Probabilistic Verification of Fairness Properties via Concentration

OSBERT BASTANI, University of Pennsylvania, USA
XIN ZHANG, MIT, USA
ARMANDO SOLAR-LEZAMA, MIT, USA

As machine learning systems are increasingly used to make real world legal and financial decisions, it is of paramount importance that we develop algorithms to verify that these systems do not discriminate against minorities. We design a scalable algorithm for verifying fairness specifications. Our algorithm obtains strong correctness guarantees based on adaptive concentration inequalities; such inequalities enable our algorithm to adaptively take samples until it has enough data to make a decision. We implement our algorithm in a tool called VeriFair, and show that it scales to large machine learning models, including a deep recurrent neural network that is more than five orders of magnitude larger than the largest previously-verified neural network. While our technique only gives probabilistic guarantees due to the use of random samples, we show that we can choose the probability of error to be extremely small.

CCS Concepts: • **Theory of computation → Program verification**.

Additional Key Words and Phrases: probabilistic verification, machine learning, fairness

ACM Reference Format:
Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. 2019. Probabilistic Verification of Fairness Properties via Concentration. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 118 (October 2019), 33 pages. https://doi.org/10.1145/3360544

1 INTRODUCTION

Machine learning is increasingly being used to inform sensitive decisions, including legal decisions such as whether to offer bail to a defendant [Lakkaraju et al. 2017], and financial decisions such as whether to give a loan to an applicant [Hardt et al. 2016]. In these settings, for both ethical and legal reasons, it is of paramount importance that decisions are made fairly and without discrimination [Barocas and Selbst 2016; Zarsky 2014]. Indeed, one of the motivations for introducing machine learning in these settings is the expectation that machines would not be subject to the same implicit biases that may affect human decision makers. However, designing machine learning models that satisfy fairness criterion has proven to be quite challenging, since these models have a tendency to internalize biases present in the data. Even if sensitive features such as race and gender are withheld from the model, it often internally reconstructs sensitive features.

Our goal is to verify whether a given fairness specification holds for a given machine learning model, focusing on specifications that have been proposed in the machine learning literature. In particular, our goal is *not* to devise new specifications. There has been previous work on trying to verify probabilistic specifications [Gehr et al. 2016; Sampson et al. 2014; Sankaranarayanan et al. 2013], including work specifically targeting fairness [Albarghouthi et al. 2017]. Approaches based

Authors' addresses: Osbert Bastani, University of Pennsylvania, USA, obastani@seas.upenn.edu; Xin Zhang, MIT, USA, xzhang@csail.mit.edu; Armando Solar-Lezama, MIT, USA, asolar@csail.mit.edu.

2475-1421/2019/10-ART118
https://doi.org/10.1145/3360544

on symbolic integration [Gehr et al. 2016] and numerical integration [Albarghouthi et al. 2017] have been proposed. However, these approaches can be extremely slow—indeed, previous work using numerical integration to verify fairness properties only scales to neural networks with a single hidden layer containing just three hidden units [Albarghouthi et al. 2017], whereas state-of-the-art neural networks can have dozens of layers and millions of hidden units. There has also been prior work aiming to verify probabilistic specifications using approximate techniques such as belief propagation and sampling [Sampson et al. 2014]. While these techniques are much more scalable, they typically cannot give soundness guarantees; thus, they can be useful for bug-finding, but are not suitable for verifying fairness properties, where the ability to guarantee the fairness of a given model is very important.

Our approach is to do probabilistic verification by leveraging sampling, using concentration inequalities to provide strong soundness guarantees. [1] In particular, we provide guarantees of the form:

$$\Pr[\hat{Y} = Y] \geq 1 - \Delta, \tag{1}$$

where $\hat{Y}$ is the response provided by our algorithm (i.e., whether the specification holds for the given model), and Y is the true answer. To enable such guarantees, we rely on *adaptive concentration inequalities* [Zhao et al. 2016], which are concentration inequalities where our verification algorithm can improve its estimate $\hat{Y}$ of Y until Eq. 1 holds.

We prove that our verification algorithm is both sound and precise in this high-probability sense. While in principle the probabilistic guarantee expressed by the formula is weaker than a traditional soundness guarantee, we show that our approach allows us to efficiently prove the above property with Δ very close to zero. For example, in our evaluation on a deep neural network, we take $\Delta = 10^{-10}$, meaning there is only a 10^{-10} probability that a program that our algorithm verifies to be fair is actually unfair. In contrast, the probability of winning the October 2018 Powerball was about 3 times higher (roughly 3×10^{-9}) [Picchi 2019]. To the best of our knowledge, our work is the first to use adaptive concentration inequalities to design a probabilistically sound and precise verification algorithm.

Furthermore, while our algorithm is incomplete and can fail to terminate on certain problem instances, we show that nontermination can only occur under an unlikely condition. Intuitively, nontermination can happen in cases where a specification "just barely holds"—i.e., for a random variable X, we want to show that $\mathbb{E}[X] \geq 0$, but $\mathbb{E}[X] = 0$. Then, the error in our estimate of $\mathbb{E}[X]$ will never be small enough to determine whether $\mathbb{E}[X] \geq 0$ holds. Except in these cases, we prove that our algorithm terminates in finite time with probability 1.

We implement our algorithm in a tool called VeriFair, which can be used to verify fairness properties of programs. [2] In particular, we compare VeriFair to the state-of-the-art fairness verification tool FairSquare [Albarghouthi et al. 2017]; our tool outperforms theirs on each of the 12 largest problem instances in their benchmark. Furthermore, the FairSquare benchmarks are implemented in Python; compiling their problem instances to native code can yield more than a 200× increase in the performance of VeriFair (in contrast, the running time of their tool is not increased this way, since they use symbolic techniques). Finally, we evaluate VeriFair on a much larger benchmark: we study a deep neural network used to classify human-drawn sketches of various objects [Google 2018; Ha and Eck 2017]. Our benchmark consists of neural networks with about 16 million parameters, which is more than 5 orders of magnitude larger than the largest neural network in the FairSquare benchmark, which has 37 parameters. On this benchmark,

[1] We discuss limitations of our approach in Section 7. Furthermore, while our approach is not a priori specific to fairness, there are several challenges to applying it more broadly, which we also discuss in Section 7.

[2] VeriFair is available at https://github.com/obastani/verifair.

```
def offer_job(col_rank, years_exp)
   if col_rank <= 5:
      return true
   elif years_exp > 5:
      return true
   else:
      return false
```

```
def population_model():
   is_male ~ bernoulli(0.5)
   col_rank ~ normal(25, 10)
   if is_male:
      years_exp ~ normal(15, 5)
   else:
      years_exp ~ normal(10, 5)
   return col_rank, years_exp
```

Fig. 1. Left: A classifier $f_{\text{job}} : \mathbb{R}^2 \to \{\text{true}, \text{false}\}$ for deciding whether to offer a job to a candidate (adapted from [Albarghouthi et al. 2017]). This classifier takes as input two features—the candidate's college ranking (`col_rank`), and the candidate's years of work experience (`years_exp`). Right: A population model P_{job} over the features `is_male`, `col_rank`, and `years_exp` of job candidates. Note that a candidate's years of work experience is affected by their gender.

VeriFair terminates in just 697 seconds (with probability of error $\Delta = 10^{-10}$). This result shows that VeriFair can scale to large image classification tasks, for which fairness is often an important property—for example, login systems based on face recognition have been shown to make more mistakes detecting minority users than detecting majority users [Simon 2009]. In summary, our contributions are

- We propose an algorithm for verifying fairness properties of machine learning models based on adaptive concentration inequalities (Section 4).
- We prove that our algorithm is sound and precise in a high-probability sense, and guarantee termination except in certain pathelogical cases—most importantly, fairness does not "just barely" hold (Section 5).
- We implement our algorithm in a tool called VeriFair. We show that VeriFair substantially outperforms the state-of-the-art fairness verifier FairSquare, and can furthermore scale to problem instances more than $10^6\times$ larger than FairSquare (Section 6).

2 MOTIVATING EXAMPLE

Consider the simple classifier f_{job} shown on the left-hand side of Figure 1 (adapted from [Albarghouthi et al. 2017]). This classifier predicts whether a given candidate should be offered a job based on two features: the ranking of the college they attended and their number of years of work experience. For both legal and ethical reasons, we may want to ensure that f_{job} does not discriminate against minorities. There are a number of ways to formalize nondiscrimination. In this section, we show how our techniques can be applied to checking a fairness specification called *demographic parity* [Calders et al. 2009]; we discuss additional fairness specifications of interest in Section 3.2. Demographic parity is based on legal guideline for avoiding hiring discrimination is the "80% rule" [Biddle 2006]. This rule says that the rate at which minority candidates are offered jobs should be at least 80% of the rate at which majority candidates are offered jobs:

$$Y_{\text{job}} \equiv \left(\frac{\mu_{\text{female}}}{\mu_{\text{male}}} \geq 0.8 \right),$$

where

$$\mu_{\text{male}} = \Pr[\text{offer} = 1 \mid \text{gender} = \text{male}]$$
$$\mu_{\text{female}} = \Pr[\text{offer} = 1 \mid \text{gender} = \text{female}].$$

Then, f_{job} satisfies demographic parity if $Y_{\text{job}} = \text{true}$.

Note that the demographic parity specification assumes given a distribution P of features for job candidates, which we call a *population model* [Albarghouthi et al. 2017], since μ_{male} and μ_{female} are conditional expectations over this distribution. In general, a population model is specified as a probabilistic program that takes no inputs, and returns the features (i.e., college ranking and years of work experience) for a randomly sampled member of that population. For example, on the right-hand side of Figure 1, we show a population model P_{job} over job candidates. We refer to $P_{\text{job}} \mid \text{gender} = \text{male}$ as the *majority subpopulation*, and $P_{\text{job}} \mid \text{gender} = \text{female}$ as the *minority subpopulation*. In this example, male candidates have more years of experience on average than female candidates, but they have the same college ranking on average. We discuss how population models can be obtained in Section 7.

Given classifier f_{job}, demographic parity specification Y_{job} with population model P_{job}, and a desired confidence level $\Delta \in \mathbb{R}_+$, the goal of our verification algorithm is to check whether Y_{job} holds. In particular, our algorithm estimates the fairness of f by iteratively sampling random values

$$V_{a,1}, ..., V_{a,n} \sim P_{\text{job}} \mid \text{gender} = a$$

for each $a \in \{\text{male}, \text{female}\}$, and then using these samples to estimate μ_{male} and μ_{female}:

$$\hat{\mu}_a = \frac{1}{n} \sum_{i=1}^{n} f(V_{a,i}).$$

Then, our algorithm uses $\hat{\mu}_{\text{male}}$ and $\hat{\mu}_{\text{female}}$ to estimate Y_{job}:

$$\hat{Y}_{\text{job}} \equiv \left(\frac{\hat{\mu}_{\text{female}}}{\hat{\mu}_{\text{male}}} \geq 0.8 \right).$$

Note that $\hat{Y}_{\text{job}}$ is easy to compute; the difficulty is bounding the probability of error, namely, $\gamma = \Pr[\hat{Y}_{\text{job}} \neq Y_{\text{job}}] \in \mathbb{R}_+$. In particular, our estimates $\hat{\mu}_{\text{male}}$ and $\hat{\mu}_{\text{female}}$ may have errors; thus, $\hat{Y}_{\text{job}}$ may differ from the true value Y_{job}. It is well known that $\gamma \to 0$ as the number of samples n goes to infinity; thus, while we can never guarantee that fairness holds, we can do so with arbitrarily high confidence. In particular, for any $\Delta \in \mathbb{R}_+$, our algorithm returns $\hat{Y}_{\text{job}}$ satisfying

$$\Pr[\hat{Y}_{\text{job}} = Y_{\text{job}}] \geq 1 - \Delta. \tag{2}$$

The key challenge is establishing finite sample bounds on γ, and furthermore, doing so in an adaptive way so it can collect as much data as needed to ensure that Eq. 2 holds (i.e., $\gamma \leq \Delta$). In particular, there are two key techniques our algorithm uses to establish Eq. 2. First, our algorithm uses an *adaptive concentration inequality* (from [Zhao et al. 2016]) to establish lemmas on the error of the estimates $\hat{\mu}_{\text{male}}$ and $\hat{\mu}_{\text{female}}$, e.g.,

$$\Pr[|\hat{\mu}_a - \mu_a| \leq \varepsilon] \geq 1 - \delta_a \tag{3}$$

for $a \in \{\text{male}, \text{female}\}$. Standard concentration inequalities can only establish bounds of the form Eq. 3 for a fixed number of samples n. However, our algorithm cannot a priori know how many samples it needs to establish Eq. 2; instead, it adaptively takes new samples until it determines that Eq. 2 holds. To enable this approach, we use adaptive concentration inequalities, which we describe in Section 4.2.

Second, it uses the lemmas in Eq. 3 to derive a bound

$$\Pr[\hat{Y}_{\text{job}} = Y_{\text{job}}] \geq 1 - \gamma.$$

We describe how our algorithm does so in Section 4.3.

Finally, our algorithm terminates once $\gamma \leq \Delta$, at which point we guarantee that the estimate $\hat{Y}_{\text{job}}$ satisfies Eq. 2, i.e., our algorithm accurately outputs whether fairness holds with high probability. In

$$
\begin{aligned}
T ::= {} & \mu_Z \mid \ldots \\
& \mid c \mid \ldots \\
& \mid T + T \\
& \mid -T \\
& \mid T \cdot T \\
& \mid T^{-1} \\
S ::= {} & T \geq 0 \\
& \mid S \wedge S \\
& \mid S \vee S \\
& \mid \neg S.
\end{aligned}
\qquad
\begin{aligned}
[\![\mu_Z]\!] &= \mu_Z \\
[\![c]\!] &= c \\
[\![X + X']\!] &= [\![X]\!] + [\![X']\!] \\
[\![-X]\!] &= -[\![X]\!] \\
[\![X \cdot X']\!] &= [\![X]\!] \cdot [\![X']\!] \\
[\![X^{-1}]\!] &= [\![X]\!]^{-1} \\
[\![X \geq 0]\!] &= \mathbb{I}[[\![X]\!] \geq 0] \\
[\![Y \wedge Y']\!] &= [\![Y]\!] \wedge [\![Y']\!] \\
[\![Y \vee Y']\!] &= [\![Y]\!] \vee [\![Y']\!] \\
[\![\neg Y]\!] &= \neg [\![Y]\!].
\end{aligned}
$$

Fig. 2. Left: Specification syntax. Here, S and T are nonterminal symbols (with S being the start symbol), and the remaining symbols are terminals. The terminal symbols $\mu_Z, \ldots$ represent the respective means of given Bernoulli random variables $Z, \ldots$. In our setting, $Z, \ldots$ typically encode the distribution of some statistic (e.g., rate of positive decisions) of f for some subpopulation. The terminal symbols $c, \ldots \in \mathbb{R}$ represent real-valued constants. Right: Specification semantics. Here, $X \in \mathcal{L}(T)$ and $Y \in \mathcal{L}(S)$ (where $\mathcal{L}(A)$ is the context-free language generated by A). The indicator function $\mathbb{I}[C]$ returns true if C holds and false otherwise.

particular, our algorithm is sound and precise in a probabilistic sense. Furthermore, our algorithm terminates with probability 1 unless the problem instance is pathelogical in one of two ways: (i) $\mu_{\text{male}} = 0$ (so Y_{job} contains a division by zero), or (ii) fairness "just barely" holds, i.e., $\frac{\mu_{\text{female}}}{\mu_{\text{male}}} = 0.8$. In our evaluation, we show that even for $\Delta = 10^{-10}$, our algorithm terminates quickly on a deep neural network benchmark—i.e., we can feasibly require that our algorithm make a mistake with probability at most 10^{-10}.

3 PROBLEM FORMULATION

We formalize the fairness properties that our algorithm can verify; our formulation is based on previous work [Albarghouthi et al. 2017].

3.1 Verification Algorithm Inputs

Classification program. Our goal is to verify fairness properties for a deterministic program $f : \mathcal{V} \to \{0, 1\}$ that maps given members of a population $\mathcal{V}$ (e.g., job applicants) to a single binary output $r \in \mathcal{R} = \{0, 1\}$ (e.g., whether to offer the applicant a job). For example, f may be a machine learning classifier such as a neural network. Note that f may use parameters learned from training data; in this case, our verification algorithm operates on the output of the training algorithm. Our verification algorithm only requires blackbox access to f, i.e., for any chosen input $v \in \mathcal{V}$, it can execute f on v to obtain the corresponding output $r = f(v)$.

Population model. We assume we are given a probability distribution $P_{\mathcal{V}}$ over $\mathcal{V}$, which we refer to as the *population model*, encoded as a probabilistic program that takes no inputs and construct a random member $V \sim P_{\mathcal{V}}$ of the population. Furthermore, we assume that our algorithm can sample conditional distributions $P_{\mathcal{V}} \mid C$, for some logical predicate C over $\mathcal{V}$ (i.e., $C : \mathcal{V} \to \{\text{true}, \text{false}\}$). For example, assuming $\mathcal{V}$ is discrete, our algorithm can do so using rejection sampling—we randomly sample $V \sim P_{\mathcal{V}}$ until $C(V) = \text{true}$, and return this V. The predicate C is dependent on the fairness property that we are trying to prove; in our evaluation, we show that for

the fairness properties that we study, the necessary predicates have sufficiently large support that rejection sampling is reasonably efficient.

Specification language. The syntax and semantics of the specifications that we aim to verify are shown in Figure 2. The start symbol of the grammar is S. In this grammar, the symbol μ_Z (where Z is a Bernoulli random variable) represents the expected value of Z, and $c \in \mathbb{R}$ is a numerical constant. The remainder of this grammar enables us to construct arithmetic expressions of the expectated values μ_Z and the constants c. Intuitively, this specification language enables us to encode arithmetic relationships between various conditional expectations that should hold. The advantage of introducing a specification language is that we can flexibly verify a wide range of fairness specifications in the same framework. As we show in Section 3.2, a number of fairness specifications that have been proposed in the literature can be expressed in our specification language.

3.2 Fairness Specifications

Next, we describe how three fairness specifications from the machine learning literature can be formalized in our specification language; the best fairness specification to use is often context specific. We discuss additional specifications that can be represented in our language in Section 7. We first establish some notation. For any probability distribution $P_{\mathcal{Z}}$ over a space $\mathcal{Z}$ with corresponding random variable $Z \sim P_{\mathcal{Z}}$, we let $\mu_Z = \mathbb{E}_{Z \sim P_{\mathcal{Z}}}[Z]$ denote the expectation of Z. Recall that for a Bernoulli random variable $Z \sim P_{\mathcal{Z}}$, we have $\mu_Z = \Pr_{Z \sim P_{\mathcal{Z}}}[Z = 1]$.

Demographic parity. Intuitively, our first property says that minority members should be classified as $f(V) = 1$ at approximately the same rate as majority candidates [Calders et al. 2009].

Definition 3.1. *Let*

$$V_{maj} \sim P_{\mathcal{V}} \mid A = maj$$
$$V_{min} \sim P_{\mathcal{V}} \mid A = min$$

*be conditional random variables for members of the majority and minority subpopulations, respectively. Let $R_{maj} = f(V_{maj})$ and $R_{min} = f(V_{min})$ be the Bernoulli random variables denoting whether the classifier f offers a favorable outcome to a member of the majority and minority subpopulation, respectively. Given $c \in [0, 1]$, the **demographic parity** property is*

$$Y_{parity} \equiv \left(\frac{\mu_{R_{min}}}{\mu_{R_{maj}}} \geq 1 - c \right).$$

In our example of hiring, the majority subpopulation is $P_{\text{job}} \mid \text{gender} = \text{male}$, the minority subpopulation is $P_{\text{job}} \mid \text{gender} = \text{female}$, and the classifier $f_{\text{job}} : \mathbb{R}^2 \to \{0, 1\}$ determines whether a candidate with the given years of experience and college rank is offered a job. Then, demographic parity says that for every male candidate offered a job, at least $1 - c$ female candidates should be offered a job.

Equal opportunity. Intuitively, our second property says that *qualified* members of the minority subpopulation should be classified as $f(V) = 1$ at roughly the same rate as qualified members of the majority subpopulation [Hardt et al. 2016].

Definition 3.2. *Let $q \in Q = \{qual, unqual\}$ indicate whether the candidate is qualified, and let*

$$V_{maj} \sim P_{\mathcal{V}} \mid A = maj,\ Q = qual$$
$$V_{min} \sim P_{\mathcal{V}} \mid A = min,\ Q = qual$$

be conditional random variables over $\mathcal{V}$ representing qualified members of the majority and minority subpopulations, respectively. Let $R_{maj} = f(V_{maj})$ and $R_{min} = f(V_{min})$ denote whether candidates V_{maj} and V_{min} are offered jobs according to f, respectively. Then, the ***equal opportunity*** *property is*

$$Y_{equal} \equiv \left(\frac{\mu_{R_{min}}}{\mu_{R_{maj}}} \geq 1 - c \right)$$

for a given constant $c \in [0, 1]$.

Continuing our example, this property says that for every job offered to a *qualified* male candidate, at least $1 - c$ *qualified* female candidates should be offered a job as well.

Path-specific causal fairness. Intuitively, our third property says that the outcome (e.g., job offer) should not depend directly on a sensitive variable (e.g., gender), but may depend indirectly on the sensitive covariate through other *mediator covariates* deemed directly relevant to predicting job performance (e.g., college degree) [Nabi and Shpitser 2018]. For simplicity, we assume that the mediator covariate $\mathcal{M} = \{0, 1\}$ is binary, that we are given a distribution $P_{\mathcal{M}}$ over $\mathcal{M}$, and that the classifier $f : \mathcal{V} \times \mathcal{M} \to \{0, 1\}$ is extended to be a function of $\mathcal{M}$.

Definition 3.3. *Let*

$$\begin{aligned} V_{maj} &\sim P_{\mathcal{V}} \mid A = maj \\ M_{maj} &\sim P_{\mathcal{M}} \mid A = maj,\ V = V_{maj} \\ R_{maj} &= f(V_{maj}, M_{maj}) \end{aligned}$$

be how a member of the majority subpopulation is classified by f, and let

$$\begin{aligned} V_{min} &\sim P_{\mathcal{V}} \mid A = min \\ M_{min} &\sim P_{\mathcal{M}} \mid A = maj,\ V = V_{min} \\ R_{min} &= f(V_{min}, M_{min}) \end{aligned}$$

be how a member of the minority subpopulation is classified by f, except that their mediator covariate M is drawn as if they were a member of the majority subpopulation. Given $c \in [0, 1]$, the ***path-specific causal fairness*** *property is*

$$Y_{causal} \equiv (\mu_{R_{min}} - \mu_{R_{maj}} \geq -c).$$

The key insight in this specification is how we sample the mediator variable M_{min} for a member V_{min} of the minority population. In particular, we sample M_{min} conditioned on the characteristics V_{min}, except that we change the sensitive attribute to $A = \text{maj}$ instead of $A = \text{min}$. Intuitively, M_{min} is the value of the mediator variable if V_{min} were instead a member of the majority population, but everything else about them stays the same. In our example, suppose that we have a mediator covariate college (either yes or no) and a non-mediator covariate `years_exp`. Then, the path-specific causal fairness property says that a female candidate should be given a job offer with similar probability as a male candidate—except she went to college as if she were a male candidate (but everything else about her—i.e., her years of job experience—stays the same). Thus, this specification measures the effect of gender on job offer, but ignoring the effect of gender on whether they went to college.

4 VERIFICATION ALGORITHM

We now describe our verification algorithm.

Algorithm 1 Algorithm for verifying the given specification $Y \in \mathcal{L}(S)$. The quantity $\varepsilon(\delta_Z, n)$ is defined in Eq. 10. The rules for checking $\Gamma \vdash Y : (I, \gamma)$, for $I \in \{\text{true}, \text{false}\}$, are shown in Figure 3.

```
procedure VERIFY(P_𝒵, Y, Δ)
    s ← 0
    n ← 0
    while true do
        Z ~ P_𝒵
        s ← s + Z
        n ← n + 1
        δ_Z ← Δ/⟦Y⟧_δ
        ε_Z ← ε(δ_Z, n)
        Γ ← {μ_Z : (s/n, ε_Z, δ_Z)}
        if Γ ⊢ Y : (true, γ) and γ ≤ Δ then
            return true
        else if Γ ⊢ Y : (false, γ) and γ ≤ Δ then
            return false
```

4.1 High-Level Algorithm

The intuition behind our algorithm is that for a Bernoulli random variable Z with distribution $P_{\mathcal{Z}}$, we can use a fixed number of random samples $Z_1, ..., Z_n \sim P_{\mathcal{Z}}$ to estimate μ_Z:

$$\hat{\mu}_Z = \frac{1}{n}\sum_{i=1}^{n} Z_i. \tag{4}$$

Note that no matter how many samples we take, there may always be some error ε between our estimate $\hat{\mu}_Z$ and the true expected value μ_Z. Our algorithm uses adaptive concentration inequalities to prove high-probability bounds on this error. Then, it uses these bounds to establish high-probability bounds on the output of our algorithm—i.e., whether the fairness specification holds. We describe each of these components in more detail in the remainder of this section.

Adaptive concentration inequalities. We can use concentration inequalities to establish high-probability bounds on the error $|\hat{\mu}_Z - \mu_Z|$ of our estimate $\hat{\mu}_Z$ of μ_Z of the form

$$\Pr_{Z_1,...,Z_n \sim P_{\mathcal{Z}}}[|\hat{\mu}_Z - \mu_Z| \leq \varepsilon] \geq 1 - \delta. \tag{5}$$

Note that the probability is taken over the (independent) random samples $Z_1, ..., Z_n \sim P_{\mathcal{Z}}$ used in the estimate $\hat{\mu}_Z$; when there is no ambiguity, we omit this notation.

Our algorithm uses adaptive concentration inequalities to establish bounds of the form Eq. 5. In particular, they enable the algorithm to continue to take samples to improve its estimate $\hat{\mu}_Z$. Once our algorithm terminates, the adaptive concentration inequality guarantees that a bound of the form Eq. 5 holds (for a given $\delta \in \mathbb{R}_+$; then, ε is a function of δ specified by the inequality). We describe the adaptive concentration inequalities we use in Section 4.2.

Concentration for expressions. Next, consider an expression $X \in \mathcal{L}(T)$. We can use substitute $\hat{\mu}_Z$ for μ_Z in X to obtain an estimate E for $[\![X]\!]$. Then, given that Eq. 5 holds, we show how to derive high-probability bounds of the form

$$\Pr[|E - [\![X]\!]| \leq \varepsilon] \geq 1 - \delta. \tag{6}$$

We use the notation $X : (E, \varepsilon, \delta)$ to denote that Eq. 6 holds; we call this relationship a *lemma*. Similarly, for $Y \in \mathcal{L}(S)$, we can substitute $\hat{\mu}_Z$ for μ_Z in Y to obtain an estimate I for $[\![Y]\!]$, and derive

high-probability bounds of the form

$$\Pr[I = [\![Y]\!]] \geq 1 - \gamma. \tag{7}$$

Unlike Eq. 6, we can establish that I exactly equals $[\![Y]\!]$ with high probability; this difference arises because $[\![Y]\!] \in \{\text{true}, \text{false}\}$ are discrete values, whereas $[\![X]\!] \in \mathbb{R}$ are continuous values. We describe inference rules used to derive these lemmas $X : (E, \varepsilon, \delta)$ and $Y : (I, \gamma)$ in Section 4.3.

Verification algorithm. Given a classifier $f : \mathcal{V} \to \{0, 1\}$, a population model $P_{\mathcal{V}}$, a specification $Y \in \mathcal{L}(S)$, and a confidence level $\Delta \in \mathbb{R}_+$, our goal is determine whether Y is true with probability at least $1 - \Delta$. For simplicity, we assume that Y only has a single subexpression of the form μ_Z (where Z is a Bernoulli random variable with distribution $P_{\mathcal{Z}}$); it is straightforward to generalize to the case where Y contains multiple such subexpressions. At a high level, our algorithm iteratively computes more and more accurate estimates $\hat{\mu}_Z$ of μ_Z until $\hat{\mu}_Z$ is sufficiently accurate such that it can be used to compute an estimate I of $[\![Y]\!]$ satisfying Eq. 7. In particular, on the nth iteration, our algorithm performs these steps:

(1) Draw a random sample $Z_n \sim P_{\mathcal{Z}}$, and update its estimate $\hat{\mu}_Z$ of μ_Z according to Eq. 4.
(2) Establish a lemma $\mu_Z : (\hat{\mu}_Z, \varepsilon_Z, \delta_Z)$ using the adaptive concentration inequality (for a chosen value of δ_Z).
(3) Use the inferences rules to derive a lemma $Y : (I, \gamma)$ from the lemma in the previous step.
(4) Terminate if $\gamma \leq \Delta$; otherwise, continue.

The full algorithm is shown in Algorithm 1. In the body of the algorithm, s is a running sum of the n samples $Z_1, \ldots, Z_n \sim P_{\mathcal{Z}}$ taken so far, so $\hat{\mu}_Z = \frac{s}{n}$. The variables δ_Z and ε_Z come from our adaptive concentration inequality, described in Section 4.2. Furthermore, δ_Z is chosen to be sufficiently small such that we can compute an estimate I of $[\![Y]\!]$ with the desired confidence level Δ, as we describe in Section 4.4.

4.2 Adaptive Concentration Inequalities

Concentration inequalities can be used to establish bounds of the form Eq. 5. For example, Hoeffding's inequality says Eq. 5 holds for $\delta = 2e^{-2n\varepsilon^2}$ (equivalently, $\varepsilon = \sqrt{\frac{1}{2n} \log \frac{2}{\delta}}$) [Hoeffding 1963]:

$$\Pr_{Z_1, \ldots, Z_n \sim P_{\mathcal{Z}}}[|\hat{\mu}_Z - \mu_Z| \leq \varepsilon] \geq 1 - 2e^{-2n\varepsilon^2}. \tag{8}$$

Then, for any $\varepsilon, \delta \in \mathbb{R}_+$, we can establish Eq. 5 by taking n sufficiently large—in particular, because $2e^{-2n\varepsilon^2} \to 0$ as $n \to \infty$, so for sufficiently large n, we have $\delta \leq 2e^{-2n\varepsilon^2}$.

A priori, we cannot know how large n must be, since we do not know how small ε must be for us to be able to prove or disprove the fairness specification. For example, for a specification of form $Y \equiv (\mu_Z \geq d)$, if μ_Z is very close to d, then we need ε to be very small to ensure that our estimate $\hat{\mu}_Z$ is close to μ_Z. For example, consider the two conditions

$$\begin{aligned} C_0 &: \hat{\mu}_Z - d - \varepsilon \geq 0 \\ C_1 &: \hat{\mu}_Z - d + \varepsilon < 0 \end{aligned} \tag{9}$$

If C_0 holds, then together with the fact that $|\hat{\mu}_Z - \mu_Z| \leq \varepsilon$, we can conclude that

$$\mu_Z \geq \hat{\mu}_Z - \varepsilon \geq d,$$

Similarly, if C_1 holds, then we can conlude that

$$\mu_Z \leq \hat{\mu}_Z + \varepsilon < d,$$

However, a prior, we do not know $\hat{\mu}_Z$, so we cannot directly use these conditions to determine how small to take ε. Instead, our algorithm iteratively samples more and more points so ε becomes smaller and smaller (for fixed δ) until one of the two conditions C_0 and C_1 in Eq. 9 holds.

To implement this strategy, we have to account for multiple hypothesis testing. In particular, we need to establish a series of bounds for the estimates $\hat{\mu}_Z^{(0)}$, $\hat{\mu}_Z^{(1)}$, ... of μ_Z on successive iterations of our algorithm. For simplicitly, suppose that we apply Eq. 8 to two estimates $\hat{\mu}_Z^{(0)}$ and $\hat{\mu}_Z^{(1)}$ of μ_Z:

$$\Pr[|\hat{\mu}_Z^{(0)} - \mu_Z| \leq \varepsilon] \geq 1 - \delta$$
$$\Pr[|\hat{\mu}_Z^{(1)} - \mu_Z| \leq \varepsilon] \geq 1 - \delta,$$

where $\delta = 2e^{-2n\varepsilon^2}$. The problem is that while we have established that each of the two events $|\hat{\mu}_Z^{(0)} - \mu_Z| \leq \varepsilon$ and $|\hat{\mu}_Z^{(1)} - \mu_Z| \leq \varepsilon$ occur with high probability $1 - \delta$, we need for *both* of these events to hold with high probability. One way we can do so is to take a union bound, in which case we get

$$\Pr[|\hat{\mu}_Z^{(0)} - \mu_Z| \leq \varepsilon \wedge |\hat{\mu}_Z^{(1)} - \mu_Z| \leq \varepsilon] \geq 1 - 2\delta.$$

Rather than building off of Hoeffding's inequality, our algorithm uses *adaptive* concentration inequalities, which naturally account for multiple hypothesis testing. In particular, they enable our algorithm to continue to take samples to improve its estimate $\hat{\mu}_Z$. Upon termination, our algorithm has obtained J samples $Z_i \sim P_{\mathcal{Z}}$. Note that J is a random variable, since it depends on the previously taken samples Z_i, which our algorithm uses to decide when to terminate. Then, an adaptive concentration inequality guarantees that a bound of the form Eq. 5 holds, where J is substituted for n and ε is specified by the bound. In particular, we use the following adaptive concentration inequality based on [Zhao et al. 2016].

Theorem 4.1. *Given a Bernoulli random variable Z with distribution $P_{\mathcal{Z}}$, let $\{Z_i \sim P_{\mathcal{Z}}\}_{i \in \mathbb{N}}$ be i.i.d. samples of Z, let*

$$\hat{\mu}_Z^{(n)} = \frac{1}{n} \sum_{i=1}^{n} Z_i,$$

let J be a random variable on $\mathbb{N} \cup \{\infty\}$ such that $Pr[J < \infty] = 1$, and let

$$\varepsilon(\delta, n) = \sqrt{\frac{\frac{3}{5} \cdot \log(\log_{11/10} n + 1) + \frac{5}{9} \cdot \log(24/\delta)}{n}}. \tag{10}$$

Then, given any $\delta \in \mathbb{R}_+$, we have

$$Pr[|\hat{\mu}_Z^{(J)} - \mu_Z| \leq \varepsilon(\delta, J)] \geq 1 - \delta.$$

We give a proof in Appendix A.1.

4.3 Concentration for Specifications

Now, we describe how our algorithm derives estimates E for $[\![X]\!]$ (where $X \in \mathcal{L}(T)$) and estimates I for $[\![Y]\!]$ (where $Y \in \mathcal{L}(S)$), as well as high-probability bounds on these estimates. We use the notation $X : (E, \varepsilon, \delta)$ to denote that $E \in \mathbb{R}$ is an estimate for $[\![X]\!]$ with corresponding high-probability bound

$$\Pr[|E - [\![X]\!]| \leq \varepsilon] \geq 1 - \delta, \tag{11}$$

where $\varepsilon, \delta \in \mathbb{R}_+$. We call Eq. 11 a *lemma*. Similarly, we use the notation $Y : (I, \gamma)$ to denote that $I \in \{\text{true}, \text{false}\}$ is an estimate of $[\![Y]\!]$ with corresponding high-probability bound

$$\Pr[I = [\![Y]\!]] \geq 1 - \gamma, \tag{12}$$

$$\frac{\mu_Z : (E, \varepsilon, \delta) \in \Gamma}{\Gamma \vdash \mu_Z : (E, \varepsilon, \delta)} \text{ (random variable)} \qquad \frac{c \in \mathbb{R}}{\Gamma \vdash (c, 0, 0)} \text{ (constant)} \qquad \frac{\Gamma \vdash X : (E, \varepsilon, \delta),\ \Gamma \vdash X' : (E', \varepsilon', \delta')}{\Gamma \vdash X + X' : (E + E', \varepsilon + \varepsilon', \delta + \delta')} \text{ (sum)}$$

$$\frac{\Gamma \vdash X : (E, \varepsilon, \delta)}{\Gamma \vdash -X : (-E, \varepsilon, \delta)} \text{ (negative)} \qquad \frac{\Gamma \vdash X : (E, \varepsilon, \delta),\ |E| > \varepsilon}{\Gamma \vdash X^{-1} : (E^{-1}, \frac{\varepsilon}{|E| \cdot (|E| - \varepsilon)}, \delta)} \text{ (inverse)}$$

$$\frac{\Gamma \vdash X : (E, \varepsilon, \delta),\ \Gamma \vdash X' : (E', \varepsilon', \delta')}{\Gamma \vdash X \cdot X' : (E \cdot E', |E| \cdot \varepsilon' + |E'| \cdot \varepsilon + \varepsilon \cdot \varepsilon', \delta + \delta')} \text{ (product)}$$

$$\frac{\Gamma \vdash X : (E, \varepsilon, \delta),\ E - \varepsilon \geq 0}{\Gamma \vdash X \geq 0 : (\text{true}, \delta)} \text{ (inequality true)} \qquad \frac{\Gamma \vdash X : (E, \varepsilon, \delta),\ E + \varepsilon < 0}{\Gamma \vdash X \geq 0 : (\text{false}, \delta)} \text{ (inequality false)}$$

$$\frac{\Gamma \vdash Y : (I, \gamma),\ \Gamma \vdash Y' : (I', \gamma')}{\Gamma \vdash Y \wedge Y' : (I \wedge I', \gamma + \gamma')} \text{ (and)} \qquad \frac{\Gamma \vdash Y : (I, \gamma),\ \Gamma \vdash Y' : (I', \gamma')}{\Gamma \vdash Y \vee Y' : (I \vee I', \gamma + \gamma')} \text{ (or)} \qquad \frac{\Gamma \vdash Y : (I, \gamma)}{\Gamma \vdash \neg Y : (\neg I, \gamma)} \text{ (not)}$$

Fig. 3. Inference rules used to derive lemmas $X : (E, \varepsilon, \delta)$ and $Y : (I, \gamma)$ for specifications $X \in \mathcal{L}(T)$ and $Y \in \mathcal{L}(S)$.

where $\gamma \in \mathbb{R}_+$. Then, let $\Gamma = \{\mu_Z : (\hat{\mu}_Z, \varepsilon, \delta)\}$ be an environment of lemmas for the subexpressions μ_Z. In Figure 3, we show the inference rules that our algorithm uses to derive lemmas for expressions $X \in \mathcal{L}(T)$ and $Y \in \mathcal{L}(S)$ given Γ. The rules for expectations μ_Z and constants c are straightforward. Next, consider the rule for sums—its premise is

$$\Pr[|E - [\![X]\!]| \leq \varepsilon] \geq 1 - \delta$$
$$\Pr[|E' - [\![X']\!]| \leq \varepsilon'] \geq 1 - \delta'.$$

By a union bound, the events $|E - [\![X]\!]| \leq \varepsilon$ and $|E' - [\![X']\!]| \leq \varepsilon'$ hold with probability at least $1 - (\delta + \delta')$, so

$$\begin{aligned} |(E + E') - ([\![X]\!] + [\![X']\!])| &\leq |E - [\![X]\!]| + |E' - [\![X']\!]| \\ &\leq \varepsilon + \varepsilon'. \end{aligned}$$

Thus, we have lemma $X + X' : (E + E', \varepsilon + \varepsilon', \delta + \delta')$, which is exactly the conclusion of the rule for sums. The rules for products, inverses, and if-then-else statements hold using similar arguments; the only subtlety is that for inverses, a constraint $|E| > \varepsilon$ in the premise of the rule is needed to ensure that that $[\![X]\!] \neq 0$ with probability at least $1 - \delta$. The rules for conjunctions, disjunctions, and negations also follow using similar arguments. There are two rules for inequalities $X \geq 0$—one for the case where the inequality evaluates to true, and one for the case where it evaluates to false. Note that at most one rule may apply (but it may be the case that neither rule applies). We describe the rule for the former case; the rule for the latter case is similar.

Note that the inequality evaluates to true as long as $[\![X]\!] \geq 0$. Thus, suppose that E is an estimate of X satisfying the premise of the rule, i.e.,

$$\Pr[|E - [\![X]\!]| \leq \varepsilon] \geq 1 - \delta$$
$$E - \varepsilon \geq 0.$$

Rearranging the inequality $E - [\![X]\!] \leq \varepsilon$ gives

$$[\![X]\!] \geq E - \varepsilon \geq 0.$$

$$\begin{aligned}
[\![\mu_Z]\!]_\delta &= 1 & [\![X^{-1}]\!]_\delta &= [\![X]\!]_\delta \\
[\![c]\!]_\delta &= 0 & [\![X \geq 0]\!]_\delta &= [\![X]\!]_\delta \\
[\![X + X']\!]_\delta &= [\![X]\!]_\delta + [\![X']\!]_\delta & [\![X \wedge X']\!]_\delta &= [\![X]\!]_\delta + [\![X']\!]_\delta \\
[\![-X]\!]_\delta &= [\![X]\!]_\delta & [\![X \vee X']\!]_\delta &= [\![X]\!]_\delta + [\![X']\!]_\delta \\
[\![X \cdot X']\!]_\delta &= [\![X]\!]_\delta + [\![X']\!]_\delta & [\![\neg X]\!]_\delta &= [\![X]\!]_\delta
\end{aligned}$$

Fig. 4. Inference rules used to compute δ_Z, in particular, $\delta_Z = \Delta/[\![Y]\!]_\delta$, where $Y \in \mathcal{L}(S)$ is the specification to be verified and $\delta \in \mathbb{R}_+$ is the desired confidence.

Thus, $[\![X \geq 0]\!] = \text{true}$ with probability at least $1 - \delta$ (since the original inequality holds with probability at least $1 - \delta$). In other words, we can conclude that $X \geq 0 : (\text{true}, \delta)$, which is exactly the conclusion of the rule for the inequality evaluating to true. In summary, we have:

Theorem 4.2. *The inference rules in Figure 3 are sound.*

We give a proof in Appendix A.2. As an example, we describe how to apply the inference rules to infer whether the demographic parity specification Y_{parity} holds. Recall that this specification is a function of the Bernoulli random variables R_{maj} and R_{min}. Suppose that

$$\begin{aligned}
\mu_{R_{\text{maj}}} &: (E_{\text{maj}}, \varepsilon_{\text{maj}}, \delta_{\text{maj}}) \\
\mu_{R_{\text{min}}} &: (E_{\text{min}}, \varepsilon_{\text{min}}, \delta_{\text{min}}),
\end{aligned}$$

and that $|E_{\text{maj}}| > \varepsilon_{\text{maj}}$. Let

$$\begin{aligned}
E_{\text{parity}} &= E_{\text{min}} \cdot E_{\text{maj}}^{-1} - (1 - c) \\
\varepsilon_{\text{parity}} &= |E_{\text{maj}}|^{-1} \cdot \varepsilon_{\text{min}} + \frac{\varepsilon_{\text{maj}} \cdot (|E_{\text{min}}| + \varepsilon_{\text{min}})}{|E_{\text{maj}}|(|E_{\text{maj}}| - \varepsilon_{\text{maj}})}
\end{aligned}$$

Now, if $E_{\text{parity}} - \varepsilon_{\text{parity}} \geq 0$, then $Y_{\text{parity}} : (\text{true}, \delta_{\text{maj}} + \delta_{\text{min}})$, and if $E_{\text{parity}} + \varepsilon_{\text{parity}} < 0$, then $Y_{\text{parity}} : (\text{false}, \delta_{\text{maj}} + \delta_{\text{min}})$.

4.4 Choosing δ_Z

To ensure that Algorithm 1 terminates, we have to ensure that for any given problem instance, we eventually either prove or disprove the given specification Y.[3] More precisely, as $n \to \infty$ (where n is the number of samples taken so far), we must derive $\Gamma \vdash Y : (I, \gamma)$ for some $\gamma \leq \Delta$ (where Δ is the given confidence level) and $I \in \{\text{true}, \text{false}\}$, with probability 1. In particular, the value γ depends on the environment $\Gamma = \{\mu_Z : (s/n, \varepsilon_Z, \delta_Z)\}$. In Γ, our algorithm can choose the value $\delta_Z \in \mathbb{R}_+$ (which determines $\varepsilon_Z = \varepsilon(\delta_Z, n)$ via Eq. 10). Thus, to ensure termination, we have to choose δ_Z so that we eventually derive $Y : (I, \gamma)$ such that $\gamma \leq \Delta$.

In fact, γ is a simple function of δ_Z—each inference rule in Figure 3 adds the values of δ (or γ) for each subexpression of the current expression, so γ equals the sum of the values of δ for each leaf in the syntax tree of Y. Since we have assumed there is a single Bernoulli random variable Z, each leaf in the syntax tree has either $\delta = \delta_Z$ (for leaves labeled μ_Z) or $\delta = 0$ (for leaves labeled $c \in \mathbb{R}$). Thus, γ has the form $\gamma = m \cdot \delta_Z$ for some $m \in \mathbb{N}$. The rules in Figure 4 compute this value $m = [\![Y]\!]_\delta$—the base cases are $[\![\mu_Z]\!]_\delta = 1$ and $[\![c]\!]_\delta = 0$, and the remaining rules add together the values of m for each subexpression of the current expression.

As a consequence, for any $\Delta \in \mathbb{R}_+$, we can derive $Y : (I, \gamma)$ with $\gamma \leq \Delta$ from Γ by choosing $\delta_Z = \Delta/m$.

[3] We require a technical condition on the problem instance; see Section 5.

Theorem 4.3. *Let $(P_{\mathcal{Z}}, Y)$ be a well-defined problem instance, and let $\Delta \in \mathbb{R}_+$ be arbitrary. Let $\delta_Z = \Delta/[\![Y]\!]_\delta$, and let*

$$\Gamma^{(n)} = \{\mu_Z : (E^{(n)}, \varepsilon(\delta_Z, n), \delta_Z)\}$$

be the lemma established on the nth iteration of Algorithm 1 (i.e., using n random samples $Z \sim P_{\mathcal{Z}}$). Then, for any $\delta_0 \in \mathbb{R}_+$, there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, we have

$$\Gamma^{(n)} \vdash Y : (I, \gamma)$$

where $\gamma \leq \Delta$ with probability at least $1 - \delta_0$.

We give a proof in Appendix A.3. Note that the probability is taken over the n random samples $Z \sim P_{\mathcal{Z}}$ used to construct $E^{(n)}$. Also, note that the success of the inference is in a high-probability, asymptotic sense—this approach is necessary since adversarial sequences of random samples $Z \sim P_{\mathcal{Z}}$ may cause nontermination, but the probability of such adversarial samples becomes arbitrarily small as $n \to \infty$. Finally, we have focused on the case where there is a single Bernoulli random variable μ_Z. In the general case, we use the same $\delta_Z = \Delta/[\![Y]\!]_\delta$ for each Bernoulli random variable Z; Theorem 4.3 follows with exactly the same reasoning.

Continuing our example, we describe how $\delta_{R_{\text{maj}}}$ and $\delta_{R_{\text{min}}}$ are computed for Y_{parity}. In particular, the inference rules in Figure 4 give $[\![Y_{\text{parity}}]\!]_\delta = 2$, so it suffices to choose

$$\delta_{R_{\text{maj}}} = \delta_{R_{\text{min}}} = \frac{\Delta}{2}.$$

Recall from Section 4.3 that we actually have $\gamma = \delta_{R_{\text{maj}}} + \delta_{R_{\text{min}}}$, so this choice indeed suffices to ensure that $\gamma \leq \Delta$.

5 THEORETICAL GUARANTEES

We prove that Algorithm 1 terminates with probability 1 as long as the given problem instance satisfies a technical condition. Futhermore, we prove that Algorithm 1 is sound and precise in a probabilistic sense.

5.1 Termination

Algorithm 1 terminates as long as it the given problem instance satisfies the following condition:

Definition 5.1. *Given a problem instance consisting of an expression $W \in \mathcal{L}(T) \cup \mathcal{L}(S)$ together with a distribution $P_{\mathcal{Z}}$ for each μ_Z occuring in W, we say the problem instance is **well-defined** if its subexpressions are well-defined. If $W \equiv (X \geq 0)$ or $W \equiv X^{-1}$, we furthermore require that $[\![X]\!] \neq 0$.*

If Y contains a subexpression X^{-1} such that $[\![X]\!] = 0$, then $[\![X^{-1}]\!]$ is infinite. As a consequence, Algorithm 1 fails to terminate since it cannot estimate of $[\![X^{-1}]\!]$ to any finite confidence level. Next, the constraint on subexpressions of the form $X \geq 0$ is due to the nature of our problem formulation. In particular, consider an expression $X \geq 0$, where $[\![X]\!] = 0$. In our setting, we cannot compute $[\![\mu_Z]\!]$ exactly since we are treating the Bernoulli random variables $Z, \ldots$ as blackboxes. Therefore, we also cannot compute $[\![X]\!]$ exactly (assuming it contains subexpressions of the form μ_Z). Thus, we can never determine with certainty whether $[\![X]\!] \geq 0$.

Theorem 5.2. *Given a well-defined problem instance, Algorithm 1 terminates with probability 1, i.e.,*

$$\lim_{n \to \infty} Pr[\textit{Algorithm 1 terminates}] = 1,$$

where n is the number of samples taken so far.

The proof of this thoerem is somewhat subtle. In particular, our algorithm only terminates if we can prove that $\hat{\mu}_Z^{(n)} \to \mu_Z$ as $n \to \infty$, where $\hat{\mu}_Z^{(n)}$ is the estimate of μ_Z established on the nth iteration. However, we cannot use our adaptive concentration inequality in Theorem 4.1 to prove this guarantee, since our adaptive concentration inequality *assumes* that our algorithm terminates with probability 1. Thus, we have to directly prove that our estimates converge, and then use this fact to prove that our algorithm terminates. We give a full proof in Appendix A.4.

The restriction to well-defined properties is not major—for typical problem instances, having $[\![X]\!] = 0$ hold exactly is very unlikely. Furthermore, this restriction to well-defined problem instances is implicitly assumed by current state-of-the-art systems, including FairSquare [Albarghouthi et al. 2017]. In particular, it is a necessary restriction for any system that does not exactly evaluate the expectations μ_Z. For example, FairSquare relies on a technique similar to numerical integration, and can only obtain estimates $\mu_Z \in [E - \varepsilon, E + \varepsilon]$; therefore, it will fail to terminate given an ill-defined problem instance.

5.2 Probabilistic Soundness and Precision

Let $Y \in \mathcal{L}(S)$ be a specification, and consider a verification algorithm tasked with computing $[\![Y]\!]$. Typically, the algorithm is sound if it only returns true when $[\![Y]\!] = \text{true}$, and it is precise if it only returns false when $[\![Y]\!] = \text{false}$. However, because our algorithm uses random samples to evaluate $[\![Y]\!]$, it cannot guarantee soundness or precision—e.g., adversarial sequences of samples can cause the algorithm to fail. Instead, we need probabilistic notions of soundness and precision.

Definition 5.3. *Let $\Delta \in \mathbb{R}_+$. We say a verification algorithm is Δ-**sound** if it returns true only if*

$$Pr[[\![Y]\!] = \mathit{true}] \geq 1 - \Delta,$$

*where the probability is taken over the random samples drawn by the algorithm. Furthermore, if the algorithm takes Δ as a parameter, and is Δ-sound for any given $\Delta \in \mathbb{R}_+$, then we say that the algorithm is **probabilistically sound**.*

Definition 5.4. *Let $\Delta \in \mathbb{R}_+$. We say a verification algorithm is Δ-**precise** if it returns false only if*

$$Pr[[\![Y]\!] = \mathit{false}] \geq 1 - \Delta$$

*where the probability is taken over the random samples drawn by the algorithm. Furthermore, if the algorithm takes Δ as a parameter, and is Δ-precise for any given $\Delta \in \mathbb{R}_+$, then we say that the algorithm is **probabilistically precise**.*

Theorem 5.5. *Algorithm 1 is probabilistically sound and probabilistically precise.*

We give a proof in Appendix A.5. For ill-defined problem instances, Algorithm 1 may fail to terminate, but nontermination is allowed by probabilistic soundness and precision.

6 EVALUATION

We have implemented our algorithm a tool called VeriFair, which we evaluate on two benchmarks. First, we compare to FairSquare on their benchmark, where the goal is to verify whether demographic parity holds [Albarghouthi et al. 2017]. In particular, we show that VeriFair scales substantially better than FairSquare on every large problem instance in their benchmark (with $\Delta = 10^{-10}$).

However, the FairSquare benchmark fails to truly demonstrate the scalability of VeriFair. In particular, it exclusively contains tiny classifiers—e.g., the largest neural network in their benchmark has a single hidden layer with just two hidden units. This tiny example already causes FairSquare to time out. Indeed, the scalability of FairSquare depends on the complexity the internal structure

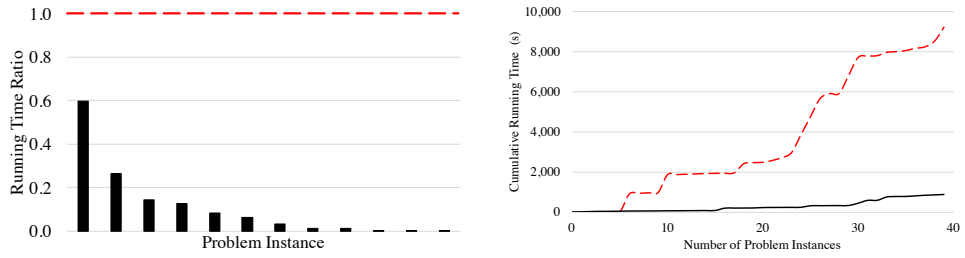


Fig. 5. (a) Results for the largest problem instances from the FairSquare benchmark. The y-axis is the ratio of the VeriFair running time to the FairSquare running time (so lower is better). The problem instances are along the x-axis; we have sorted them from highest to lowest. The red, dashed line at $y = 1$ denotes the FairSquare running time; for all instances below this line, VeriFair outperforms FairSquare. (b) The cumulative running time of VeriFair (black, solid) and FairSquare (red, dashed). In particular, we sorted all 39 problem instances from smallest to largest (in terms of lines of code), and plot the cumulative running time from running the first i benchmarks. The x-axis is i, and the y-axis is the running time.

of the classifier and population model, whereas the scalability of VeriFair only depends on the time it takes to execute these models.

Thus, for our second benchmark, we use a state-of-the-art deep recurrent neural network (RNN) designed to classify sketches [Google 2018], together with a state-of-the-art deep generative model for randomly sampling sketches similar to those produced by humans [Ha and Eck 2017]. Together, these two deep neural networks have more than 16 million parameters, which is 5 orders of magnitude larger than the largest neural network in the FairSquare benchmark. We show that VeriFair scales to this benchmark, and furthermore study how its scalability depends on various hyperparameters. In fact, FairSquare cannot even be applied to this benchmark, since FairSquare can only be applied to straight line programs but the RNN computation involves a possibly unbounded loop operation.

6.1 FairSquare Benchmark

We begin by comparing our tool, VeriFair, to FairSquare, a state-of-the-art fairness verification tool. The results on this benchmark were run on a machine with a 2.2GHz Intel Xeon CPU with 20 cores and 128 GB of memory.

Benchmark. The FairSquare benchmark contains 39 problem instances. Each problem instance consists of a classifier $f : \mathcal{V} \to \{0, 1\}$, where $\mathcal{V} = \mathbb{R}^d$ with $d \in [1, 6]$, together with a population model encoding a distribution $P_{\mathcal{V}}$ over $\mathcal{V}$. The classifiers include decision trees with up to 44 nodes, SVMs, and neural networks with up to 2 hidden units. The population models include one where the features are assumed to be independent and two Bayes net models. The goal is to check whether demographic parity holds, taking $c = 0.15$ in Definition 3.1. We run VeriFair using $\Delta = 10^{-10}$ (i.e., the probability of an incorrect response is at most 10^{-10}).

In theory, FairSquare provides stronger guarantees than VeriFair, since FairSquare never responds incorrectly. Intuitively, the guarantees provided by FairSquare are analogous to using VeriFair with $\Delta = 0$. However, as we discuss below, because we have taken the parameters to be so small, they have essentially no effect on the outputs of VeriFair. Also, the population models in the FairSquare benchmark often involve conditional probabilities. There are many ways to sample such a probability distribution. We use the simplest technique, i.e., rejection sampling; we

Table 1. Results from comparing VeriFair to FairSquare [Albarghouthi et al. 2017]. For each problem instance (i.e., a classifier and population model), we show the total number of lines of code (LOC), the response of each tool, the running time of each tool (in seconds, timed out after 900 seconds), the ratio of the running time of VeriFair to that of FairSquare (lower is better), and for the rejection sampling strategy used by VeriFair, the number of accepted samples, total samples, and the acceptance rate. In the ratio of running times, we conservatively assume FairSquare takes 900 seconds to run if it times out; this ratio sometimes equals 0 due to rounding error.

Classifier	Pop. Model	LOC	Is Fair?		Running Time (s)			Samples		
			VeriFair	FairSquare	VeriFair	FairSquare	Ratio	Accepted	Total	Accept Rate
DT_4	Ind.	17	1	1	21.2	2.1	9.9	91710	443975	20.7%
DT_{14}	Ind.	34	1	1	120.4	4.1	29.3	365503	1768404	20.7%
DT_{16}	Ind.	38	1	1	17.3	5.6	3.1	49095	236822	20.7%
DT^{α}_{16}	Ind.	42	1	1	3.1	6.4	0.5	7221	35377	20.4%
DT_{44}	Ind.	95	1	1	33.3	19.5	1.7	68078	329859	20.6%
SVM_3	Ind.	15	1	1	9.4	2.4	3.9	34304	166274	20.6%
SVM_4	Ind.	17	1	1	9.6	3.5	2.7	33158	159964	20.7%
SVM^{α}_4	Ind.	19	1	1	1.7	3.0	0.6	5437	26013	20.9%
SVM_5	Ind.	19	1	1	10.7	6.4	1.7	36315	175729	20.7%
SVM_6	Ind.	21	1	1	7.8	5.4	1.4	28140	136722	20.6%
$NN_{2,1}$	Ind.	22	1	1	2.3	3.9	0.6	9364	45289	20.7%
$NN_{2,2}$	Ind.	25	1	1	2.9	6.1	0.5	11407	55102	20.7%
$NN_{3,2}$	Ind.	27	1	1	6.4	435.6	0.0	20856	100855	20.7%
DT_4	B.N. 1	27	0	0	1.6	3.5	0.5	6208	29689	20.9%
DT_{14}	B.N. 1	48	1	1	156.0	21.8	7.1	442872	2147170	20.6%
DT_{16}	B.N. 1	51	0	0	2.4	15.3	0.2	5698	27422	20.8%
DT^{α}_{16}	B.N. 1	55	1	1	24.4	27.7	0.9	64691	313671	20.6%
DT_{44}	B.N. 1	111	0	0	17.5	353.2	0.0	33750	163661	20.6%
SVM_3	B.N. 1	25	0	0	3.0	4.0	0.7	10347	49845	20.8%
SVM_4	B.N. 1	30	0	0	4.6	5.8	0.8	15009	72556	20.7%
SVM^{α}_4	B.N. 1	32	1	1	5.2	10.4	0.5	16846	81355	20.7%
SVM_5	B.N. 1	35	0	0	3.5	11.1	0.3	12116	58197	20.8%
SVM_6	B.N. 1	40	0	0	3.0	19.0	0.2	9193	44575	20.6%
$NN_{2,1}$	B.N. 1	36	1	1	2.9	57.0	0.1	10345	50183	20.6%
$NN_{2,2}$	B.N. 1	39	1	1	4.8	32.7	0.1	14449	69779	20.7%
$NN_{3,2}$	B.N. 1	40	1	T.O.	88.3	T.O.	0.1	308228	1489839	20.7%
DT_4	B.N. 2	33	0	0	1.4	5.8	0.2	4790	23232	20.6%
DT_{14}	B.N. 2	54	1	T.O.	190.1	T.O.	0.2	524166	2535812	20.7%
DT_{16}	B.N. 2	57	0	0	3.1	35.4	0.1	7002	34194	20.5%
DT^{α}_{16}	B.N. 2	61	1	1	24.0	60.0	0.4	61027	295445	20.7%
DT_{44}	B.N. 2	117	0	T.O.	22.1	T.O.	0.0	40841	197689	20.7%
SVM_3	B.N. 2	31	0	0	4.3	8.7	0.5	14392	69596	20.7%
SVM_4	B.N. 2	36	0	0	3.8	24.2	0.2	11113	53831	20.6%
SVM^{α}_4	B.N. 2	38	1	1	5.9	22.1	0.3	18664	89394	20.9%
SVM_5	B.N. 2	41	0	0	3.8	496.7	0.0	12147	58115	20.9%
SVM_6	B.N. 2	42	0	0	3.9	87.8	0.0	11765	56820	20.7%
$NN_{2,1}$	B.N. 2	38	1	1	2.9	52.2	0.1	9717	47162	20.6%
$NN_{2,2}$	B.N. 2	41	1	1	4.1	126.4	0.0	12729	61965	20.5%
$NN_{3,2}$	B.N. 2	42	1	T.O.	110.9	T.O.	0.1	387860	1880146	20.6%

discuss the performance implications below. Finally, the problem instances in the FairSquare benchmark are implemented as Python programs. While we report results using the original Python implementations, below we discuss how compiling the benchmarks can substantially speed up execution.

Results. For both tools, we set a timeout of 900 seconds. We give a detailed results in Table 1. For each problem instance, we show the running times of VeriFair and FairSquare, as well as the ratio

$$\frac{\text{running time of VeriFair}}{\text{running time of FairSquare}},$$

where we conservatively assume that FairSquare runs in 900 seconds for problem instances in which it times out. We also show the number of lines of code and some statistics about the rejection sampling approach we use to sample the population models.

VeriFair outperforms FairSquare on 30 of the 39 problem instances. More importantly, VeriFair scales much better to large problem instances—whereas FairSquare times out on 4 problem instances, VeriFair terminates on all 39 in within 200 seconds. In particular, while FairSquare relies on numerical integration that may scale exponentially in the problem size, VeriFair relies on sampling, which linearly in the time required to execute the population model and classifier.

In Figure 5 (a), we show results for 12 of the largest problem instances. In particular, we include the largest two each of decision tree, SVM, and neural network classifiers, using each of the two Bayes net population models. As can be seen, VeriFair runs faster than FairSquare on all of the problem instances, and more than twice as fast in all but one.

Similarly, in Figure 5 (b), we plot the cumulative running time of each tool across all 39 problem instances. For this plot, we sort the problem instances from smallest to largest based on number of lines of code. Then, the plot shows the cumulative running time of the first i problem instances, as a function of i. As before, we conservatively assume that FairSquare terminates in 900 seconds when it times out. As can be seen, VeriFair scales significantly better than FairSquare—VeriFair becomes faster than FairSquare after the first 9 problem instances, and substantially widens that lead as the problem instances become larger.

Compiled problem instances. The running time of VeriFair depends linearly on the time taken by a single execution of the population model and classifier. Because the benchmarks are implemented in Python, the running time can be made substantially faster if they are compiled to native code. To demonstrate this speed up, we manually implement two of the problem instances in C++:

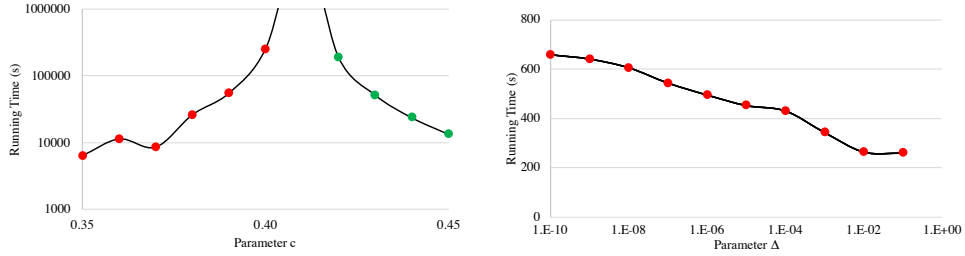
- The decision tree with 14 nodes with the independent population model; in this problem instance, VeriFair is slowest relative to FairSquare (29.3× slower). VeriFair runs the compiled version of this model in just 0.40 seconds, which is a 301× speed up, and more than 10× faster than FairSquare.
- The decision tree with 14 nodes with Bayes net 2 as the popluation model; in this problem instance, VeriFair is slowest overall (190.1 seconds). VeriFair runs the compiled version of this model in just 0.58 seconds, which is a 327× speed up.

Note that compiling problem instances would not affect FairSquare, since it translates them to SMT formula.

Comparison of guarantees. We ran the VeriFair ten times on the benchmark; the responses were correct on all iterations. Indeed, because we have set $\Delta = 10^{-10}$, it is extremely unlikely that the response of VeriFair is incorrect.

Rejection sampling. When the population model contains conditional probabilities, VeriFair uses rejection sampling to sample the model. The acceptance rate is always between 20-21%. This consistency is likely due to the fact that the models in the FairSquare benchmark are always modeling the same population. Thus, rejection sampling is an effective strategy for the FairSquare benchmark. Furthermore, we discuss possible alternatives to rejection sampling in Section 7.

Path-specific causal fairness. We check whether path-specific causal fairness Y_{causal} holds for three FairSquare problem instances—the largest classifier of each kind using the Bayes net 2 population model. We use the number of years of education as the mediator covariate. We use $\Delta = 10^{-10}$. VeriFair concludes that all of the problem instances are fair. The running time for the



(a) (b)

Fig. 6. We plot the running time of VeriFair on the Quick Draw benchmark as a function of (a) the parameter c, and (b) the parameter Δ. The running times in (b) are averaged over 10 runs; the running times in (a) are reported for a single run since they were too expensive to run multiple times. In each figure, a green marker denotes a response of "fair" and a red marker denotes a response of "unfair". In (a), the curve diverges because VeriFair times out when $c = 0.41$.

decision tree DT_{44} with 44 nodes is 2.47 seconds, for the SVM SVM_6 with $d = 6$ features terminates is 8.89 seconds, and for the neural network $\text{NN}_{3,2}$ with $d = 3$ features and 2 neurons is 0.35 seconds.

6.2 Quick Draw Benchmark

Our next benchmark consists of a deep recurrent neural network (RNN) classifier and a deep sequence-to-sequence variational autoencoder (VAE) population model [Ha and Eck 2017]. Recall that VeriFair scales linearly with the running time of the classifier and population model; therefore, VeriFair should scale to these very complex models as long as executing the model can be executed in a reasonable amount of time. In this benchmark, we use VeriFair to verify the equal opportunity property described in Definition 3.2. Finally, we study how the running time of VeriFair on this benchmark depends on various problem parameters. The results on this benchmark were run on a machine with an Intel Core i7-6700K 4GHz quad core CPU, an Nvidia GeForce GTX 745 with 4GB of GPU memory (used to run the deep neural networks), and 16GB of memory. Note that we cannot run FairSquare on this benchmark, since it can only handle straight-line models but recurrent neural networks involve a loop operation.

Benchmark. The classifier in our benchmark is an RNN $f : \mathcal{X} \to \{0, 1\}$, where $x \in \mathcal{X}$ is representation of a 256×256 image that is a black and white sketch drawn by a human. This image is represented as a sequence of strokes (x, y, p), where (x, y) is the displacementand p is a command (pen down, pen up, or finish drawing). Each input is a drawing of one of 345 different categories of objects, including dogs, cats, firetrucks, gardens, etc. To obtain a binary classifier, we train a classifier to predict a binary label $y \in \{0, 1\}$ indicating whether the input image is a drawing of a dog. The neural network was trained on a dataset $X \subseteq \mathcal{X}$ containing 70K training examples, 2.5K cross-validation examples, and 2.5K test examples; overall, 0.3% of the training images are dogs. Its accuracy is 18.8%, which is 626× better than random. Our population model is the decoder portion of a VAE [Ha and Eck 2017], which generates a random sequence in the form described above.

We have the country of origin for each image; we consider images from the United States to be the majority subpopulation (43.9% of training examples), and images from other countries to be the minority subpopulation. We train two population models: (i) a decoder d_{maj} trained to generate sketches of dogs from the United States, and (ii) d_{min} to generate sketches of dogs from other countries.

We aim to verify the equal opportunity property. Recall that this property says that the classifier f should not make mistakes (in particular, false negatives) much more frequently for members of the minority class than for members of the majority class. For example, a classifier that always responds randomly is fine, but the classifier cannot respond accurately for majority members and randomly for minority members. In the context of the Quick Draw benchmark, this fairness property says that the classifier should not perform worse for people from outside of the United States. This guarantee is important for related tasks—e.g., classifiers for detecting skin cancer from photographs [Esteva et al. 2017] and login systems based on face recognition [Simon 2009]; for example, certain face recognition systems have been shown to have more difficulty detecting minority users than detecting majority users. As before, we use parameter $c = 0.15$ in the fairness specification.

Batched samples. Typical deep learning frameworks are much more efficient when they operate on batches of data. Thus, we batch the samples taken by VeriFair—on each iteration, it samples 1000 images $X_{\text{maj}} \sim d_{\text{maj}}$ and 1000 images $X_{\text{min}} \sim d_{\text{min}}$ as a batch, and computes $f(X_{\text{maj}})$ and $f(X_{\text{min}})$ as a batch as well. As a consequence, Algorithm 1 may sample up to 999 more images than needed, but we find that execution time improves significantly—sampling a single image takes about 0.5 seconds, whereas sampling 1000 images takes about 30 seconds, for a speed up of about 17×.

Results. We ran VeriFair on our benchmark; using $\Delta = 10^{-5}$, VeriFair terminates in 301 seconds and uses 14,000 samples, and using $\Delta = 10^{-10}$, VeriFair terminated in 606 seconds and uses 28,000 samples.

Varying c**.** Besides the running time of the classifier f and population models d_{maj} and d_{min}, the most important factor affecting the running time of VeriFair is the value of the parameter c. In particular, in the specification

$$Y_{\text{equal}} \equiv \left(\frac{\mu_{R_{\text{min}}}}{\mu_{R_{\text{maj}}}} \geq 1 - c \right),$$

as the left-hand side and right-hand side of the inequality become closer together, then we need increasingly accurate estimates of $\mu_{R_{\text{min}}}$ and $\mu_{R_{\text{maj}}}$ to check whether the specification holds. Thus, VeriFair needs to take a larger number of samples to confidently determine whether Y_{equal} holds.

We ran VeriFair on with values of c near

$$c_0 = 1 - \frac{\mu_{R_{\text{min}}}}{\mu_{R_{\text{maj}}}} \approx 0.41,$$

in particular, $c \in \{0.35, 0.36, \ldots, 0.45\}$ (with $\Delta = 10^{-5}$). In Figure 6 (a), we plot the running time of VeriFair on Quick Draw as a function of c. VeriFair terminated for all choices of c except $c = 0.41$, which timed out after 96 hours. For the remaining choices of c, the longest running time was $c = 0.40$, which terminated after 84 hours. We also show whether VeriFair concludes that the specification is true (green marker) or false (red marker)—VeriFair concludes that Quick Draw is fair if $c > 0.41$ and unfair if $c \leq 0.40$.

In practice, c is unlikely to be very close to c_0. Furthermore, approaches based on numerical integration would suffer from a similar divergence near $c = c_0$, since their estimate of Y_{equal} is subject to numerical errors that must be reduced by increasing precision, which increases running time.

Varying Δ**.** We study the running time of VeriFair on Quick Draw as a function of Δ, which controls the probability that VeriFair may respond incorrectly. In particular, we ran VeriFair

on Quick Draw with values $\Delta \in \{10^{-10}, 10^{-9}, ..., 10^{-1}\}$ (with $c = 0.15$). In Figure 6 (b), we plot the running time of VeriFair as a function of Δ. As expected, the running time increases as Δ becomes smaller. Even using $\Delta = 10^{-10}$, the running time is only about 10 minutes. In particular, VeriFair scales very well as a function of Δ—the running time only increases linearly even as we decrease Δ exponentially.

7 DISCUSSION

In this section, we discuss various aspects of our algorithm.

Population models. A key input to our algorithm is the population model encoding the distribution over population members. Intuitively, population models are analogous to preconditions. Population models are required for most fairness definitions, since these definitions are typically constraints on statistical properties of the classifier for different subpopulations. Without a population model, we cannot reason about the distribution of outputs. Population models can easily be obtained by fitting a density estimation model (e.g., a GAN, Bayesian network, VAE, etc.) to the data.

An advantage of our approach compared to previous work is that we only require blackbox access to the population model. Thus, if a population model is unavailable, our tool can actually be run online as real population members arrive over time. In this setting, it may be possible that an unfair model is deployed in production for some amount of time, but our tool will eventually detect the unfairness, upon which the model can be removed.

Additional fairness specifications. While we have focused on a small number of fairness specifications, many others have been proposed in the literature. Indeed, the exact notion of fairness can be context-dependent; a major benefit of our approach is that it can be applied to a wide range of specifications. For example, we can straightforwardly support other kinds of fairness for supervised learning [Galhotra et al. 2017; Kleinberg et al. 2017; Zafar et al. 2017]. We can also straightforwardly extend our techniques to handle multiple minority subgroups; for example, the extension of demographic parity to a set $\mathcal{M}_{\text{min}}$ of minority subgroups is

$$Y_{\text{parity}} \equiv \bigwedge_{m \in \mathcal{M}_{\text{min}}} \left(\frac{\mu_{R_m}}{\mu_{R_{\text{maj}}}} \geq 1 - c \right),$$

where $R_m = f(V_m)$ and $V_m = P_{\mathcal{V}} \mid A = m$. Furthermore, we can support extensions of these properties to regression and multi-class classification; for example, for regression, an analog of demographic parity is

$$Y_{\text{reg}} \equiv |\mu_{R_{\text{maj}}} - \mu_{R_{\text{min}}}| \leq c,$$

where $f : \mathcal{V} \to [0, 1]$ is a real-valued function (where $[0, 1]$ is the unit interval), $R_{\text{maj}} = f(V_{\text{maj}})$ and $R_{\text{min}} = f(V_{\text{min}})$ are as before, and $c \in \mathbb{R}_+$ is a constant. [4] In other words, the outcomes for members of the majority and minority subpopulations are similar on average. In the same way, we can support extensions to the reinforcement learning setting [Wen et al. 2019]. We can also support counterfactual fairness [Kusner et al. 2017] and causal fairness [Kilbertus et al. 2017], which are variants of path-specific causal fairness without a mediator variable.

Another approach to fairness is *individual fairness* [Dwork et al. 2012], which intuitively says that people with similar observed covariates should be treated similarly. Traditionally, this notion

[4] The constraint that $f(V) \in [0, 1]$ is needed for our concentration inequality, Theorem 4.1, to apply.

is defined over a finite set of individuals $x, y \in \mathcal{X}$, in which case it says:

$$\bigwedge_{V \in \mathcal{V}} \bigwedge_{V' \in \mathcal{V}} \|f(V) - f(V')\|_1 \leq \lambda \cdot \|V - V'\|_1 \tag{13}$$

where $f(x) \in \mathbb{R}^k$ are the outcomes and $\lambda \in \mathbb{R}_+$ is a given constant. This finite notion is trivial to check by enumerating over $V, V' \in \mathcal{V}$. We can check an extension to the case of continous $V, V' \in \mathcal{V}$, except where we only want Eq. 13 to hold with high probability:

$$Y_{\text{ind}} \equiv (\mu_R \geq 1 - c),$$

where $c \in \mathbb{R}_+$ is a constant, and where $V \sim \mathcal{V}$, $V' \sim \mathcal{V}$, and

$$R = \mathbb{I}\left[\|f(V) - f(V')\|_1 \leq \|V - V'\|_1\right].$$

In particular, note that R is a Bernoulli random variable that indicates whether Eq. 13 holds for a random pair $V, V' \sim \mathcal{V}$. Thus, the specification Y_{ind} says that the probability that Eq. 13 holds for random individuals V and V' is at least $1 - c$.

Sampling algorithm. Recall that VeriFair uses rejection sampling, which we find works well for typical fairness definitions. In particular, most definitions only condition on being a member of the majority or minority subpopulation, or other fairly generic qualifications. These events are not rare, so there is no need to use more sophisticated sampling techniques. We briefly discuss how our approach compares to symbolic methods, as well as a possible approach to speeding up sampling by using importance sampling.

First, we note that existing approaches based on symbolic methods—in particular, FairSquare; see Figure 6 in [Albarghouthi et al. 2017]—would also have trouble scaling to specifications that condition on rare events. The reason is that FairSquare requires that the user provides a piecewise constant distribution $\tilde{P}_{\mathcal{V}}$ that approximates the true distribution $P_{\mathcal{V}}$. Their approach performs integration by computing regions of the input space that have high probability according to this approximate distribution $\tilde{P}_{\mathcal{V}}$; once an input region is chosen, it computes the actual volume according to the true distribution $P_{\mathcal{V}}$. Thus, if the approximation $\tilde{P}_{\mathcal{V}}$ is poor, then the actual volume could be much smaller than the volume according to the approximation, so their approach would also scale poorly.

Furthermore, if our algorithm has access to a good approximation $\tilde{P}_{\mathcal{V}}$, then we may be able to use it to speed up sampling. In particular, suppose that we have access to a piecewise constant $\tilde{P}_{\mathcal{V}}$, where each piece is on a polytope A_i (for $i \in [h]$) of the input space, and the probability on A_i is a constant $p_i \in [0, 1]$. We consider the problem of sampling from $P_{\mathcal{V}} \mid C$, where we assume (as in FairSquare) that the constraints C are affine. In this context, we can use $\tilde{P}_{\mathcal{V}}$ to improve the scalability of sampling by using *importance sampling*. First, to sample $\tilde{P}_{\mathcal{V}} \mid C$, we can efficiently compute the volume of each of constrained polytope $v_i = \text{Vol}(A_i \cap A_C)$, where A_C is the polytope corresponding to the constraints C [Lawrence 1991]. Next, we can directly sample $V \sim \tilde{P}_{\mathcal{V}}$ as follows: (i) sample a random polytope according to their probabilities according to $\tilde{P}_{\mathcal{V}} \mid C$, i.e., $i \sim \text{Categorical}(p_1 \cdot v_1, ..., p_h \cdot v_h)$, and (ii) randomly sample a point $V \sim \text{Uniform}(A_i \cap A_C)$; the second step can be accomplished efficiently [Chen et al. 2018]. Finally, for a random variable X that is a function of V, we have the following identity:

$$\mathbb{E}_{V \sim P_{\mathcal{V}}}[X \mid C] = \mathbb{E}_{V \sim \tilde{P}_{\mathcal{V}}}\left[\frac{X \cdot f_{P_{\mathcal{V}}}(V)}{f_{\tilde{P}_{\mathcal{V}}}(V)} \,\middle|\, C\right],$$

where $f_{P_\mathcal{V}}$ and $f_{\tilde{P}_\mathcal{V}}$ are the density functions of $P_\mathcal{V}$ and $\tilde{P}_\mathcal{V}$, respectively, and where we assume that the support of $\tilde{P}_\mathcal{V}$ contains the support of $P_\mathcal{V}$. Thus, the importance sampling estimator is

$$\hat{\mu}_X = \sum_{i=1}^{n} \frac{X_i \cdot f_{P_\mathcal{V}}(V_i)}{f_{\tilde{P}_\mathcal{V}}(V_i)}, \tag{14}$$

for samples $V_1, ..., V_n$ and where the corresponding values of X are $X_1, ..., X_n$. Assuming

$$\arg\max_{V \in \mathcal{V}} \frac{X \cdot f_{P_\mathcal{V}}(V)}{f_{\tilde{P}_\mathcal{V}}(V)} \leq 1,$$

then Theorem 4.1 continues to hold for Eq. 14; in general, it is straightforward to scale X so that the theorem applies.

One caveat is that this approach is that it requires that $P_\mathcal{V}$ has bounded domain (since the support of $\tilde{P}_\mathcal{V}$ must contain the support of $P_\mathcal{V}$). Technically, the same is true for FairSquare; in particular, since tails of most distributions are small (e.g., Gaussian distributions have doubly exponentially decaying tails), truncating the distribution yields a very good approximation. However, the FairSquare algorithm remains sound since its upper and lower bounds account for the error due to truncation; thus, it converges as long as the truncation error is smaller than the tolerance ε. Similarly, we can likely bound the error for our algorithm, but we leave this approach to future work.

Limitations. As we have already discussed, our algorithm suffers from several limitations. Unlike FairSquare, it is only able to provide high-probability fairness guarantees. Nevertheless, in practice, our experiments show that we can make the failure probability vanishingly small (e.g., $\Delta = 10^{-10}$). Furthermore, our termination guarantee is not absolute, and there are inputs for which our algorithm would fail to terminate (i.e., where fairness "just barely" holds). However, existing tools such as FairSquare would fail to terminate on these problem instances as well. Finally, our approach would have difficulty if the events conditioned on in the population model have very low probability since it relies on rejection sampling.

Challenges for specifications beyond fairness. We focus on fairness properties since sampling population models tends to be very scalable in this setting. In particular, we find that sampling the population model is usually efficient—as above, they are often learned probabilistic models, which are designed to be easy to sample. Furthermore, we find that the conditional statements in the fairness specifications usually do not encode rare events—e.g., in the case of demographic parity, we do not expect minority and majority subpopulations to be particularly rare. In more general settings, there are often conditional sampling problems that are more challenging. For these settings, more sophisticated sampling algorithms would need to be developed, possibly along the lines of what we described above.

Furthermore, our specification language is tailored to fairness specifications, which typically consist of inequalities over arithmetic formulas, and boolean formulas over these inequalities. For other specifications, other kinds of logical operators such as temporal operators may be needed.

Finally, we note that our approach cannot be applied to verifying adversarial properties such as robustness [Goodfellow et al. 2014], which inherently require solving an optimization problem over the inputs of the machine learning model. In contrast, fairness properties are probabilistic in the sense that they can be expressed as expectations over the outputs of the machine learning model.

8 RELATED WORK

Verifying fairness. The work most closely related to ours is [Albarghouthi et al. 2017], which uses numerical integration to verify fairness properties of machine learning models including decision trees, SVMs, and neural networks. Because they rely on constraint solving techniques (in particular, SMT solvers), their tool is substantially less scalable than ours—whereas their tool does not even scale to a neural network with 37 parameters (including those in the Bayes net population model), our tool scales to deep neural networks with 16 million parameters. In contrast to their work, our algorithm may return an incorrect result; however, in our evaluation, we show that these events are very unlikely to happen.

Checking fairness using hypothesis testing. There has also been recent work on checking whether fairness holds by using hypothesis testing [Galhotra et al. 2017]. There are two major advantages of our work compared to their approach. First, they use p-values, which are asymptotic, so they cannot give any formal guarantees; furthermore, they do not account for multiple hypothesis testing, which can yield misleading results. In contrast, our approach establishes concrete, high-probability fairness guarantees. Second, their approach is tailored to a single fairness definition. In contrast, our algorithm can be used with a variety of fairness specifications (including theirs).

Fairness in machine learning. There has been a large literature attempting to devise new fairness specifications, including demographic parity [Calders et al. 2009], equal opportunity [Hardt et al. 2016], and approaches based on causality [Kilbertus et al. 2017; Kusner et al. 2017]. There has also been a large literature focusing on how to train fair machine learning classifiers [Calders and Verwer 2010; Corbett-Davies et al. 2017; Dwork et al. 2012, 2018; Fish et al. 2016; Pedreshi et al. 2008] and transforming the data into fair representations [Calmon et al. 2017; Feldman et al. 2015; Hajian and Domingo-Ferrer 2013; Zemel et al. 2013]. Finally, there has been work on quantifying the influence of input variables on the output of a machine learning classifier; this technique can be used to study fairness, but does not provide any formal fairness guarantees [Datta et al. 2017]. In contrast, our work takes fairness properties as given, and aims to design algorithms for verifying the correctness of existing machine learning systems, which are treated as blackbox functions.

Verifying probabilistic properties. There has been a long history of work attempting to verify probabilistic properties, including program analysis [Albarghouthi et al. 2017; Sampson et al. 2014; Sankaranarayanan et al. 2013], symbolic execution [Filieri et al. 2013; Geldenhuys et al. 2012], and model checking [Clarke and Zuliani 2011; Grosu and Smolka 2005; Kwiatkowska et al. 2002; Younes et al. 2002]. Many of these tools rely on techniques such as numerical integration, which do not scale in our setting [Albarghouthi et al. 2017]. Alternatively, abstraction interpretation has been extended to probabilistic programs [Claret et al. 2013; Monniaux 2000, 2001a,b]; see [Gordon et al. 2014] for a survey. However, these approaches may be imprecise and incomplete (even on non-pathelogical problem instances).

Statistical model checking. There has been work on using statistical hypothesis tests to check probabilistic properties [Clarke and Zuliani 2011; Grosu and Smolka 2005; Herault et al. 2006; Sampson et al. 2014; Sankaranarayanan et al. 2013; Younes et al. 2002; Younes and Simmons 2002].

One line of work relies on a fixed sample size [Hérault et al. 2004; Sampson et al. 2014; Sen et al. 2004, 2005]. Then, they use a statistical test to compute a bound on the probability that the property holds. Assuming a concentration inequality such as Hoeffding's inequality is used [Hérault

et al. 2004], [5] then they can obtain high-probability bounds such as ours. A key drawback is that because they do not adaptively collect data, there is a chance that the statistical test will be able to neither prove nor disprove the specification. Furthermore, simply re-running the algorithm is not statistically sound, since it runs into the problem of multiple hypothesis testing [Johari et al. 2017; Zhao et al. 2016].

An alternative approach that has been studied is to leverage adaptive statistical hypothesis tests—in particular, Wald's sequential probability ratio test (SPRT) [Wald 1945]. Like the adaptive concentration inequalities used in our work, SPRT continues to collect data until the specification is either proven or disproven [Legay et al. 2010; Younes et al. 2002; Younes and Simmons 2002, 2006; Younes 2004]. SPRT can distinguish two hypotheses of the form

$$H_0 \equiv \mu_Z \leq d_0 \quad \text{vs.} \quad H_1 \equiv \mu_Z \geq d_1,$$

where Z is a Bernoulli random variable and $d_1 > d_0$. There are two key shortcomings of these approaches. First, we need to distinguish H_0 vs. $\neg H_0$ (or equivalently, the case $d_0 = d_1$). This limitation is fundamental to approaches based on Wald's test—it computes a statistic S_0 based on d_0 and a statistic S_1 based on d_1, and compares them; if $d_0 = d_1$, then we always have $S_0 = S_1$, so the test can never distinguish H_0 from H_1. Second, Wald's test requires that the distribution of the random variables is known (but the parameters of the distribution may be unknown). While we have made this assumption (i.e., they are Bernoulli), our techniques are much more general. In particular, we only require a bound on the random variables. Indeed, our techniques directly apply to the setting where $R_{\text{min}} = f(V_{\text{min}})$ and $R_{\text{maj}} = f(V_{\text{maj}})$ are only known to satisfy $R_{\text{min}}, R_{\text{maj}} \in [0, 1]$. In particular, Theorem 4.1 applies as stated to random variables with domain $[0, 1]$.

Finally, for verifying fairness properties, we need to compare a ratio of means $\frac{\mu_{R_{\text{min}}}}{\mu_{R_{\text{maj}}}}$ rather than a single mean μ_Z. Prior work has focused on developing inference rules for handling formulas in temporal logics such as continuous stochastic logic (CSL) [Sen et al. 2004; Younes and Simmons 2002] and linear temporal logic (LTL) [Hérault et al. 2004] rather than arithmetic formulas such as ours. The inference rules we develop enable us to do so.

Verifying machine learning systems. More broadly, there has been a large amount of recent work on verifying machine learning systems; the work has primarily focused on verifying robustness properties of deep neural networks [Bastani et al. 2016; Gehr et al. 2018; Goodfellow et al. 2014; Huang et al. 2017; Katz et al. 2017; Raghunathan et al. 2018; Tjeng and Tedrake 2017]. At a high level, robustness can be thought of as an optimization problem (e.g., MAX-SMT), whereas fairness properties involve integration and are therefore more similar to counting problems (e.g., COUNTING-SMT). In general, counting is harder than optimization [Valiant 1979], at least when asking for exact solutions. In our setting, we can obtain high-probability approximations of the counts.

9 CONCLUSION

We have designed an algorithm for verifying fairness properties of machine learning systems. Our algorithm uses a sampling-based approach in conjunction with adaptive concentration inequalities to achieve probabilistic soundness and precision guarantees. As we have shown, our implementation VeriFair can scale to large machine learning models, including a deep recurrent neural network benchmark that is more than six orders of magnitude larger than the largest neural network in the FairSquare benchmark. While we have focused on verifying fairness, we believe that our

[5] We note that Hoeffding's inequality is sometimes called the Chernoff-Hoeffding inequality. It handles an additive error $|\hat{\mu}_Z - \mu_Z| \leq \varepsilon$. The variant of the bound that handles multiplicative error $|\hat{\mu}_Z - \mu_Z| \leq \varepsilon)\mu_Z$ is typically called Chernoff's inequality.

approach of using adaptive concentration inequalities can be applied to verify other probabilistic properties as well.

ACKNOWLEDGMENTS

This work was supported by ONR N00014-17-1-2699.

REFERENCES

Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V Nori. 2017. FairSquare: probabilistic verification of program fairness. In *OOPSLA*.

Solon Barocas and Andrew D Selbst. 2016. Big data's disparate impact. *Cal. L. Rev.* 104 (2016), 671.

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*. 2613–2621.

Dan Biddle. 2006. *Adverse impact and test validation: A practitioner's guide to valid and defensible employment testing*. Gower Publishing, Ltd.

Toon Calders, Faisal Kamiran, and Mykola Pechenizkiy. 2009. Building classifiers with independency constraints. In *ICDMW*. 13–18.

Toon Calders and Sicco Verwer. 2010. Three naive Bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery* 21, 2 (2010), 277–292.

Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. 2017. Optimized Pre-Processing for Discrimination Prevention. In *Advances in Neural Information Processing Systems*. 3995–4004.

Yuansi Chen, Raaz Dwivedi, Martin J Wainwright, and Bin Yu. 2018. Fast MCMC sampling algorithms on polytopes. *The Journal of Machine Learning Research* 19, 1 (2018), 2146–2231.

Guillaume Claret, Sriram K Rajamani, Aditya V Nori, Andrew D Gordon, and Johannes Borgström. 2013. Bayesian inference using data flow analysis. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 92–102.

Edmund M Clarke and Paolo Zuliani. 2011. Statistical model checking for cyber-physical systems. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 1–12.

Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. 2017. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 797–806.

Anupam Datta, Shayak Sen, and Yair Zick. 2017. Algorithmic transparency via quantitative input influence. In *Transparent Data Mining for Big and Small Data*. Springer, 71–94.

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*. ACM, 214–226.

Cynthia Dwork, Nicole Immorlica, Adam Tauman Kalai, and Mark DM Leiserson. 2018. Decoupled Classifiers for Group-Fair and Efficient Machine Learning. In *Conference on Fairness, Accountability and Transparency*. 119–133.

Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542, 7639 (2017), 115.

Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 259–268.

Antonio Filieri, Corina S Păsăreanu, and Willem Visser. 2013. Reliability analysis in symbolic pathfinder. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 622–631.

Benjamin Fish, Jeremy Kun, and Ádám D Lelkes. 2016. A confidence-based approach for balancing fairness and accuracy. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 144–152.

Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 498–510.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *IEEE Symposium on Security and Privacy*.

Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. Psi: Exact symbolic inference for probabilistic programs. In *CAV*.

Jaco Geldenhuys, Matthew B Dwyer, and Willem Visser. 2012. Probabilistic symbolic execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ACM, 166–176.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. In *ICLR*.

Google. 2018. Recurrent Neural Networks for Drawing Classification. https://www.tensorflow.org/versions/master/tutorials/recurrent_quickdraw. Accessed: 2018-04-15.

Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. 2014. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*. ACM, 167–181.

Radu Grosu and Scott A Smolka. 2005. Monte carlo model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 271–286.

David Ha and Douglas Eck. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017).

Sara Hajian and Josep Domingo-Ferrer. 2013. A methodology for direct and indirect discrimination prevention in data mining. *IEEE transactions on knowledge and data engineering* 25, 7 (2013), 1445–1459.

Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of opportunity in supervised learning. In *NIPS*. 3315–3323.

Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. 2004. Approximate probabilistic model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 73–84.

Thomas Herault, Richard Lassaigne, and Sylvain Peyronnet. 2006. APMC 3.0: Approximate verification of discrete and continuous time Markov chains. In *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*. IEEE, 129–130.

Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.

Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. 2017. Peeking at a/b tests: Why it matters, and what to do about it. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1517–1525.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

Niki Kilbertus, Mateo Rojas Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, and Bernhard Schölkopf. 2017. Avoiding discrimination through causal reasoning. In *Advances in Neural Information Processing Systems*. 656–666.

Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. 2017. Inherent trade-offs in the fair determination of risk scores. In *ITCS*.

Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. In *Advances in Neural Information Processing Systems*. 4069–4079.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2002. PRISM: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 200–204.

Himabindu Lakkaraju, Jon Kleinberg, Jure Leskovec, Jens Ludwig, and Sendhil Mullainathan. 2017. The Selective Labels Problem: Evaluating Algorithmic Predictions in the Presence of Unobservables. In *KDD*.

Jim Lawrence. 1991. Polytope volume computation. *Math. Comp.* 57, 195 (1991), 259–271.

Axel Legay, Benoît Delahaye, and Saddek Bensalem. 2010. Statistical model checking: An overview. In *International conference on runtime verification*.

David Monniaux. 2000. Abstract interpretation of probabilistic semantics. In *International Static Analysis Symposium*. Springer, 322–339.

David Monniaux. 2001a. An abstract Monte-Carlo method for the analysis of probabilistic programs. In *ACM SIGPLAN Notices*, Vol. 36. ACM, 93–101.

David Monniaux. 2001b. Backwards abstract interpretation of probabilistic programs. In *European Symposium on Programming*. Springer, 367–382.

Razieh Nabi and Ilya Shpitser. 2018. Fair inference on outcomes. In *AAAI*, Vol. 2018.

Dino Pedreshi, Salvatore Ruggieri, and Franco Turini. 2008. Discrimination-aware data mining. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 560–568.

Aimee Picchi. 2019. Odds of winning $1 billion Mega Millions and Powerball: 1 in 88 quadrillion. *CBS News* (2019). https://www.cbsnews.com/news/odds-of-winning-1-billion-mega-millions-and-powerball-1-in-88-quadrillion

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified defenses against adversarial examples. In *ICLR*.

Adrian Sampson, Pavel Panchekha, Todd Mytkowicz, Kathryn S McKinley, Dan Grossman, and Luis Ceze. 2014. Expressing and verifying probabilistic assertions. In *PLDI*.

Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *PLDI*. 447–458.

Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2004. Statistical model checking of black-box probabilistic systems. In *International Conference on Computer Aided Verification*. Springer, 202–215.

Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2005. On statistical model checking of stochastic systems. In *International Conference on Computer Aided Verification*. Springer, 266–280.

Mallory Simon. 2009. HP looking into claim webcams can't see black people. http://www.cnn.com/2009/TECH/12/22/hp.webcams/index.html

Vincent Tjeng and Russ Tedrake. 2017. Verifying Neural Networks with Mixed Integer Programming. *arXiv preprint arXiv:1711.07356* (2017).

Leslie G Valiant. 1979. The complexity of computing the permanent. *Theoretical computer science* 8, 2 (1979), 189–201.

Abraham Wald. 1945. Sequential tests of statistical hypotheses. *The annals of mathematical statistics* 16, 2 (1945), 117–186.

Min Wen, Osbert Bastani, and Ufuk Topcu. 2019. Fairness with Dynamics. *arXiv preprint arXiv:1901.08568* (2019).

Håkan LS Younes, David J Musliner, et al. 2002. Probabilistic plan verification through acceptance sampling. In *Proceedings of the AIPS-02 Workshop on Planning via Model Checking*. Citeseer, 81–88.

Håkan LS Younes and Reid G Simmons. 2002. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification*. Springer, 223–235.

Håkan LS Younes and Reid G Simmons. 2006. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation* 204, 9 (2006), 1368–1409.

Hakan Lorens Samir Younes. 2004. *Verification and Planning for Stochastic Processes with Asynchronous Events*. Ph.D. Dissertation. Pittsburgh, PA, USA.

Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. 2017. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1171–1180.

Tal Z Zarsky. 2014. Understanding discrimination in the scored society. *Wash. L. Rev.* 89 (2014), 1375.

Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning fair representations. In *International Conference on Machine Learning*. 325–333.

Shengjia Zhao, Enze Zhou, Ashish Sabharwal, and Stefano Ermon. 2016. Adaptive Concentration Inequalities for Sequential Decision Problems. In *NIPS*. 1343–1351.

A PROOFS OF THEORETICAL RESULTS

We prove a number of correctness results for Algorithm 1.

A.1 Proof of Theorem 4.1

First, we have the following well-known definition, which is a key component for the adaptive concentration inequality we use [Zhao et al. 2016].

Definition A.1. *A random variable Z is d-*subgaussian *if $\mu_Z = 0$ and*

$$\mathbb{E}[e^{rZ}] \leq e^{d^2 r^2/2}$$

for all $r \in \mathbb{R}$.

Theorem A.2. *Suppose that Z is a $\frac{1}{2}$-subgaussian random variable with probability distribution $P_{\mathcal{Z}}$. Let*

$$\hat{\mu}_Z^{(n)} = \frac{1}{n}\sum_{i=1}^{n} Z_i,$$

where $\{Z_i \sim P_{\mathcal{Z}}\}_{i\in\mathbb{N}}$ are i.i.d. samples from $P_{\mathcal{Z}}$, let J be a random variable on $\mathbb{N} \cup \{\infty\}$, let

$$\varepsilon_b(n) = \sqrt{\frac{\frac{3}{5}\cdot\log(\log_{11/10} n + 1) + b}{n}}$$

for some constant $b \in \mathbb{R}$, and let $\delta_b = 24e^{-9b/5}$. Then,

$$Pr[J < \infty \wedge (|\hat{\mu}_Z^{(J)}| \geq \varepsilon_b(J))] \leq \delta_b.$$

Using this result, we first prove the following slight variant of Theorem 4.1, which accounts for the case $\Pr[J < \infty] < 1$.

Theorem A.3. *Given a Bernoulli random variable Z with probability distribution $P_{\mathcal{Z}}$, let $\{Z_i \sim P_{\mathcal{Z}}\}_{i\in\mathbb{N}}$ be i.i.d. samples of Z, let*

$$\hat{\mu}_Z^{(n)} = \frac{1}{n}\sum_{i=1}^{n} Z_i,$$

let J be a random variable on $\mathbb{N} \cup \{\infty\}$, and let

$$\varepsilon(\delta, n) = \sqrt{\frac{\frac{3}{5}\cdot\log(\log_{11/10} n + 1) + \frac{5}{9}\cdot\log(24/\delta)}{n}}$$

for a given $\delta \in \mathbb{R}_+$. Then,

$$Pr[J < \infty \wedge (|\hat{\mu}_Z^{(J)} - \mu_Z| \geq \varepsilon(\delta, J))] \leq \delta.$$

Proof. As described in [Zhao et al. 2016], any distribution bounded in an interval of length $2d$ is d-subgaussian. Thus, for any Bernoulli random variable Z, the random variable $Z - \mu_Z$ is $\frac{1}{2}$-subgassian. Then, the claim follows by applying Theorem A.2 (noting that $b = \frac{5}{9}\cdot\log(24/\delta_b)$). □

Note that Theorem 4.1 follows immediately from Theorem A.3 since it assumes that $\Pr[J < \infty] = 1$, so this term can be dropped from the probability event. □

A.2 Proof of Theorem 4.2

We prove by structural induction on the derivation.

Random variable. This case follows by our assumption that the initial environment Γ is correct.

Constant. This case follows by definition since a constant c satisfies $[\![c]\!] = c$.

Sum. By assumption, $|E - [\![X]\!]| \leq \varepsilon$ with probability at least $1 - \delta$, and $|E' - [\![X']\!]| \leq \varepsilon'$ with probability at least $1 - \delta'$. By a union bound, both of these hold with probability at least $1 - (\delta + \delta')$. Then,

$$\begin{aligned} |(E + E') - [\![X + X']\!]| &= |(E + E') - ([\![X]\!] + [\![X']\!])| \\ &\leq |E - [\![X]\!]| + |E' - [\![X']\!]| \\ &\leq \varepsilon + \varepsilon'. \end{aligned}$$

In other words, we can conclude that $X + X' : (E + E', \varepsilon + \varepsilon', \delta + \delta')$.

Negative. By assumption, $|E - [\![X]\!]| \leq \varepsilon$ with probability at least $1 - \delta$. Then,

$$|(-E) - [\![-X]\!])| = |E - [\![X]\!]| \leq \varepsilon$$

In other words, we can conclude that $-X : (-E, \varepsilon, \delta)$.

Product. By assumption, $|E - [\![X]\!]| \leq \varepsilon$ with probability at least $1 - \delta$, and $|E' - [\![X']\!]| \leq \varepsilon'$ with probability at least $1 - \delta'$. a union bound, both of these hold with probability at least $1 - (\delta + \delta')$. Then,

$$\begin{aligned} |E' - E' + [\![X']\!]| &= |E' - E' + [\![X']\!]| \\ &\leq |E'| + | - E' + [\![X']\!]| \\ &\leq |E'| + \varepsilon', \end{aligned}$$

so

$$\begin{aligned} &|E \cdot E' - [\![X \cdot X']\!]| \\ &= |E \cdot E' - [\![X]\!] \cdot [\![X']\!]| \\ &= |E \cdot E' - E \cdot [\![X']\!] + E \cdot [\![X']\!] - [\![X]\!] \cdot [\![X']\!]| \\ &= |E \cdot (E' - [\![X']\!]) + [\![X']\!] \cdot (E - [\![X]\!])| \\ &\leq |E| \cdot |E' - [\![X']\!]| + |[\![X']\!]| \cdot |E - [\![X]\!]| \\ &\leq |E| \cdot \varepsilon' + |[\![X']\!]| \cdot \varepsilon \\ &\leq |E| \cdot \varepsilon' + (|E'| + \varepsilon') \cdot \varepsilon \\ &= |E| \cdot \varepsilon' + |E'| \cdot \varepsilon + \varepsilon \cdot \varepsilon'. \end{aligned}$$

In other words, we can conclude that $X \cdot X' : (E \cdot E', E \cdot \varepsilon' + E' \cdot \varepsilon + \varepsilon \cdot \varepsilon', \delta + \delta')$.

Inverse. By assumption, $|E - [\![X]\!]| \leq \varepsilon$ with probability at least $1 - \delta$. Then,

$$\begin{aligned} |E| &= |E - [\![X]\!] + [\![X]\!]| \\ &\leq |E - [\![X]\!]| + |[\![X]\!]| \\ &\leq \varepsilon + |[\![X]\!]|, \end{aligned}$$

i.e., $|[\![X]\!]| \geq |E| - \varepsilon$, so

$$\begin{aligned} |E^{-1} - [\![X^{-1}]\!]| &= |E^{-1} - [\![X]\!]^{-1}| \\ &= \left| \frac{[\![X]\!] - E}{E \cdot [\![X]\!]} \right| \\ &\leq \frac{\varepsilon}{|E| \cdot |[\![X]\!]|} \\ &\leq \frac{\varepsilon}{|E| \cdot (|E| - \varepsilon)}, \end{aligned}$$

where the last step follows since we have assumed that $|E| - \varepsilon > 0$. In other words, we can conclude that $X^{-1} : (E^{-1}, \frac{\varepsilon}{|E| \cdot (|E| - \varepsilon)}, \delta)$.

Inequality true. By assumption, $|E - [\![X]\!]| \leq \varepsilon$ with probability at least $1 - \delta$, and furthermore $E - \varepsilon \geq 0$. Thus,

$$E - [\![X]\!] \leq \varepsilon,$$

or equivalently,

$$[\![X]\!] \geq E - \varepsilon \geq 0.$$

In other words, we can conclude that $X \geq 0 : (\mathsf{true}, \delta)$.

Inequality false. By assumption, $|E - [\![X]\!]| \leq \varepsilon$ with probability at least $1 - \delta$, and furthermore $E + \varepsilon <$. Thus,

$$[\![X]\!] - E \leq \varepsilon,$$

or equivalently,

$$[\![X]\!] \leq E + \varepsilon < 0.$$

In other words, we can conclude that $X \geq 0 : (\mathsf{false}, \delta)$.

And. By assumption, $[\![Y]\!] = I$ with probability at least $1 - \delta$, and $[\![Y']\!] = I'$ with probability at least $1 - \delta'$. a union bound, both of these hold with probability at least $1 - (\delta + \delta')$. Then,

$$[\![Y \wedge Y']\!] = [\![Y]\!] \wedge [\![Y']\!] = I \wedge I'.$$

In other words, we can conclude that $Y \wedge Y' : (I \wedge I', \delta + \delta')$.

Or. By assumption, $[\![Y]\!] = I$ with probability at least $1 - \delta$, and $[\![Y']\!] = I'$ with probability at least $1 - \delta'$. a union bound, both of these hold with probability at least $1 - (\delta + \delta')$. Then,

$$[\![Y \vee Y']\!] = [\![Y]\!] \vee [\![Y']\!] = I \vee I'.$$

In other words, we can conclude that $Y \vee Y' : (I \vee I', \delta + \delta')$.

Not. By assumption, $[\![Y]\!] = I$ with probability at least $1 - \delta$. Then,

$$[\![\neg Y]\!] = \neg [\![Y]\!] = \neg I.$$

In other words, we can conclude that $\neg Y : (\neg I, \delta)$. □

A.3 Proof of Theorem 4.3

First, we prove the following stronger lemma, which says that as $n \to \infty$ (where n is the number of samples), our algorithm eventually infers arbitrarily tight bounds on any given well-defined problem instance. Then, Theorem 4.3 follows from the applying this lemma to the given specification Y and $\gamma = \Delta$, where $\Delta \in \mathbb{R}_+$ is the given confidence level.

Lemma A.4. *Given any well-defined problem instance $(P_{\mathcal{Z}}, X)$, where $X \in \mathcal{L}(T)$, and any $\delta \in \mathbb{R}_+$, let*

$$\Gamma^{(n)} = \{\mu_Z : (E^{(n)}, \varepsilon(\delta_Z, n), \delta_Z)\}$$

where

$$E^{(n)} = \frac{1}{n}\sum_{i=1}^{n} Z_i$$

$$\varepsilon(\delta_Z, n) = \sqrt{\frac{\frac{3}{5}\cdot\log(\log_{11/10} n + 1) + \frac{5}{9}\cdot\log(24/\delta_Z)}{n}}$$

$$\delta_Z = \delta / [\![X]\!]_\delta.$$

Intuitively, $\Gamma^{(n)}$ is the lemma established for μ_Z on the nth iteration of Algorithm 1. Then, for any $\varepsilon \in \mathbb{R}_+$ and any $\varepsilon_0, \delta_0 \in \mathbb{R}_+$, there exists $n_0 \in \mathbb{N}$ such that for any $n \geq n_0$, with probability at least $1 - \delta_0$, so

$$\Gamma^{(n)} \vdash X : (E, \varepsilon, \delta)$$

for some $E \in \mathbb{R}$ such that $|E - [\![X]\!]| \leq \varepsilon_0$. We are allowed to make the given values $\varepsilon, \delta, \varepsilon_0, \delta_0$ smaller.

Similarly, given any well-defined problem-instance $(P_{\mathcal{Z}}, Y)$, where $Y \in \mathcal{L}(S)$ and any $\gamma \in \mathbb{R}_+$, let

$$\Gamma^{(n)} = \{\mu_Z : (E^{(n)}, \varepsilon(\delta_Z, n), \delta_Z)\}$$

as before. Then, for any $\gamma \in \mathbb{R}_+$ and $\delta_0 \in \mathbb{R}_+$, there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, with probability at least $1 - \delta_0$, so

$$\Gamma^{(n)} \vdash Y : ([\![Y]\!], \gamma).$$

Again, we are allowed to make the given values γ, δ_0 smaller.

Proof. We prove by structural induction on the inference rules in Figure 3, focusing on the following cases of interest: random variables, inverses, and inequalities; the remaining cases follow similarly.

Random variable. Consider the specification μ_Z, and let $\varepsilon, \delta, \varepsilon_0, \delta_0 \in \mathbb{R}_+$ be given. Note that as $n \to \infty$, we have $\varepsilon(\delta_Z, n) \to 0$; furthermore, $\delta_Z = \delta / [\![\mu_Z]\!]_\delta = \delta$. Thus, it suffices to prove that as $E^{(n)} \to \mu_Z$ as $n \to \infty$ as well. To this end, let

$$n_0 = \frac{\log(2/\delta_0)}{2(\varepsilon_0)^2}.$$

By Hoeffding's inequality,

$$\Pr[|E^{(n)} - \mu_Z| \leq \varepsilon_0] \geq 1 - 2e^{-2n\varepsilon_0^2} \geq 1 - 2e^{-2n_0\varepsilon_0^2} = 1 - \delta_0,$$

as claimed.

Inverse. Consider the specification X^{-1}, and let $\varepsilon, \delta, \varepsilon_0, \delta_0 \in \mathbb{R}^+$ be given. Because we have assumed that the problem instance is well-defined, we must have $[\![X]\!] \neq 0$. Let $\alpha = |[\![X]\!]|$, and let

$$\begin{aligned}
\tilde{\varepsilon} &= \min\left\{\frac{\varepsilon \cdot (\alpha/2)^2}{1 + \varepsilon \cdot (\alpha/2)}, \frac{\alpha}{2}\right\} \\
\tilde{\delta} &= \delta \\
\tilde{\varepsilon}_0 &= \min\left\{\frac{\alpha}{4}, \frac{\varepsilon_0 \cdot \alpha^2}{2}\right\} \\
\tilde{\delta}_0 &= \delta_0.
\end{aligned}$$

Note that $\delta_Z = \delta/[\![X^{-1}]\!]_\delta = \tilde{\delta}/[\![X]\!]_\delta$. Therefore, by induction, there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, with probability at least $1 - \tilde{\delta}_0 = 1 - \delta_0$, our algorithm proves the lemma

$$\Gamma^{(n)} \vdash X : (\tilde{E}, \tilde{\varepsilon}, \tilde{\delta}),$$

where $|\tilde{E} - [\![X]\!]| \leq \tilde{\varepsilon}_0$. Then, note that

$$\begin{aligned}
\frac{\alpha}{2} > \tilde{\varepsilon}_0 &\geq |\tilde{E} - [\![X]\!]| \\
&\geq |[\![X]\!]| - |\tilde{E}| \\
&\geq \alpha - |\tilde{E}|,
\end{aligned}$$

from which it follows that

$$|\tilde{E}| > \frac{\alpha}{2} \geq \tilde{\varepsilon}.$$

Thus, the inference rule for inverses applies, so Algorithm 1 proves the lemma

$$\Gamma^{(n)} \vdash X^{-1} : \left(\tilde{E}^{-1}, \frac{\tilde{\varepsilon}}{|\tilde{E}|(|\tilde{E}| - \tilde{\varepsilon})}, \tilde{\delta}\right).$$

Next, note that

$$\tilde{\varepsilon} \leq \frac{\varepsilon \cdot (\alpha/2)^2}{1 + \varepsilon \cdot (\alpha/2)} \leq \frac{\varepsilon \cdot (\alpha/2) \cdot |\tilde{E}|}{1 + \varepsilon \cdot (\alpha/2)},$$

from which it follows that

$$\varepsilon \geq \frac{\tilde{\varepsilon}}{(\alpha/2) \cdot (|\tilde{E}| - \tilde{\varepsilon})} \geq \frac{\tilde{\varepsilon}}{|\tilde{E}|(|\tilde{E}| - \tilde{\varepsilon})}.$$

Furthermore, we have $\tilde{\delta} \leq \delta$. Finally, note that

$$\begin{aligned}
|\tilde{E}^{-1} - [\![X]\!]^{-1}| &= \left|\frac{\tilde{E} - [\![X]\!]}{\tilde{E} \cdot [\![X]\!]}\right| \\
&\leq \frac{\tilde{\varepsilon}_0}{\alpha^2/2} \\
&\leq \varepsilon_0,
\end{aligned}$$

which holds with probability at least $\delta_0 \leq \tilde{\delta}_0$. Note that we can make ε and ε_0 smaller so that

$$\Gamma^{(n)} \vdash X^{-1} : (E, \varepsilon, \delta),$$

where $E = \tilde{E}^{-1}$ satisfies $|E - [\![X^{-1}]\!]| \leq \varepsilon_0$, so the claim follows.

Inequality. Consider the specification $X \geq 0$, and let $\gamma, \delta_0 \in \mathbb{R}_+$ be given. Let $\alpha = |[\![X]\!]|$, and let

$$\begin{aligned}\tilde{\varepsilon} &= \frac{\alpha}{3}\\ \tilde{\delta} &= \gamma\\ \tilde{\varepsilon}_0 &= \frac{\alpha}{3}\\ \tilde{\delta}_0 &= \delta_0.\end{aligned}$$

Note that $\delta_Z = \gamma/[\![X \geq 0]\!]_\delta = \tilde{\delta}/[\![X]\!]_\delta$. Therefore, by induction, there exists $n_0 \in \mathbb{N}$ such that for any $n \geq n_0$, with probability at least $1 - \tilde{\delta}_0 = 1 - \delta_0$, our algorithm proves the lemma

$$\Gamma^{(n)} \vdash X : (\tilde{E}, \tilde{\varepsilon}, \tilde{\delta}),$$

where $|\tilde{E} - [\![X]\!]| \leq \tilde{\varepsilon}_0$. Without loss of generality, assume that $[\![X]\!] \geq 0$ (so $\alpha = [\![X]\!]$). Then, note that

$$\begin{aligned}\tilde{E} &\geq [\![X]\!] - \tilde{\varepsilon}\\ &\geq \frac{2\alpha}{3}\\ &> \tilde{\varepsilon},\end{aligned}$$

so $\tilde{E} - \tilde{\varepsilon} \geq 0$, which implies that the inference rule for true inequalities applies. Thus, our algorithm proves the lemma

$$\Gamma^{(n)} \vdash X : (\mathsf{true}, \tilde{\delta}),$$

where $\tilde{\delta} = \gamma$. Note that since $[\![X]\!] \geq 0$, we have $[\![X \geq 0]\!] = \mathsf{true}$, so the claim follows. □

A.4 Proof of Theorem 5.2

To show that Algorithm 1 terminates with probability 1, it suffices to show that for any $\delta_0 \in \mathbb{R}$, there exists $n_0 \in \mathbb{N}$ such that our algorithm terminates after $n \leq n_0$ steps with probability at least $1 - \delta_0$. Applying Lemma A.4, we have that there exists $n_0 \in \mathbb{N}$ such that with probability at least $1 - \delta_0$, so

$$\Gamma^{(n)} \vdash Y : ([\![Y]\!], \gamma),$$

where $\gamma \leq \Delta$, where Δ is the confidence level given as input to Algorithm 1. Therefore, the claim follows. □

A.5 Proof of Theorem 5.5

For simplicity, we consider the case where there is a single leaf node labeled μ_Z in the given specification Y (so $[\![Y]\!]_\delta = 1$); the general case is a straightforward extension. First, we claim that if Algorithm 1 terminates and returns an incorrect response, then it must be the case that

$$|\hat{\mu}_Z^{(J)} - \mu_Z| > \varepsilon(\delta_Z, J),$$

where

$$\delta_Z = \Delta/[\![Y]\!]_\delta = \Delta,$$

and J is the number of iterations of our algorithm. Suppose to the contrary; then, the lemma

$$\mu_Z : (s/J, \varepsilon_Z(s/n, J), \delta_Z)$$

in Γ on the Jth iteration of our algorithm holds. By Theorem 4.2, we have $\Gamma \vdash Y : (I, \gamma)$ if and only if

$$\Pr[[\![Y]\!] = I] \geq 1 - \gamma.$$

Since Algorithm 1 has terminated, then it must be the case that $\gamma \leq \Delta$. Thus, the response is correct, which is a contradiction, so the claim follows. Then,

$$\begin{aligned}
&\Pr[\text{Algorithm 1 terminates and responds incorrectly}] \\
&\leq \Pr[J < \infty \wedge |\hat{\mu}_Z^{(J)} - \mu_Z| > \varepsilon(\delta_Z, J)] \\
&\leq \delta_Z \\
&\leq \Delta.
\end{aligned}$$

The second inequality follows from Theorem A.3. Thus, Algorithm 1 is probabilistically sound and precise, as claimed. □