FairSquare: Probabilistic Verification of Program Fairness

AWS ALBARGHOUTHI, University of Wisconsin–Madison
LORIS D'ANTONI, University of Wisconsin–Madison
SAMUEL DREWS, University of Wisconsin–Madison
ADITYA NORI, Microsoft Research

With the range and sensitivity of algorithmic decisions expanding at a break-neck speed, it is imperative that we aggressively investigate *fairness and bias* in decision-making programs. First, we show that a number of recently proposed formal definitions of fairness can be encoded as probabilistic program properties. Second, with the goal of enabling rigorous reasoning about fairness, we design a novel technique for verifying probabilistic properties that admits a wide class of decision-making programs. Third, we present FairSquare, the first verification tool for automatically certifying that a program meets a given fairness property. We evaluate FairSquare on a range of decision-making programs. Our evaluation demonstrates FairSquare's ability to verify fairness for a range of different programs, which we show are out-of-reach for state-of-the-art program analysis techniques.

CCS Concepts: • **Mathematics of computing → Probabilistic inference problems**; • **Software and its engineering → Automated static analysis**;

Additional Key Words and Phrases: Algorithmic Fairness, Probabilistic Programming, Probabilistic Inference

ACM Reference Format:
Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *Proc. ACM Program. Lang.* 1, 1, Article 80 (October 2017), 30 pages.
https://doi.org/10.1145/3133904

1 INTRODUCTION

A number of very interesting applications of program analysis have been explored in the probabilistic setting: reasoning about cyber-physical systems [Sankaranarayanan et al. 2013], proving differential privacy of complex algorithms [Barthe et al. 2014], reasoning about approximate programs and hardware [Carbin et al. 2013], synthesizing control programs [Chaudhuri et al. 2014], amongst many others. In this paper, we turn our attention to the problem of *verifying fairness of decision-making programs*.

Program bias As software permeates our personal lives, corporate world, and bureaucracy, more and more of our critical decisions are being delegated to opaque algorithms. Software has thus become a powerful arbitrator of a range of significant decisions with far-reaching societal impact—hiring [Kobie 2016; Miller 2015], welfare allocation [Eubanks 2015], prison sentencing [Angwin

Authors' addresses: A. Albarghothi, L. D'Antoni, S. Drews, Department of Computer Sciences, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI, 53706, US; A. Nori, Microsoft Research Cambridge, 21 Station Road Cambridge CB1 2FB United Kingdom.

2475-1421/2017/10-ART80
https://doi.org/10.1145/3133904

et al. 2016], policing [Berg 2014; Perry 2013], amongst many others. With the range and sensitivity of algorithmic decisions expanding by the day, the problem of understanding the nature of program bias is a pressing one: Indeed, the notion of *algorithmic fairness* has recently captured the attention of a broad spectrum of experts, within computer science and without [Ajunwa et al. 2016; Angwin et al. 2016; Barocas and Selbst 2014; Calders and Verwer 2010; Datta et al. 2015; Dwork et al. 2012; Feldman et al. 2015; Sweeney 2013; Tutt 2016; Valentino-Devries et al. 2012; Zemel et al. 2013].

Fairness and justice have always been ripe topics for philosophical debate [Rawls 2009], and, of course, there are no established rigorous definitions. Nonetheless, the rise of automated decision-making has prompted the introduction of a number of formal definitions of fairness, and their utility within different contexts is being actively studied and contested [Dwork et al. 2012; Feldman et al. 2015; Friedler et al. 2016; Hardt et al. 2016; Kleinberg et al. 2017; Ruggieri 2014]. Notable formulations of fairness include *individual fairness*, which dictates that *similar* inputs must result in *similar* outputs; and *group fairness*, which dictates that a particular subset of inputs must have a similar aggregate output to the whole. In this paper, we view such notions of fairness as *probabilistic specifications* of decision-making programs.

Fairness as a probabilistic specification We think of decision-making algorithms as probabilistic programs, in the sense that they are invoked on inputs drawn from a probability distribution, e.g., representing the demographics of some population. Fairness properties are then formalized as probabilistic specifications to which the decision-making program needs to adhere.

Consider a hiring program P that takes as input a vector of arguments $\boldsymbol{v}$ representing a job applicant's record and returns a Boolean value indicating whether the applicant is hired. One of the arguments v_s in the vector $\boldsymbol{v}$ states whether the person is a member of a protected minority or not, and similarly v_q in $\boldsymbol{v}$ states whether the person is qualified or not for the job. Our goal may be to prove a group fairness property that is augmented with a notion of qualification—that the algorithm is *just as likely* to hire a qualified minority applicant as it is for other qualified non-minority applicants. Formally, we state this probabilistic condition as follows:

$$\frac{\mathbb{P}[P(\boldsymbol{v}) = \mathit{true} \mid v_s = \mathit{true} \wedge v_q = \mathit{true}]}{\mathbb{P}[P(\boldsymbol{v}) = \mathit{true} \mid v_s = \mathit{false} \wedge v_q = \mathit{true}]} > 1 - \epsilon$$

Here, ϵ is a small constant. In other words, the probability of hiring a person $\boldsymbol{v}$, conditioned on them being a qualified minority applicant, is very close to (or greater than) the probability of hiring a person conditioned on them being a qualified non-minority applicant. We note that, while most recent concerns of fairness have focused on automation of bureaucratic processes, e.g., employment and loan applications, our view of the problem is broad. For instance, fairness properties can be extended to actions and decisions of autonomous agents, like robots and self-driving cars, that interact with us and affect our environment.

Automated fairness verification We envision a future in which those who employ algorithmic decision-making in sensitive domains are required to prove fairness of their processes. Towards this vision, our goal in this paper is to develop an automated technique that can prove fairness properties of programs, like the one shown above, as well as others. With that in mind, we have two key criteria: First, we require a technique that can *construct a proof* of fairness or unfairness of a given program with respect to a specified fairness property. Second, we need to ensure that our technique can handle real-world classes of decision-making programs.

Since our aim is to construct proofs of fairness or unfairness, we focus our development on *exact* probabilistic verification techniques, in contrast with approximate techniques that may provide probabilistic guarantees. We first attempted to reason about fairness using a range of recent probabilistic static analysis techniques that provide exact guarantees [Gehr et al. 2016;

Sankaranarayanan et al. 2013], but we observed that these existing techniques are unable to handle the programs and properties we consider. We therefore set out to design a new technique that is suited for our domain of verifying fairness of decision-making programs.

We first observe that many decision-making programs—e.g., machine-learned classifiers—can be encoded logically as formulas in real arithmetic. Since our goal is to verify a probabilistic property of programs, we propose an automated technique that reduces our probabilistic verification problem to that of computing the *weighted volume* of the logical encoding of a program in real arithmetic (i.e., computing the probability of picking an assignment that satisfies the formula). To enable automatic construction of proofs, we propose a novel *symbolic-volume-computation* algorithm that exploits the power of SMT solvers to compute the weighted volume of formulas in real arithmetic. We show that our algorithm monotonically converges to the *exact* probabilities in the limit, thus resulting in a *sound and complete fairness verification procedure*. To our knowledge, this is the first probabilistic-inference algorithm for arithmetic SMT theories with this expressivity and guarantees.

FairSquare We implement our algorithm in a new tool called FairSquare. We evaluate FairSquare on a number of decision-making programs generated by a range of machine-learning algorithms from real-world data. Our evaluation demonstrates FairSquare's ability to prove/disprove fairness properties for a range of decision-making programs. Furthermore, our evaluation highlights the importance of our algorithmic contributions and design decisions in the fairness context: for example, we demonstrate how state-of-the-art, general-purpose probabilistic program analysis tools are unable to handle the majority of our benchmarks.

Contributions This paper makes a number of conceptual, algorithmic, and practical contributions:

- We frame fairness properties of programs as correctness properties in the context of program verification. Specifically, we show that a number of formal definitions of fairness can be cast as probabilistic specifications of decision-making programs. (Sec. 3)
- Motivated by the structure of decision-making programs, we address the problem of automating fairness verification by reducing it to a set of weighted-volume-computation problems. We present a novel weighted-volume-computation algorithm, for formulas over *real closed fields*, that utilizes an SMT solver as a black box, and we prove that it converges to the exact volume in the limit. (Sec. 4)
- We implement our technique in a new tool called FairSquare and use it to verify a class of fairness properties for a broad spectrum of decision-making programs generated from real-world datasets. Our evaluation demonstrates the power of our technique in the domain of fairness verification and its ability to outperform state-of-the-art probabilistic program analyses. (Sec. 6)

2 OVERVIEW AND ILLUSTRATION

Our problem setting is as follows: First, we are given a *decision-making program* $P_{\mathtt{dec}}$. Second, we have a *probabilistic precondition* defining a probability distribution over inputs of $P_{\mathtt{dec}}$. We define the probability distribution operationally as a probabilistic program $P_{\mathtt{pop}}$, which we call the *population model*. Intuitively, the population model provides a probabilistic picture of the population from which the inputs of $P_{\mathtt{dec}}$ are drawn. Third, we are given a quantitative postcondition $\varphi_{\mathtt{post}}$ that correlates the probabilities of various program outcomes. This postcondition can encode various fairness properties. Intuitively, our goal is to prove the following triple:

$$\{\boldsymbol{v} \sim P_{\mathtt{pop}}\} \quad r \leftarrow P_{\mathtt{dec}}(\boldsymbol{v}) \quad \{\varphi_{\mathtt{post}}\}$$

In this section, we consider a specific fairness property. We will discuss in Sec. 3 how several formulations of fairness can be captured by our framework.

(a)

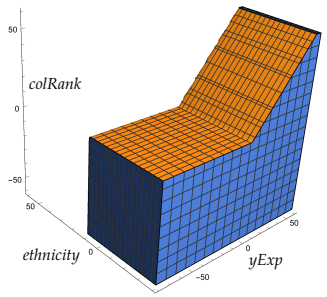
The *population model* defines a joint probability distribution on attributes of members of a population: (*i*) the rank of the college a person attended (`colRank`), (*ii*) the years of work experience they have (`yExp`), and (*iii*) their ethnicity (`ethnicity`). ***Note that*** `colRank` ***is influenced by a persons's ethnicity***.

```
1 define popModel()
2   ethnicity ~ gauss(0,10)
3   colRank ~ gauss(25,10)
4   yExp ~ gauss(10,5)
5   if (ethnicity > 10)
6     colRank ← colRank + 5
7   return colRank, yExp
```

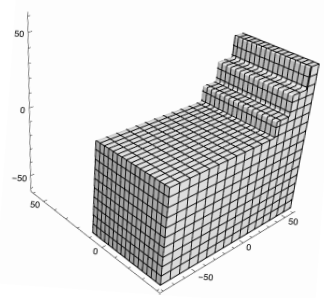
The *decision-making* program takes an applicant's record and decides whether to hire them. A person is hired if they are from a *top-5* college (`colRank <= 5`) or have lots of experience compared to their college rank (`expRank > -5`). ***Note that this program does not access an applicant's ethnicity***.

```
1 define dec(colRank, yExp)
2   expRank ← yExp - colRank
3   if (colRank <= 5)
4     hire ← true
5   elif (expRank > -5)
6     hire ← true
7   else
8     hire ← false
9   return hire
```

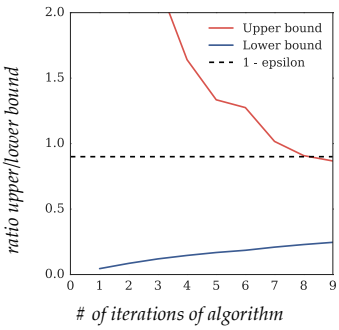
(b) Formula φ in $\mathbb{R}^3$ (blue faces are unbounded)



(c) Underapproximation of φ as a union of hyperrectangles



(d) FairSquare ratio computation on `dec` and `popModel` (unfair)



(e) FairSquare ratio computation on modified `dec` (fair)

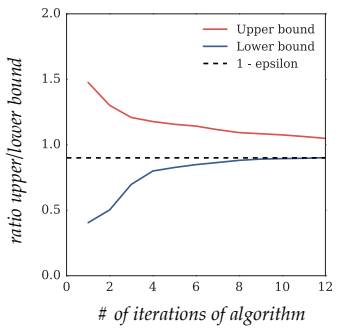


Fig. 1. Simple illustrative example

A simple verification problem Consider the two programs in Figure 1(a). The program `popModel` is a probabilistic program describing a simple model of the population. Here, a member of the population has three attributes, all of which are real-valued: (*i*) `ethnicity`; (*ii*) `colRank`, the ranking of the college the person attended (lower is better); and (*iii*) `yExp`, the years of work experience a person has. We consider a person to be a member of a protected group if `ethnicity > 10`; we call this the *sensitive condition*. The population model can be viewed as a *generative model* of records of individuals—the more likely a combination is to occur in the population, the more likely it will be generated. For instance, the years of experience an individual has (line 4) follows a Gaussian (normal) distribution with mean 10 and standard deviation 5. Observe that our model specifies that members of a protected minority will probably attend a lower-ranked college, as encoded in lines 5-6.

The program `dec` is a decision-making program that takes a job applicant's college ranking and years of experience and decides whether they get hired. The program implements a simple decision tree, perhaps one generated by a machine-learning algorithm or written by a person. A person is hired if they attended a *top-5* college (`colRank <= 5`) or have lots of experience compared to their college's ranking (`expRank > -5`). Observe that `dec` *does not access an applicant's ethnicity.*

Our goal is to establish whether the hiring algorithm `dec` discriminates against members of the protected minority. Concretely, we attempt to prove the following property:

$$\frac{\mathbb{P}[\texttt{hire} \mid \texttt{min}]}{\mathbb{P}[\texttt{hire} \mid \neg\texttt{min}]} > 1 - \epsilon$$

where `min` is shorthand for the sensitive condition `ethnicity > 10`, and ϵ is a small parameter set to 0.1 for purposes of illustration. Despite the potential shortcomings of this group-fairness property [Dwork et al. 2012], its simple formulation serves well as an illustration of our technique.

We can rewrite the above statement to eliminate conditional probabilities as follows:

$$\frac{\mathbb{P}[\texttt{hire} \wedge \texttt{min}]/\mathbb{P}[\texttt{min}]}{\mathbb{P}[\texttt{hire} \wedge \neg\texttt{min}]/\mathbb{P}[\neg\texttt{min}]} > 1 - \epsilon \tag{1}$$

Therefore, to prove the above statement, we need to compute a value for each of the probability terms: $\mathbb{P}[\texttt{hire} \wedge \texttt{min}]$, $\mathbb{P}[\texttt{min}]$, and $\mathbb{P}[\texttt{hire} \wedge \neg\texttt{min}]$. (Note that $\mathbb{P}[\neg\texttt{min}] = 1 - \mathbb{P}[\texttt{min}]$.) Observe that, to prove or disprove inequality 1, all we need are sufficiently precise bounds on probabilities—not their exact values.

For the purposes of illustration, we shall focus our description on computing $\mathbb{P}[\texttt{hire} \wedge \neg\texttt{min}]$.

Probabilistic verification conditions To compute the probability $\mathbb{P}[\texttt{hire} \wedge \neg\texttt{min}]$, we need to reason about the *composition* of the two programs, $\texttt{dec} \circ \texttt{popModel}$. That is, we want to compute the probability that (*i*) `popModel` generates a non-minority applicant, and (*ii*) `dec` hires that applicant. To do so, we begin by encoding both programs as formulas in the linear-real-arithmetic theory of first-order logic. The process is analogous to that of standard *verification-condition generation* for loop-free program fragments.

First, we encode `popModel` as follows:

$$\varphi_{\texttt{pop}} \equiv (ethnicity > 10 \Rightarrow colRank_1 = colRank + 5) \ \wedge \ (ethnicity \leqslant 10 \Rightarrow colRank_1 = colRank)$$

where subscripts are used to encode multiple occurrences of the same variable (i.e., SSA form). Note that assignments drawn from probability distributions do not appear in the encoding—we shall address them later.

Second, we encode `dec` as follows (after simplification):

$$\varphi_{\texttt{dec}} \equiv expRank = yExp^i - colRank^i \ \wedge \ (hire \iff (colRank^i \leqslant 5 \vee expRank > -5))$$

where variables with the superscript i are the input arguments to `dec`. Now, to encode the composition `dec` $\circ$ `popModel`, we simply conjoin the two formulas—$\varphi_{\texttt{pop}}$ and $\varphi_{\texttt{dec}}$—and add equalities between returns of `popModel` and arguments of `dec`.

$$\varphi_P \equiv \varphi_{\texttt{pop}} \wedge \varphi_{\texttt{dec}} \wedge \mathit{yExp}^i = \mathit{yExp} \wedge \mathit{colRank}^i = \mathit{colRank}_1$$

Our goal is to compute the probability that a non-minority applicant gets hired. Formally, we are asking, *what is the probability that the following formula is satisfied?*

$$\varphi \equiv \exists V_d.\, \varphi_P \wedge \mathit{hire} \wedge \mathit{ethnicity} \leqslant 10$$

V_d is the set of variables that are not probabilistically assigned to—that is, all variables *other than* the three variables $V_p = \{\mathit{ethnicity}, \mathit{colRank}, \mathit{yExp}\}$. Intuitively, by *projecting out* all non-probabilistic variables with existential quantifiers, we get a formula φ whose models are the set of all probabilistic samplings that lead to a non-minority applicant being generated and hired.

Weighted volume computation To compute the probability that φ is satisfied, we begin by noting that φ is, geometrically, a region in $\mathbb{R}^3$, because it has three free, real-valued variables, V_p. The region φ is partially illustrated in Figure 1(b). Informally, the probability of satisfying φ is the probability of drawing values for the variables in V_p that end up falling in the region described by φ. Therefore, the probability of satisfying φ is its *volume* in $\mathbb{R}^3$, *weighted* by the probability density of each of the three variables. Formally:

$$\mathbb{P}[\texttt{hire} \wedge \neg\texttt{min}] = \int_{\varphi} p_e p_y p_c \; dV_p$$

where, e.g., p_e is the *probability density function* of the distribution `gauss(0,10)`—the distribution from which the value of `ethnicity` is drawn in line 2 of `popModel`. Specifically, p_e is a function of $\mathit{ethnicity}$, namely, $p_e(\mathit{ethnicity}) = \frac{1}{10\sqrt{2\pi}} e^{-\frac{\mathit{ethnicity}^2}{200}}$.

The primary challenge here is that the region of integration is specified by an arbitrary SMT formula over an arithmetic theory. So, *how do we compute a numerical value for this integral?* We make two interdependent observations: (*i*) if the formula represents a *hyperrectangular region* in $\mathbb{R}^n$—i.e., a box—then integration is typically simple, due to the constant upper/lower bounds of all dimensions; (*ii*) we can *symbolically decompose* an SMT formula into an (infinite) set of hyperrectangles.

Specifically, given our formula φ, we construct a new formula, $\boxplus_\varphi$, where each model $m \models \boxplus_\varphi$ corresponds to a hyperrectangle that underapproximates φ. Therefore, by systematically finding disjoint hyperrectangles inside of φ and computing their weighted volume, we iteratively improve a *lower bound* on the exact weighted volume of φ. Figure 1(c) shows a possible underapproximation of φ composed of four hyperrectangles. The hyperrectangles form a ladder shape that underapproximates the slanted face of φ. We can analogously compute an *upper bound* on the weighted volume of φ: we simply find a lower bound for $\neg\varphi$ and apply the fact that $\mathbb{P}[\varphi] = 1 - \mathbb{P}[\neg\varphi]$. Sec. 4 formalizes this technique and proves its convergence for decidable arithmetic theories.

Proofs of group fairness We demonstrated how our technique reduces the problem of computing probabilities to weighted volume computation. Figure 1(d) illustrates a run of our tool, FairSquare, on this example. FairSquare iteratively improves lower and upper bounds for the probabilities in the ratio, and, therefore, the ratio itself. Observe how the upper bound (red) of the ratio is decreasing and its lower bound (blue) is increasing. This example is not group fair for $\epsilon = 0.1$, because the upper bound goes below 0.9.

Recall that applicants of a protected minority tend to attend lower-ranked colleges, as defined by `popModel`. Looking at `dec`, we can point out that the cause for unfairness is the importance of college ranking for hiring. Let us attempt to fix this by modifying line 2 of `dec` to

```
expRank ← 5*yExp - colRank
```

In other words, we have made the algorithm value job experience far more than college ranking. The run of FairSquare on the modified `dec` is shown in Figure 1(e), where the lower bound on the ratio exceeds 0.9, thus proving our group fairness property.

3 A FRAMEWORK FOR VERIFYING FAIRNESS PROPERTIES

In this section, we formally define our program model, show how a number of fairness properties can be modeled as probabilistic properties, and present a general framework for specifying and verifying such properties.

3.1 Program model and semantics

Programs A program P is a sequence of statements S:

$$
\begin{array}{rl}
S \coloneqq & V \leftarrow E \qquad \text{assignment statement} \\
\mid & V \sim \mathit{Dist} \qquad \text{probabilistic assignment} \\
\mid & \texttt{if}\ B\ \texttt{then}\ S\ \texttt{else}\ S \qquad \text{conditional} \\
\mid & S; S \qquad \text{sequence of statements}
\end{array}
$$

where V is the set of *real-valued variables* that can appear in P, $e \in E$ is an arithmetic expression over variables in V, and $b \in B$ is a Boolean expression over variables in V. A probabilistic assignment is made by sampling from a probability distribution $p \in \mathit{Dist}$. A probability distribution can be, for example, a *Gaussian distribution*, denoted by $\texttt{gauss}(\mu, \sigma)$, where $\mu, \sigma \in \mathbb{R}$ are the mean and standard deviation of the Gaussian. Without loss of generality, we shall restrict distributions to be *univariate*. We will also assume distributions have only constant parameters, e.g., mean and standard deviation of a Laplacian or Gaussian—that is, we assume independence of probabilistic assignments.[1] Given a probabilistic assignment $x \sim p$, we shall treat $p(x)$ as a *probability density function* (PDF) of the distribution from which the value assigned to x is drawn. For instance, if the distribution p is `gauss(0,1)`, then $p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

We use $\boldsymbol{v}_i$ to denote a vector of *input variables* of P, and $\boldsymbol{v}_o$ to denote a vector of *output variables* of P; these variables appear in V and denote the arguments and returns of P. We say that a program is *closed* if it has no inputs, i.e., $\boldsymbol{v}_i$ is empty. We shall refer to the following subsets of V.

- $V_p \subseteq V$ is the set of *probabilistic variables*: those that get assigned to in probabilistic assignments.
- $V_d = V \setminus V_p$ is the set of *deterministic variables*: those that do not appear in probabilistic assignments.

This simple language can be used to describe typical machine-learning classifiers such as decision trees, support vector machines, Bayesian networks, neural networks, as well as loop-free probabilistic programs (loops with constant bounds can be unrolled). As demonstrated in Sec. 2, the same language is used to define population models programmatically.

Operational semantics The operational semantics of our program model is standard, following those introduced by Kozen [1981] and used by other recent papers on the topic [Chistikov et al.

[1] Gaussian distributions with non-constant parameters can be handled through properties of Gaussians. E.g., $y \sim \texttt{gauss}(x, \sigma)$, where $x \in V$ and $\sigma \in \mathbb{R}$, can be transformed into an equivalent sequence of assignments $y \sim \texttt{gauss}(0, \sigma); y \leftarrow y + x$.

2015; Sampson et al. 2014; Sankaranarayanan et al. 2013]. We refer the reader to these texts for an account of the semantics.

3.2 Fairness as a probabilistic program property

We now formalize probabilistic pre- and postconditions and use them to define the probabilistic verification problem. We then show how many fairness definitions can be expressed in our verification framework.

Probabilistic verification problems A verification problem is a triple $(P_{\texttt{pop}}, P_{\texttt{dec}}, \varphi_{\texttt{post}})$, where

- $P_{\texttt{pop}}$, called the *population model*, is a closed program over variables $V^{\texttt{pop}}$ and output variables $\boldsymbol{v}_o^{\texttt{pop}}$.
- $P_{\texttt{dec}}$, called the *decision-making program*, is an open program over variables $V^{\texttt{dec}}$; its input arguments are $\boldsymbol{v}_i^{\texttt{dec}}$, with $|\boldsymbol{v}_i^{\texttt{dec}}| = |\boldsymbol{v}_o^{\texttt{pop}}|$; and its output variables are $\boldsymbol{v}_o^{\texttt{dec}}$. (We assume that $V^{\texttt{pop}} \cap V^{\texttt{dec}} = \emptyset$.)
- $\varphi_{\texttt{post}}$ is a *probabilistic postcondition*, which is a Boolean expression over probabilities of program outcomes. Specifically, $\varphi_{\texttt{post}}$ is defined as follows:

$$\varphi_{\texttt{post}} \in \text{BExp} \coloneqq \text{PExp} > c \mid \text{BExp} \wedge \text{BExp} \mid \neg\text{BExp}$$
$$\text{PExp} \coloneqq \mathbb{P}[\varphi] \mid c \mid \text{PExp}\ \{+, -, \div, \times\}\ \text{PExp}$$

  where $c \in \mathbb{R}$ and φ is a linear arithmetic formula over input and output variables of $P_{\texttt{dec}}$; e.g., $\varphi_{\texttt{post}}$ might be of the form

$$\mathbb{P}[x > 0] > 0.5 \wedge \mathbb{P}[y + z > 7] - \mathbb{P}[t > 5] > 0$$

The goal of verification is to prove that $\varphi_{\texttt{post}}$ is true for the program $P_{\texttt{dec}} \circ P_{\texttt{pop}}$, i.e., the composition of the two programs where we first run $P_{\texttt{pop}}$ to generate an input for $P_{\texttt{dec}}$. Since $P_{\texttt{pop}}$ is closed, $P_{\texttt{dec}} \circ P_{\texttt{pop}}$ is also closed. To avoid division-by-zero problems, we assume that divisors never have value zero. We will use the following definition when stating the meta-properties of our algorithm: we say that the postcondition is *robust* iff, for any subformula of the form PExp $> c$, the value of the expression PExp is not exactly c.

Fairness properties We now show how prominent fairness definitions from the literature can be encoded as probabilistic postconditions. At a high-level, all proposed fairness definitions aim to ensure fair decision making, and while some focus on fairness at the granularity of groups, others focus on fairness at the individual level.

We first consider group fairness formulations. Feldman et al. [2015] introduced the following definition, inspired by *Equality of Employment Opportunity Commission's* recommendation in the US [EEOC 2014]:

$$\frac{\mathbb{P}[r = \mathit{true} \mid \mathit{min}(\boldsymbol{v}) = \mathit{true}]}{\mathbb{P}[r = \mathit{true} \mid \mathit{min}(\boldsymbol{v}) = \mathit{false}]} > 1 - \epsilon$$

Assuming $P_{\texttt{dec}}$ returns a Boolean value r—indicating whether an applicant $\boldsymbol{v}$ is hired—this *group fairness* property states that the selection rate from a protected minority group, $\mathit{min}(\boldsymbol{v}) = \mathit{true}$, is as good as the selection rate from the rest of the population. One can thus view this verification problem as proving a probabilistic property involving *two sets of program traces*: one set where the input $\mathit{min}(\boldsymbol{v})$ is true, and another where it is false. Alternatively, the above definition could be strengthened by conjoining that the reciprocal of the ratio is also at least $1 - \epsilon$, thus ensuring that the selection rate of the two groups is nearly the same (*demographic parity*). Further, we could additionally condition on *qualified* individuals, e.g., if the job has some minimum qualification, we do not want to characterize group fairness for arbitrary applicants, but only within the qualified

subpopulation. Various comparable notions of group fairness have been proposed and used in the literature, e.g., [Datta et al. 2016; Feldman et al. 2015; Zemel et al. 2013].

While the above definition is concerned with fairness at the level of subsets of the domain of the decision-making program, *individual fairness* [Dwork et al. 2012] is concerned with similar outcomes for similar elements of the domain. In our hiring example, one potential formulation is as follows:

$$\mathbb{P}[r_1 = r_2 \mid \boldsymbol{v}_1 \sim \boldsymbol{v}_2] > 1 - \epsilon$$

In other words, for any two similar individuals (denoted $\boldsymbol{v}_1 \sim \boldsymbol{v}_2$), we want them to receive similar outcomes ($r_1 = r_2$) with a high probability. This is a *hyperproperty*—as it considers two copies of P_{dec}—and can be encoded through *self-composition* [Barthe et al. 2004]. This property is close in nature to differential privacy [Dwork 2006] and robustness [Bastani et al. 2016; Chaudhuri et al. 2011].

Of course, various definitions of fairness have their merits, shortcomings, and application domains, and there is an ongoing discussion on this subject [Ajunwa et al. 2016; Dwork et al. 2012; Feldman et al. 2015; Friedler et al. 2016; Hardt et al. 2016]. Our contribution is not to add to this debate, but to cast fairness as a quantitative property of programs, and therefore enable automated reasoning about fairness of decision-making programs.

3.3 Probabilistic inference through volume computation

Now that we have defined our program model and the properties we are interested in verifying, we switch attention to constructing *probabilistic verification conditions*.

Following Chistikov et al. [2015], we reduce the problem of computing the probability that the program terminates in a state satisfying φ to *weighted volume computation* (WVC) over formulas describing regions in $\mathbb{R}^n$. In what follows, we begin by formalizing the WVC problem.

Volume of a formula We will use $\mathcal{L}$ to denote first-order formulas in *linear real arithmetic* and the strictly richer *real closed fields*—Boolean combinations of polynomial inequalities. Given a formula $\varphi \in \mathcal{L}$, a model m of φ, denoted by $m \models \varphi$, is a point in $\mathbb{R}^n$, where n is the number of free variables of φ. Thus, we view φ as a region in $\mathbb{R}^n$, i.e., $\varphi \subseteq \mathbb{R}^n$. We use $X_\varphi = \{x_1, \ldots, x_n\}$ to denote the free variables of φ.

The (unweighted) *volume* of a formula φ is $\int_\varphi 1 \, dX_\varphi$, where dX_φ is short for $dx_1 dx_2 \ldots dx_n$. For example, if φ is in $\mathbb{R}^2$, then $\int_\varphi 1 \, dX_\varphi$ is the area of φ.

Weighted volume of a formula We now define the *weighted volume* of a formula. We assume we are given a pair (φ, D), where $\varphi \in \mathcal{L}$ and $D = \{p_1, \ldots, p_n\}$ is a set of probability density functions such that each variable $x_i \in X_\varphi$ is associated with a density function $p_i(x_i)$ of the probability distribution of its values. The weighted volume of φ with respect to D, denoted by $\textsc{vol}(\varphi, D)$, is defined as follows:

$$\int_\varphi \prod_{x_i \in X_\varphi} p_i(x_i) \, dX_\varphi$$

Example 3.1. Consider the formula $\varphi \equiv x_1 + x_2 \geqslant 0$, and let $D = \{p_1, p_2\}$, where p_1 and p_2 are the PDF of the Gaussian distribution with mean 0 and standard deviation 1. Then,

$$\textsc{vol}(\varphi, D) = \int_{x_1 + x_2 \geqslant 0} p_1(x_1) p_2(x_2) \, dx_1 dx_2 = 0.5$$

Intuitively, if we are to randomly draw two values for x_1 and x_2 from the Gaussian distribution, we will land in the region $x_1 + x_2 \geqslant 0$ with probability 0.5. ■

$$\frac{}{\langle x = [\![e]\!], \emptyset\rangle \rhd x \leftarrow e}\ \textsc{vc-asn} \qquad \frac{}{\langle true, \{p_i\}\rangle \rhd x_i \sim p_i}\ \textsc{vc-pasn}$$

$$\frac{\langle \varphi_1, D_1\rangle \rhd S_1 \qquad \langle \varphi_2, D_2\rangle \rhd S_2}{\langle \varphi_1 \wedge \varphi_2, D_1 \cup D_2\rangle \rhd S_1 S_2}\ \textsc{vc-seq}$$

$$\frac{\langle \varphi_1, D_1\rangle \rhd S_1 \qquad \langle \varphi_2, D_2\rangle \rhd S_2}{\langle ite([\![b]\!], \varphi_1, \varphi_2), D_1 \cup D_2\rangle \rhd \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2}\ \textsc{vc-cond}$$

Fig. 2. Probabilistic verification condition generation. Above, $ite(a, b, c) \triangleq (a \Rightarrow b) \wedge (\neg a \Rightarrow c)$.

Probabilistic verification conditions Recall that our goal is to compute the probability of some predicate φ at the end of a program execution, denoted $\mathbb{P}[\varphi]$. We now show how to encode this problem as weighted volume computation. First, we encode program executions as a formula φ_P. The process is similar to standard verification condition generation (as used by verification [Barnett and Leino 2005] and bounded model checking tools [Clarke et al. 2004]), with the difference that probabilistic assignments populate a set D of probability density functions.

Figure 2 inductively defines the construction of a *probabilistic verification condition* for a program P, denoted by a function $\textsc{pvc}(P)$, which returns a pair $\langle \varphi_P, D\rangle$. Without loss of generality, to simplify our exposition, we assume programs are in *static single assignment* (SSA) form [Cytron et al. 1991]. Given a Boolean expression b, the denotation $[\![b]\!]$ is the same expression interpreted as an $\mathcal{L}$ formula. The same applies to arithmetic expressions e. For example, $[\![\texttt{x + y > 0}]\!] \triangleq x + y > 0$. Intuitively, the construction generates (*i*) a formula φ_P that encodes program executions, treating probabilistic assignments as non-deterministic, and (*ii*) a set D of the PDFs of distributions in probabilistic assignments (rule VC-PASN).

Now, suppose we are given a closed program P and a Boolean formula φ over its output variables. Then,

$$\mathbb{P}[\varphi] = \textsc{vol}(\exists V_d.\, \varphi_P \wedge \varphi, D)$$

That is, we project out all non-probabilistic variables V_d from $\varphi_P \wedge \varphi$ and compute the weighted volume with respect to the densities $p_i \in D$. Intuitively, each model m of $\exists V_d.\, \varphi_P \wedge \varphi$ corresponds to a sequence of values drawn in probabilistic assignments in an execution of P. We note that our construction is closely related to that of Chistikov et al. [2015], to which we refer the reader for a measure-theoretic formalization.

Example 3.2. Consider the following closed program P:

```
x ~ gauss(0,2);
y ~ gauss(-1,1);
z ← x + y
```

where z is the return variable. Using the encoding in Figure 2, we compute the pair $\langle \varphi_P, D\rangle \rhd P$, where $\varphi_P \triangleq z = x + y$ and $D = \{p_x, p_y\}$, where p_x and p_y are the PDFs of the two distributions from which values of x and y are drawn.

Suppose that we would like to compute the probability that z is positive when the program terminates: $\mathbb{P}[z \geqslant 0]$. Then, we can compute the following weighted volume: $\textsc{vol}(\exists z.\, \varphi_P \wedge z \geqslant 0, D)$, which is ~ 0.327. ■

1: **function** VERIFY($P_{\texttt{pop}}, P_{\texttt{dec}}, \varphi_{\texttt{post}}$)
2: $\langle \varphi_{\texttt{pop}}, D_{\texttt{pop}} \rangle \leftarrow \textsc{pvc}(P_{\texttt{pop}})$
3: $\langle \varphi_{\texttt{dec}}, D_{\texttt{dec}} \rangle \leftarrow \textsc{pvc}(P_{\texttt{dec}})$
4: $\langle \varphi_P, D \rangle \leftarrow \langle \varphi_{\texttt{pop}} \wedge \varphi_{\texttt{dec}} \wedge \boldsymbol{v}_i^{\texttt{dec}} = \boldsymbol{v}_o^{\texttt{pop}}, D_{\texttt{pop}} \cup D_{\texttt{dec}} \rangle$
5: $V_d \leftarrow V_d^{\texttt{pop}} \cup V_d^{\texttt{dec}}$
6: $m \leftarrow \emptyset$
7: **for each** expression $\mathbb{P}[\varphi]$ in $\varphi_{\texttt{post}}$ **do**
8: $m \leftarrow m[\mathbb{P}[\varphi] \mapsto \textsc{vol}(\exists V_d.\, \varphi_P \wedge \varphi, D)]$
9: **return** $m \models \varphi_{\texttt{post}}$

Fig. 3. Abstract verification algorithm

Verification algorithm We now describe an idealized verification algorithm that assumes the existence of an oracle VOL for measuring probability expressions appearing in the postcondition (we formally define these quantities in Sec. 3.3). The algorithm VERIFY, shown in Figure 3, takes a verification problem and returns whether the probabilistic postcondition holds.

VERIFY begins by encoding the composition of the two programs, $P_{\texttt{dec}} \circ P_{\texttt{pop}}$, as the pair $\langle \varphi_P, D \rangle$ and adds the constraint $\boldsymbol{v}_i^{\texttt{dec}} = \boldsymbol{v}_o^{\texttt{pop}}$ to connect the outputs of $P_{\texttt{pop}}$ to the inputs of $P_{\texttt{dec}}$ (recall the example from Sec. 2 for an illustration). For each term of the form $\mathbb{P}[\varphi]$ appearing in $\varphi_{\texttt{post}}$, the algorithm computes its numerical value and maintains it in a map m. If m satisfies the $\varphi_{\texttt{post}}$—i.e., by replacing all terms $\mathbb{P}[\varphi]$ with their values in m—then the postcondition holds.

4 SYMBOLIC PROBABILISTIC INFERENCE

We now turn our attention to our probabilistic inference algorithm, which reduces the problem to computing the weighted volume of a formula. Recall that we are given (*i*) a formula φ over real arithmetic constraints, encoding the semantics of a program, and (*ii*) a set D defining the PDFs of the distributions of free variables of φ. Our goal is to evaluate the integral $\int_\varphi \prod_{x_i \in X_\varphi} p_i(x_i)\, dX_\varphi$. We begin by describing limitations of existing approaches.

Existing techniques In general, there is no systematic technique for computing an exact value for such an integral. Moreover, even simpler linear versions of the volume computation problem, not involving probability distributions, are #P-hard [Dyer and Frieze 1988]. Existing techniques suffer from one or more of the following: they (*i*) restrict φ to a conjunction of linear inequalities [De Loera et al. 2012; Sankaranarayanan et al. 2013], (*ii*) restrict integrands to polynomials or uniform distributions [Belle et al. 2015a,b; Chistikov et al. 2015; De Loera et al. 2012], (*iii*) compute approximate solutions with probabilistic guarantees [Belle et al. 2015b; Chistikov et al. 2015; Vempala 2005], (*iv*) restrict φ to bounded regions of $\mathbb{R}^n$ [Chistikov et al. 2015], or (*v*) have no convergence guarantees, e.g., computer algebra tools that find closed-form solutions, like Mathematica and PSI [Gehr et al. 2016]. (See Sec. 7 for details.)

Symbolic weighted volume computation Our approach is novel in its generality and its algorithmic core. The following are the high-level properties of our algorithm:

(1) It is guaranteed to converge to the exact value of the weighted volume in the limit. *This allows us to produce a sound and complete procedure for verifying fairness properties.*

$$\frac{}{vol \leftarrow 0 \quad \Psi \leftarrow \boxplus_{\varphi}} \text{ HDECOMP}$$

$$\frac{m \models \Psi}{vol \leftarrow vol + \text{VOL}(H^m, D) \quad \Psi \leftarrow \Psi \wedge block(H^m)} \text{ HSAMPLE}$$

$$\text{where } block(H^m) \equiv \bigvee_{x \in X_\varphi} u_x < H^m_l(x) \vee l_x > H^m_u(x)$$

Fig. 4. SYMVOL: weighted volume computation algorithm

(2) It imposes no restrictions on PDFs, only that we can evaluate the *cumulative distribution functions* (CDFs) associated with the PDFs in D.[2] *This provides us with flexibility in defining population models.*
(3) It accepts formulas in the decidable yet rich theory of *real closed fields*: Boolean combinations of polynomial inequalities. *This provides a rich language for encoding many decision-making programs, as we demonstrate in Sec. 6.*

At the algorithmic level, our approach makes the following contributions:

(1) It exploits the power of SMT solvers and uses them as a black box, allowing it to directly benefit from future advances in solver technology.
(2) It employs the idea of dividing the space into rectangular regions that are easy to integrate over. While this idea has been employed in various guises in verification [Asarin et al. 2000; Bournez et al. 1999; Li et al. 2014; Sankaranarayanan et al. 2013], we utilize it in a new symbolic way to enable volume computation over SMT formulas.
(3) It introduces a novel technique for approximately encoding PDFs as formulas and using them to guide the SMT solver towards making large leaps to the exact solution. This technique is crucial when dealing with decision-making programs comprised of halfspaces, as we show experimentally in Sec. 6.

4.1 Weighted volume computation algorithm

To compute the integral over the region φ, we exploit the observation that if φ is a *hyperrectangular region*, i.e., an n-dimensional rectangle in $\mathbb{R}^n$, then we can evaluate the integral, because each dimension has constant lower and upper bounds. For instance, consider the following formula representing a rectangle in $\mathbb{R}^2$:

$$\varphi \equiv 0 \leqslant x_1 \leqslant 100 \wedge 4 \leqslant x_2 \leqslant 10$$

The following holds:
$$\int_\varphi p_1(x_1)p_2(x_2)\, dx_1 dx_2 = \left(\int_0^{100} p_1(x_1)\, dx_1\right)\left(\int_4^{10} p_2(x_2)\, dx_2\right)$$
$$= (F_1(10) - F_1(4))(F_2(100) - F_2(0))$$

[2]The *cumulative distribution function* of a real-valued random variable X is the function $f : \mathbb{R} \to \mathbb{R}$, such that $f(x) = \mathbb{P}[X \leqslant x]$. In practice, *evaluating* a CDF means either computing the value of the function exactly or approximating its value to specified high degree of precision; see Sec. 5.

where $F_i = \int_{-\infty}^{x} p_i(t)\ dt$ is the CDF of $p_i(x_i)$. That is, we independently compute the integral along each dimension of the rectangle and take the product. This holds since all variables are independently sampled.

Our algorithm is primarily composed of two steps: First, the *hyperrectangular decomposition* phase represents the formula φ as a set of hyperrectangles. Note that this set is likely to be infinite. Thus, we present a technique for defining all hyperrectangles that lie in φ symbolically as a formula $\boxplus_\varphi$, where each model of $\boxplus_\varphi$ corresponds to a hyperrectangle that lies inside the region φ. Second, after characterizing the set $\boxplus_\varphi$ of all hyperrectangles in φ, we can iteratively *sample hyperrectangles* in φ, which can be done using an off-the-shelf SMT solver to find models of $\boxplus_\varphi$. For each hyperrectangle we sample, we compute its weighted volume and add it to our current solution. Therefore, the current solution maintained by the algorithm is the weighted volume of an underapproximation of φ—that is, a lower bound on the exact weighted volume of φ.

Hyperrectangular decomposition We begin by defining hyperrectangles as special formulas.

Definition 4.1 (Hyperrectangles and their weighted volume). A formula $H \in \mathcal{L}$ is a hyperrectangle if it can be written in the form $\bigwedge_{x \in X_H} c_x \leqslant x \leqslant c'_x$, where $c_x, c'_x \in \mathbb{R}$ are the lower and upper bounds of dimension x. We use $H_l(x)$ and $H_u(x)$ to denote the lower and upper bounds of x in H.

The weighted volume of H, given a set D, is as follows:

$$\text{VOL}(H, D) = \prod_{x_i \in X_H} \int_{H_l(x_i)}^{H_u(x_i)} p_i(x_i)\ dx_i$$ ■

Ideally, we would take a formula φ and rewrite it as a disjunction of hyperrectangles $\bigvee H$, but this disjunction is most likely infinite. To see why, consider the simple formula representing a triangular polytope in Figure 5(a). Here, there is no finite number of rectangles whose union is the full region in $\mathbb{R}^2$ enclosed by the triangle.

While the number of hyperrectangles enclosed in φ is infinite, we can characterize them symbolically using universal quantifiers, as shown by Li et al. [2014]. Specifically, we define the hyperrectangular decomposition of φ as follows:

Definition 4.2 (Hyperrectangular decomposition). Given φ, its *hyperrectangular decomposition* $\boxplus_\varphi$ is:

$$\boxplus_\varphi \equiv \left(\bigwedge_{x \in X_\varphi} l_x < u_x \right) \wedge \forall X_\varphi . \left(\left(\bigwedge_{x \in X_\varphi} l_x \leqslant x \leqslant u_x \right) \Rightarrow \varphi \right)$$

where l_x, u_x are fresh free variables introduced for each $x \in X_\varphi$, and $\forall X_\varphi$ is short for $\forall x_1, \ldots, x_n$, for $x_i \in X_\varphi$.

Given a model $m \models \boxplus_\varphi$, we say that H^m is the *hyperrectangle induced by* m, as defined below:

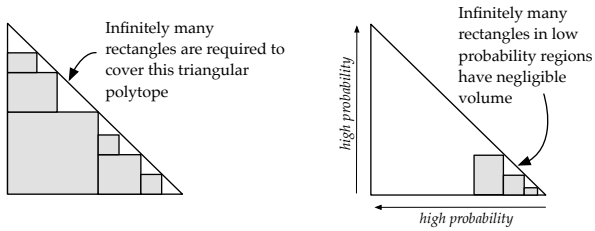
$$H^m \equiv \bigwedge_{x \in X_\varphi} m(l_x) \leqslant x \leqslant m(u_x)$$ ■

Intuitively, $\boxplus_\varphi$ characterizes every possible hyperrectangle that is subsumed by φ. The idea is that the hyperrectangle H^m induced by each model m of $\boxplus_\varphi$ is subsumed by φ, that is, $H^m \Rightarrow \varphi$. The following example illustrates this process.

Example 4.3. Consider the formula $\varphi \equiv x \geqslant y \wedge y \geqslant 0$, illustrated in Figure 5(c) as a gray, unbounded polygon. The formula $\boxplus_\varphi$, after eliminating the universal quantifier, is:

$$l_x < u_x \wedge l_y < u_y \wedge l_y \geqslant 0 \wedge l_x \geqslant u_y$$

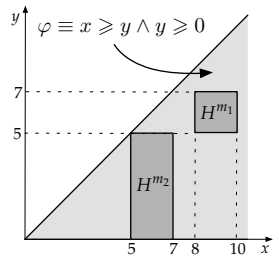
Figure 5(c) shows two models $m_1, m_2 \models \boxplus_\varphi$ and their graphical representation as rectangles H^{m_1}, H^{m_2} in $\mathbb{R}^2$. Observe that both rectangles are subsumed by φ. ■



(a) (b)

Two examples of models inducing rectangles

$m_1 \models \boxplus_\varphi$	$m_2 \models \boxplus_\varphi$
$l_x = 8$	$l_x = 5$
$u_x = 10$	$u_x = 7$
$l_y = 5$	$l_y = 0$
$u_y = 7$	$u_y = 5$



(c)

Fig. 5. (a) $\mathbb{R}^2$ view of hyperrectangular decomposition. (b) Hyperrectangle sampling, where density is concentrated in the top-left corner. (c) Illustration of models of $\boxplus_\varphi$.

The following theorem states the soundness and completeness of hyperrectangular decomposition: models of $\boxplus_\varphi$ characterize all hyperrectangles in φ and no others.

Theorem 4.4 (Correctness of $\boxplus$). *Let $\varphi \in \mathcal{L}$.*

- ***Soundness**: Let $m \models \boxplus_\varphi$. Then, $H^m \Rightarrow \varphi$ is valid.*
- ***Completeness**: Let H be a hyperrectangle such that $H \Rightarrow \varphi$. Then, the following is satisfiable:*

$$\boxplus_\varphi \wedge \bigwedge_{x \in X_\varphi} l_x = H_l(x) \wedge u_x = H_u(x)$$

Hyperrectangle sampling Our symbolic weighted volume computation algorithm, SYMVOL, is shown in Figure 4 as two transition rules. Given a pair (φ, D), the algorithm maintains a state consisting of two variables: (*i*) *vol*, the current lower bound of the weighted volume, and (*ii*) Ψ, a constraint that encodes the *remaining* rectangles in the hyperrectangular decomposition of φ.

The algorithm is presented as guarded rules. Initially, using the rule HDECOMP, *vol* is set to 0 and Ψ is set to $\boxplus_\varphi$. The algorithm then proceeds by iteratively applying the rule HSAMPLE. Informally, the rule HSAMPLE is used to find arbitrary hyperrectangles in φ and compute their weighted volume. Specifically, HSAMPLE finds a model m of Ψ, computes the weighted volume of the hyperrectangle H^m induced by m, and adds the result to *vol*.

To maintain soundness, HSAMPLE ensures that it never samples two overlapping hyperrectangles, as otherwise we would overapproximate the volume. To do so, every time a hyperrectangle H^m is sampled, we conjoin an additional constraint to Ψ—denoted $block(H^m)$ and defined in Figure 4—that ensures that for all models $m' \models \Psi$, $H^{m'}$ does not overlap with H^m, i.e., $H^{m'} \wedge H^m$ is unsatisfiable.

Informally, the $block(H^m)$ constraint specifies that any newly sampled hyperrectangle should be to the *left* or *right* of H^m for at least one of the dimensions.

The following theorem states the correctness of *block*: it removes all hyperrectangles that overlap with H^m (soundness), and it does not overconstrain Ψ by removing hyperrectangles that do not overlap with H^m (completeness).

Theorem 4.5 (Correctness of *block*). *Given φ, let $\Psi \Rightarrow \boxplus_\varphi$, and let $m_1, m_2 \models \Psi$.*

- ***Soundness**: If $H^{m_1} \wedge H^{m_2}$ is satisfiable, then $m_2 \not\models \Psi \wedge block(H^{m_1})$.*
- ***Completeness**: If $H^{m_1} \wedge H^{m_2}$ is unsatisfiable, then $m_2 \models \Psi \wedge block(H^{m_1})$.*

Lower and upper bounds The following theorem states the soundness of SYMVOL: it maintains a lower bound on the exact weighted volume.

Theorem 4.6 (Soundness of SYMVOL). *The following is an invariant of SYMVOL(φ, D): $vol \leqslant$ VOL(φ, D).*

Proof. At any point in the execution, $vol = \sum_{i=1}^{l} \int_{H_i} \prod p_i(x_i)\, dX_\varphi$, where l is the number of applications of HSAMPLE and H_i is the hyperrectangle sampled at step i. By definition, $\bigvee H_i \Rightarrow \varphi$. Since PDFs are positive functions, $vol \leqslant$ VOL(φ, D). ∎

It follows from the above theorem that we can use SYMVOL to compute an upper bound on the exact volume. Specifically, because we are integrating over PDFs, we know that VOL(φ, D)+VOL$(\neg\varphi, D) = 1$. Therefore, by using SYMVOL to compute the weighted volume of $\neg\varphi$, we get an upper bound on the exact volume of φ.

Corollary 4.7 (Upper bounds). *The following is an invariant of SYMVOL$(\neg\varphi, D)$: $1 - vol \geqslant$ VOL(φ, D)*

Proof. By definition of PDFs and integration,

$$\int_{\mathbb{R}^n} \prod p_i(x_i)\, dX_\varphi = \int_{\varphi} \prod p_i(x_i)\, dX_\varphi + \int_{\neg\varphi} \prod p_i(x_i)\, dX_\varphi$$

for any $\varphi \subseteq \mathbb{R}^n$. From Theorem 4.6, it follows that at any point in the execution of SYMVOL$(\neg\varphi, D)$, we have $1 - vol \geqslant$ VOL(φ, D). ∎

4.2 Density-directed sampling

While the SYMVOL algorithm is sound, it provides no progress guarantees. Consider, for example, that the algorithm might diverge by sampling hyperrectangles in φ that appear in very low probability density regions, as illustrated in Figure 5(b) on a triangular polytope in $\mathbb{R}^2$.

Ideally, the rule HSAMPLE would always find a model m yielding the hyperrectangle H^m with the *largest weighted volume*. Finding such a model amounts to solving the optimization problem:

$$\arg\max_{m \models \Psi} \prod_{x_i \in X_\varphi} \int_{H_l^m(x_i)}^{H_u^m(x_i)} p_i(x_i)\, dx_i$$

From a practical perspective, there are no known tools or techniques for finding models of first-order formulas that maximize such complex objective functions—with integrals over probability density functions.

However, we make the key observation that if $p(x)$ is a *step function*—i.e., piecewise constant—then we can symbolically encode $\int p(x)\, dx$ in linear arithmetic. As such, we propose to (*i*) *approximate* each density function $p(x)$ with a step function $step(x)$, (*ii*) *encode* the integrals $\int step(x)\, dx$ as

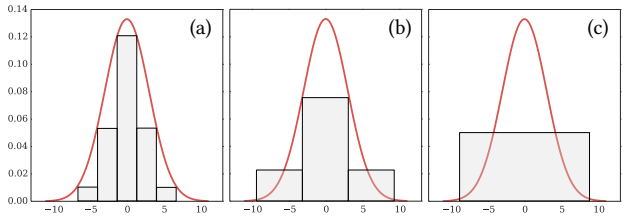


Fig. 6. Three ADFs (gray) of a Gaussian PDF (red) with mean 0 and a SD 3: (a) *fine-grained*; (b) *coarse*; (c) *uniform*.

linear arithmetic formulas, and (*iii*) *direct* sampling towards hyperrectangles that maximize these integrals, thus finding hyperrectangles of large volume.

Approximate density functions We begin by defining *approximate density functions* (ADFs).

Definition 4.8 (Approximate density functions). An approximate density function $step(x)$ is of the form:

$$step(x) = \begin{cases} c_i, & x \in [a_i, b_i) \text{ for } 1 \leqslant i \leqslant n \\ 0, & \text{otherwise} \end{cases}$$

where $c_i, a_i, b_i \in \mathbb{R}$, $c_i > 0$, and all $[a_i, b_i)$ are disjoint. ■

We now show how to encode a formula $step^{\phi}(x)$ over the free variables δ_x, l_x, u_x, where for any model $m \models step^{\phi}(x)$, the value $m(\delta_x)$ is the area under $step(x)$ between $m(l_x)$ and $m(u_x)$, i.e.: $m(\delta_x) = \int_{m(l_x)}^{m(u_x)} step(x)\, dx$. Intuitively, the value of this integral is the sum of the areas of each *bar* in $step(x)$, restricted to $[m(l_x), m(u_x)]$.

Definition 4.9 (Encoding area under an ADF). Given an ADF $step(x)$, we define $step^{\phi}(x)$ as follows:

$$step^{\phi}(x) \equiv \delta_x = \sum_{i=1}^{n} c_i \cdot \Big|[a_i, b_i) \cap [l_x, u_x]\Big| \qquad ■$$

The finite sum in $step^{\phi}(x)$ computes the size of the intersection of $[l_x, u_x]$ with each interval $[a_i, b_i)$ in $step(x)$, and multiplies the intersection with c_i, the value of the *step* in that interval. Note that the constraint $step^{\phi}(x)$ is directly expressible in linear arithmetic, since

$$\Big|[a_i, b_i) \cap [l_x, u_x]\Big| = \max(\min(b_i, u_x) - \max(a_i, l_x), 0)$$

The following theorem states the correctness of the ADF encoding:

THEOREM 4.10 (CORRECTNESS OF $step^{\phi}$). *Fix an ADF $step(x)$.*

- ***Soundness**: For any model $m \models step^{\phi}$, the following is true: $m(\delta_x) = \int_{m(l_x)}^{m(u_x)} step(x)\, dx$.*
- ***Completeness**: For any constants $a, b, c \in \mathbb{R}$ such that $c = \int_a^b step(x)\, dx$, the following formula is satisfiable: $\delta_x = c \wedge l_x = a \wedge u_x = b \wedge step^{\phi}(x)$.*

ADF-directed volume computation We now present the algorithm ADF-SYMVOL (Figure 7), an extension of our volume computation algorithm SYMVOL that uses ADFs to steer the sampling process. The ADFs are only used for guiding the rule HSAMPLE towards dense hyperrectangles, and thus do not affect soundness of the volume computation. For example, Figure 6 shows three

$$\frac{}{vol \leftarrow 0 \quad \Psi \leftarrow \square\!\!\square_\varphi \quad lb \leftarrow 1}\ \textsc{hdecomp} \qquad \frac{}{lb \leftarrow \lambda * lb}\ \textsc{decay}$$

$$\frac{m \models \Psi \wedge \bigwedge_{x_i \in X_\varphi} \exists \delta_{x_i}.\, step_i^\phi(x_i) \wedge \delta_{x_i} \geqslant lb}{vol \leftarrow vol + \textsc{vol}(H^m, D) \qquad \Psi \leftarrow \Psi \wedge block(H^m)}\ \textsc{hsample}$$

Fig. 7. ADF-SYMVOL: ADF-directed volume computation

approximations of a Gaussian; all three are valid approximations. In Sec. 6, we discuss the impact of different ADFs on performance.

Formally, we create a set of ADFs $\mathcal{A} = \{step_1, \ldots, step_n\}$, where, for each variable $x_i \in X_\varphi$, we associate the ADF $step_i(x_i)$. The rule HSAMPLE now encodes $step_i^\phi(x_i)$ and attempts to find a hyperrectangle such that for each dimension x, δ_x is greater than some lower bound lb, which is initialized to 1. Of course, we need to reduce the value lb as we run out of hyperrectangles of a given volume. Therefore, the rule DECAY is used to shrink lb using a fixed *decay rate* $\lambda \in (0, 1)$ and can be applied when HSAMPLE fails to find a sufficiently large hyperrectangle.

Example 4.11. Suppose we want to find the weighted volume of a single-variable formula

$$\varphi \equiv 0 \leqslant x \leqslant 1$$

where x is uniformly distributed over the interval $[0, 1]$. Its hyperrectangular decomposition is

$$\square\!\!\square_\varphi \equiv l_x < u_x \wedge \forall x.\, (l_x \leqslant x \leqslant u_x \Rightarrow 0 \leqslant x \leqslant 1)$$

or equivalently, if we eliminate the quantifier,

$$\square\!\!\square_\varphi \equiv l_x < u_x \wedge 0 \leqslant l_x \leqslant 1 \wedge 0 \leqslant u_x \leqslant 1$$

It's clear that an arbitrary model of $\square\!\!\square_\varphi$ can be any single interval $I \subseteq [0, 1]$, and since x is uniformly distributed over $[0, 1]$, the weighted volume of I is exactly its size. Thus, we would like the models to be *large* intervals; to do so, we employ a constraint based on the ADF of x.

Since x is uniformly distributed, we can use its actual distribution as its ADF. We then have that

$$step^\phi(x) \equiv \delta_x = 1 \cdot \Big|[0, 1) \cap [l_x, u_x]\Big| \equiv \delta_x = \max(\min(1, u_x) - \max(0, l_x), 0)$$

Here, δ_x represents the weighted volume contribution of the variable x (which happens to be the only variable in φ), and so if we obtain a model not of $\square\!\!\square_\varphi$, but instead of the formula $\square\!\!\square_\varphi \wedge \exists \delta_x.\, (step^\phi(x) \wedge \delta_x \geqslant lb)$, explicitly written as

$$l_x < u_x \wedge 0 \leqslant l_x \leqslant 1 \wedge 0 \leqslant u_x \leqslant 1 \wedge \exists \delta_x.\, \Big(\delta_x = \max(\min(1, u_x) - \max(0, l_x), 0) \wedge \delta_x \geqslant lb\Big)$$

then the weighted volume of the *worst* model approximately increases as a function of lb. In fact, when $lb = 1$, the only model is the whole unit interval (where $l_x = 0$ and $u_x = 1$), which contains all of the probability mass of x. ■

Note that, ideally, we would look for a model m such that $\prod_{x \in X_\varphi} \delta_x$ is maximized, thus, finding the hyperrectangle with the largest weighted volume with respect to the ADFs. However, this constraint is non-linear. To lower the complexity of the problem to that of linear arithmetic, we set a decaying lower bound and attempt to find a model where each δ_x is greater than the lower bound.

4.3 Convergence of algorithm

We now discuss the convergence properties of ADF-SYMVOL. Suppose we are given a formula φ, a set D, and a set $\mathcal{A}$. Let $R \subset \mathbb{R}^n$ be the region where all the ADFs in $\mathcal{A}$ are non-zero. We will show that ADF-SYMVOL monotonically converges, in the limit, to the exact weighted volume restricted to R; that is, ADF-SYMVOL converges to

$$\int_{\varphi \cap R} \prod_{x_i \in X_\varphi} p_i(x_i)\, dX_\varphi$$

The fascinating part here is that we do not impose any restrictions on the ADFs: they do not have to have any correspondence with the PDFs they approximate; they need only be step functions. Of course, in practice, the quality of the approximation dictates the rate of convergence, but we delay this discussion to Sec. 6.

The following theorem states convergence of ADF-SYMVOL; it assumes that HSAMPLE is applied iteratively and DECAY is only applied when HSAMPLE cannot find a model.

THEOREM 4.12 (MONOTONE CONVERGENCE TO R). *Assume ADF-SYMVOL is run on (φ, D) and a set of ADFs $\mathcal{A}$ that are non-zero for $R \subset \mathbb{R}^n$. Let vol_i be the value of vol after i applications of HSAMPLE. Then,*

$$\lim_{i \to \infty} vol_i = \int_{\varphi \cap R} \prod_{x_i \in X_\varphi} p_i(x_i)\, dX_\varphi \qquad \text{and} \qquad \forall j \geqslant k \geqslant 1.\ vol_j \geqslant vol_k$$

PROOF. The algorithm constructs two series in parallel: the actual volume computation series $\sum v_i$ and the approximated series $\sum a_i$, where each v_i and a_i correspond to the actual and approximate volume of the i'th sampled hyperrectangle (note that the latter is not explicitly maintained in the algorithm). Each series corresponds to a sequence of partial sums: Let

$$v_i^\Sigma = \sum_{j=1}^{i} v_j \qquad\qquad a_i^\Sigma = \sum_{j=1}^{i} a_j$$

It is maintained that

$$\forall i.\ v_i^\Sigma \leqslant EVol_{R \cap \varphi} = \int_{R \cap \varphi} \prod p(x)\, dX_\varphi$$

$$\forall i.\ a_i^\Sigma \leqslant AVol = \int_{R \cap \varphi} \prod step(x)\, dX_\varphi$$

Since v_i^Σ and a_i^Σ are non-decreasing sequences bounded from above, they converge to *some* limit; call the limits v^Σ and a^Σ, respectively. It does not matter what the value of a^Σ is, but we would like to ensure that v^Σ is actually equal to $EVol_{R \cap \varphi}$. Since the a_i determine which hyperrectangles we sample, the potential concern is that they negatively affect the limit v^Σ; we will prove below that this is not possible.

Suppose, for the sake of obtaining a contradiction, that our sequence of samples to construct $\{v_i^\Sigma\}$ and $\{a_i^\Sigma\}$ results in the limit v^Σ being strictly less than the actual weighted volume $EVol_{R \cap \varphi}$. Then there is some subregion $R' \subseteq R$ that is completely disjoint from the infinite set of hyperrectangles we sample and has non-zero weighted volume. In particular, there must exist some hyperrectangle $H \subseteq R'$ contained in this unsampled region that also has non-zero weighted volume.

In the limit, a_i^Σ approaches a^Σ: by the definition of a limit, for all $\epsilon > 0$, there exists N such that for all $n > N$, $a^\Sigma - a_n^\Sigma < \epsilon$. Let $\delta = \int_H \prod step(x)\, dX_\varphi$: at some point when we have fixed a threshold

$\tau < \delta$ and have run out of samples in $R \setminus R'$ with $a_n \geqslant \tau$ (guaranteed when $a^\Sigma - a^\Sigma_n < \tau$ by letting $\epsilon = \tau$) we would have sampled $H \subseteq R'$. This property ensures that the limit $v^\Sigma = EVol_{R \cap \varphi}$. ■

Note that the above theorem directly gives us a way to approach the exact volume. Specifically, by performing runs of ADF-SYMVOL on subsets in an infinite partition of $\mathbb{R}^n$ induced by the ADFs, we can ensure that the sum over the ADF-SYMVOL processes approaches the exact volume. For all i, let $\mathcal{A}_i$ be a set of ADFs corresponding to an ADF-SYMVOL process P_i, where $R_i \subset \mathbb{R}^n$ is the non-zero region of $\mathcal{A}_i$. We require an infinite set of P_i to partition $\mathbb{R}^n$: (*i*) for all $i \neq j$, $R_i \cap R_j = \emptyset$, and (*ii*) $\bigcup_{i=1}^{\infty} R_i = \mathbb{R}^n$. The following theorem formalizes the argument:

THEOREM 4.13 (MONOTONE CONVERGENCE). *Let $P_1, P_2, \ldots$ be ADF-SYMVOL processes that partition $\mathbb{R}^n$. Assume an execution where each P_i executes infinitely often and each P_i performs HSAMPLE infinitely often, and let vol_n be the total computed volume across all P_i after n successful calls to HSAMPLE. Then,*

$$\lim_{n \to \infty} vol_n = \textsc{vol}(\varphi, D) \qquad \textit{and} \qquad \forall j \geqslant k \geqslant 1.\ vol_j \geqslant vol_k$$

PROOF. We require that ADF-SYMVOL calls HSAMPLE on each P_i (and its $\mathcal{A}_i$ defined over R_i) infinitely often: we can refer to H_n as the nth hyperrectangle obtained by HSAMPLE in the serialized execution. Clearly $\sum_{n=1}^{i} \textsc{vol}(H_n, D)$ is a non-decreasing series. It is bounded above by its supremum, which is exactly $\textsc{vol}(\varphi, D)$ since each individual P_i converges to the weighted volume restricted to R_i. This completes the proof, since the limit of any non-decreasing sequence bounded above by its supremum is identically its supremum. ■

Completeness in verification Given that we have established monotone convergence of ADF-SYMVOL, we can use it to construct a verification procedure that is complete whenever the postcondition is *robust* (as defined in Sec. 3). Given a robust postcondition $\varphi_{\texttt{post}}$, for any subformula $\mathbb{P}[\varphi] > c$ in the postcondition we have that $\mathbb{P}[\varphi] \neq c$. Using this property, we can use ADF-SYMVOL to iteratively improve a lower and an upper bound for $\mathbb{P}[\varphi]$, one of which will prove or disprove the subformula $\mathbb{P}[\varphi] > c$.

5 IMPLEMENTATION

We implemented our algorithms in a new tool called FairSquare, which employs Z3 [De Moura and Bjørner 2008] for SMT solving and Redlog[3] for quantifier elimination. FairSquare accepts as input the population model and the decision-making program in a Python-like syntax, where the definitions of predicates in the probability events are provided as program annotations.

FairSquare computes upper and lower bounds for each probability in the postcondition using weighted volume computation. A *round* of sampling involves (*i*) obtaining a sample (hyperrectangle) for each of the quantities, (*ii*) computing these samples' weighted volumes, (*iii*) updating the bounds on each quantity and (*iv*) checking if the bounds are precise enough to determine the validity of the postcondition, i.e., to prove *fairness* or *unfairness*. Rounds of sampling are performed until a proof is found or a timeout is reached.

Sample maximization A key optimization implemented in FairSquare is the maximization of hyperrectangles obtained during sampling. We use Z3's optimization capability to maximize and minimize the finite bounds of all hyperrectangles, while still satisfying the formula Ψ (in Figures 4 and 7). This process is performed greedily by extending a hyperrectangle in one dimension at a time to find a maximal hyperrectangle. If a dimension extends to infinity, then we drop that bound, thus resulting in an unbounded hyperrectangle.

[3] http://www.redlog.eu/

Numerical precision All of the arithmetic performed by FairSquare is over arbitrary precision rationals, and therefore we do not encounter any loss of precision. The only place where floating point numbers appear is when we evaluate CDFs with `scipy`. We truncate (underapproximate) the result and convert it to a rational number. Truncating the results ensures that our implementation is sound—that at any point in our volume computation, the current volume is a lower bound—at the cost of a small possibility of incompleteness.

6 EVALUATION

In this section, we evaluate the effectiveness and performance of FairSquare. Specifically, we investigate the following questions:[4]

Q1 Can FairSquare verify fairness properties of real machine-learned programs? (Sec. 6.2)
Q2 Do ADFs and sample maximization improve the performance of FairSquare? (Sec. 6.3)
Q3 Can FairSquare verify fairness properties other probabilistic analysis tools cannot? (Sec. 6.4)
Q4 Can FairSquare verify the benchmarks solved by other probabilistic analysis tools? (Sec. 6.4)

6.1 Benchmarks

Fairness postconditions In our experiments, we consider a *group fairness* postcondition augmented with a notion of qualification. We ultimately obtained our benchmarks by datamining a popular income dataset[5] used in related research on algorithmic fairness [Calders and Verwer 2010; Feldman et al. 2015; Zemel et al. 2013]; accordingly, our postconditions are defined in terms of that dataset's features. Specifically, they are of the form:

$$\frac{\mathbb{P}[\mathit{high\ income} \mid \mathit{female} \wedge \mathit{qual}(\boldsymbol{v})]}{\mathbb{P}[\mathit{high\ income} \mid \mathit{male} \wedge \mathit{qual}(\boldsymbol{v})]} > 1 - \epsilon$$

Suppose, for example, machine-learned models inferred from the dataset would be used to determine the salary of an employee: *high* (> \$50,000) or *low*. We consider $\mathit{qual}(\boldsymbol{v})$ in two different scenarios—first, the case when *qual* is tautologically true, and second, when individuals are qualified if they are at least 18 years of age. In short, we would like to verify whether salary decisions are fair to qualified female employees. Throughout, we fix $\epsilon = 0.15$.

Decision-making programs We obtained our set of decision-making programs by training a variety of machine-learning models on the income dataset to classify *high* vs *low* income. Using the Weka machine learning suite [Hall et al. 2009], we learned 11 different decision-making programs (see, e.g., Bishop [2006] for background), which are listed in Figure 8: (*i*) four *decision trees*, named DT_n, where n is the number of conditionals in the program, and the number of variables and the depth of the tree each varies from 2 to 3; (*ii*) four *support vector machines* with linear kernels, named SVM_n, where n is the number of variables in the linear separator; (*iii*) three *neural networks* using *rectified linear units* [Nair and Hinton 2010], named $\text{NN}_{n,m}$, where n is the number of input variables, and m is the number of nodes in the single hidden layer.

As we will show in the next section, some of these programs do not satisfy the fairness property we consider. We introduced modifications of DT_{16} and SVM_4, called DT^{α}_{16} and SVM^{α}_{4}, that implement rudimentary forms of *affirmative action* for female applicants. For DT^{α}_{16}, there is a 15% chance it will flip a decision to give the low salary; for SVM^{α}_{4}, the linear separator is moved to increase the likelihood of hiring.

[4] All experiments are performed on an Intel Core i7 4.00GHz CPU with 16 GB of RAM.
[5] https://archive.ics.uci.edu/ml/datasets/Adult/

Decision program	Acc	Population Model: Independent				Bayes Net 1				Bayes Net 2			
		Res	#	*Vol*	*QE*	*Res*	#	*Vol*	*QE*	*Res*	#	*Vol*	*QE*
DT_4	0.79	✓	10	1.3	0.5	✗	12	2.2	0.9	✗	18	6.6	2.2
DT_{14}	0.71	✓	20	4.2	1.4	✓	38	52.3	11.4	✓	73	130.9	33.6
DT_{16}	0.79	✓	21	7.7	2.0	✗	22	15.3	6.3	✗	22	38.2	14.3
DT^{α}_{16}	0.76	✓	18	5.1	3.0	✓	34	32.0	8.2	✓	40	91.0	19.4
DT_{44}	0.82	✓	55	63.5	9.8	✗	113	178.9	94.3	✗	406	484.0	222.4
SVM_3	0.79	✓	10	2.6	0.6	✗	10	3.7	1.7	✗	10	10.8	6.2
SVM_4	0.79	✓	10	2.7	0.8	✗	18	13.3	3.1	✗	14	33.7	20.1
SVM^{α}_4	0.78	✓	10	3.0	0.8	✓	22	15.7	3.2	✓	14	33.4	63.2
SVM_5	0.79	✓	10	8.5	1.3	✗	10	12.2	6.3	TO_q	-	-	TO
SVM_6	0.79	$^{0.02}_{35.3}$	634	TO	2.4	$^{0.09}_{3.03}$	434	TO	12.8	TO_q	-	-	TO
$\text{NN}_{2,1}$	0.65	✓	78	21.6	0.8	✓	466	456.1	3.4	✓	154	132.9	7.2
$\text{NN}_{2,2}$	0.67	✓	62	27.8	2.0	✓	238	236.5	7.2	✓	174	233.5	18.2
$\text{NN}_{3,2}$	0.74	$^{0.03}_{674.7}$	442	TO	10.0	$^{0.00}_{5.24}$	34	TO	55.9	TO_q	-	-	TO

Fig. 8. Results of FairSquare applied to 39 fairness verification problems. *Res*: ✓ for fair; ✗ for unfair. *Vol*: time (s) of the sampling procedure; #: number of SMT calls. If sampling timed out (900s), *Res* denotes the latest bounds on the fairness ratio. *QE*: time (s) of the quantifier elimination procedure used prior to sampling; if *QE* times out (900s), no sampling is performed, denoted by TO_q for *Res*. *Acc*: training set accuracy of the programs.

Population models For our population models, we used three different probabilistic programs that were inferred from the same dataset: (*i*) a set of *independently distributed* variables (Ind), (*ii*) a *Bayesian network* using a simple graph structure (BN1), and (*iii*) the same Bayesian network, but with an integrity constraint in the form of an inequality between two of the variables (BN2). Note that the first model is sometimes a trivial case: since there is there is no dependence between variables, a program will be fair if it does not access an individual's sex; this simplicity serves well as a baseline for our evaluation. The Bayesian models permit correlations between the variables, allowing for more subtle sources of fairness or unfairness. The benchmarks we use are derived from each combination of population models with decision-making programs.

6.2 Effectiveness of FairSquare

Figure 8 shows the results of applying FairSquare to 39 fairness verification problems, as described earlier. Only the instances using the tautologically true notion of qualification are shown, since the qualitative results are quite similar to the non-trivial qualification. FairSquare was able to solve 32 of the 39 problems within a timeout period of 900 seconds each, proving 21 fair and 11 unfair.

Consider the results for DT_4: FairSquare proved it fair with respect to the independent population model after 0.5 seconds of an initial quantifier elimination procedure and 1.3 seconds of the actual volume computation algorithm, which required 10 SMT queries. The more sophisticated Bayesian network models took longer for sampling, but due to the correlations between variables, were proved unfair.

In contrast, consider the results for DT_{44} under the Bayes Net 1 population model: FairSquare was unable to conclude fairness or unfairness after 900 seconds of volume computation (denoted by TO in the *Vol* column). The lower and upper bounds of the fairness ratio it had computed at that time are listed in the *Res* column: in this case, the value of the fairness ratio is within $[0.70, 0.88]$, which

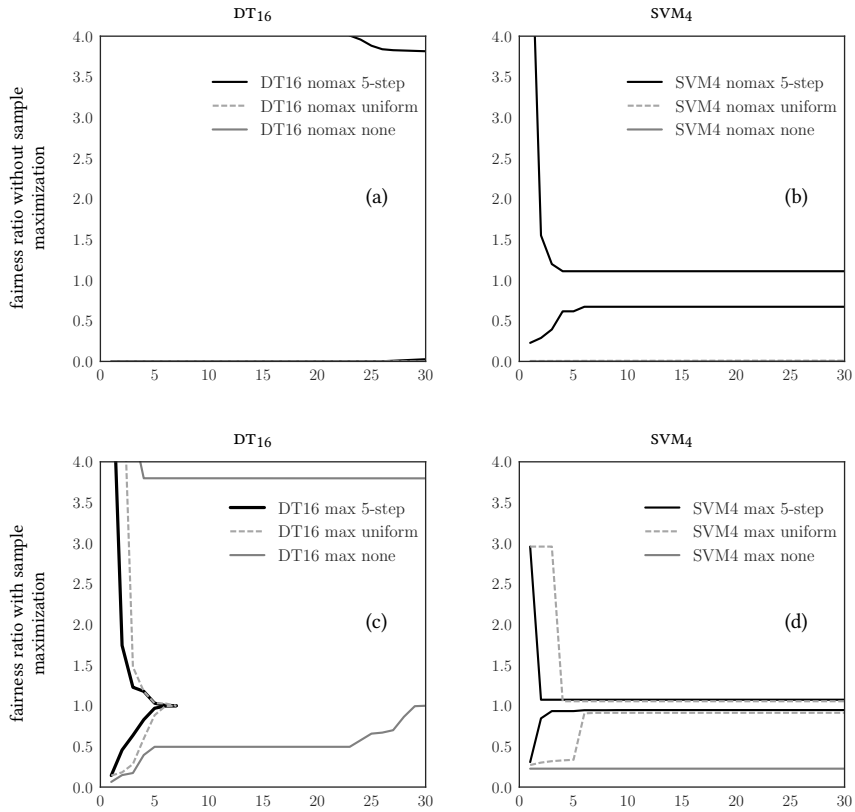


Fig. 9. Fairness ratio vs. rounds of sampling for DT_{16} and SVM_4 (Ind pop model) differing on ADFs and sample maximization. In (c) two runs end at the exact value. Outside the visible range are: (a)(b) upper and lower bounds of *uniform* and *none*; (d) upper bounds of *none*.

is not precise enough for the $\epsilon = 0.15$ requirement (but would be precise enough for ϵ outside of $[0.12, 0.30]$).

In general, all conclusive results using the independent population model were proved to be fair, as expected, but many are unfair with respect to the clusters and Bayes net models because of the correlations those population models capture. This difference illustrates the sensitivity of fairness to the population model; in particular, none of the decision trees syntactically access *sex*, yet several are unfair.

Figure 8 shows that the affirmative action modifications in DT_{16}^{α} and SVM_4^{α} are sufficient to make the programs fair with respect to every population model without substantially impacting the training set accuracy.

In summary, the answer to **Q1** is that **FairSquare is powerful enough to reason about group fairness for many non-trivial machine-learned programs**.

6.3 Effect of parameters

The experiments in Figure 8 were all performed using sample maximization (as described in Sec. 5); additionally, to guide volume computation, all Gaussian distributions with mean μ and variance

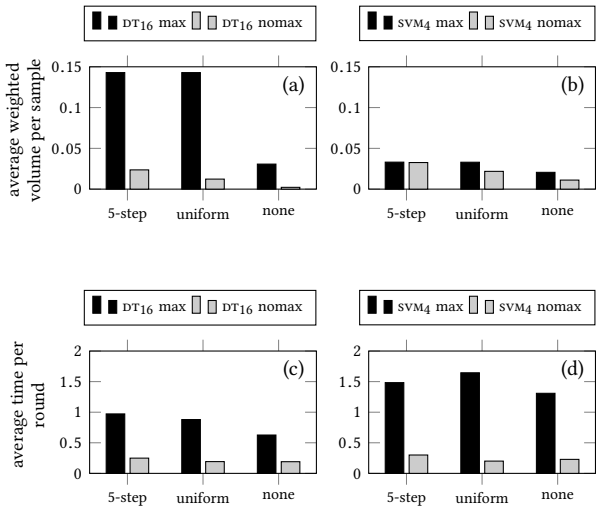


Fig. 10. Effect of optimizations on FairSquare for DT_{16} and SVM_4 (Ind pop model). (a) and (b) show the average weighted volume per sample (averaged across all probabilities). (c) and (d) show the average time (s) per round of sampling.

σ^2 use ADFs (see Sec. 4.2) with 5 equal-width steps spanning $(\mu - 3\sigma^2, \mu + 3\sigma^2)$—analogous to Figure 6(a). In this section, we explore the effects of the approximate density functions and of the sample maximization optimizations. These results are captured in Figures 9 and 10.

There are three instances of ADFs in Figure 9 used to guide the sampling to high-probability regions: (*i*) *none* indicates that no ADF is used, i.e., we used SYMVOL instead of ADF-SYMVOL; (*ii*) *uniform* indicates that each `gauss`(μ, σ^2) is approximated by a uniform function spanning $(\mu - 3\sigma^2, \mu + 3\sigma^2)$ (similar to Figure 6(c)); and (*iii*) *5-step* indicates that each Gaussian is approximated by a step function of 5 equal-width regions spanning that same domain (similar to Figure 6(a)). Another variable, *max* or *nomax*, denotes whether the sample maximization optimization is enabled.

Each combination of these techniques is run on two of our benchmarks: DT_{16} and SVM_4 under the independent population model. Figure 9(a) and (b) show how convergence to the fairness ratio is improved by the choice of ADFs when sample maximization is not employed: in particular, the runs using *uniform* and *none* are not even visible, as the bounds never fall within $[0.01, 4.0]$. Plots (c) and (d) show that when sample maximization *is* employed, the choice between the uniform and 5-step ADFs is not as substantial on these benchmarks, although (*i*) the better approximation gets better bounds faster, and (*ii*) using none results in substantially worse bounds.

Figure 10 plot (a) and (b) show that employing ADFs and using sample maximization each increases the average weighted volume per sample, allowing volume computation to be done with fewer samples. Plots (c) and (d) illustrate the trade-off: the average time per sampling round tends to be greater for more complex optimizations.

We present these results for two particular problems and observe the same results across our suite. In summary, the answer to **Q2** is that **ADFs and sample maximization improve the performance of FairSquare**, and FairSquare requires both of these features to verify most benchmarks.

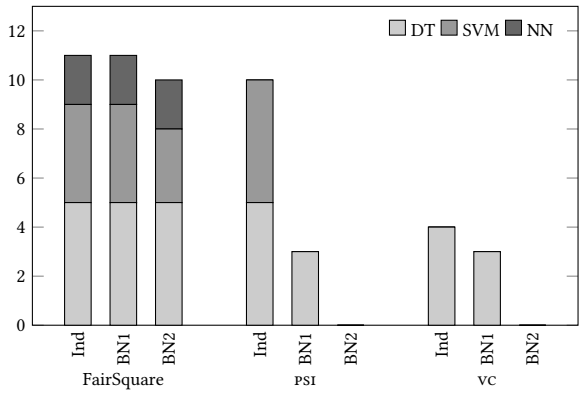


Fig. 11. Comparison of the number of benchmarks that FairSquare, PSI [Gehr et al. 2016], and VC [Sankaranarayanan et al. 2013] were able to solve.

6.4 Comparison to other tools

We ran our benchmarks on the two other recent probabilistic program analysis tools that accept the same class of problems and provide exact guarantees on probabilities. First, we compare to the tool of Sankaranarayanan et al. [2013] (VC),[6] which is algorithmically similar to our tool: it finds bounds for probabilities on individual paths by approximating convex polytopes with bounding and inscribed hyperrectangles. Second, we compare to PSI [Gehr et al. 2016],[7] which symbolically computes representations of the posterior distributions of variables.

Figure 11 shows the number of benchmarks solved per category per tool. Tools were deemed to have failed on a benchmark when they timed out after a 900s period or returned an inconclusive solution (in the case of PSI). For instance, in the case of the population model BN1, FairSquare solved 11 benchmarks, while PSI and VC only solved 3 (the decision trees). For BN2, neither PSI nor VC was able to complete any benchmark.

The figure illustrates some qualitative properties of the applicability of the tools. In general, most of the decision trees are solvable because they partition the decision space with inequalities between a single variable and a *constant*. However, inequalities involving multiple variables can result in (*i*) the lack of closed form posterior CDFs, as reflected in the output of PSI, and (*ii*) angled boundaries in the decision space that are hard to approximate with hyperrectangles; these inequalities occur in the SVMs, neural networks, and the BN2 population model. Consequently, VC fails to produce good bounds in these cases. Similarly, PSI fails because the integrals do not have closed forms or cannot be constructed within the timeout period.

In summary, the answer to **Q3** is that **FairSquare can verify fairness properties that other tools cannot** and therefore extends the class of problems that can be solved by state-of-the-art probabilistic analysis tools.

We now discuss the results of applying the weighted volume computation algorithm of FairSquare to the benchmarks from VC. (We omit a comparison to the benchmarks from PSI, since the output of PSI is a posterior distribution—which can be used to compute probabilities, but not vice-versa.) We first focus on VC's three loop-free benchmark programs, which have thousands of paths; VC computes various probabilities within two hours for each program. In FairSquare, however, the

[6] Acquired directly from the authors.
[7] Artifact available from http://psisolver.org/.

quantifier elimination procedure employed *before beginning sampling* does not terminate within two hours.

Second, we consider two of vc's programs, *cart* and *invPend*, which have loops explicitly bounded by constants and could be encoded in our framework using loop unrolling. The programs' loops have maximum depths of 5 and 10 iterations, respectively, (and the loop bodies contain if statements and probabilistic assignments). The fully unrolled versions of these programs are very large and cause FairSquare's quantifier elimination to timeout; to better understand the limitations of FairSquare, we tried unrolling the programs up to 4 and 7 iterations, respectively, which were the largest unrollings for which the quantifier elimination procedure would terminate within two hours. However, we found that the program paths corresponding to these fewer number of loop iterations have zero probability mass, so FairSquare was not able to compute non-trivial bounds for any probabilities.

In summary, the answer to **Q4** is that **FairSquare currently cannot solve the verification benchmarks solved by the tool vc**.

Quantifier elimination is difficult on programs with this many paths. vc is well-designed for these tasks because it performs weighted volume computation only on the most *important* paths. Specifically, vc heuristically picks a program path π through simulation, with the assumption that traversed paths will likely have a larger probability mass for the event of interest. vc then computes the probability of executing π and a given property being true at the end. By iteratively choosing more and more paths through the program, it improves the computed bounds. Our approach considers the full set of paths symbolically by encoding them as a formula. As described above, this methodology works well for decision-making programs. Our evaluation indicates that our two techniques can complement each other, providing an important direction for future work. Specifically, we plan to investigate a lazily-evaluated quantifier elimination procedure, where we heuristically sample disjuncts (i.e., program paths), so that FairSquare can scale to benchmarks used by vc—where explicit quantifier elimination is prohibitively expensive.

7 DISCUSSION AND RELATED WORK

Algorithmic fairness Our work is inspired by recent concern in the fairness of modern decision-making programs [Barocas and Selbst 2014; Zarsky 2014]. A number of recent works have explored algorithmic fairness [Calders and Verwer 2010; Datta et al. 2016, 2015; Dwork et al. 2012; Feldman et al. 2015; Hardt et al. 2016; Pedreshi et al. 2008; Zemel et al. 2013]. Most works are interested in fairness from a machine learning perspective: how does one learn a fair classifier from data? For example, Zemel et al. [2013] and Feldman et al. [2015] aim to transform training data so as to erase correlations between the sensitive attributes of individuals and the rest of their features. Within this context, classification utility is important. Hardt et al. [2016] recently proposed a new fairness definition—equality of opportunity—that improves on demographic parity in terms of classification utility. Discrimination in *black-box* systems has been studied through the lens of statistical analysis [Datta et al. 2016, 2015; Sweeney 2013]. Notably, Datta et al. [2015] created an automated tool that analyzes online advertising: it operates dynamically by surveying the ads produced by Google.

In this paper, we viewed fairness through the lens of program specification and verification. We are given a decision-making program—perhaps written by an expert or automatically generated from data—and we would like to prove that it satisfies some fairness criterion with respect to a probabilistic model of the population. We envision that in the future, for instance, governing bodies might issue population models, and those employing automated decision-making have to certify fairness of their procedures with respect to those models. Along those lines, in the US, two recent White House reports [WH 2014, 2016] warn that *"Powerful algorithms ... raise the potential of*

encoding discrimination in automated decisions," and recommend that federal agencies *"should take extra care to ensure the efficacy and fairness of those systems, based on evidence-based verification and validation."*

Probabilistic abstract interpretation We refer the reader to Gordon et al. [2014] for a thorough survey on probabilistic program analysis. A number of works tackled analysis of probabilistic programs from an abstract interpretation perspective [Claret et al. 2013; Mardziel et al. 2011; Monniaux 2000, 2001a,b]. The comparison between our solution through volume computation and abstract interpretation is perhaps analogous to SMT solving and software model checking versus abstract interpretation. For example, techniques proposed by Monniaux [2000] sacrifice precision of the analysis (through joins, abstraction, etc.) for the benefit of efficiency. Our approach, on the other hand, is aimed at eventually producing a proof, or iteratively improving probability bounds while guaranteeing convergence.

Sampling-based inference In probabilistic verification, some techniques perform probabilistic inference by compiling programs or program paths to Bayesian networks [Koller and Friedman 2009] and applying hypothesis testing [Sampson et al. 2014]. The verification technique proposed by Sampson et al. [2014] applies to properties of the form $\mathbb{P}[\varphi] > c$. The approach relies on concentration inequalities to determine a number of samples (executions) that would provide a result within an ϵ additive error with $1 - \delta$ probability. In the case of properties where we have a ratio over two probabilities—like the ones considered here—we cannot a priori determine the number of samples required to achieve (ϵ, δ) guarantees.

Probabilistic programming languages often rely on sampling to approximate the posterior distribution of a program. The Church [Goodman et al. 2008] programming language, for instance, employs the *Metropolis–Hastings* algorithm [Chib and Greenberg 1995], a *Markov Chain Monte Carlo* (MCMC) technique. In MCMC techniques, there is usually no guarantee on how different the Markov chain is from the actual distribution at any point in execution, although the Markov chain is guaranteed to converge in the limit.

Volume computation The computation of weighted volume is known to be hard—even for a convex polytope, volume computation is #P-hard [Khachiyan 1993]. Two general approaches exist: approximate and exact solutions. Note that in general, any approximate technique at best can prove facts *with high probability*.

Our volume computation algorithm is inspired by (*i*) the formula decomposition procedure of Li et al. [2014], where quantifier elimination is used to underapproximate an LRA constraint as a Boolean combination of monadic predicates; and (*ii*) the technique for bounding the weighted volume of a polyhedron introduced by Sankaranarayanan et al. [2013], which is the closest volume computation work to ours. (The general technique of approximating complex regions with unions of orthogonal polyhedra is well-studied in the hybrid systems literature [Bournez et al. 1999].)

A number of factors differentiate our work from that of Sankaranarayanan et al. [2013], which we compared with experimentally in Sec. 6. First, our approach is more general, in that it can operate on Boolean formulas over linear and polynomial inequalities, as opposed to just conjunctions of linear inequalities. Second, our approach employs ADFs to guide the sampling of hyperrectangles with large volume, which, as we have demonstrated experimentally, is a crucial feature of our approach. Third, we provide theoretical convergence guarantees.

LattE is a tool that performs exact integration of polynomial functions over polytopes [De Loera et al. 2012]. Belle et al. [2016, 2015a] compute the volume of a linear real arithmetic (LRA) formula by, effectively, decomposing it into DNF—a set of polytopes—and using LattE to compute the volume of each polyhedron with respect to piece-wise polynomial densities. Our volume computation

algorithm is more general in that it (*i*) handles formulas over real closed fields, which subsumes LRA, and (*ii*) handles probability distributions for which we can evaluate the CDF. Our implementation also supports polynomial approximations of the ADFs. Polynomial approximations provide better samples, but since the polynomials introduce non-linear constraints, the actual SMT calls become dramatically slower due to the lack of scalable solvers for non-linear arithmetic. This includes Z3's non-linear solver [Jovanović and de Moura 2013], which implements a variant of cylindrical algebraic decomposition (CAD) [Basu et al. 2006], a technique for solving non-linear constraints implemented in tools such as Mathematica and Maple. Although Z3 was shown to be faster than all other non-linear solvers [Jovanović and de Moura 2013], it still does not scale to formulas of size we consider, making polynomial approximations currently ineffective in practice. The same applies to Redlog, which we used for quantifier elimination, and other state-of-the-art tools such as QEPCAD [Brown 2003]; they implement CAD and are comparable to Z3's non-linear solver in performance [Jovanović and de Moura 2013].

Chistikov et al. [2015] present a framework for approximate counting with probabilistic guarantees in SMT theories, which they specialize for bounded LRA. In contrast, our technique (*i*) handles unbounded formulas in LRA as well as real closed fields, (*ii*) handles arbitrary distributions, and (*iii*) provides converging lower-bound guarantees. It is important to note that there is a also a rich body of work investigating randomized polynomial algorithms for approximating the volume of a polytope, beginning with the seminal work of Dyer et al. [1991] (see Vempala [2005] for a survey).

Probabilistic verification with model counting A number of works have also addressed probabilistic analysis through symbolic execution [Filieri et al. 2013; Geldenhuys et al. 2012; Sampson et al. 2014; Sankaranarayanan et al. 2013]. Filieri et al. [2013] and Geldenhuys et al. [2012] attempt to find the probability a safety invariant is preserved. Both methods reduce to a weighted model counting approach and are thus effectively restricted to variables over finite domains. Note that our technique is more general than a model counting approach, as we can handle discrete cases with a proper encoding of the variables into a continuous domain without loss of precision.

8 CONCLUSION

We formalized notions of fairness as probabilistic postconditions of decision-making programs. We presented a novel probabilistic verification technique that is well-suited in its expressiveness, performance, and guarantees to the fairness verification problem. We implemented our proposed ideas and applied them to a range of decision-making programs. Our results highlight the power of our approach and the importance of our design decisions.

An important direction for future research is investigating how to repair an unfair program. In recent work [Albarghouthi et al. 2017], we began investigating this problem for a simple class of loop-free programs and properties. Another interesting problem is pinpointing parts of the program that lead to unfairness—in other words, explaining why the program is unfair. In traditional verification, a counterexample is a clear artifact that falsifies a postcondition. In the probabilistic setting, however, there is no single execution trace that explains why a postcondition does not hold. Exploring debugging in the probabilistic setting is an interesting problem for future work.

Acknowledgements We would like to thank Thomas Reps and Aaron Roth for giving us feedback on earlier drafts of the paper, Shuchi Chawla and Jerry Zhu for long and detailed discussions, Sriram Sankaranarayanan for help with his tool, and the OOPSLA reviewers for their suggestions. This paper is based upon work supported by the National Science Foundation under Grant numbers 1566015 and 1704117.

REFERENCES

Ifeoma Ajunwa, Sorelle Friedler, Carlos E Scheidegger, and Suresh Venkatasubramanian. 2016. Hiring by algorithm: predicting and preventing disparate impact. *Available at SSRN 2746078* (2016).

Aws Albarghouthi, Loris D'Antoni, and Samuel Drews. 2017. *Repairing Decision-Making Programs Under Uncertainty.* Springer, Cham, 181–200.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine Bias: There's Software Used Across the Country to Predict Future Criminals. And it's Biased Against Blacks. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing. (May 2016). (Accessed on 06/18/2016).

Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. 2000. Approximate reachability analysis of piecewise-linear dynamical systems. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 20–31.

Mike Barnett and K Rustan M Leino. 2005. Weakest-precondition of unstructured programs. In *ACM SIGSOFT Software Engineering Notes*, Vol. 31. ACM, 82–87.

Solon Barocas and Andrew D Selbst. 2014. Big data's disparate impact. *Available at SSRN 2477899* (2014).

Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. 2004. Secure Information Flow by Self-Composition. In *CSFW*.

Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, César Kunz, and Pierre-Yves Strub. 2014. Proving Differential Privacy in Hoare Logic. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. 411–424. https://doi.org/10.1109/CSF.2014.36

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. *CoRR* abs/1605.07262 (2016). http://arxiv.org/abs/1605.07262

Saugata Basu, Richard Pollack, and Marie-Françoise Roy. 2006. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Vaishak Belle, Guy Van den Broeck, and Andrea Passerini. 2016. Component Caching in Hybrid Domains with Piecewise Polynomial Densities. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. 3369–3375. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12542

Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. 2015a. Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. 2770–2776. http://ijcai.org/Abstract/15/392

Vaishak Belle, Guy Van den Broeck, and Andrea Passerini. 2015b. Hashing-based approximate probabilistic inference in hybrid domains. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI)*.

Nate Berg. 2014. Predicting crime, LAPD-style. https://www.theguardian.com/cities/2014/jun/25/predicting-crime-lapd-los-angeles-police-data-analysis-algorithm-minority-report. (June 2014). (Accessed on 06/18/2016).

Christopher M Bishop. 2006. Pattern recognition. *Machine Learning* 128 (2006).

Olivier Bournez, Oded Maler, and Amir Pnueli. 1999. *Orthogonal Polyhedra: Representation and Computation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 46–60. https://doi.org/10.1007/3-540-48983-5_8

Christopher W. Brown. 2003. QEPCAD B: A Program for Computing with Semi-algebraic Sets Using CADs. *SIGSAM Bull.* 37, 4 (Dec. 2003), 97–108. https://doi.org/10.1145/968708.968710

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

Michael Carbin, Sasa Misailovic, and Martin C. Rinard. 2013. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*. 33–52. https://doi.org/10.1145/2509136.2509546

Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. 2014. Bridging boolean and quantitative synthesis using smoothed proof search. In *POPL*, Vol. 49. ACM, 207–220.

Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara Navidpour. 2011. Proving Programs Robust. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 102–112. https://doi.org/10.1145/2025113.2025131

Siddhartha Chib and Edward Greenberg. 1995. Understanding the metropolis-hastings algorithm. *The american statistician* 49, 4 (1995), 327–335.

Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. 2015. Approximate Counting in SMT and Value Estimation for Probabilistic Programs. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. 320–334. https://doi.org/10.1007/978-3-662-46681-0_26

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 Clarke, Daniel Kroening, and Flavio Lerda. 2004. A tool for checking ANSI-C programs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 168–176.

Ron Cytron, Jeanne Ferrante, Barry K Rosen, Mark N Wegman, and F Kenneth Zadeck. 1991. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 13, 4 (1991), 451–490.

Anupam Datta, Shayak Sen, and Yair Zick. 2016. Algorithmic Transparency via Quantitative Input Influence. In *Proceedings of 37th IEEE Symposium on Security and Privacy*.

Amit Datta, Michael Carl Tschantz, and Anupam Datta. 2015. Automated experiments on Ad privacy settings. *Proceedings on Privacy Enhancing Technologies* 2015, 1 (2015), 92–112.

JA De Loera, Brandon Dutra, Matthias Koeppe, Stanislav Moreinis, Gregory Pinto, and Jianqiu Wu. 2012. Software for exact integration of polynomials over polyhedra. *ACM Communications in Computer Algebra* 45, 3/4 (2012), 169–172.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

Cynthia Dwork. 2006. Differential privacy. In *Automata, languages and programming*. Springer, 1–12.

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2012. Fairness through awareness. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. 214–226. https://doi.org/10.1145/2090236.2090255

Martin Dyer, Alan Frieze, and Ravi Kannan. 1991. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM (JACM)* 38, 1 (1991), 1–17.

Martin E. Dyer and Alan M. Frieze. 1988. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.* 17, 5 (1988), 967–974.

EEOC. 2014. Code of Federal Regulations. https://www.gpo.gov/fdsys/pkg/CFR-2014-title29-vol4/xml/CFR-2014-title29-vol4-part1607.xml. (July 2014). (Accessed on 06/18/2016).

Virginia Eubanks. 2015. The dangers of letting algorithms enforce policy. http://www.slate.com/articles/technology/future_tense/2015/04/the_dangers_of_letting_algorithms_enforce_policy.html. (April 2015). (Accessed on 06/18/2016).

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, Sydney, NSW, Australia, August 10-13, 2015*. 259–268. https://doi.org/10.1145/2783258.2783311

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.

Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. 2016. On the (im)possibility of fairness. *CoRR* abs/1609.07236 (2016). http://arxiv.org/abs/1609.07236

Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer aided verification*. Springer.

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.

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*. 220–229.

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.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.

Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of Opportunity in Supervised Learning. *CoRR* abs/1610.02413 (2016). http://arxiv.org/abs/1610.02413

Dejan Jovanović and Leonardo de Moura. 2013. Solving Non-linear Arithmetic. *ACM Commun. Comput. Algebra* 46, 3/4 (Jan. 2013), 104–105. https://doi.org/10.1145/2429135.2429155

Leonid Khachiyan. 1993. *Complexity of polytope volume computation*. Springer.

Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. 2017. Inherent Trade-Offs in the Fair Determination of Risk Scores. In *ITCS*.

Nicole Kobie. 2016. Who do you blame when an algorithm gets you fired? http://www.wired.co.uk/article/make-algorithms-accountable. (January 2016). (Accessed on 06/18/2016).

Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350.

Yi Li, Tian Huat Tan, and Marsha Chechik. 2014. Management of time requirements in component-based systems. In *FM 2014: Formal Methods*. Springer, 399–415.

Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. 2011. Dynamic enforcement of knowledge-based security policies. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*. IEEE, 114–128.

Claire Cain Miller. 2015. Can an Algorithm Hire Better Than a Human? http://www.nytimes.com/2015/06/26/upshot/can-an-algorithm-hire-better-than-a-human.html. (June 2015). (Accessed on 06/18/2016).

David Monniaux. 2000. Abstract interpretation of probabilistic semantics. In *Static Analysis*. 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 *Programming Languages and Systems*. Springer, 367–382.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 807–814.

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.

Walt L Perry. 2013. *Predictive policing: The role of crime forecasting in law enforcement operations*. Rand Corporation.

John Rawls. 2009. *A theory of justice*. Harvard university press.

Salvatore Ruggieri. 2014. Using t-closeness anonymity to control for non-discrimination. *Transactions on Data Privacy* 7, 2 (2014), 99–129.

Adrian Sampson, Pavel Panchekha, Todd Mytkowicz, Kathryn S McKinley, Dan Grossman, and Luis Ceze. 2014. Expressing and verifying probabilistic assertions. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 112–122.

Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*. 447–458. https://doi.org/10.1145/2462156.2462179

Latanya Sweeney. 2013. Discrimination in online ad delivery. *Queue* 11, 3 (2013), 10.

Andrew Tutt. 2016. An FDA for Algorithms. *Available at SSRN 2747994* (2016).

Jennifer Valentino-Devries, Jeremy Singer-Vine, and Ashkan Soltani. 2012. Websites Vary Prices, Deals Based on Users' Information. http://www.wsj.com/articles/SB10001424127887323777204578189391813881534. (December 2012). (Accessed on 06/18/2016).

Santosh Vempala. 2005. Geometric random walks: a survey. *Combinatorial and computational geometry* 52, 573-612 (2005), 2.

WH. 2014. Big Data: Seizing Opportunities, Preserving Values. https://www.whitehouse.gov/sites/default/files/docs/big_data_privacy_report_may_1_2014.pdf. (May 2014). (Accessed on 06/18/2016).

WH. 2016. Preparing for the Future of Artificial Intelligence. https://www.whitehouse.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf. (Octorber 2016). (Accessed on 10/15/2016).

Tal Zarsky. 2014. Understanding discrimination in the scored society. *Washington Law Review* 89, 4 (2014).

Richard S. Zemel, Yu Wu, Kevin Swersky, Toniann Pitassi, and Cynthia Dwork. 2013. Learning Fair Representations. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. 325–333. http://jmlr.org/proceedings/papers/v28/zemel13.html