source vertex. Building on this, we present a practical implementation of the frequentist monitor that can estimate the value of a given PSE from an observed finite sequence of states. For the coin example, after every new toss, the frequentist monitor will update its estimate of probability of seeing heads by computing the fraction of times the coin came up heads so far, and then by using concentration bounds to find a tight error bound for a given confidence level. On the other hand, the central idea of the *Bayesian monitor* is that we begin with a prior belief about the transition probabilities of M, and having seen a finite sequence of observations, we can obtain an updated posterior belief about M. For a given confidence level, the output of the monitor is computed by applying concentration inequalities to find a tight error bound around the mean of the posterior belief. For the coin example, the Bayesian monitor will begin with a prior belief about the degree of fairness, and, after observing the outcome of each new toss, will compute a new posterior belief. If the prior belief agrees with the true model with a high probability, then the Bayesian monitor's output converges to the true value of the PSE more quickly than the frequentist monitor. In general, both monitors can efficiently estimate more complicated PSEs, such as the ratio and the squared difference of the probabilities of heads of two different coins. The choice of the monitor for a particular application depends on whether an objective or a subjective evaluation, with respect to a given prior, is desired.

Both frequentist and Bayesian monitors use registers (and counters as a restricted class of registers) to keep counts of the relevant events and store the intermediate results. If the size of the given PSE is n, then, in theory, the frequentist monitor uses $\mathcal{O}(n^4 2^n)$ registers and computes its output in $\mathcal{O}(n^4 2^n)$ time after each new observation, whereas the Bayesian monitor uses $\mathcal{O}(n^2 2^n)$ registers and computes its output in $\mathcal{O}(n^2 2^n)$ time after each new observation.

The computation time and the required number of registers get drastically reduced to $O(n^2)$ for the frequentist monitor with PSEs that contain up to one division operator, and for the Bayesian monitor with polynomial PSEs (possibly having negative exponents in the monomials). This shows that under given circumstances, one or the other type of the monitor can be favorable computation-wise. These special, efficient cases cover many algorithmic fairness properties of interest, such as demographic parity and equal opportunity.

Our experiments confirm that our monitors are fast in practice. Using a prototype implementation in Rust, we monitored a couple of decision-making systems adapted from the literature. In particular, we monitor if a bank is fair in lending money to applicants from different demographic groups [48], and if a college is fair in admitting students without creating an unreasonable financial burden on the society [54]. In our experiments, both monitors took, on an average, less than a millisecond to update their verdicts after each observation, and only used tens of internal registers to operate, thereby demonstrating their practical usability at runtime.

In short, we advocate that runtime verification introduces a new set of tools in the area of algorithmic fairness, using which we can monitor biases of deployed AI and machine-learned systems in real-time. While existing monitoring approaches only support sequential decision making problems and use only the frequentist statistical viewpoint, we present monitors for the more general class of Markov chain system models using both frequentist and Bayesian statistical viewpoints.

All proofs can be found in the longer version of the paper [33].

1.1 Motivating Examples

We first present two real-world examples from the algorithmic fairness literature to motivate the problem; these examples will later be used to illustrate the technical developments.

The Lending Problem [48]**:** Suppose a bank lends money to individuals based on certain attributes, like credit score, age group, etc. The bank wants to maximize profit by lending money to only those who will repay the loan in time—called the "true individuals." There is a sensitive attribute (e.g., ethnicity) classifying the population into two groups g and $\overline{g}$. The bank will be considered fair (in lending money) if its lending policy is independent of an individual's membership in g or $\overline{g}$. Several *group fairness* metrics from the literature are relevant in this context. *Disparate impact* [25] quantifies the *ratio* of the probability of an individual from g getting the loan to the probability of an individual from $\overline{g}$ getting the loan, which should be close to 1 for the bank to be considered fair. *Demographic parity* [21] quantifies the *difference* between the probability of an individual from g getting the loan and the probability of an individual from $\overline{g}$ getting the loan, which should be close to 0 for the bank to be considered fair. *Equal opportunity* [32] quantifies the *difference* between the probability of a *true* individual from g getting the loan and the probability of a *true* individual from $\overline{g}$ getting the loan, which should be close to 0 for the bank to be considered fair.

A discussion on the relative merit of various different algorithmic fairness notions is out of scope of this paper, but can be found in the literature [15,22,43,62]. We show how we can monitor whether a given group fairness criteria is fulfilled by the bank, by observing a sequence of lending decisions.

The College Admission Problem [54]**:** Consider a college that announces a cutoff of grades for admitting students through an entrance examination. Based on the merit, every truly qualified student belongs to group g, and the rest to group $\overline{g}$. Knowing the cutoff, every student can choose to invest a sum of money—proportional to the gap between the cutoff and their true merit—to be able to reach the cutoff, e.g., by taking private tuition classes. On the other hand, the college's utility is in minimizing admission of students from $\overline{g}$, which can be accomplished by raising the cutoff to a level that is too expensive to be achieved by the students from $\overline{g}$ and yet easy to be achieved by the students from g. The *social burden* associated to the college's cutoff choice is the expected expense of every student from g, which should be close to 0 for the college to be considered fair (towards the society). We show how we can monitor the social burden, by observing a sequence of investment decisions made by the students from g.

1.2 Related Work

There has been a plethora of work on algorithmic fairness from the machine learning standpoint [10,12,21,32,38,42,45,46,52,59,63,66]. In general, these works improve algorithmic fairness through de-biasing the training dataset (pre-processing), or through incentivizing the learning algorithm to make fair decisions (in-processing), or through eliminating biases from the output of the machine-learned model (post-processing). All of these are interventions in the design of the system, whereas our monitors treat the system as already deployed.

Recently, formal methods-inspired techniques have been used to guarantee algorithmic fairness through the verification of a learned model [2,9,29,53,61], and enforcement of robustness [6,30,39]. All of these works verify or enforce algorithmic fairness *statically* on all runs of the system with high probability. This requires certain knowledge about the system model, which may not be always available. Our runtime monitor dynamically verifies whether the current run of an opaque system is fair.

Our frequentist monitor is closely related to the novel work of Albarghouthi et al. [3], where the authors build a programming framework that allows runtime monitoring of algorithmic fairness properties on programs. Their monitor evaluates the algorithmic fairness of repeated "single-shot" decisions made by machine-learned functions on a sequence of samples drawn from an underlying unknown but fixed distribution, which is a special case of our more general Markov chain model of the generator. They do not consider the Bayesian point of view. Moreover, we argue and empirically show in Sect. 4 that our frequentist approach produces significantly tighter statistical estimates than their approach on most PSEs. On the flip side, their specification language is more expressive, in that they allow atomic variables for expected values of events, which is useful

for specifying individual fairness criteria [21]. We only consider group fairness, and leave individual fairness as part of future research. Also, they allow logical operators (like boolean connectives) in their specification language. However, we obtain tighter statistical estimates for the core arithmetic part of algorithmic fairness properties (through PSEs), and point out that we can deal with logical operators just like they do in a straightforward manner.

Shortly after the first manuscript of this paper was written, we published a separate work for monitoring long-run fairness in sequential decision making problems, where the feature distribution of the population may dynamically change due to the actions of the individuals [34]. Although this other work generalizes our current paper in some aspects (support for dynamic changes in the model), it only allows sequential decision making models (instead of Markov chains) and does not consider the Bayesian monitoring perspective.

There is a large body of research on monitoring, though the considered properties are mainly temporal [5,7,19,24,40,50,60]. Unfortunately, these techniques do not directly extend to monitoring algorithmic fairness, since checking algorithmic fairness requires statistical methods, which is beyond the limit of finite automata-based monitors used by the classical techniques. Although there are works on quantitative monitoring that use richer types of monitors (with counters/registers like us) [28,35,36,56], the considered specifications do not easily extend to statistical properties like algorithmic fairness. One exception is the work by Ferrère et al. [26], which monitors certain statistical properties, like mode and median of a given sequence of events. Firstly, they do not consider algorithmic fairness properties. Secondly, their monitors' outputs are correct only as the length of the observed sequence approaches infinity (asymptotic guarantee), whereas our monitors' outputs are *always* correct with high confidence (finite-sample guarantee), and the precision gets better for longer sequences.

Although our work uses similar tools as used in statistical verification [1,4,14,17,64], the goals are different. In traditional statistical verification, the system's runs are chosen probabilistically, and it is verified if any run of the system satisfies a boolean property with a certain probability. For us, the run is given as input to the monitor, and it is this run that is verified against a quantitative algorithmic fairness property with statistical error bounds. To the best of our knowledge, existing works on statistical verification do not consider algorithmic fairness properties.

2 Preliminaries

For any alphabet Σ, the notation Σ^* represents the set of all finite words over Σ. We write $\mathbb{R}$, $\mathbb{N}$, and $\mathbb{N}^+$ to denote the sets of real numbers, natural numbers (including zero), and positive integers, respectively. For a pair of real (natural) numbers a, b with $a < b$, we write $[a, b]$ ($[a\,..\,b]$) to denote the set of all real (natural) numbers between and including a and b. For a given $c, r \in \mathbb{R}$, we write $[c \pm r]$ to denote the set $[c - r, c + r]$. For simpler notation, we will use $|\cdot|$ to denote both the cardinality of a set and the absolute value of a real number, whenever the intended use is clear.

For a given vector $v \in \mathbb{R}^n$ and a given $m \times n$ real matrix M, for some m, n, we write v_i to denote the i-th element of v and write M_{ij} to denote the element at the i-th row and the j-th column of M. For a given $n \in \mathbb{N}^+$, a *simplex* is the set of vectors $\Delta(n) := \{x \in [0,1]^{n+1} \mid \sum_{i=1}^{n+1} x_i = 1\}$. Notice that the dimension of $\Delta(n)$ is $n+1$ (and not n), a convention that is standard due to the interpretation of $\Delta(n)$ as the $n+1$ vertices of an n-dimensional polytope. A *stochastic matrix* of dimension $m \times m$ is a matrix whose every row is in $\Delta(m-1)$, i.e. $M \in \Delta(m-1)^m$. Random variables will be denoted using uppercase symbols from the Latin alphabet (e.g. X), while the associated outcomes will be denoted using lowercase font of the same symbol (x is an outcome of X). We will interchangeably use the expected value $\mathbb{E}(X)$ and the mean μ_X of X. For a given set S, define $\mathcal{D}(S)$ as the set of every random variable—called a *probability distribution*[1]—with set of outcomes being 2^S. A Bernoulli random variable that produces "1" (the alternative is "0") with probability p is written as $Bernoulli(p)$.

2.1 Markov Chains as Randomized Generators of Events

We use finite Markov chains as sequential randomized generators of events. A (finite) Markov chain $\mathcal{M}$ is a triple (Q, M, π), where $Q = [1\,..\,N]$ is a set of states for a finite N, $M \in \Delta(N-1)^N$ is a stochastic matrix called the transition probability matrix, and $\pi \in \mathcal{D}(Q)$ is the distribution over initial states. We often refer to a pair of states $(i, j) \in Q \times Q$ as an *edge*. The Markov chain $\mathcal{M}$ generates an infinite sequence of random variables $X_0 = \pi, X_1, \ldots$, with $X_i \in \mathcal{D}(Q)$ for every i, such that the Markov property is satisfied: $\mathbb{P}(X_{n+1} = i_{n+1} \mid X_0 = i_0, \ldots, X_n = i_n) = \mathbb{P}(X_{n+1} = i_{n+1} \mid X_n = i_n)$, which is $M_{i_n i_{n+1}}$ in our case. A finite *path* $\vec{x} = x_0, \ldots, x_n$ of $\mathcal{M}$ is a finite word over Q such that for every $t \in [0; n]$, $\mathbb{P}(X_t = x_t) > 0$. Let $Paths(\mathcal{M})$ be the set of every finite path of $\mathcal{M}$.

We use Markov chains to model the probabilistic interaction between a machine-learned decision maker with its environment. Intuitively, the Markov assumption on the model puts the restriction that the decision maker does not change over time, e.g., due to retraining.

In Fig. 1 we show the Markov chains for the lending and the college admission examples from Sect. 1.1. The Markov chain for the lending example captures the sequence of loan-related probabilistic events, namely, that a loan applicant is randomly sampled and the group information (g or $\overline{g}$) is revealed, a probabilistic decision is made by the decision-maker and either the loan was granted (gy or $\overline{g}y$, depending on the group) or refused ($\overline{y}$), and if the loan is granted then with some probabilities it either gets repaid (z) or defaulted ($\overline{z}$). The Markov chain for the college admission example captures the sequence of admission events, namely, that a candidate is randomly sampled and the group is revealed ($g, \overline{g}$), and when the candidate is from group g (truly qualified) then the amount of money invested for admission is also revealed.

[1] An alternate commonly used definition of probability distribution is directly in terms of the probability measure induced over S, instead of through the random variable.

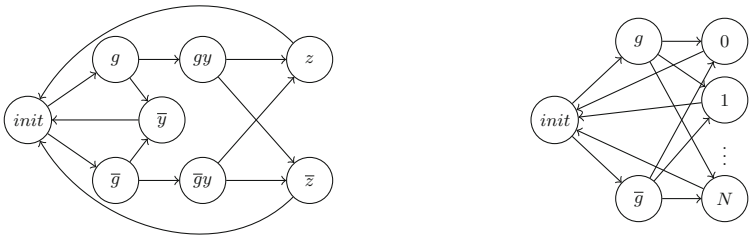


Fig. 1. Markov chains for the lending and the college-admission examples. (**left**) The lending example: The state *init* denotes the initiation of the sampling, and the rest represent the selected individual, namely, g and $\overline{g}$ denote the two groups, (gy) and $(\overline{g}y)$ denote that the individual is respectively from group g and group $\overline{g}$ and the loan was granted, $\overline{y}$ denotes that the loan was refused, and z and $\overline{z}$ denote whether the loan was repaid or not. (**right**) The college admission example: The state *init* denotes the initiation of the sampling, the states $g, \overline{g}$ represent the group identity of the selected candidate, and the states $\{0, \ldots, N\}$ represent the amount of money invested by a truly eligible candidate.

2.2 Randomized Register Monitors

Randomized register monitors, or simply monitors, are adapted from the (deterministic) polynomial monitors of Ferrère et al. [27]. Let R be a finite set of integer variables called registers. A function $v \colon R \to \mathbb{N}$ assigning concrete value to every register in R is called a valuation of R. Let $\mathbb{N}^R$ denote the set of all valuations of R. Registers can be read and written according to relations in the signature $S = \langle 0, 1, +, -, \times, \div, \leq \rangle$. We consider two basic operations on registers:

- A *test* is a conjunction of atomic formulas over S and their negation;
- An *update* is a mapping from variables to terms over S.

We use $\Phi(R)$ and $\Gamma(R)$ to respectively denote the set of tests and updates over R. *Counters* are special registers with a restricted signature $S = \langle 0, 1, +, -, \leq \rangle$.

Definition 1 (Randomized register monitor). *A randomized register monitor is a tuple $(\Sigma, \Lambda, R, \lambda, T)$ where Σ is a finite input alphabet, Λ is an output alphabet, R is a finite set of registers, $\lambda \colon \mathbb{N}^R \to \Lambda$ is an output function, and $T \colon \Sigma \times \Phi(R) \to \mathcal{D}(\Gamma(R))$ is the randomized transition function such that for every $\sigma \in \Sigma$ and for every valuation $v \in \mathbb{N}^R$, there exists a unique $\phi \in \Phi(R)$ with $v \models \phi$ and $T(\sigma, \phi) \in \mathcal{D}(\Gamma(R))$. A deterministic register monitor is a randomized register monitor for which $T(\sigma, \phi)$ is a Dirac delta distribution, if it is defined.*

A *state* of a monitor $\mathcal{A}$ is a valuation of its registers $v \in \mathbb{N}^R$. The monitor $\mathcal{A}$ *transitions* from state v to a *distribution* over states given by the random variable $Y = T(\sigma, \phi)$ on input $\sigma \in \Sigma$ if there exists ϕ such that $v \models \phi$. Let γ be an outcome of Y with $\mathbb{P}(Y = \gamma) > 0$, in which case the registers are updated as $v'(x) = v(\gamma(x))$ for every $x \in R$, and the respective concrete transition is

written as $v \xrightarrow{\sigma} v'$. A *run* of $\mathcal{A}$ on a word $w_0 \dots w_n \in \Sigma^*$ is a sequence of concrete transitions $v_0 \xrightarrow{w_0} v_1 \xrightarrow{w_1} \dots \xrightarrow{w_n} v_{n+1}$. The probabilistic transitions of $\mathcal{A}$ induce a probability distribution over the sample space of finite runs of the monitor, denoted $\widehat{\mathbb{P}}(\cdot)$. For a given finite word $w \in \Sigma^*$, the *semantics* of the monitor $\mathcal{A}$ is given by a random variable $[\![\mathcal{A}]\!](w) \coloneqq \lambda(Y)$ inducing the probability measure $\mathbb{P}_{\mathcal{A}}$, where Y is the random variable representing the distribution over the final state in a run of $\mathcal{A}$ on the word w, i.e., $\mathbb{P}_{\mathcal{A}}(Y = v) \coloneqq \widehat{\mathbb{P}}(\{r = r_0 \dots r_m \in \Sigma^* \mid r \text{ is a run of } \mathcal{A} \text{ on } w \text{ and } r_m = v\})$.

Example: A Monitor for Detecting the (Unknown) Bias of a Coin. We present a simple deterministic monitor that computes a PAC estimate of the bias of an unknown coin from a sequence of toss outcomes, where the outcomes are denoted as "h" for heads and "t" for tails. The input alphabet is the set of toss outcomes, i.e., $\Sigma = \{h, t\}$, the output alphabet is the set of every bias intervals, i.e., $\Gamma = \{[a, b] \mid 0 \leq a < b \leq 1\}$, the set of registers is $R = \{r_n, r_h\}$, where r_n and r_h are counters counting the total number of tosses and the number of heads, respectively, and the output function λ maps every valuation of r_n, r_h to an interval estimate of the bias that has the form $\lambda \equiv {}^{v(r_h)}\!/_{v(r_n)} \pm \varepsilon(r_n, \delta)$, where $\delta \in [0, 1]$ is a given upper bound on the probability of an incorrect estimate and $\varepsilon(r_n, \delta)$ is the estimation error computed using PAC analysis. For instance, after observing a sequence of 67 tosses with 36 heads, the values of the registers will be $v(r_n) = 67$ and $v(r_h) = 36$, and the output of the monitor will be $\lambda(67, 36) = {}^{36}\!/_{67} \pm \varepsilon(n, \delta)$ for some appropriate $\varepsilon(\cdot)$. Now, suppose the next input to the monitor is h, in which case the monitor's transition is given as $T(h, \cdot) = (r_n + 1, r_h + 1)$, which updates the registers to the new values $v'(r_n) = 67 + 1 = 68$ and $v'(r_h) = 36 + 1 = 37$. For this example, the tests $\Phi(R)$ over the registers are redundant, but they can be used to construct monitors for more complex properties.

3 Algorithmic Fairness Specifications and Problem Formulation

3.1 Probabilistic Specification Expressions

To formalize algorithmic fairness properties, like the ones in Sect. 1.1, we introduce *probabilistic specification expressions* (PSE). A PSE φ over a given finite set Q is an algebraic expression with some restricted set of operations that uses variables labeled v_{ij} with $i, j \in Q$ and whose domains are the real interval $[0, 1]$. The syntax of φ is:

$$\xi ::= v \in \{v_{ij}\}_{i,j \in Q} \mid \xi \cdot \xi \mid 1 \div \xi, \tag{1a}$$

$$\varphi ::= \kappa \in \mathbb{R} \mid \xi \mid \varphi + \varphi \mid \varphi - \varphi \mid \varphi \cdot \varphi \mid (\varphi), \tag{1b}$$

where $\{v_{ij}\}_{i,j \in Q}$ are the variables with domain $[0, 1]$ and κ is a constant. The expression ξ in (1a) is called a *monomial* and is simply a product of powers of variables with integer exponents. A *polynomial* is a weighted sum of monomials

with constant weights.[2] Syntactically, polynomials form a strict subclass of the expressions definable using (1b), because the product of two polynomials is not a polynomial, but is a valid expression according to (1b). A PSE φ is *division-free* if there is no division operator involved in φ. The *size* of an expression φ is the total number of arithmatic operators (i.e. $+, -, \cdot, \div$) in φ. We use V_φ to denote the set of variables appearing in the expression φ, and for every $V \subseteq V_\varphi$ we define $Dom(V) \coloneqq \{i \in Q \mid \exists v_{ij} \in V \vee \exists v_{ki} \in V\}$ as the set containing any state of the Markov chain that is involved in some variable in V.

The semantics of a PSE φ is interpreted *statically* on the unknown Markov chain M: we write $\varphi(M)$ to denote the evaluation or the value of φ by substituting every variable v_{ij} in φ with M_{ij}. E.g., for a Markov chain with state space $\{1, 2\}$ and transition probabilities $M_{11} = 0.2$, $M_{12} = 0.8$, $M_{21} = 0.4$, and $M_{22} = 0.6$, the expression $\varphi = v_{11} - v_{21}$ has the evaluation $\varphi(M) = 0.2 - 0.4 = -0.2$. We will assume that for every expression $(1 \div \xi)$, $\xi(M) \neq 0$.

Example: Group Fairness. Using PSEs, we can express the group fairness properties for the lending example described in Sect. 1.1, with the help of the Markov chain in the left subfigure of Fig. 1:

Disparate impact [25]: $v_{gy} \div v_{\overline{g}y}$

Demographic parity [21]: $v_{gy} - v_{\overline{g}y}$

The equal opportunity criterion requires the following probability to be close to zero: $p = \mathbb{P}(y \mid g, z) - \mathbb{P}(y \mid \overline{g}, z)$, which is tricky to monitor as p contains the counter-factual probabilities representing "the probability that an individual from a group would repay had the loan been granted." We apply Bayes' rule, and turn p into the following equivalent form: $p' = \frac{\mathbb{P}(z|g,y)\cdot\mathbb{P}(y|g)}{\mathbb{P}(z|g)} - \frac{\mathbb{P}(z|\overline{g},y)\cdot\mathbb{P}(y|\overline{g})}{\mathbb{P}(z|\overline{g})}$. Assuming $\mathbb{P}(z \mid g) = c_1$ and $\mathbb{P}(z \mid \overline{g}) = c_2$, where c_1 and c_2 are known constants, the property p' can be encoded as a PSE as below:

Equal opportunity [32]: $(v_{(gy)z} \cdot v_{gy}) \div c_1 - (v_{(\overline{g}y)z} \cdot v_{\overline{g}y}) \div c_2$.

Example: Social Burden. Using PSEs, we can express the social burden of the college admission example described in Sect. 1.1, with the help of the Markov chain depicted in the right subfigure of Fig. 1:

Social burden [54]: $1 \cdot v_{g1} + \ldots + N \cdot v_{gN}$.

3.2 The Monitoring Problem

Informally, our goal is to build monitors that observe a single long path of a Markov chain and, after each observation, output a new estimate for the value of the PSE. Since the monitor's estimate is based on statistics collected from

[2] Although monomials and polynomials usually only have positive exponents, we take the liberty to use the terminologies even when negative exponents are present.

a finite path, the output may be incorrect with some probability, where the source of this probability is different between the frequentist and the Bayesian approaches. In the frequentist approach, the underlying Markov chain is fixed (but unknown), and the randomness stems from the sampling of the observed path. In the Bayesian approach, the observed path is fixed, and the randomness stems from the uncertainty about a prior specifying the Markov chain's parameters. The commonality is that, in both cases, we want our monitors to estimate the value of the PSE up to an error with a fixed probabilistic confidence.

We formalize the monitoring problem separately for the two approaches. A *problem instance* is a triple (Q, φ, δ), where $Q = [1\,..\,N]$ is a set of states, φ is a PSE over Q, and $\delta \in [0, 1]$ is a constant. In the frequentist approach, we use $\mathbb{P}_s$ to denote the probability measure induced by *sampling* of paths, and in the Bayesian approach we use $\mathbb{P}_\theta$ to denote the probability measure induced by the *prior* probability density function $p_\theta \colon \Delta(n-1)^n \to \mathbb{R} \cup \{\infty\}$ over the transition matrix of the Markov chain. In both cases, the output alphabets of the monitors contain every real interval.

Problem 1 (Frequentist monitor). *Suppose (Q, φ, δ) is a problem instance given as input. Design a monitor $\mathcal{A}$ such that for every Markov chain $\mathcal{M}$ with transition probability matrix M and for every finite path $\vec{x} \in Paths(\mathcal{M})$:*

$$\mathbb{P}_{s,\mathcal{A}}\left(\varphi(M) \in [\![\mathcal{A}]\!](\vec{x})\right) \geq 1 - \delta, \tag{2}$$

where $\mathbb{P}_{s,\mathcal{A}}$ is the joint probability measure of $\mathbb{P}_s$ and $\mathbb{P}_{\mathcal{A}}$.

Problem 2 (Bayesian monitor). *Suppose (Q, φ, δ) is a problem instance and p_θ is a prior density function, both given as inputs. Design a monitor $\mathcal{A}$ such that for every Markov chain $\mathcal{M}$ with transition probability matrix M and for every finite path $\vec{x} \in Paths(\mathcal{M})$:*

$$\mathbb{P}_{\theta,\mathcal{A}}\left(\varphi(M) \in [\![\mathcal{A}]\!](\vec{x}) \mid \vec{x}\right) \geq 1 - \delta, \tag{3}$$

where $\mathbb{P}_{\theta,\mathcal{A}}$ is the joint probability measure of $\mathbb{P}_\theta$ and $\mathbb{P}_{\mathcal{A}}$.

Notice that the state space of the Markov chain and the input alphabet of the monitor are the same, and so, many times, we refer to observed states as (input) symbols, and vice versa. The estimate $[l, u] = [\![\mathcal{A}]\!](\vec{x})$ is called the $(1-\delta)\cdot 100\%$ *confidence interval* for $\varphi(M)$.[3] The radius, given by $\varepsilon = 0.5 \cdot (u - l)$, is called the *estimation error*, and the quantity $1 - \delta$ is called the *confidence*. The estimate gets more precise as the error gets smaller and the confidence gets higher.

In many situations, we are interested in a *qualitative* question of the form "is $\varphi(M) \leq c$?" for some constant c. We point out that, once the quantitative problem is solved, the qualitative questions can be answered using standard procedures by setting up a hypothesis test [44, p. 380].

[3] While in the Bayesian setting *credible intervals* would be more appropriate, we use confidence intervals due to uniformity and the relative ease of computation. To relate the two, our confidence intervals are over-approximations of credible intervals (non-unique) that are centered around the posterior mean.

4 Frequentist Monitoring

Suppose the given PSE is only a single variable $\varphi = v_{ij}$, i.e., we are monitoring the probability of going from state i to another state j. The frequentist monitor $\mathcal{A}$ for φ can be constructed in two steps: (1) empirically compute the average number of times the edge (i, j) was taken per visit to the state i on the observed path of the Markov chain, and (2) compute the $(1-\delta)\cdot 100\%$ confidence interval using statistical concentration inequalities.

Now consider a slightly more complex PSE $\varphi' = v_{ij} + v_{ik}$. One approach to monitor φ', proposed by Albarghouthi et al. [3], would be to first compute the $(1-\delta)\cdot 100\%$ confidence intervals $[l_1, u_1]$ and $[l_2, u_2]$ separately for the two constituent variables v_{ij} and v_{ik}, respectively. Then, the $(1-2\delta)\cdot 100\%$ confidence interval for φ' would be given by the sum of the two intervals $[l_1, u_1]$ and $[l_2, u_2]$, i.e., $[l_1+l_2, u_1+u_2]$; notice the drop in overall confidence due to the union bound. The drop in the confidence level and the additional error introduced by the interval arithmetic accumulate quickly for larger PSEs, making the estimate unusable. Furthermore, we lose all the advantages of having any dependence between the terms in the PSE. For instance, by observing that v_{ij} and v_{ik} correspond to the mutually exclusive transitions i to j and i to k, we know that $\varphi'(M)$ is always less than 1, a feature that will be lost if we use plain merging of individual confidence intervals for v_{ij} and v_{ik}. We overcome these issues by estimating the value of the PSE as a whole as much as possible. In Fig. 2, we demonstrate how the ratio between the estimation errors from the two approaches vary as the number of summands (i.e., n) in the PSE $\varphi = \sum_{i=1}^{n} v_{1n}$ changes; in both cases we fixed the overall δ to 0.05 (95% confidence). The ratio remains the same for different observation lengths. Our approach is always at least as accurate as their approach [3], and is significantly better for larger PSEs.

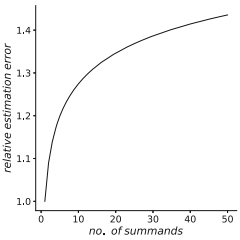


Fig. 2. Variation of ratio of the est. error using the existing approach [3] to est. error using our approach, w.r.t. the size of the chosen PSE.

4.1 The Main Principle

We first explain the idea for division-free PSEs, i.e., PSEs that do not involve any division operator; later we extend our approach to the general case.

Divison-Free PSEs: In our algorithm, for every variable $v_{ij} \in V_\varphi$, we introduce a *Bernoulli*(M_{ij}) random variable Y^{ij} with the mean M_{ij} unknown to us. We make an observation y_p^{ij} for every p-th visit to the state i on a run, and if j follows immediately afterwards then record $y_p^{ij} = 1$ else record $y_p^{ij} = 0$. This gives us a sequence of observations $\vec{y}^{ij} = y_1^{ij}, y_2^{ij}, \ldots$ corresponding to the sequence of i.i.d. random variables $\vec{Y}^{ij} = Y_1^{ij}, Y_2^{ij}, \ldots$. For instance, for the run 121123 we obtain $\vec{y}^{12} = 1, 0, 1$ for the variable v_{12}.

The heart of our algorithm is an aggregation procedure of every sequence of random variable $\{\vec{Y}^{ij}\}_{v_{ij} \in V_\varphi}$ to a single i.i.d. sequence $\vec{W}$ of an auxiliary random variable W, such that the mean of W is $\mu_W = \mathbb{E}(W) = \varphi(M)$. We can then use known concentration inequalities on the sequence $\vec{W}$ to estimate μ_W. Since μ_W exactly equals $\varphi(M)$ by design, we obtain a tight concentration bound on $\varphi(M)$. We informally explain the main idea of constructing $\vec{W}$ using simple examples; the details can be found in Algorithm 2.

Sum and Difference: Let $\varphi = v_{ij} + v_{kl}$. We simply combine $\vec{Y}^{ij}$ and $\vec{Y}^{kl}$ as $W_p = Y_p^{ij} + Y_p^{kl}$, so that $w_p = y_p^{ij} + y_p^{kl}$ is the corresponding observation of W_p. Then $\mu_{W_p} = \varphi(M)$ holds, because $\mu_{W_p} = \mathbb{E}(W_p) = \mathbb{E}(Y_p^{ij} + Y_p^{kl}) = \mathbb{E}(Y_p^{ij}) + \mathbb{E}(Y_p^{kl}) = M_{ij} + M_{kl}$. Similar approach works for $\varphi = v_{ij} - v_{kl}$.

Multiplication: For multiplications, the same linearity principle will not always work, since for random variables A and B, $\mathbb{E}(A \cdot B) = \mathbb{E}(A) \cdot \mathbb{E}(B)$ *only if* A and B are statistically independent, which will not be true for specifications of the form $\varphi = v_{ij} \cdot v_{ik}$. In this case, the respective Bernoulli random variables Y_p^{ij} and Y_p^{ik} are dependent: $\mathbb{P}(Y_p^{ij} = 1) \cdot \mathbb{P}(Y_p^{ik} = 1) = M_{ij} \cdot M_{ik}$, but $\mathbb{P}(Y_p^{ij} = 1 \wedge Y_p^{ik} = 1)$ is always 0 (since *both* j and k cannot be visited following the p-th visit to i).

To benefit from independence once again, we temporally shift one of the random variables by defining $W_p = Y_{2p}^{ij} \cdot Y_{2p+1}^{ik}$, with $w_p = y_{2p}^{ij} \cdot y_{2p+1}^{ik}$. Since the random variables Y_{2p}^{ij} and Y_{2p+1}^{ik} are independent, as they use separate visits of state i, hence we obtain $\mu_{W_p} = M_{ij} \cdot M_{ik}$. For independent multiplications of the form $\varphi = v_{ij} \cdot v_{kl}$ with $i \neq k$, we can simply use $W_p = Y_p^{ij} \cdot Y_p^{ik}$.

In general, we use the ideas of aggregation and temporal shift on the syntax tree of the PSE φ, inductively. With an aggregated sequence of observations for the auxiliary variable W for φ, we can find an estimate for $\varphi(M)$ using the Hoeffding's inequality. We present the detailed algorithm of this monitor, namely `FreqMonitorDivFree`, in Algorithm 1.

The General Case (PSEs With Division Operators): We observe that every arbitrary PSE φ of size n can be transformed into a semantically equivalent PSE of the form $\varphi_a + \frac{\varphi_b}{\varphi_c}$ of size $\mathcal{O}(n^2 2^n)$, where φ_a, φ_b, and φ_c are all division-free. Once in this form, we can employ three different `FreqMonitorDivFree` monitors from Algorithm 1 to obtain separate interval estimates for φ_a, φ_b, and φ_c, which are then combined using standard interval arithmetic and the resulting confidence of the estimate is obtained through the union bound. The steps for constructing the (general-case) `FrequentistMonitor` are shown in Algorithm 2, and the detailed analysis can be found in the proof of Theorem 1.

Bounding Memory: Consider a PSE $\varphi = v_{ij} + v_{kl}$. The outcome w_p for φ can only be computed when both the Bernoulli outcomes y_p^{ij} and y_p^{kl} are available. If at any point only one of the two is available, then we need to store the available one so that it can be used later when the other one gets available. It can be shown that the storage of "unmatched" outcomes may need unbounded memory.

To bound the memory, we use the insight that a *random reshuffling* of the i.i.d. sequence y_p^{ij} would still be i.i.d. with the same distribution, so that we do

not need to store the exact order in which the outcomes appeared. Instead, for every $v_{ij} \in V_\varphi$, we only store the number of times we have seen the state i and the edge (i, j) in counters c_i and c_{ij}, respectively. Observe that $c_i \geq \sum_{v_{ik} \in V_\varphi} c_{ik}$, where the possible difference accounts for the visits to irrelevant states, denoted as a dummy state $\top$. Given $\{c_{ik}\}_k$, whenever needed, we generate in x_i a *random reshuffling* of the sequence of states, together with $\top$, seen after the past visits to i. From the sequence stored in x_i, for every $v_{ik} \in V_\varphi$, we can consistently determine the value of y_p^{ik} (consistency dictates $y_p^{ik} = 1 \Rightarrow y_p^{ij} = 0$). Moreover, we reuse space by resetting x_i whenever the sequence stored in x_i is no longer needed. It can be shown that the size of every x_i can be at most the size of the expression [33, Proof of Thm. 2]. This random reshuffling of the observation sequences is the cause of the probabilistic transitions of the frequenitst monitor.

4.2 Implementation of the Frequentist Monitor

Fix a problem instance (Q, φ, δ), with size of φ being n. Let φ be transformed into φ^l by relabeling duplicate occurrences of v_{ij} using distinct labels $v_{ij}^1, v_{ij}^2, \ldots$. The set of labeled variables in φ^l is V_φ^l, and $|V_\varphi^l| = \mathcal{O}(n)$. Let $SubExpr(\varphi)$ denote the set of every subexpression in the expression φ, and use $[l_\varphi, u_\varphi]$ to denote the range of values the expression φ can take for every valuation of every variable as per the domain $[0, 1]$. Let $Dep(\varphi) = \{i \mid \exists v_{ij} \in V_\varphi\}$, and every subexpression $\varphi_1 \cdot \varphi_2$ with $Dep(\varphi_1) \cap Dep(\varphi_2) \neq \emptyset$ is called a *dependent multiplication*.

Implementation of `FreqMonitorDivFree` in Algorithm 1 has two main functions. *Init* initializes the registers. *Next* implements the transition function of the monitor, which attempts to compute a new observation w for $\overrightarrow{W}$ (Line 4) after observing a new input σ', and if successful it updates the output of the monitor by invoking the *UpdateEst* function. In addition to the registers in *Init* and *Next* labeled in the pseudocode, following registers are used internally:

- x_i, $i \in Dom(V_\varphi)$: reshuffled sequence of states that followed i.
- t_{ij}^l: the index of x_i that was used to obtain the latest outcome of v_{ij}^l.

Now, we summarize the main results for the frequentist monitor.

Theorem 1 (Correctness). *Let (Q, φ, δ) be a problem instance. Algorithm 2 implements a monitor for (Q, φ, δ) that solves Problem 1.*

Theorem 2 (Computational resources). *Let (Q, φ, δ) be a problem instance and $\mathcal{A}$ be the monitor implemented using the* `FrequentistMonitor` *routine of Algorithm 2. Suppose the size of φ is n. The monitor $\mathcal{A}$ requires $\mathcal{O}(n^4 2^{2n})$ registers, and takes $\mathcal{O}(n^4 2^{2n})$ time to update its output after receiving a new input*

Algorithm 1. `FreqMonitorDivFree`

Parameters: Q, φ, δ
Output: Λ

function $Init(\sigma)$
  $\varphi^l \xleftarrow{\text{unique labeling}} \varphi$
  for all $v_{ij} \in V_\varphi$ **do**
    $c_{ij} \leftarrow 0$ ▷# of (i,j)
    $c_i \leftarrow 0$ ▷# of i
  $n \leftarrow 0$ ▷length of $\vec{w}$
  $\sigma \leftarrow \sigma$ ▷prev. symbol
  $\mu_\Lambda \leftarrow \bot$ ▷est. mean
  $\varepsilon_\Lambda \leftarrow \bot$ ▷est. error
  $ResetX()$ ▷reset x_i-s
  Compute l_φ, u_φ ▷int. arith.

function $Next(\sigma')$
  $c_\sigma \leftarrow c_\sigma + 1$ ▷update counters
  $c_{\sigma\sigma'} \leftarrow c_{\sigma\sigma'} + 1$
  $w \leftarrow Eval(\varphi^l)$
  if $w \neq \bot$ **then**
    $n \leftarrow n + 1$
    $\Lambda \leftarrow UpdateEst(w, n)$
    $ResetX()$
  $\sigma \leftarrow \sigma'$
  return Λ

function $Eval(\varphi^l)$
  if $r_{\varphi^l} = \bot$ **then**
    if $\varphi^l \equiv \varphi_1^l + \varphi_2^l$ **then**
      $r_{\varphi^l} \leftarrow Eval(\varphi_1^l) + Eval(\varphi_2^l)$
    else if $\varphi^l \equiv \varphi_1^l - \varphi_2^l$ **then**
      $r_{\varphi^l} \leftarrow Eval(\varphi_1^l) - Eval(\varphi_2^l)$
    else if $\varphi^l \equiv \varphi_1^l \cdot \varphi_2^l$ **then**
      if $Dep(V_{\varphi_1}^l) \cap Dep(V_{\varphi_2}^l) = \emptyset$ **then**
        $r_{\varphi^l} \leftarrow Eval(\varphi_1^l) \cdot Eval(\varphi_2^l)$
      else ▷dep. mult.
        for $v_{ij}^l \in V_{\varphi_2}^l \cap Dep(V_{\varphi_1}^l)$ **do**
          $t_{ij}^l \leftarrow \max(\{t_{ik}^m \mid v_{ik}^m \in V_{\varphi_1}^l\})$
          $t_{ij}^l \leftarrow t_{ij}^l + 1$ ▷make indep.
        $r_{\varphi^l} \leftarrow Eval(\varphi_1^l) \cdot Eval(\varphi_2^l)$
    else if $\varphi^l \equiv v_{ij}^l$ **then**
      if $x_i[t_{ij}^l + 1] = \bot$ **then**
        $ExtractOutcome(x_i, t_{ij}^l + 1)$
      if $x_i[t_{ij}^l + 1] = j \neq \bot$ **then**
        $r_{\varphi^l} \leftarrow 1$
      else
        $r_{\varphi^l} \leftarrow 0$
    else if $\varphi^l \equiv c$ **then**
      $r_{\varphi^l} \leftarrow c$
  return r_{φ^l}

function $UpdateEst(w, n)$
  $\mu_\Lambda \leftarrow \frac{\mu_\Lambda \cdot (n-1) + w}{n}$
  $\varepsilon_\Lambda \leftarrow \sqrt{-\frac{(u_\varphi - l_\varphi)^2}{2n} \cdot \ln\left(\frac{\delta}{2}\right)}$
  return $[\mu_\Lambda \pm \varepsilon_\Lambda]$

function $ExtractOutcome(x_i, t)$
  ▷generate a shuffled sequence of symbols seen after i so that $|x_i| = t$
  Let $U \leftarrow \{j \in Q \mid v_{ij} \in V_\varphi\}$
  for $p = |x_i| + 1, \ldots, t$ **do**
    $q \leftarrow \forall u \in U$.
      pick u w/ prob. $\frac{c_{iu}}{c_i}$,
      pick $\top$ w/ prob. $\frac{(c_i - \sum_j c_{ij})}{c_i}$
    $c_i \leftarrow c_i - 1$
    if $q \neq \top$ **then**
      $c_{iq} \leftarrow c_{iq} - 1$
    $x_i[|x_i| + 1] \leftarrow q$

function $ResetX()$
  for all $i \in Dom(V_\varphi)$ **do**
    $x_i \leftarrow \emptyset$
  for all $v_{ij}^l \in V_\varphi^l$ **do**
    $t_{ij}^l \leftarrow 0$

Algorithm 2. `FrequentistMonitor`

Parameters: Q, φ, δ
Output: Λ

function $Init(\sigma)$
  $\varphi_a + \frac{\varphi_b}{\varphi_c} \xleftarrow{\text{change form}} \varphi^l \xleftarrow{\text{labeling}} \varphi$
  $\mathcal{A}_a \leftarrow$ `FreqMonitorDivFree`$(Q, \varphi_a, \delta/3)$
  $\mathcal{A}_b \leftarrow$ `FreqMonitorDivFree`$(Q, \varphi_b, \delta/3)$
  $\mathcal{A}_c \leftarrow$ `FreqMonitorDivFree`$(Q, \varphi_c, \delta/3)$
  $\mathcal{A}_a.Init(\sigma)$
  $\mathcal{A}_b.Init(\sigma)$
  $\mathcal{A}_c.Init(\sigma)$

function $Next(\sigma')$
  $[\mu_a \pm \varepsilon_a] \leftarrow \mathcal{A}_a.Next(\sigma')$
  $[\mu_b \pm \varepsilon_b] \leftarrow \mathcal{A}_b.Next(\sigma')$
  $[\mu_c \pm \varepsilon_c] \leftarrow \mathcal{A}_c.Next(\sigma')$
  if $\mu_a \neq \bot \wedge \mu_b \neq \bot \wedge \mu_c \neq \bot$ **then**
    $[\mu_\Lambda \pm \varepsilon_\Lambda] \leftarrow [\mu_a \pm \varepsilon_a] + \frac{[\mu_b \pm \varepsilon_b]}{[\mu_c \pm \varepsilon_c]}$
  return $[\mu_\Lambda \pm \varepsilon_\Lambda]$

symbol. For the special case of φ containing at most one division operator (division by constant does not count), $\mathcal{A}$ requires only $\mathcal{O}(n^2)$ registers, and takes only $\mathcal{O}(n^2)$ time to update its output after receiving a new input symbol.

There is a tradeoff between the estimation error, the confidence, and the length of the observed sequence of input symbols. For instance, for a fixed confidence, the longer the observed sequence is, the smaller is the estimation error. The following theorem establishes a lower bound on the length of the sequence for a given upper bound on the estimation error and a fixed confidence.

Theorem 3 (Convergence speed). *Let (Q, φ, δ) be a problem instance where φ does not contain any division operator, and let $\mathcal{A}$ be the monitor computed using Algorithm 2. Suppose the size of φ is n. For a given upper bound on estimation error $\bar{\varepsilon} \in \mathbb{R}$, the minimum number of visits to every state in $Dom(V_\varphi)$ for obtaining an output with error at most $\bar{\varepsilon}$ and confidence at least $1-\delta$ on any path is given by:*

$$-\frac{(u_\varphi - l_\varphi)^2 \ln\left(\frac{\delta}{2}\right) n}{2\bar{\varepsilon}^2}, \tag{4}$$

where $[l_\varphi, u_\varphi]$ is the set of possible values of φ for every valuation of every variable (having domain $[0,1]$) in φ.

The bound follows from the Hoeffding's inequality, together with the fact that every dependent multiplication increments the required number of samples by 1. A similar bound for the general case with division is left open.

5 Bayesian Monitoring

Fix a problem instance $(Q = [1\,..\,N], \varphi, \delta)$. Let $\mathbb{M} = \Delta(N-1)^N$ be the shorthand notation for the set of transition probability matrices of the Markov chains with state space Q. Let $p_\theta \colon \mathbb{M} \to [0,1]$ be the prior probability density function over $\mathbb{M}$, which is assumed to be specified using the matrix beta distribution (the definition can be found in standard textbooks on Bayesian statistics [37, pp. 280]). Let $\mathbb{1}$ be a matrix, with its size dependent on the context, whose every element is 1. We make the following common assumption [31,37, p. 50]:

Assumption 1 (Prior). *We are given a parameter matrix $\theta \geq \mathbb{1}$, and p_θ is specified using the matrix beta distribution with parameter θ. Moreover, the initial state of the Markov chain is fixed.*

When $\theta = \mathbb{k}$, then p_θ is the uniform density function over $\mathbb{M}$. After observing a path $\vec{x}$, using Bayes' rule we obtain the *posterior* density function $p_\theta(\cdot \mid \vec{x})$, which is known to be efficiently computable due to the so-called conjugacy property that holds due to Assumption 1. From the posterior density, we obtain the expected posterior semantic value of φ as: $\mathbb{E}_\theta(\varphi(M) \mid \vec{x}) := \int_{\mathbb{M}} \varphi(M) \cdot p_\theta(M \mid \vec{x}) dM$. The heart of our Bayesian monitor is an efficient incremental computation of $\mathbb{E}_\theta(\varphi(M) \mid \vec{x})$—free from numerical integration. Once we can compute $\mathbb{E}_\theta(\varphi(M) \mid \vec{x})$, we can also compute the posterior variance S^2 of $\varphi(M)$ using the known expression $S^2 = \mathbb{E}_\theta(\varphi^2(M) \mid \vec{x}) - \mathbb{E}_\theta(\varphi(M) \mid \vec{x})$, which enables us to compute a confidence interval for $\varphi(M)$ using the Chebyshev's inequality. In the following, we summarize our procedure for estimating $\mathbb{E}_\theta(\varphi(M) \mid \vec{x})$.

5.1 The Main Principle

The incremental computation of $\mathbb{E}_\theta(\varphi(M) \mid \vec{x})$ is implemented in `BayesExpMonitor`. We first transform the expression φ into the polynomial form $\varphi' = \sum_l \kappa_l \xi_l$, where $\{\kappa_l\}_l$ are the weights and $\{\xi_l\}_l$ are monomials. If the size of φ is n then the size of φ' is $\mathcal{O}(n 2^{\frac{n}{2}})$. Then we can use linearity to compute the overall expectation as the weighted sum of expectations of the individual monomials: $\mathbb{E}_\theta(\varphi(M) \mid \vec{x}) = \mathbb{E}_\theta(\varphi'(M) \mid \vec{x}) = \sum_l \kappa_l \mathbb{E}_\theta(\xi_l(M) \mid \vec{x})$. In the following, we summarize the procedure for estimating $\mathbb{E}_\theta(\xi(M) \mid \vec{x})$ for every monomial ξ.

Let ξ be a monomial, and let $\vec{x}ab \in Q^*$ be a sequence of states. We use d_{ij} to store the exponent of the variable v_{ij} in the monomial ξ, and define $d_a := \sum_{j \in [1..N]} d_{aj}$. Also, we record the sets of (i,j)-s and i-s with positive and negative d_{ij} and d_i entries: $D_i^+ := \{j \mid d_{ij} > 0\}$, $D_i^- := \{j \mid d_{ij} < 0\}$, $D^+ := \{i \mid d_i > 0\}$, and $D^- := \{i \mid d_i < 0\}$.

For any given word $\vec{w} \in Q^*$, let $c_{ij}(\vec{w})$ denote the number of ij-s in $\vec{w}$ and let $c_i(\vec{w}) := \sum_{j \in Q} c_{ij}(\vec{w})$. Define $\bar{c}_i(\vec{w}) := c_i(\vec{w}) + \sum_{j \in [1..N]} \theta_{ij}$ and $\bar{c}_{ij}(\vec{w}) := c_{ij}(\vec{w}) + \theta_{ij}$. Let $\mathcal{H} \colon Q^* \to \mathbb{R}$ be defined as:

$$\mathcal{H}(\vec{w}) := \frac{\prod_{i=1}^N \prod_{j \in D_i^+} {}^n P_{(\bar{c}_{ij}(\vec{w})-1)+|d_{ij}|} |d_{ij}|}{\prod_{i \in D^+} {}^n P_{(\bar{c}_i(\vec{w})-1)+|d_i|} |d_i|} \cdot \frac{\prod_{i \in D^-} {}^n P_{(\bar{c}_i(\vec{w})-1)} |d_i|}{\prod_{i=1}^N \prod_{j \in D_i^-} {}^n P_{(\bar{c}_{ij}(\vec{w})-1)} |d_{ij}|}, \tag{5}$$

where ${}^n P_n k := \frac{n!}{(n-k)!}$ is the number of permutations of $k > 0$ items from $n > 0$ objects, for $k \leq n$, and we use the convention that for $S = \emptyset$, $\prod_{s \in S} \ldots = 1$. Below, in Lemma 1, we establish that $\mathbb{E}_\theta(\xi(M) \mid \vec{w}) = \mathcal{H}(\vec{w})$, and present an efficient incremental scheme to compute $\mathbb{E}_\theta(\xi(M) \mid \vec{x}ab)$ from $\mathbb{E}_\theta(\xi(M) \mid \vec{x}a)$.

Lemma 1 (Incremental computation of $\mathbb{E}(\cdot \mid \cdot)$). *If the following consistency condition*

$$\forall i, j \in [1 \,..\, N] \,.\, \bar{c}_{ij}(\vec{w}) + d_{ij} > 0 \tag{6}$$

is met, then the following holds:

$$\mathbb{E}(\xi(M) \mid \vec{x}ab) = \mathcal{H}(\vec{x}ab) = \mathcal{H}(\vec{x}a) \cdot \frac{\bar{c}_{ab}(\vec{x}) + d_{ab}}{\bar{c}_{ab}(\vec{x})} \cdot \frac{\bar{c}_a(\vec{x})}{\bar{c}_a(\vec{x}) + d_a}. \tag{7}$$

Algorithm 3. `BayesExpMonitor`

Parameters: $Q, \varphi = \sum_{l=1}^{p} \kappa_l \xi_l, \theta$
Output: E

1: **function** *Init*($\sigma = 1$)
2: **for** $v_{ij} \in V_\varphi$ **do**
3: $\overline{c}_{ij} \leftarrow \theta_{ij}$
4: $\overline{c}_i \leftarrow \sum_{j \in [1..N]} \theta_{ij}$
5: $m_{ij} \leftarrow \min_{l \in [1..p]} d_{ij}^l$ ▷cache
6: active $\leftarrow$ *false* ▷Eq. 6 not true
7: $\sigma \leftarrow \sigma$ ▷prev. state
8: $E \leftarrow \bot$ ▷expect. val.

1: **function** *Next*(σ')
2: $\overline{c}_\sigma \leftarrow \overline{c}_\sigma + 1$ ▷update counters
3: $\overline{c}_{\sigma\sigma'} \leftarrow \overline{c}_{\sigma\sigma'} + 1$
4: **if** active = *false* **then**
5: **if** $(\forall v_{ij} \in V_\varphi \,.\, \overline{c}_{ij} + m_{ij} > 0)$ **then**
6: active $\leftarrow$ *true* ▷Eq. 6 is true
7: **for** $l \in [1\,..\,p]$ **do** ▷Eq. 5
8: $h^l \leftarrow \mathcal{H}^l\left(\{\overline{c}_{ij}\}_{i,j}, \{\overline{c}_i\}_i\right)$
9: **else**
10: **for** $l \in [1\,..\,p]$ **do** ▷Eq. 7
11: $h^l \leftarrow h^l \cdot \frac{\overline{c}_{\sigma\sigma'}-1+d_{\sigma\sigma'}^l}{\overline{c}_{\sigma\sigma'}-1} \cdot \frac{\overline{c}_\sigma - 1}{\overline{c}_\sigma - 1 + d_\sigma^l}$
12: **if** active = *true* **then**
13: $E \leftarrow \sum_{l=1}^{p} \kappa_l \cdot h^l$ ▷overall expect.
14: $\sigma \leftarrow \sigma'$
15: **return** E

Condition (6) guarantees that the permutations in (5) are well-defined. The first equality in (7) follows from Marchal et al. [51], and the rest uses the conjugacy of the prior. Lemma 1 forms the basis of the efficient update of our Bayesian monitor. Observe that on any given path, once (6) holds, it continues to hold forever. Thus, initially the monitor keeps updating $\mathcal{H}$ internally without outputting anything. Once (6) holds, it keeps outputting $\mathcal{H}$ from then on.

5.2 Implementation of the Bayesian Monitor

We present the Bayesian monitor implementation in `BayesConfIntMonitor` (Algorithm 4), which invokes `BayesExpMonitor` (Algorithm 3) as subroutine. `BayesExpMonitor` computes the expected semantic value of an expression φ in polynomial form, by computing the individual expected value of each monomial using Propostion 1, and combining them using the linearity property. We drop the arguments from $\overline{c}_i(\cdot)$ and $\overline{c}_{ij}(\cdot)$ and simply write $\overline{c}_i$ and $\overline{c}_{ij}$ as constants associated to appropriate words. The symbol m_{ij} in Line 5 of *Init* is used as a bookkeeping variable for quickly checking the consistency condition (Eq. 6) in Line 5 of *Next*. In `BayesConfIntMonitor`, we compute the expected value and the variance of φ, by invoking `BayesExpMonitor` on φ and φ^2 respectively, and then compute the confidence interval using the Chebyshev's inequality. It can be observed in the *Next* subroutines of `BayesConfIntMonitor` and `BayesExpMonitor` that a deterministic transition function suffices for the Bayesian monitors.

Theorem 4 (Correctness). *Let (Q, φ, δ) be a problem instance, and p_θ be given as the prior distribution which satisfies Assumption 1. Algorithm 4 produces a monitor for (Q, φ, δ) that solves Problem 2.*

Theorem 5 Computational resources). *Let (Q, φ, δ) be a problem instance and $\mathcal{A}$ be the monitor computed using the* `BayesConfIntMonitor` *routine of*

Algorithm 4. BayesConfIntMonitor

Parameters: Q, φ, θ
Output: Λ
1: **function** $Init(\sigma = 1)$
2: $\overline{\varphi} \xleftarrow{\text{polyn.}} \varphi$, $\overline{\varphi^2} \xleftarrow{\text{polyn.}} \varphi^2$ ▷polyn. form
3: $EXP \leftarrow$ `BayesExpMonitor`$(Q, \overline{\varphi}, \theta)$
4: $EXP2 \leftarrow$ `BayesExpMonitor`$(Q, \overline{\varphi^2}, \theta)$
5: $EXP.Init(\sigma)$
6: $EXP2.Init(\sigma)$
7: $\Lambda \leftarrow \bot$

1: **function** $Next(\sigma')$
2: $E \leftarrow EXP.Next(\sigma')$
3: $E2 \leftarrow EXP2.Next(\sigma')$
4: **if** $E \neq \bot$ and $E2 \neq \bot$ **then**
5: $S \leftarrow E2 - E^2$ ▷variance
6: $\Lambda \leftarrow \left[E \pm \sqrt{\frac{S}{\delta}}\right]$ ▷Chebysh.
7: **return** Λ

Algorithm 4. Suppose the size of φ is n. The monitor $\mathcal{A}$ requires $\mathcal{O}(n^2 2^n)$ registers, and takes $\mathcal{O}(n^2 2^n)$ time to update its output after receiving a new input symbol. For the special case of φ being in polynomial form, $\mathcal{A}$ requires only $\mathcal{O}(n^2)$ registers, and takes only $\mathcal{O}(n^2)$ time to update its output after receiving a new input symbol.

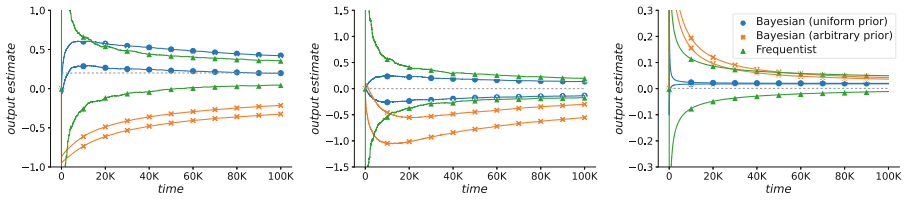
A bound on the convergence speed of the Bayesian monitor is left open. This would require a bound on the change in variance with respect to the length of the observed path, which is not known for the general case of PSEs. Note that the efficient (quadratic) cases are different for the frequentist and Bayesian monitors, suggesting the use of different monitors for different specifications.

6 Experiments

We implemented our frequentist and Bayesian monitors in a tool written in Rust, and used the tool to design monitors for the lending and the college admission examples taken from the literature [48,54] (described in Sect. 1.1). The generators are modeled as Markov chains (see Fig. 1)—unknown to the monitors—capturing the sequential interactions between the decision-makers (i.e., the bank or the college) and their respective environments (i.e., the loan applicants or the students), as described by D'Amour et al. [16]. The setup of the experiments is as follows: We created a multi-threaded wrapper program, where one thread simulates one long run of the Markov chain, and a different thread executes the monitor. Every time a new state is visited by the Markov chain on the first thread, the information gets transmitted to the monitor on the second thread, which then updates the output. The experiments were run on a Macbook Pro 2017 equipped with a 2,3 GHz Dual-Core Intel Core i5 processor and 8GB RAM. The tool can be downloaded from the following url, where we have also included the scripts to reproduce our experiments: https://github.com/ista-fairness-monitoring/fmlib.

We summarize the experimental results in Fig. 3, and, from the table, observe that both monitors are extremely lightweight: they take less than a millisecond per update and small numbers of registers to operate. From the plots, we observe that the frequentist monitors' outputs are always centered around the ground

truth values of the properties, empirically showing that they are always objectively correct. On the other hand, the Bayesian monitors' outputs can vary drastically for different choices of the prior, empirically showing that the correctness of outputs is subjective. It may be misleading that the outputs of the Bayesian monitors are wrong as they often do not contain the ground truth values. We reiterate that from the Bayesian perspective, the ground truth does not exist. Instead, we only have a probability distribution over the true values that gets updated after observing the generated sequence of events. The choice of the type of monitor ultimately depends on the application requirements.



Scenario	Size of expression	Av. comp. time/step		# registers	
		Freq.	Bayes.	Freq.	Bayes.
Lending (bias) + dem. par.	1	13.0µs	29.3µs	15	17
Lending (fair) + eq. opp.	5	21.6µs	31.0µs	29	27
Admission + soc. burden	19	53.8µs	184.6µs	46	102

Fig. 3. The plots show the 95% confidence intervals estimated by the monitors over time, averaged over 10 different sample paths, for the lending with demographic parity (left), lending with equalized opportunity (middle), and the college admission with social burden (right) problems. The horizontal dotted lines are the ground truth values of the properties, obtained by analyzing the Markov chains used to model the systems (unknown to the monitors). The table summarizes various performance metrics.

7 Conclusion

We showed how to monitor algorithmic fairness properties on a Markov chain with unknown transition probabilities. Two separate algorithms are presented, using the frequentist and the Bayesian approaches to statistics. The performances of both approaches are demonstrated, both theoretically and empirically.

Several future directions exist. Firstly, more expressive classes of properties need to be investigated to cover a broader range of algorithmic fairness criteria. We believe that boolean logical connectives, as well as min and max operators can be incorporated straightforwardly using ideas from the related literature [3]. This also adds support for absolute values, since $|x| = \max\{x, -x\}$. On the other hand, properties that require estimating how often a state is visited would require more information about the dynamics of the Markov chain, including its mixing time. Monitoring statistical hyperproperties [18] is another important direction,

which will allow us to encode individual fairness properties [21]. Secondly, more liberal assumptions on the system model will be crucial for certain practical applications. In particular, hidden Markov models, time-inhomogeneous Markov models, Markov decision processes, etc., are examples of system models with widespread use in real-world applications. Finally, better error bounds tailored for specific algorithmic fairness properties can be developed through a deeper mathematical analysis of the underlying statistics, which will sharpen the conservative bounds obtained through off-the-shelf concentration inequalities.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. (TOMACS) **28**(1), 1–39 (2018)
2. Albarghouthi, A., D'Antoni, L., Drews, S., Nori, A.V.: Fairsquare: probabilistic verification of program fairness. Proc. ACM Program. Lang. **1**(OOPSLA), 1–30 (2017)
3. Albarghouthi, A., Vinitsky, S.: Fairness-aware programming. In: Proceedings of the Conference on Fairness, Accountability, and Transparency, pp. 211–219 (2019)
4. Ashok, P., Křetínský, J., Weininger, M.: PAC statistical model checking for Markov decision processes and stochastic games. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 497–519. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_29
5. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003). https://doi.org/10.1109/TSE.2003.1205180
6. Balunovic, M., Ruoss, A., Vechev, M.: Fair normalizing flows. In: International Conference on Learning Representations (2021)
7. Bartocci, E., et al.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 135–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
8. Bartocci, E., Falcone, Y.: Lectures on Runtime Verification. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-75632-5
9. Bastani, O., Zhang, X., Solar-Lezama, A.: Probabilistic verification of fairness properties via concentration. Proc. ACM Program. Lang. **3**(OOPSLA), 1–27 (2019)
10. Bellamy, R.K., et al.: Ai fairness 360: an extensible toolkit for detecting and mitigating algorithmic bias. IBM J. Res. Dev. **63**(4/5), 4–1 (2019)
11. Berk, R., et al.: A convex framework for fair regression. arXiv preprint arXiv:1706.02409 (2017)
12. Bird, S., et al.: Fairlearn: a toolkit for assessing and improving fairness in ai. Microsoft, Technical Report. MSR-TR-2020-32 (2020)
13. Chouldechova, A.: Fair prediction with disparate impact: a study of bias in recidivism prediction instruments. Big Data **5**(2), 153–163 (2017)
14. Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 1–12. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_1
15. Corbett-Davies, S., Pierson, E., Feller, A., Goel, S., Huq, A.: Algorithmic decision making and the cost of fairness. In: Proceedings of the 23rd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining, pp. 797–806 (2017)

16. D'Amour, A., Srinivasan, H., Atwood, J., Baljekar, P., Sculley, D., Halpern, Y.: Fairness is not static: deeper understanding of long term fairness via simulation studies. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, FAT* 2020, pp. 525–534 (2020)
17. David, A., Du, D., Guldstrand Larsen, K., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 352–367. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_24
18. Dimitrova, R., Finkbeiner, B., Torfah, H.: Probabilistic hyperproperties of markov decision processes (2020). https://doi.org/10.48550/ARXIV.2005.03362, https://arxiv.org/abs/2005.03362
19. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
20. Dressel, J., Farid, H.: The accuracy, fairness, and limits of predicting recidivism. Sci. Adv. **4**(1), eaao5580 (2018)
21. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.: Fairness through awareness. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 214–226 (2012)
22. Dwork, C., Ilvento, C.: Individual fairness under composition. In: Proceedings of Fairness, Accountability, Transparency in Machine Learning (2018)
23. Ensign, D., Friedler, S.A., Neville, S., Scheidegger, C., Venkatasubramanian, S.: Runaway feedback loops in predictive policing. In: Conference on Fairness, Accountability and Transparency, pp. 160–171. PMLR (2018)
24. Faymonville, P., Finkbeiner, B., Schwenger, M., Torfah, H.: Real-time stream-based monitoring. arXiv preprint arXiv:1711.03829 (2017)
25. Feldman, M., Friedler, S.A., Moeller, J., Scheidegger, C., Venkatasubramanian, S.: Certifying and removing disparate impact. In: proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 259–268 (2015)
26. Ferrère, T., Henzinger, T.A., Kragl, B.: Monitoring event frequencies. In: Fernández, M., Muscholl, A. (eds.) 28th EACSL Annual Conference on Computer Science Logic (CSL 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 152, pp. 20:1–20:16. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl (2020). https://doi.org/10.4230/LIPIcs.CSL.2020.20, https://drops.dagstuhl.de/opus/volltexte/2020/11663
27. Ferrère, T., Henzinger, T.A., Saraç, N.E.: A theory of register monitors. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 394–403 (2018)
28. Finkbeiner, B., Sankaranarayanan, S., Sipma, H.: Collecting statistics over runtime executions. Electron. Notes Theor. Comput. Sci. **70**(4), 36–54 (2002)
29. Ghosh, B., Basu, D., Meel, K.S.: Justicia: a stochastic sat approach to formally verify fairness. arXiv preprint arXiv:2009.06516 (2020)
30. Ghosh, B., Basu, D., Meel, K.S.: Algorithmic fairness verification with graphical models. arXiv preprint arXiv:2109.09447 (2021)
31. Gómez-Corral, A., Insua, D.R., Ruggeri, F., Wiper, M.: Bayesian inference of markov processes. In: Wiley StatsRef: Statistics Reference Online, pp. 1–15 (2014)
32. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. Adv. Neural Inf. Process. Syst. **29** (2016)

33. Henzinger, T.A., Karimi, M., Kueffner, K., Mallik, K.: Monitoring algorithmic fairness. arXiv preprint arXiv:2305.15979 (2023)
34. Henzinger, T.A., Karimi, M., Kueffner, K., Mallik, K.: Runtime monitoring of dynamic fairness properties. arXiv preprint arXiv:2305.04699 (2023). to appear in FAccT '23
35. Henzinger, T.A., Saraç, N.E.: Monitorability under assumptions. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 3–18. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_1
36. Henzinger, T.A., Saraç, N.E.: Quantitative and approximate monitoring. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–14. IEEE (2021)
37. Insua, D., Ruggeri, F., Wiper, M.: Bayesian Analysis of Stochastic Process Models. John Wiley & Sons, Hoboken (2012)
38. Jagielski, M., et al.: Differentially private fair learning. In: International Conference on Machine Learning, pp. 3000–3008. PMLR (2019)
39. John, P.G., Vijaykeerthy, D., Saha, D.: Verifying individual fairness in machine learning models. In: Conference on Uncertainty in Artificial Intelligence, pp. 749–758. PMLR (2020)
40. Junges, S., Torfah, H., Seshia, S.A.: Runtime monitors for markov decision processes. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12760, pp. 553–576. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81688-9_26
41. Kamiran, F., Calders, T.: Data preprocessing techniques for classification without discrimination. Knowl. Inf. Syst. **33**(1), 1–33 (2012)
42. Kearns, M., Neel, S., Roth, A., Wu, Z.S.: Preventing fairness gerrymandering: auditing and learning for subgroup fairness. In: International Conference on Machine Learning, pp. 2564–2572. PMLR (2018)
43. Kleinberg, J., Mullainathan, S., Raghavan, M.: Inherent trade-offs in the fair determination of risk scores. In: Papadimitriou, C.H. (ed.) 8th Innovations in Theoretical Computer Science Conference (ITCS 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 67, pp. 43:1–43:23. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl (2017). https://doi.org/10.4230/LIPIcs.ITCS.2017.43, http://drops.dagstuhl.de/opus/volltexte/2017/8156
44. Knight, K.: Mathematical Statistics. CRC Press, Boca Raton (1999)
45. Konstantinov, N.H., Lampert, C.: Fairness-aware pac learning from corrupted data. J. Mach. Learn. Res. **23** (2022)
46. Kusner, M.J., Loftus, J., Russell, C., Silva, R.: Counterfactual fairness. Adv. Neural Inf. Process. Syst. **30** (2017)
47. Lahoti, P., Gummadi, K.P., Weikum, G.: ifair: learning individually fair data representations for algorithmic decision making. In: 2019 IEEE 35th International Conference on Data Engineering (icde), pp. 1334–1345. IEEE (2019)
48. Liu, L.T., Dean, S., Rolf, E., Simchowitz, M., Hardt, M.: Delayed impact of fair machine learning. In: International Conference on Machine Learning, pp. 3150–3158. PMLR (2018)
49. Lum, K., Isaac, W.: To predict and serve? Significance **13**(5), 14–19 (2016)
50. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
51. Marchal, O., Arbel, J.: On the sub-gaussianity of the beta and dirichlet distributions. Electron. Commun. Probabil. **22**, 1–14 (2017)

52. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning. ACM Comput. Surv. (CSUR) **54**(6), 1–35 (2021)
53. Meyer, A., Albarghouthi, A., D'Antoni, L.: Certifying robustness to programmable data bias in decision trees. Adv. Neural Inf. Process. Syst. **34**, 26276–26288 (2021)
54. Milli, S., Miller, J., Dragan, A.D., Hardt, M.: The social cost of strategic classification. In: Proceedings of the Conference on Fairness, Accountability, and Transparency, pp. 230–239 (2019)
55. Obermeyer, Z., Powers, B., Vogeli, C., Mullainathan, S.: Dissecting racial bias in an algorithm used to manage the health of populations. Science **366**(6464), 447–453 (2019)
56. Otop, J., Henzinger, T.A., Chatterjee, K.: Quantitative automata under probabilistic semantics. Logical Methods Comput. Sci. **15** (2019)
57. Scheuerman, M.K., Paul, J.M., Brubaker, J.R.: How computers see gender: an evaluation of gender classification in commercial facial analysis services. In: Proceedings of the ACM on Human-Computer Interaction, vol. 3, no. CSCW, pp. 1–33 (2019)
58. Seyyed-Kalantari, L., Liu, G., McDermott, M., Chen, I.Y., Ghassemi, M.: Chexclusion: fairness gaps in deep chest x-ray classifiers. In: BIOCOMPUTING 2021: Proceedings of the Pacific Symposium, pp. 232–243. World Scientific (2020)
59. Sharifi-Malvajerdi, S., Kearns, M., Roth, A.: Average individual fairness: algorithms, generalization and experiments. Adv. Neural Inf. Process. Syst. **32** (2019)
60. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15
61. Sun, B., Sun, J., Dai, T., Zhang, L.: Probabilistic verification of neural networks against group fairness. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) FM 2021. LNCS, vol. 13047, pp. 83–102. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_5
62. Wachter, S., Mittelstadt, B., Russell, C.: Bias preservation in machine learning: the legality of fairness metrics under eu non-discrimination law. W. Va. L. Rev. **123**, 735 (2020)
63. Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F., Wilson, J.: The what-if tool: Interactive probing of machine learning models. IEEE Trans. Vis. Comput. Graph. **26**(1), 56–65 (2019)
64. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_17
65. Zafar, M.B., Valera, I., Gomez-Rodriguez, M., Gummadi, K.P.: Fairness constraints: a flexible approach for fair classification. J. Mach. Learn. Res. **20**(1), 2737–2778 (2019)
66. Zemel, R., Wu, Y., Swersky, K., Pitassi, T., Dwork, C.: Learning fair representations. In: International Conference on Machine Learning, pp. 325–333. PMLR (2013)