Perfectly Parallel Fairness Certification of Neural Networks

CATERINA URBAN, INRIA and DIENS, École Normale Supérieure, CNRS, PSL University, France
MARIA CHRISTAKIS, MPI-SWS, Germany
VALENTIN WÜSTHOLZ, ConsenSys, Germany
FUYUAN ZHANG, MPI-SWS, Germany

Recently, there is growing concern that machine-learned software, which currently assists or even automates decision making, reproduces, and in the worst case reinforces, bias present in the training data. The development of tools and techniques for certifying fairness of this software or describing its biases is, therefore, critical. In this paper, we propose a *perfectly parallel* static analysis for certifying *fairness* of feed-forward neural networks used for classification of tabular data. When certification succeeds, our approach provides definite guarantees, otherwise, it describes and quantifies the biased input space regions. We design the analysis to be *sound*, in practice also *exact*, and configurable in terms of scalability and precision, thereby enabling *pay-as-you-go certification*. We implement our approach in an open-source tool called LIBRA and demonstrate its effectiveness on neural networks trained on popular datasets.

CCS Concepts: • **Software and its engineering → Formal software verification**; • **Theory of computation → Program analysis**; **Abstraction**; • **Computing methodologies →** *Neural networks*; • **Social and professional topics → Computing / technology policy**.

Additional Key Words and Phrases: Fairness, Neural Networks, Abstract Interpretation, Static Analysis

ACM Reference Format:
Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. 2020. Perfectly Parallel Fairness Certification of Neural Networks. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 185 (November 2020), 30 pages. https://doi.org/10.1145/3428253

1 INTRODUCTION

Due to the tremendous advances in machine learning and the vast amounts of available data, machine-learned software has an ever-increasing important role in assisting or even autonomously making decisions that impact our lives. We are already witnessing the wide adoption and societal impact of such software in criminal justice, health care, and social welfare, to name a few examples. It is not far-fetched to imagine a future where most of the decision-making is automated.

However, several studies have recently raised concerns about the fairness of such systems. For instance, consider a commercial recidivism-risk assessment algorithm that was found racially biased [Larson et al. 2016]. Similarly, a commercial algorithm that is widely used in the U.S. health care system falsely determined that Black patients were healthier than other equally sick patients by using health costs to represent health needs [Obermeyer et al. 2019]. There is also empirical evidence of gender bias in image searches, for instance, there are fewer results depicting women when searching for certain occupations, such as CEO [Kay et al. 2015]. Commercial facial recognition

Authors' addresses: Caterina Urban, INRIA and DIENS, École Normale Supérieure, CNRS, PSL University, Paris, France, caterina.urban@inria.fr; Maria Christakis, MPI-SWS, Germany, maria@mpi-sws.org; Valentin Wüstholz, ConsenSys, Germany, valentin.wustholz@consensys.net; Fuyuan Zhang, MPI-SWS, Germany, fuyuan@mpi-sws.org.

2475-1421/2020/11-ART185
https://doi.org/10.1145/3428253

algorithms, which are increasingly used in law enforcement, are less effective for women and darker skin types [Buolamwini and Gebru 2018].

In other words, machine-learned software may reproduce, or even reinforce, bias that is directly or indirectly present in the training data. This awareness will certainly lead to regulations and strict audits in the future. It is, therefore, critical to develop tools and techniques for certifying fairness of machine-learned software or understanding the circumstances of its biased behavior.

Feed-Forward Neural Networks. We make a step forward in meeting these needs by designing a novel static analysis framework for certifying fairness of *feed-forward neural networks* used *for classification of tabular data* (e.g., stored in Excel files or relational databases).

While, in general, machine learning approaches such as random forests or gradient boosting are preferred over neural networks for analyzing tabular data, for some datasets, feed-forward neural networks provide better accuracy and flexibility at the cost of putting slightly more effort into hyperparameter optimizations. This is the case, for instance, for very large datasets (i.e., with billions of rows) or data that comes in batches over time. As these kinds of datasets are becoming more and more common, so are neural network-based decision-making software systems.

The other challenge is that neural networks are harder to interpret than such machine learning approaches and, thus, it becomes harder to ensure that neural network-based decision-making software is doing what intended. This is precisely what our work aims to address.

Dependency Fairness. Our static analysis framework is designed for certifying *dependency fairness* [Galhotra et al. 2017][1] of feed-forward neural networks used for classification tasks. Specifically, given a choice (e.g., driven by a causal model [Pearl 2009] or a correlation analysis) of input features that are considered (directly or indirectly) sensitive to bias, *a neural network is fair if the output classification is not affected by different values of the chosen features.* We chose this definition over other fairness notions because it does not require an oracle and it is specifically designed to be testable and, thus, also amenable to static analysis. It is also a stronger notion than group-based fairness notions (see Section 12 for a more in-depth comparison).

Of course, an obvious way to avoid dependencies between the classification and the sensitive features is to entirely remove the sensitive features from the training data [Grgić-Hlača et al. 2016]. However, this in general does not work for three main reasons. First, neural networks learn from latent variables (e.g., [Lum and Isaac 2016; Udeshi et al. 2018]). For instance, a credit-screening algorithm that does not take race (or gender) as an input might still be biased with respect to it, say, by using the ZIP code of applicants as proxy for race (or their first name as proxy for gender). Therefore, simply removing a sensitive feature does not necessarily free the training data or the corresponding trained neural network from bias. Second, the training data is only a relatively small sample of the entire input space, on portions of which the neural network might end up being biased even if trained with fair data. For example, if women are under-represented in the training data, this leaves freedom to the training process and thus the resulting classifier might end up being biased against them. Third, the information provided by a sensitive feature might be necessary for business necessity [Barocas and Selbst 2016], for instance, to introduce intended bias in a certain input region. Assume a credit-screening model that should discriminate with respect to age only above a particular threshold. Above this age threshold, the higher the requested credit amount, the lower the chances of receiving it. In such cases, removing the sensitive feature is not even possible.

[1] Note that, Galhotra et al. [Galhotra et al. 2017] use the term *causal fairness* instead of *dependency fairness*. We renamed it to avoid confusion since this fairness notion does not presume a given *causal model* [Pearl 2009] but instead looks at whether the classification *depends* on the value of the sensitive features.

Our Approach. Static analysis of neural networks is still in its infancy. Most of the work in the area focuses on certifying local robustness of neural networks against adversarial examples [Gehr et al. 2018; Huang et al. 2017; Singh et al. 2019, etc.] or proving functional properties of neural networks [Katz et al. 2017; Wang et al. 2018, etc.]. On the other hand, dependency fairness is a *global* property, relative to the entire input space, instead of only those inputs within a particular distance metric or that satisfy a given functional specification. It is thus much harder to verify. The approach that we propose in this paper brings us closer to the aspiration for a practical general verification approach of global neural network properties.

Our approach certifies dependency fairness of feed-forward neural networks by employing a combination of a forward and a backward static analysis. On a high level, the forward pass aims to reduce the overall analysis effort. At its core, it divides the input space of the network into independent partitions. The backward analysis then attempts to certify fairness of the classification within each partition (in a *perfectly parallel* fashion) with respect to a chosen (set of) feature(s), which may be directly or indirectly sensitive, for instance, race or ZIP code. In the end, our approach reports for which regions of the input space the neural network is certified to be fair and for which there is bias. Note that we do not necessarily need to analyze the entire input space; our technique is also able to answer specific bias queries about a fraction of the input space, e.g., are Hispanics over 45 years old discriminated against with respect to gender?

The scalability-vs-precision tradeoff of our approach is configurable. Partitions that do not satisfy the given configuration are excluded from the analysis and may be resumed later, with a more flexible configuration. This enables usage scenarios in which our approach adapts to the available resources, e.g., time or CPUs, and is run incrementally. In other words, we designed a *pay-as-you-go certification* approach that the more resources it is given, the larger the region of the input space it is able to analyze.

Related Work. In the literature, related work on determining fairness of machine-learning models has focused on providing probabilistic guarantees [Bastani et al. 2019]. In contrast, our approach gives definite guarantees for those input partitions that satisfy the analysis configuration. Similarly to our approach, there is work that also aims to provide definite guarantees [Albarghouthi et al. 2017b] (although for different fairness criteria). However, it has been shown to scale only up to neural networks with two hidden neurons. We demonstrate that our approach can effectively analyze networks with hundreds (or, in some cases, even thousands) of hidden neurons. Ours is also the first approach in this area that is configurable in terms of scalability and precision.

Contributions. We make the following contributions:

(1) We propose a novel perfectly parallel static analysis approach for certifying dependency fairness of feed-forward neural networks used for classification of tabular data. If certification fails, our approach can describe and quantify the biased input space region(s).
(2) We rigorously formalize our approach and show that it is sound and, in practice, exact for the analyzed regions of the input space.
(3) We discuss the configurable scalability-vs-precision tradeoff of our approach that enables pay-as-you-go certification.
(4) We implement our approach in an open-source tool called LIBRA and extensively evaluate it on neural networks trained with popular datasets. We show the effectiveness of our approach in detecting injected bias and answering bias queries. We also experiment with the precision and scalability of the analysis and discuss the tradeoffs.

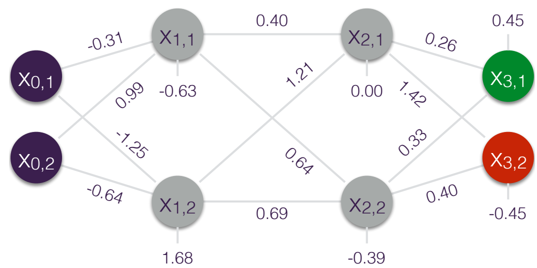


Fig. 1. Small, constructed example of trained feed-forward neural network for credit approval.

2 OVERVIEW

In this section, we give an overview of our approach using a small constructed example, which is shown in Figure 1.

Example. The figure depicts a feed-forward neural network for credit approval. There are two inputs $x_{0,1}$ and $x_{0,2}$ (shown in purple). Input $x_{0,1}$ denotes the requested credit amount and $x_{0,2}$ denotes age. Both inputs have continuous values in the range $[0, 1]$. Output $x_{3,2}$ (shown in green) denotes that the credit request is approved, whereas $x_{3,1}$ (in red) denotes that it is denied. The neural network also consists of two hidden layers with two nodes each (in gray).

Now, let us assume that this neural network is trained to deny requests for large credit amounts from older people. Otherwise, the network should not discriminate with respect to age for small credit amounts. There should also not be discrimination for younger people with respect to the requested credit amount. When choosing age as the sensitive input, our approach can certify fairness with respect to different age groups for small credit amounts. Our approach is also able to find (as well as quantify) bias with respect to age for large credit amounts. Note that, in general, bias found by the analysis may be intended or accidental — our analysis does not aim to address this question. It is up to the user to decide whether the result of the analysis is expected or not.

Our approach does not require age to be an input of the neural network. For example, $x_{0,2}$ could denote the ZIP code of credit applicants, and the network could still use it as proxy for age. That is, requests for large credit amounts are denied for a certain range of ZIP codes (where older people tend to live), yet there is no discrimination between ZIP codes for small credit amounts. When choosing the ZIP code as the sensitive input, our approach would similarly be able to detect bias with respect to it for large credit amounts. In general, the choice of the sensitive features is up to the user, e.g., it can be driven by knowledge of proxies coming from a separate correlation analysis.

Below, we present on a high level how our approach achieves these results.

Naïve Approach. In theory, the simplest way to certify dependency fairness is to first analyze the neural network backwards starting from each output node, in our case $x_{3,1}$ and $x_{3,2}$. This allows us to determine the regions of the input space (i.e., age and requested credit amount) for which credit is approved and denied. For example, assume that we find that requests are denied for credit amounts larger than 10 000 (i.e., $10\,000 < x_{0,1}$) and age greater than 60 (i.e., $60 < x_{0,2}$), while they are approved for $x_{0,1} \leq 10\,000$ and $60 < x_{0,2}$ or for $x_{0,2} \leq 60$.

The second step is to forget the value of the sensitive input (i.e., age) or, in other words, to project these regions over the credit amount. In our example, after projection we have that credit requests are denied for $10\,000 < x_{0,1}$ and approved for any value of $x_{0,1}$. A non-empty intersection between the projected input regions indicates bias with respect to the sensitive input. In our example, the intersection is non-empty for $10\,000 < x_{0,1}$: there exist people that differ in age but request the same credit amount (greater than 10 000), some of whom receive the credit while others do not.

This approach, however, is not practical. Specifically, in neural networks with ReLU activation functions (see Section 3 for more details, other activation functions are discussed in Section 9), each hidden node effectively represents a disjunction between two activation statuses (active and inactive). In our example, there are 2^4 possible activation patterns for the 4 hidden nodes. To retain maximum precision, the analysis would have to explore all of them, which does not scale in practice.

Our Approach. Our analysis is based on the observation that *there might exist many activation patterns that do not correspond to a region of the input space* [Hanin and Rolnick 2019]. Such patterns can, therefore, be ignored during the analysis. We push this idea further by defining *abstract activation patterns*, which fix the activation status of only certain nodes and thus represent sets of (concrete) activation patterns. Typically, *a relatively small number of abstract activation patterns is sufficient for covering the entire input space*, without necessarily representing and exploring all possible concrete patterns.

Identifying those patterns that definitely correspond to a region of the input space is only possible with a forward analysis. Hence, we combine a forward pre-analysis with a backward analysis. The pre-analysis partitions the input space into independent partitions corresponding to abstract activation patterns. Then, the backward analysis tries to prove fairness of the neural network for each such partition.

More specifically, we set an upper bound U on the number of tolerated disjunctions (i.e., on the number of nodes with an unknown activation status) per abstract activation pattern. Our forward pre-analysis uses a cheap abstract domain (e.g., the boxes domain [Cousot and Cousot 1976]) to *iteratively* partition the input space along the *non-sensitive* input dimensions to obtain *fair* input partitions (i.e., boxes). Each partition satisfies one of the following conditions: (a) its classification is already fair because only one network output is reachable for all inputs in the region, (b) it has an abstract activation pattern with at most U unknown nodes, or (c) it needs to be partitioned further. We call partitions that satisfy condition (b) *feasible*.

In our example, let $U = 2$. At first, the analysis considers the entire input space, that is, $x_{0,1} : [0, 1]$ (credit amount) and $x_{0,2} : [0, 1]$ (age). (Note that we could also specify a subset of the input space for analysis.) The abstract activation pattern corresponding to this initial partition I is ϵ (i.e., no hidden nodes have fixed activation status) and, thus, the number of disjunctions would be 4, which is greater than U. Therefore, I needs to be divided into I_1 ($x_{0,1} : [0, 0.5].x_{0,2} : [0, 1]$) and I_2 ($x_{0,1} : [0.5, 1].x_{0,2} : [0, 1]$). Observe that the input space is not split with respect to $x_{0,2}$, which is the sensitive input. Now, I_1 is feasible since its abstract activation pattern is $x_{1,2}x_{2,1}x_{2,2}$ (i.e., 3 nodes are always active), while I_2 must be divided further since its abstract activation pattern is ϵ.

To control the number of partitions, we impose a lower bound L on the size of each of their dimensions. Partitions that require a dimension of a smaller size are *excluded*. In other words, they are not considered until more analysis *budget* becomes available, that is, a larger U or a smaller L.

In our example, let $L = 0.25$. The forward pre-analysis further divides I_2 into $I_{2,1}$ ($x_{0,1} : [0.5, 0.75].x_{0,2} : [0, 1]$) and $I_{2,2}$ ($x_{0,1} : [0.75, 1].x_{0,2} : [0, 1]$). Now, $I_{2,1}$ is feasible, with abstract pattern $x_{1,2}x_{2,1}$, while $I_{2,2}$ is not. However, $I_{2,2}$ may not be split further because the size of the only non-sensitive dimension $x_{0,1}$ has already reached the lower bound L. As a result, $I_{2,2}$ is excluded, and only the remaining 75% of the input space is considered for analysis.

Next, feasible input partitions (within bounds L and U) are grouped by abstract activation patterns. In our example, the pattern corresponding to I_1, namely $x_{1,2}x_{2,1}x_{2,2}$, is subsumed by the (more abstract) pattern of $I_{2,1}$, namely $x_{1,2}x_{2,1}$. Consequently, we group I_1 and $I_{2,1}$ under pattern $x_{1,2}x_{2,1}$.

The backward analysis is then run *in parallel* for each representative abstract activation pattern, in our example $x_{1,2}x_{2,1}$. This analysis determines the region of the input space (within a given partition group) for which each output of the neural network is returned, e.g., credit is approved for $c_1 \leq x_{0,1} \leq c_2$ and $a_1 \leq x_{0,2} \leq a_2$. To achieve this, the analysis uses an expensive abstract domain, for instance, disjunctive or powerset polyhedra [Cousot and Cousot 1979; Cousot and Halbwachs 1978], and leverages abstract activation patterns to avoid disjunctions. For instance, pattern $x_{1,2}x_{2,1}$ only requires reasoning about two disjunctions from the remaining hidden nodes $x_{1,1}$ and $x_{2,2}$.

Finally, fairness is checked for each partition in the same way that it is done by the naïve approach for the entire input space. In our example, we prove that the classification within I_1 is fair and determine that within $I_{2,1}$ the classification is biased. Concretely, our approach determines that bias occurs for $0.54 \leq x_{0,1} \leq 0.75$, which corresponds to 21% of the entire input space (assuming a uniform probability distribution). In other words, the network returns different outputs for people that request the same credit in the above range but differ in age. Recall that partition $I_{2,2}$, where $0.75 \leq x_{0,1} \leq 1$, was excluded from analysis, and therefore, we cannot draw any conclusions about whether there is any bias for people requesting credit in this range.

Note that bias may also be quantified according to a probability distribution of the input space. In particular, it might be that credit requests in the range $0.54 \leq x_{0,1} \leq 0.75$ are more (resp. less) common in practice. Given their probability distribution, our analysis computes a tailored percentage of bias, which in this case would be greater (resp. less) than 21%.

3 FEED-FORWARD DEEP NEURAL NETWORKS

Formally, a *feed-forward deep neural network* consists of an input layer (L_0), an output layer (L_N), and a number of hidden layers ($L_1, \ldots, L_{N-1}$) in between. Each layer L_i contains $|L_i|$ nodes and, with the exception of the input layer, is associated to a $|L_i| \times |L_{i-1}|$-matrix W_i of weight coefficients and a vector B_i of $|L_i|$ bias coefficients. In the following, we use X to denote the set of all nodes, X_i to denote the set of nodes of the ith layer, and $x_{i,j}$ to denote the jth node of the ith layer of a neural network. We focus here on neural networks used for *classification* tasks. Thus, $|L_N|$ is the number of target classes (e.g., 2 classes in Figure 1).

The value of the input nodes is given by the input data: continuous data is represented by one input node (e.g., $x_{0,1}$ or $x_{0,2}$ in Figure 1), while categorical data is represented by multiple input nodes via one-hot encoding. In the following, we use K to denote the subset of input nodes considered (directly or indirectly) *sensitive* to bias (e.g., $x_{0,2}$ in Figure 1) and $\overline{K} \stackrel{\text{def}}{=} X_0 \setminus K$ to denote the input nodes not deemed sensitive to bias.

The value of each hidden and output node $x_{i,j}$ is computed by an *activation function* f applied to a linear combination of the values of all nodes in the preceding layer [Goodfellow et al. 2016], i.e., $x_{i,j} = f\left(\sum_k^{|L_{i-1}|} w^i_{j,k} \cdot x_{i-1,k} + b_{i,j}\right)$, where $w^i_{j,k}$ and $b_{i,j}$ are weight and bias coefficients in W_i and B_i, respectively. In a *fully-connected neural network*, all $w^i_{j,k}$ are non-zero. Weights and biases are adjusted during the *training phase* of the neural network. In what follows, we focus on already trained neural networks, which we call *neural-network models*.

Nowadays, the most commonly used activation for hidden nodes is the Rectified Linear Unit (ReLU) [Nair and Hinton 2010]: $\text{ReLU}(x) = \max(x, 0)$. In this case, the activation used for output nodes is the identity function. The output values are then normalized into a probability distribution on the target classes [Goodfellow et al. 2016]. We discuss other activation functions in Section 9.

4 TRACE SEMANTICS

In the following, we represent neural-network models as sequential programs. These programs consist of assignments for computing the activation value of each node (e.g., $x_{1,1} = -0.31 * x_{0,1} + 0.99 * x_{0,2} - 0.63$ in Figure 1) and implementations of activation functions (e.g., if-statements for ReLUs). As is standard practice in static program analysis, we define a semantics for these programs that is tailored to our property of interest, dependency fairness, and use it to prove soundness of our approach. In particular, this allows us to formally prove soundness of the *parallelization* of our analysis as well (cf. Section 9), unlike for most of the existing approaches in which the parallelization is only introduced as part of the implementation and never proven sound.

The *semantics* of a neural-network model is a mathematical characterization of its behavior when executed for all possible input data. We model the operational semantics of a feed-forward neural-network model M as a transition system $\langle \Sigma, \tau \rangle$, where Σ is a (finite but exceedingly large) set of states and the *acyclic* transition relation $\tau \subseteq \Sigma \times \Sigma$ describes the possible transitions between states [Cousot 2002; Cousot and Cousot 1977].

More specifically, a state $s \in \Sigma$ maps neural-network nodes to their values. Here, for simplicity, we assume that nodes have real values, i.e., $s\colon \mathrm{X} \to \mathbb{R}$. (We discuss floating-point values in Section 9.) In the following, we often only care about the values of a subset of the neural-network nodes in certain states. Thus, let $\Sigma_{|_Y} \stackrel{\text{def}}{=} \left\{ s_{|_Y} \mid s \in \Sigma \right\}$ be the restriction of Σ to a domain of interest Y. Sets $\Sigma_{|_{\mathrm{X}_0}}$ and $\Sigma_{|_{\mathrm{X}_\mathrm{N}}}$ denote restrictions of Σ to the network nodes in the input and output layer, respectively. With a slight abuse of notation, let $\mathrm{X}_{i,j}$ denote $\Sigma_{|_{\{x_{i,j}\}}}$, i.e., the restriction of Σ to the singleton set containing $\mathrm{x}_{i,j}$. Transitions happen between states with different values for consecutive nodes in the same layer, i.e., $\tau \subseteq \mathrm{X}_{i,j} \times \mathrm{X}_{i,j+1}$, or between states with different values for the last and first node of consecutive layers of the network, i.e., $\tau \subseteq \mathrm{X}_{i,|\mathrm{L}_i|} \times \mathrm{X}_{i+1,0}$. The set $\Omega \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma\colon \langle s, s' \rangle \notin \tau\}$ is the set of final states of the neural network. These are partitioned in a set of outcomes $\mathbb{O} \stackrel{\text{def}}{=} \left\{ \left\{ s \in \Omega \mid \max \mathrm{X}_\mathrm{N} = \mathrm{x}_{\mathrm{N},i} \right\} \mid 0 \le i \le |\mathrm{L}_\mathrm{N}| \right\}$, depending on the output node with the highest value (i.e., the target class with highest probability).

Let $\Sigma^n \stackrel{\text{def}}{=} \{s_0 \cdots s_{n-1} \mid \forall i < n\colon s_i \in \Sigma\}$ be the set of all sequences of exactly n states in Σ. Let $\Sigma^+ \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}^+} \Sigma^n$ be the set of all non-empty finite sequences of states. A *trace* is a sequence of states that respects the transition relation τ, i.e., $\langle s, s' \rangle \in \tau$ for each pair of consecutive states s, s' in the sequence. We write $\overline{\Sigma}^n$ for the set of all traces of n states: $\overline{\Sigma}^n \stackrel{\text{def}}{=} \{s_0 \cdots s_{n-1} \in \Sigma^n \mid \forall i < n-1\colon \langle s_i, s_{i+1} \rangle \in \tau\}$. The *trace semantics* $\Upsilon \in \mathcal{P}(\Sigma^+)$ generated by a transition system $\langle \Sigma, \tau \rangle$ is the set of all non-empty traces terminating in Ω [Cousot 2002]:

$$\Upsilon \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}^+} \left\{ s_0 \dots s_{n-1} \in \overline{\Sigma}^n \mid s_{n-1} \in \Omega \right\} \tag{1}$$

In the rest of the paper, we write $[\![\mathrm{M}]\!]$ to denote the trace semantics of a neural-network model M.

The trace semantics fully describes the behavior of M. However, reasoning about a certain property of M does not need all this information and, in fact, is facilitated by the design of a semantics that abstracts away from irrelevant details. In the following sections, we formally define our property of interest, dependency fairness, and systematically derive, using *abstract interpretation* [Cousot and Cousot 1977], a semantics tailored to reasoning about this property.

5 DEPENDENCY FAIRNESS

A *property* is specified by its extension, that is, by the set of elements having such a property [Cousot and Cousot 1977, 1979]. Properties of neural-network models are properties of their semantics. Thus, properties of network models with trace semantics in $\mathcal{P}(\Sigma^+)$ are sets of sets of traces in $\mathcal{P}(\mathcal{P}(\Sigma^+))$. In particular, the set of neural-network properties forms a complete boolean lattice $\langle \mathcal{P}(\mathcal{P}(\Sigma^+)), \subseteq, \cup, \cap, \emptyset, \mathcal{P}(\Sigma^+) \rangle$ for subset inclusion, that is, logical implication. The strongest property is the standard *collecting semantics* $\Lambda \in \mathcal{P}(\mathcal{P}(\Sigma^+))$:

$$\Lambda \stackrel{\text{def}}{=} \{\Upsilon\} \tag{2}$$

Let $(\!|\mathrm{M}|\!)$ denote the collecting semantics of a particular neural-network model M. Then, model M satisfies a given property $\mathcal{H}$ if and only if its collecting semantics is a subset of $\mathcal{H}$:

$$\mathrm{M} \models \mathcal{H} \Leftrightarrow (\!|\mathrm{M}|\!) \subseteq \mathcal{H} \tag{3}$$

Here, we consider the property of *dependency fairness*, which expresses that the classification determined by a network model does not depend on sensitive input data. In particular, the property might interest the classification of all or just a fraction of the input space.

More formally, let $\mathbb{V}$ be the set of all possible value choices for all sensitive input nodes in K, e.g., for $\mathrm{K} = \{\mathrm{x}_{0,i}, \mathrm{x}_{0,j}\}$ one-hot encoding, say, gender information, $\mathbb{V} = \{\{1,0\}, \{0,1\}\}$; for $\mathrm{K} = \{x_{0,k}\}$ encoding continuous data, say, in the range $[0,1]$, we use binning so a possibility is, e.g., $\mathbb{V} = \{[0, 0.25], [0.25, 0.75], [0.75, 1]\}$. In the following, given a trace $\sigma \in \mathcal{P}(\Sigma^+)$, we write σ_0 and σ_ω to denote its initial and final state, respectively. We also write $\sigma_0 =_{\overline{\mathrm{K}}} \sigma'_0$ to indicate that the states σ_0 and σ'_0 agree on all values of all non-sensitive input nodes, and $\sigma_\omega \equiv \sigma'_\omega$ to indicate that σ and σ' have the same outcome $\mathrm{O} \in \mathbb{O}$. We can now formally define when the sensitive input nodes in K are *unused* with respect to a set of traces $T \in \mathcal{P}(\Sigma^+)$ [Urban and Müller 2018]. For one-hot encoded sensitive inputs[2], we have

$$\textsc{unused}_{\mathrm{K}}(T) \stackrel{\text{def}}{=} \forall \sigma \in T, \mathrm{V} \in \mathbb{V} \colon \sigma_0(\mathrm{K}) \neq \mathrm{V} \Rightarrow \exists \sigma' \in T \colon \sigma_0 =_{\overline{\mathrm{K}}} \sigma'_0 \wedge \sigma'_0(\mathrm{K}) = \mathrm{V} \wedge \sigma_\omega \equiv \sigma'_\omega, \tag{4}$$

where $\sigma_0(\mathrm{K}) \stackrel{\text{def}}{=} \{\sigma_0(\mathrm{x}) \mid \mathrm{x} \in \mathrm{K}\}$ is the image of K under σ_0. Intuitively, the sensitive input nodes in K are unused if any possible outcome in T (i.e., any outcome σ_ω of any trace σ in T) is possible from all possible value choices for K (i.e., there exists a trace σ' in T for each value choice for K with the same outcome as σ). That is, each outcome is independent of the value choice for K.

Example 5.1. Let us consider again our example in Figure 1. We write $\langle c, a \rangle \rightsquigarrow o$ for a trace starting in a state with $\mathrm{x}_{0,1} = c$ and $\mathrm{x}_{0,2} = a$ and ending in a state where o is the node with the highest value (i.e., the output class). The sensitive input $\mathrm{x}_{0,2}$ (age) is *unused* in $T = \{\langle 0.5, a \rangle \rightsquigarrow \mathrm{x}_{3,2} \mid 0 \leq a \leq 1\}$. It is instead *used* in $T' = \{\langle 0.75, a \rangle \rightsquigarrow \mathrm{x}_{3,2} \mid 0 \leq a < 0.51\} \cup \{\langle 0.75, a \rangle \rightsquigarrow \mathrm{x}_{3,1} \mid 0.51 \leq a \leq 1\}$.

The dependency-fairness property $\mathcal{F}_{\mathrm{K}}$ can now be defined as $\mathcal{F}_{\mathrm{K}} \stackrel{\text{def}}{=} \{[\![M]\!] \mid \textsc{unused}_{\mathrm{K}}([\![M]\!])\}$, that is, as the set of all neural-network models (or rather, their semantics) that do not use the values of the sensitive input nodes for classification. In practice, the property might interest just a fraction of the input space, i.e., we define

$$\mathcal{F}_{\mathrm{K}}[Y] \stackrel{\text{def}}{=} \left\{[\![\mathrm{M}]\!]^Y \mid \textsc{unused}_{\mathrm{K}}([\![\mathrm{M}]\!]^Y)\right\}, \tag{5}$$

where $Y \in \mathcal{P}(\Sigma)$ is a set of initial states of interest and the restriction $T^Y \stackrel{\text{def}}{=} \{\sigma \in T \mid \sigma_0 \in Y\}$ only contains traces of $T \in \mathcal{P}(\Sigma^+)$ that start with a state in Y. Similarly, in the rest of the paper, we write $S^Y \stackrel{\text{def}}{=} \{T^Y \mid T \in S\}$ for the set of sets of traces restricted to initial states in Y. Thus, from Equation 3, we have the following:

Theorem 5.2. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow (\!|\mathrm{M}|\!)^Y \subseteq \mathcal{F}_{\mathrm{K}}[Y]$

Proof. The proof follows trivially from Equation 3 and the definition of $\mathcal{F}_{\mathrm{K}}[Y]$ (cf. Equation 5) and $(\!|\mathrm{M}|\!)^Y$. □

6 DEPENDENCY SEMANTICS

We now use abstract interpretation to systematically derive, by successive abstractions of the collecting semantics Λ, a *sound and complete* semantics $\Lambda_{\rightsquigarrow}$ that contains only and exactly the information needed to reason about $\mathcal{F}_{\mathrm{K}}[Y]$.

[2]For continuous sensitive inputs, we can replace $\sigma_0(\mathrm{K}) \neq \mathrm{V}$ (resp. $\sigma'_0(\mathrm{K}) = \mathrm{V}$) with $\sigma_0(\mathrm{K}) \not\subseteq \mathrm{V}$ (resp. $\sigma'_0(\mathrm{K}) \subseteq \mathrm{V}$).

6.1 Outcome Semantics

Let $T_Z \stackrel{\text{def}}{=} \{\sigma \in T \mid \sigma_\omega \in Z\}$ be the set of traces of $T \in \mathcal{P}(\Sigma^+)$ that end with a state in $Z \in \mathcal{P}(\Sigma)$. As before, we write $S_Z \stackrel{\text{def}}{=} \{T_Z \mid T \in S\}$ for the set of sets of traces restricted to final states in Z. From the definition of $\mathcal{F}_K[Y]$ (and in particular, from the definition of $\textsc{unused}_K$, cf. Equation 4), we have:

Lemma 6.1. $(\!|M|\!)^Y \subseteq \mathcal{F}_K[Y] \Leftrightarrow \forall O \in \mathbb{O}\colon (\!|M|\!)^Y_O \subseteq \mathcal{F}_K[Y]$

Proof. Let $(\!|M|\!)^Y \subseteq \mathcal{F}_K[Y]$. From the definition of $(\!|M|\!)^Y$ (cf. Equation 2), we have that $[\![M]\!]^Y \in \mathcal{F}_K[Y]$. Thus, from the definition of $\mathcal{F}_K[Y]$ (cf. Equation 5), we have $\textsc{unused}_K([\![M]\!]^Y)$. Now, from the definition of $\textsc{unused}_K$ (cf. Equation 4), we equivalently have $\forall O \in \mathbb{O}\colon \textsc{unused}_K([\![M]\!]^Y_O)$. Thus, we can conclude that $\forall O \in \mathbb{O}\colon (\!|M|\!)^Y_O \subseteq \mathcal{F}_K[Y]$. ◻

In particular, this means that in order to determine whether a neural-network model M satisfies dependency fairness, we can independently verify, for each of its possible target classes $O \in \mathbb{O}$, that the values of its sensitive input nodes are unused.

We use this insight to abstract the collecting semantics Λ by *partitioning*. More specifically, let $\bullet \stackrel{\text{def}}{=} \left\{\Sigma^+_O \mid O \in \mathbb{O}\right\}$ be a trace partition with respect to outcome. We have the following Galois connection

$$\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma^+\right)\right), \subseteq\rangle \underset{\alpha_\bullet}{\overset{\gamma_\bullet}{\leftrightarrows}} \langle \mathcal{P}\left(\mathcal{P}\left(\Sigma^+\right)\right), \dot{\subseteq}\rangle, \tag{6}$$

where $\alpha_\bullet(S) \stackrel{\text{def}}{=} \{T_O \mid T \in S \wedge O \in \mathbb{O}\}$. The order $\dot{\subseteq}$ is the pointwise ordering between sets of traces with the same outcome, i.e., $A \mathrel{\dot{\subseteq}} B \stackrel{\text{def}}{=} \bigwedge_{O \in \mathbb{O}} \dot{A}_O \subseteq \dot{B}_O$, where $\dot{S}_Z$ denotes the only non-empty set of traces in S_Z. We can now define the *outcome semantics* $\Lambda_\bullet \in \mathcal{P}(\mathcal{P}(\Sigma^+))$ by abstraction of Λ:

$$\Lambda_\bullet \stackrel{\text{def}}{=} \alpha_\bullet(\Lambda) = \{\Upsilon_O \mid O \in \mathbb{O}\} \tag{7}$$

In the rest of the paper, we write $(\!|M|\!)_\bullet$ to denote the outcome semantics of a particular neural-network model M.

6.2 Dependency Semantics

We observe that, to reason about dependency fairness, we do not need to consider all intermediate computations between the initial and final states of a trace. Thus, we can further abstract the outcome semantics into a set of dependencies between initial states and outcomes of traces.

To this end, we define the following Galois connection[3]

$$\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma^+\right)\right), \dot{\subseteq}\rangle \underset{\alpha_{\rightsquigarrow}}{\overset{\gamma_{\rightsquigarrow}}{\leftrightarrows}} \langle \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma\right)\right), \dot{\subseteq}\rangle, \tag{8}$$

where $\alpha_{\rightsquigarrow}(S) \stackrel{\text{def}}{=} \{\{\langle \sigma_0, \sigma_\omega\rangle \mid \sigma \in T\} \mid T \in S\}$ [Urban and Müller 2018] abstracts away all intermediate states of any trace. We finally derive the *dependency semantics* $\Lambda_{\rightsquigarrow} \in \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma))$:

$$\Lambda_{\rightsquigarrow} \stackrel{\text{def}}{=} \alpha_{\rightsquigarrow}(\Lambda_\bullet) = \{\{\langle \sigma_0, \sigma_\omega\rangle \mid \sigma \in \Upsilon_O\} \mid O \in \mathbb{O}\} \tag{9}$$

In the following, let $(\!|M|\!)_{\rightsquigarrow}$ denote the dependency semantics of a particular network model M.

Let $R^Y \stackrel{\text{def}}{=} \{\langle s, \_\rangle \in R \mid s \in Y\}$ restrict a set of pairs of states to pairs whose first element is in Y and, similarly, let $S^Y \stackrel{\text{def}}{=} \left\{R^Y \mid R \in S\right\}$ restrict a set of sets of pairs of states to first elements in Y. The next result shows that $\Lambda_{\rightsquigarrow}$ is sound and complete for proving dependency fairness:

Theorem 6.2. $M \models \mathcal{F}_K[Y] \Leftrightarrow (\!|M|\!)^Y_{\rightsquigarrow} \mathrel{\dot{\subseteq}} \alpha_{\rightsquigarrow}(\alpha_\bullet(\mathcal{F}_K[Y]))$

[3]Note that here and in the following, for convenience, we abuse notation and reuse the order symbol $\dot{\subseteq}$ defined over sets of sets of traces, instead of its abstraction, defined over sets of sets of pairs of states.

Proof. Let $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y]$. From Theorem 5.2, we have that $(\!|\mathrm{M}|\!)^{Y} \subseteq \mathcal{F}_{\mathrm{K}}[Y]$. Thus, from the Galois connections in Equation 6 and 8, we have $\alpha_{\rightsquigarrow}(\alpha_{\bullet}((\!|\mathrm{M}|\!)^{Y})) \sqsubseteq \alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y]))$. From the definition of $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow}$ (cf. Equation 9), we can then conclude that $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \sqsubseteq \alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y]))$. □

Corollary 6.3. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow (\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \subseteq \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$

Proof. The proof follows trivially from the definition of $\sqsubseteq$ (cf. Equation 6 and 8) and Lemma 6.1. □

Furthermore, we observe that partitioning with respect to outcome induces a partition of the space of values of the input nodes *used* for classification. For instance, partitioning T' in Example 5.1 induces a partition on the values of (the indeed used node) $\mathrm{x}_{0,2}$. Thus, we can equivalently verify whether $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \subseteq \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$ by checking if the dependency semantics $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow}$ induces a partition of $Y_{|\overline{\mathrm{K}}}$. Let $R_0 \stackrel{\text{def}}{=} \{s \mid \langle s, \_\rangle \in R\}$ (resp. $R_\omega \stackrel{\text{def}}{=} \{s \mid \langle \_, s\rangle \in R\}$) be the selection of the first (resp. last) element from each pair in a set of pairs of states. We formalize this observation below.

Lemma 6.4. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow \forall A, B \in (\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \colon (A_\omega \neq B_\omega \Rightarrow A_{0|\overline{\mathrm{K}}} \cap B_{0|\overline{\mathrm{K}}} = \emptyset)$

Proof. Let $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y]$. From Corollary 6.3, we have that $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \subseteq \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$. Thus, from the definition of $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow}$ (cf. Equation 9), we have $\forall \mathrm{O} \in \mathbb{O} \colon \alpha_{\rightsquigarrow}([\![\mathrm{M}]\!]^{Y}_{\mathrm{O}}) \in \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$. In particular, from the definition of $\alpha_{\rightsquigarrow}$ and $\mathcal{F}_{\mathrm{K}}[Y]$ (cf. Equation 5), we have that $\textsc{unused}_{\mathrm{K}}([\![\mathrm{M}]\!]^{Y}_{\mathrm{O}})$ for each $\mathrm{O} \in \mathbb{O}$. From the definition of $\textsc{unused}_{\mathrm{K}}$ (cf. Equation 4), for each pair of *non-empty* $[\![\mathrm{M}]\!]^{Y}_{\mathrm{O}_1}$ and $[\![\mathrm{M}]\!]^{Y}_{\mathrm{O}_2}$ for different $\mathrm{O}_1, \mathrm{O}_2 \in \mathbb{O}$ (the case in which one or both are empty is trivial), it must necessarily be the value of the non-sensitive input nodes in $\overline{\mathrm{K}}$ that causes the different outcome O_1 or O_2. We can thus conclude that $\forall A, B \in (\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \colon (A_\omega \neq B_\omega \Rightarrow A_{0|\overline{\mathrm{K}}} \cap B_{0|\overline{\mathrm{K}}} = \emptyset)$. □

7 NAÏVE DEPENDENCY-FAIRNESS ANALYSIS

In this section, we present a first static analysis for dependency fairness that computes a *sound* over-approximation $\Lambda^{\natural}_{\rightsquigarrow}$ of the dependency semantics $\Lambda_{\rightsquigarrow}$, i.e., $\Lambda_{\rightsquigarrow} \sqsubseteq \Lambda^{\natural}_{\rightsquigarrow}$. This analysis corresponds to the naïve approach we discussed in Section 2. While it is too naïve to be practical, it is still useful for building upon later in the paper.

For simplicity, we consider ReLU activation functions. (We discuss extensions to other activation functions in Section 9.) The naïve static analysis is described in Algorithm 1. It takes as input (cf. Line 14) a neural-network model M, a set of sensitive input nodes K of M, a (representation of a) set of initial states of interest Y, and an abstract domain A to be used for the analysis. The analysis proceeds backwards for each outcome (i.e., each target class $\mathrm{x}_{\mathrm{N},j}$) of M (cf. Line 17) in order to determine an over-approximation of the initial states that satisfy Y and lead to $\mathrm{x}_{\mathrm{N},j}$ (cf. Line 18).

More specifically, the transfer function $\textsc{outcome}_{\mathrm{A}}[\![\mathrm{x}]\!]$ (cf. Line 2) modifies a given abstract-domain element to assume the given outcome x, that is, to assume that $\max \mathrm{X}_{\mathrm{N}} = \mathrm{x}$. The transfer functions $\overleftarrow{\textsc{relu}}_{\mathrm{A}}[\![\mathrm{x}_{i,j}]\!]$ and $\overleftarrow{\textsc{assign}}_{\mathrm{A}}[\![\mathrm{x}_{i,j}]\!]$ (cf. Line 5) respectively consider a ReLU operation and replace $\mathrm{x}_{i,j}$ with the corresponding linear combination of nodes in the preceding layer (see Section 3).

Finally, the analysis checks whether the computed over-approximations satisfy dependency fairness with respect to K (cf. Line 19). In particular, it checks whether they induce a partition of $Y_{|\overline{\mathrm{K}}}$ as observed for Lemma 6.4 (cf. Lines 7-13). If so, we have proved that M satisfies dependency fairness. If not, the analysis returns a set B of abstract-domain elements over-approximating the input regions in which bias might occur.

Theorem 7.1. If $\textsc{analyze}(M, \mathrm{K}, Y, \mathrm{A})$ of Algorithm 1 returns $\textsc{true}, \emptyset$ then M satisfies $\mathcal{F}_{\mathrm{K}}[Y]$.

Algorithm 1 : A Naïve Backward Analysis

1: **function** BACKWARD(M, A, x)
2:   $a \leftarrow \textsc{outcome}_{A}[\![x]\!](\textsc{new}_{A})$
3:   **for** $i \leftarrow N-1$ **down to** 0 **do**
4:    **for** $j \leftarrow |L_i|$ **down to** 0 **do**
5:     $a \leftarrow \overleftarrow{\textsc{assign}}_{A}[\![x_{i,j}]\!](\overleftarrow{\textsc{relu}}_{A}[\![x_{i,j}]\!]a)$
6:   **return** a
7: **function** CHECK(O)
8:   $B \leftarrow \emptyset$ ▷ B: biased
9:   **for all** $o_1, a_1 \in O$ **do**
10:    **for all** $o_2 \neq o_1, a_2 \in O$ **do**
11:     **if** $a_1 \sqcap_{A_2} a_2 \neq \bot_{A_2}$ **then**
12:      $B \leftarrow B \cup \{a_1 \sqcap_{A_2} a_2\}$
13:   **return** B
14: **function** ANALYZE(M, K, Y, A)
15:   $O \leftarrow \dot{\emptyset}$
16:   **for** $j \leftarrow 0$ **up to** $|L_N|$ **do**   ▷ perfectly parallelizable
17:    $a \leftarrow \textsc{backward}(M, A, x_{N,j})$
18:    $O \leftarrow O \cup \left\{x_{N,j} \mapsto (\textsc{assume}_{A}[\![Y]\!]a)_{|\overline{K}}\right\}$
19:   $B \leftarrow \textsc{check}(O)$
20:   **return** $B = \emptyset, B$ ▷ fair: $B = \emptyset$, maybe biased: $B \neq \emptyset$

PROOF (SKETCH). ANALYZE(M, K, Y, A) in Algorithm 1 computes an *over-approximation a* of the regions of the input space that yield each target class $x_{N,j}$ (cf. Line 17). Thus, it actually computes an over-approximation $(\!|M|\!)^{Y^\natural}_{\rightsquigarrow}$ of the dependency semantics $(\!|M|\!)^{Y}_{\rightsquigarrow}$, i.e., $(\!|M|\!)^{Y}_{\rightsquigarrow} \mathrel{\dot\subseteq} (\!|M|\!)^{Y^\natural}_{\rightsquigarrow}$. Thus, if $(\!|M|\!)^{Y^\natural}_{\rightsquigarrow}$ satisfies $\mathcal{F}_K[Y]$, i.e., $\forall A, B \in (\!|M|\!)^{Y^\natural}_{\rightsquigarrow} : (A_\omega \neq B_\omega \Rightarrow A_{0|_{\overline{K}}} \cap B_{0|_{\overline{K}}} = \emptyset)$ (according to Lemma 6.4, cf. Line 19), then by transitivity we can conclude that also $(\!|M|\!)^{Y^\natural}_{\rightsquigarrow}$ necessarily satisfies $\mathcal{F}_K[Y]$. □

In the analysis implementation, there is a tradeoff between performance and precision, which is reflected in the choice of abstract domain A and its transfer functions. Unfortunately, existing numerical abstract domains that are less expressive than polyhedra [Cousot and Halbwachs 1978] would make for a rather fast but too imprecise analysis. This is because they are not able to precisely handle constraints like $\max X_N = x$, which are introduced by $\textsc{outcome}_{A}[\![x]\!]$ to partition with respect to outcome.

Furthermore, even polyhedra would not be precise enough in general. Indeed, each $\overleftarrow{\textsc{relu}}_{A}[\![x_{i,j}]\!]$ would over-approximate what effectively is a conditional branch. Let $|M| \stackrel{\text{def}}{=} |L_1| + \cdots + |L_{N-1}|$ denote the number of hidden nodes (i.e., the number of ReLUs) in a model M. On the other side of the spectrum, one could use a disjunctive completion [Cousot and Cousot 1979] of polyhedra, thus keeping a separate polyhedron for each branch of a ReLU. This would yield a precise (in fact, exact) but extremely slow analysis: even with parallelization (cf. Line 16), each of the $|L_N|$ processes would have to effectively explore $2^{|M|}$ paths!

In the rest of the paper, we improve on this naïve analysis and show how far we can go all the while remaining exact by using disjunctive polyhedra.

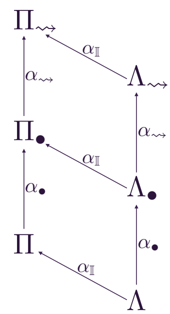


Fig. 2. Hierarchy of semantics.

8 PARALLEL SEMANTICS

We first have to take a step back and return to reasoning at the concrete-semantics level. At the end of Section 6, we observed that the dependency semantics of a neural-network model M satisfying $\mathcal{F}_{\mathrm{K}}[Y]$ effectively induces a partition of $Y_{|\overline{\mathrm{K}}}$. We call this input partition *fair*.

More formally, given a set Y of initial states of interest, we say that an input partition $\mathbb{I}$ of Y is fair if all value choices $\mathbb{V}$ for the sensitive input nodes K of M are possible in all elements of the partitions: $\forall \mathrm{I} \in \mathbb{I}, \mathrm{V} \in \mathbb{V} \colon \exists s \in \mathrm{I} \colon s(\mathrm{K}) = \mathrm{V}$. For instance, $\mathbb{I} = \{T, T'\}$, with T and T' in Example 5.1 is a fair input partition of $Y = \left\{ s \mid s(\mathrm{x}_{0,1}) = 0.5 \vee s(\mathrm{x}_{0,1}) = 0.75 \right\}$.

Given a fair input partition $\mathbb{I}$ of Y, the following result shows that we can verify whether a model M satisfies $\mathcal{F}_{\mathrm{K}}[Y]$ for each element I of $\mathbb{I}$, *independently*.

Lemma 8.1. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow \forall \mathrm{I} \in \mathbb{I} \colon \forall A, B \in (\!|\mathrm{M}|\!)^{\mathrm{I}}_{\rightsquigarrow} \colon (A_\omega \neq B_\omega \Rightarrow A_{0|_{\overline{\mathrm{K}}}} \cap B_{0|_{\overline{\mathrm{K}}}} = \emptyset)$

Proof. The proof follows trivially from Lemma 6.4 and the fact that $\mathbb{I}$ is a fair partition. □

We use this new insight to further abstract the dependency semantics $\Lambda_{\rightsquigarrow}$. We have the following Galois connection

$$\langle \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma\right)\right), \mathrel{\dot\subseteq} \rangle \underset{\alpha_{\mathbb{I}}}{\overset{\gamma_{\mathbb{I}}}{\leftrightarrows}} \langle \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma\right)\right), \mathrel{\dot\subseteq}_{\mathbb{I}} \rangle, \tag{10}$$

where $\alpha_{\mathbb{I}}(S) \overset{\text{def}}{=} \left\{ R^{\mathrm{I}} \mid R \in S \wedge \mathrm{I} \in \mathbb{I} \right\}$. Here, the order $\mathrel{\dot\subseteq}_{\mathbb{I}}$ is the pointwise ordering between sets of pairs of states restricted to first elements in the same $\mathrm{I} \in \mathbb{I}$, i.e., $A \mathrel{\dot\subseteq}_{\mathbb{I}} B \overset{\text{def}}{=} \bigwedge_{\mathrm{I} \in \mathbb{I}} \dot{A}^{\mathrm{I}} \subseteq \dot{B}^{\mathrm{I}}$, where $\dot{S}^{\mathrm{I}}$ denotes the only non-empty set of pairs in S^{I}. We can now derive the *parallel semantics* $\Pi^{\mathbb{I}}_{\rightsquigarrow} \in \mathcal{P}\left(\mathcal{P}\left(\Sigma \times \Sigma\right)\right)$:

$$\Pi^{\mathbb{I}}_{\rightsquigarrow} \overset{\text{def}}{=} \alpha_{\mathbb{I}}(\Lambda_{\rightsquigarrow}) = \left\{ \left\{ \langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Upsilon^{\mathrm{I}}_{\mathrm{O}} \right\} \mid \mathrm{I} \in \mathbb{I} \wedge \mathrm{O} \in \mathbb{O} \right\} \tag{11}$$

In fact, we derive a hierarchy of semantics, as depicted in Figure 2. We write $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow}$ to denote the parallel semantics of a particular neural-network model M. It remains to show soundness and completeness for $\Pi^{\mathbb{I}}_{\rightsquigarrow}$.

Theorem 8.2. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow \{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow} \mathrel{\dot\subseteq}_{\mathbb{I}} \alpha_{\mathbb{I}}(\alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y])))$

Proof. Let $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y]$. From Theorem 6.2, we have that $(\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow} \mathrel{\dot\subseteq} \alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y]))$. Thus, from the Galois connection in Equation 10, we have $\alpha_{\mathbb{I}}((\!|\mathrm{M}|\!)^{Y}_{\rightsquigarrow}) \mathrel{\dot\subseteq} \alpha_{\mathbb{I}}(\alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y])))$. From the definition of $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow}$ (cf. Equation 11), we can then conclude that $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow} \mathrel{\dot\subseteq}_{\mathbb{I}} \alpha_{\mathbb{I}}(\alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y])))$. □

Corollary 8.3. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow \{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow} \subseteq \alpha_{\mathbb{I}}(\alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y]))$

Proof. The proof follows trivially from the definition of $\dot{\subseteq}_{\mathbb{I}}$ (cf. Equation 6, 8, and 10) and Lemma 6.1 and 8.1. □

Finally, from Lemma 8.1, we have that we can equivalently verify whether $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow} \subseteq \alpha_{\mathbb{I}}(\alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y]))$ by checking if the parallel semantics $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow}$ induces a partition of each $\mathrm{I}_{|\overline{\mathrm{K}}}$.

Lemma 8.4. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow \forall \mathrm{I} \in \mathbb{I} \colon \forall A, B \in \{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow} \colon (A^{\mathrm{I}}_{\omega} \neq B^{\mathrm{I}}_{\omega} \Rightarrow A^{\mathrm{I}}_{0|\overline{\mathrm{K}}} \cap B^{\mathrm{I}}_{0|\overline{\mathrm{K}}} = \emptyset)$

Proof. The proof follows trivially from Lemma 8.1. □

9 PARALLEL DEPENDENCY-FAIRNESS ANALYSIS

In this section, we build on the parallel semantics to design our novel *perfectly parallel* static analysis for dependency fairness, which automatically finds a fair partition $\mathbb{I}$ and computes a sound over-approximation $\Pi^{\mathbb{I}^\natural}_{\rightsquigarrow}$ of $\Pi^{\mathbb{I}}{}_{\rightsquigarrow}$, i.e., $\Pi^{\mathbb{I}}{}_{\rightsquigarrow} \dot{\subseteq}_{\mathbb{I}} \Pi^{\mathbb{I}^\natural}_{\rightsquigarrow}$.

ReLU Activation Functions. We again only consider ReLU activation functions for now and postpone the discussion of other activation functions to the end of the section. The analysis is described in Algorithm 2. It combines a forward pre-analysis (Lines 15-24) with a backward analysis (Lines 28-38). The forward pre-analysis uses an abstract domain A_1 and builds partition $\mathbb{I}$, while the backward analysis uses an abstract domain A_2 and performs the actual dependency-fairness analysis of a neural-network model M with respect to its sensitive input nodes K and a (representation of a) set of initial states Y (cf. Line 13).

More specifically, the forward pre-analysis bounds the number of paths that the backward analysis has to explore. Indeed, not all of the $2^{|\mathrm{M}|}$ paths of a model M are necessarily viable starting from its input space.

In the rest of this section, we represent each path by an *activation pattern*, which determines the activation status of every ReLU operation in M. More precisely, an activation pattern is a sequence of flags. Each flag $\mathrm{p}_{i,j}$ represents the activation status of the ReLU operation used to compute the value of hidden node $\mathrm{x}_{i,j}$. If $\mathrm{p}_{i,j}$ is $\mathrm{x}_{i,j}$, the ReLU is always active, otherwise the ReLU is always inactive and $\mathrm{p}_{i,j}$ is $\overline{\mathrm{x}_{i,j}}$.

An *abstract activation pattern* gives the activation status of only a subset of the ReLUs of M, and thus, represents a set of activation patterns. ReLUs whose corresponding flag does not appear in an abstract activation pattern have an unknown (i.e., not fixed) activation status. Typically, *only a relatively small number of abstract activation patterns is sufficient for covering the entire input space of a neural-network model.* The design of our analysis builds on this key observation.

We set an analysis *budget* by providing an upper bound U (cf. Line 13) on the number of tolerated ReLUs with an unknown activation status for each element I of $\mathbb{I}$, i.e., on the number of paths that are to be explored by the backward analysis in each I. The forward pre-analysis starts with the trivial partition $\mathbb{I} = \{Y\}$ (cf. Line 15). It proceeds forward for each element I in $\mathbb{I}$ (cf. Lines 17-18). The transfer function $\overrightarrow{\text{RELU}}^{\mathrm{p}}_{\mathrm{A}}[\![\mathrm{x}_{i,j}]\!]$ considers a ReLU operation and additionally builds an abstract activation pattern p for I (cf. Line 5) starting from the empty pattern ϵ (cf. Line 2).

If I leads to a unique outcome (cf. Line 19), then dependency fairness is already proved for I, and there is no need for a backward analysis; I is added to the set of *completed* partitions (cf. Line 20). Instead, if abstract activation pattern p fixes the activation status of enough ReLUs (cf. Line 21), we say that the backward analysis for I is *feasible*. In this case, the pair of p and I is inserted into a map F from abstract activation patterns to feasible partitions (cf. Line 22). The insertion takes care of merging abstract activation patterns that are subsumed by other (more) abstract patterns. In other words, it groups partitions whose abstract activation patterns fix more ReLUs with partitions whose patterns fix fewer ReLUs, and therefore, represent a superset of (concrete) patterns.

Algorithm 2 : Our Analysis Based on Activation Patterns

```
1:  function FORWARD(M, A, I)
2:      a, p ← ASSUME_A⟦I⟧(NEW_A), ϵ
3:      for i ← 1 up to N do
4:          for j ← 0 up to |L_i| do
5:              a, p ← →RELU^p_A⟦x_{i,j}⟧(→ASSIGN_A⟦x_{i,j}⟧a)
6:      return a, p
7:  function BACKWARD(M, A, O, p)
8:      a ← OUTCOME_A⟦O⟧(NEW_A)
9:      for i ← N − 1 down to 0 do
10:         for j ← |L_i| down to 0 do
11:             a ← ←ASSIGN_A⟦x_{i,j}⟧(←RELU^p_A⟦x_{i,j}⟧a)
12:     return a
13: function ANALYZE(M, K, Y, A_1, A_2, L, U)
14:     F, E, C ← ∅̇, ∅̇, ∅                                ▷ F: feasible, E: excluded, C: completed
15:     𝕀 ← {Y}
16:     while 𝕀 ≠ ∅ do                                   ▷ perfectly parallelizable
17:         I ← 𝕀.GET()
18:         a, p ← FORWARD(M, A_1, I)
19:         if UNIQUELY-CLASSIFIED(a) then               ▷ I is already fair
20:             C ← C ∪ {I}
21:         else if |M| − |p| ≤ U then                    ▷ I is feasible
22:             F ← F ⊎ {p ↦ I}
23:         else if |I| ≤ L then                          ▷ I is excluded
24:             E ← E ⊎ {p ↦ I}
25:         else                                          ▷ I must be partitioned further
26:             𝕀 ← 𝕀 ∪ PARTITION_K̄(I)
27:     B ← ∅                                             ▷ B: biased
28:     for all p, 𝕀 ∈ F do                               ▷ perfectly parallelizable
29:         O ← ∅̇
30:         for j ← 0 up to |L_N| do
31:             a ← BACKWARD(M, A_2, x_{N,j}, p)
32:             O ← O ∪ {x_{N,j} ↦ a}
33:         for all I ∈ 𝕀 do
34:             O' ← ∅̇
35:             for all o, a ∈ O do
36:                 O' ← O' ∪ {o ↦ (ASSUME_{A_2}⟦I⟧a)|_K̄}
37:             B ← B ∪ CHECK(O')
38:             C ← C ∪ {I}
39:     return C, B = ∅, B, E                             ▷ fair: B = ∅, maybe biased: B ≠ ∅
```

Otherwise, I needs to be partitioned further, with respect to $\overline{K}$ (cf. Line 25). Partitioning may continue until the size of I is smaller than the given lower bound L (cf. Lines 13 and 23). At this point, I is set aside and excluded from the analysis until more resources (a larger upper bound U or a smaller lower bound L) become available (cf. Line 24).

Note that the forward pre-analysis lends itself to choosing a relatively cheap abstract domain A_1 since it does not need to precisely handle polyhedral constraints (like $\max X_N = x$, needed to partition with respect to outcome, cf. Section 7).

The analysis then proceeds backwards, independently for each abstract activation path p and associated group of partitions $\mathbb{I}$ (cf. Lines 28 and 31). The transfer function $\overleftarrow{\text{RELU}}^{p}_{A}[\![x_{i,j}]\!]$ uses p to choose which path(s) to explore at each ReLU operation, i.e., only the active (resp. inactive) path if $x_{i,j}$ (resp. $\overline{x_{i,j}}$) appears in p, or both if the activation status of the ReLU corresponding to hidden node $x_{i,j}$ is unknown. The (as we have seen, necessarily) expensive backward analysis only needs to run for each abstract activation pattern in the feasible map F. This is also why it is advantageous to merge subsumed abstract activation paths as described above.

Finally, the analysis checks dependency fairness of each element I associated to p (cf. Line 37). The analysis returns the set of input-space regions C that have been completed and a set B of abstract-domain elements over-approximating the regions in which bias might occur (cf. Line 39). If B is empty, then the given model M satisfies dependency fairness with respect to K and Y over C.

Theorem 9.1. If function ANALYZE$(M, K, Y, A_1, A_2, L, U)$ in Algorithm 2 returns C, TRUE, $\emptyset$, then M satisfies $\mathcal{F}_K[Y]$ over the input-space fraction C.

PROOF (SKETCH). ANALYZE$(M, K, Y, A_1, A_2, L, U)$ in Algorithm 2 first computes the abstract activation patterns that cover a fraction C of the input space in which the analysis is feasible (Lines 15-24). Then, it computes an *over-approximation a* of the regions of C that yield each target class $x_{N,j}$ (cf. Line 31). Thus, it actually computes an over-approximation $\{\!|M|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$ of the parallel semantics $\{\!|M|\!\}^{\mathbb{I}}_{\rightsquigarrow}$, i.e., $\{\!|M|\!\}^{\mathbb{I}}_{\rightsquigarrow} \Subset \{\!|M|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$. Thus, if $\{\!|M|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$ satisfies $\mathcal{F}_K[Y]$, i.e., $\forall I \in \mathbb{I}\colon \forall A, B \in \{\!|M|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}\colon (A^I_\omega \neq B^I_\omega \Rightarrow A^I_{0|\overline{K}} \cap B^I_{0|\overline{K}} = \emptyset)$ (according to Lemma 8.4, cf. Lines 33-37), then by transitivity we can conclude that also $\{\!|M|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$ necessarily satisfies $\mathcal{F}_K[Y]$. □

Remark. Recall that we assumed neural-network nodes to have real values (cf. Section 4). Thus, Theorem 9.1 is true for all choices of classical numerical abstract domains [Cousot and Cousot 1976; Cousot and Halbwachs 1978; Ghorbal et al. 2009; Miné 2006b, etc.] for A_1 and A_2. If we were to consider floating-point values instead, the only sound choices would be floating-point abstract domains [Chen et al. 2008; Miné 2004; Singh et al. 2019].

Other Activation Functions. Let us discuss how activation functions other than ReLUs would be handled. The only difference in Algorithm 2 would be the transfer functions $\overrightarrow{\text{RELU}}^{p}_{A}[\![x_{i,j}]\!]$ (cf. Line 5) and $\overleftarrow{\text{RELU}}^{p}_{A}[\![x_{i,j}]\!]$ (cf. Line 11), which would have to be replaced with the transfer functions corresponding to the considered activation function.

Piecewise-linear activation functions, like LEAKY ReLU$(x) = \max(x, k \cdot x)$ or HARD TanH$(x) = \max(-1, \min(x, 1))$, can be treated analogously to ReLUs. The case of LEAKY ReLUs is trivial. For HARD TanHs, the patterns p used in Algorithm 2 will consist of flags $p_{i,j}$ with three possible values, depending on whether the corresponding hidden node $x_{i,j}$ has value less than or equal to -1, greater than or equal to 1, or between -1 and 1. For these activation functions, our approach remains sound and, in practice, exact when using disjunctive polyhedra for the backward analysis.

Other activation functions, e.g., SIGMOID$(x) = \frac{1}{1+e^{-x}}$, can be soundly over-approximated [Singh et al. 2019] and similarly treated in a piecewise manner. In this case, however, we necessarily lose the exactness of the analysis, even when using disjunctive polyhedra.

10 IMPLEMENTATION

We implemented our dependency-fairness analysis described in the previous section in a tool called LIBRA. The implementation is written in PYTHON and is open source[4].

Tool Inputs. LIBRA takes as input a neural-network model M expressed as a PYTHON program (cf. Section 3), a specification of the input layer L_0 of M, an abstract domain for the forward pre-analysis, and budget constraints L and U. The specification for L_0 determines which input nodes correspond to continuous and (one-hot encoded) categorical data and, among them, which should be considered bias sensitive. We assume that continuous data is in the range $[0, 1]$. A set Y of initial states of interest is specified using an assumption at the beginning of the program representation of M.

Abstract Domains. For the forward pre-analysis, choices of the abstract domain are either boxes [Cousot and Cousot 1976] (i.e., BOXES in the following), or a combination of boxes and symbolic constant propagation [Li et al. 2019; Miné 2006a] (i.e., SYMBOLIC in the following), or the DEEPPOLY domain [Singh et al. 2019], which is designed for proving local robustness of neural networks. As previously mentioned, we use disjunctive polyhedra for the backward analysis. All abstract domains are built on top of the APRON abstract-domain library [Jeannet and Miné 2009].

Parallelization. Both the forward and backward analyses are parallelized to run on multiple CPU cores. The pre-analysis uses a queue from which each process draws a fraction I of Y (cf. Line 17). Fractions that need to be partitioned further are split in half along one of the non-sensitive dimensions (in a round-robin fashion), and the resulting (sub)fractions are put back into the queue (cf. Line 26). Feasible Is (with their corresponding abstract activation pattern p) are put into another queue (cf. Line 22) for the backward analysis.

Tool Outputs. The analysis returns the fractions of Y that were analyzed and any (sub)regions of these where bias was found. It also reports the percentage of the input space that was analyzed and (an estimate of) the percentage that was found biased according to a given probability distribution of the input space (uniform by default). To obtain the latter, we simply use the size of a box wrapped around each biased region. More precise but also costlier solutions exist [Barvinok 1994].

In general, how much of Y can be analyzed depends on the analysis configuration (i.e., chosen abstract domain for the forward pre-analysis and budget constraints L and U). For the fractions of Y that remained excluded from the analysis one can always re-run the analysis with a more powerful configuration, i.e., by restricting Y to the excluded fraction and choosing a more precise abstract domain or a higher L or a lower U. We plan on automating this process as part of our future work.

11 EXPERIMENTAL EVALUATION

In this section, we evaluate our approach by focusing on the following research questions:

RQ1: Can our analysis detect seeded (i.e., injected) bias?
RQ2: Is our analysis able to answer specific bias queries?
RQ3: How does the model structure affect the scalability of the analysis?
RQ4: How does the analyzed input-space size affect the scalability of the analysis?
RQ5: How does the analysis budget affect the scalability-vs-precision tradeoff?
RQ6: Can our analysis effectively leverage multiple CPUs?

[4] https://github.com/caterinaurban/Libra

11.1 Data

For our evaluation, we used public datasets from the UCI Machine Learning Repository and ProPublica (see below for more details) to train several neural-network models. We primarily focused on datasets discussed in the literature [Mehrabi et al. 2019] or used by related techniques (e.g., [Albarghouthi et al. 2017a,b; Albarghouthi and Vinitsky 2019; Bastani et al. 2019; Datta et al. 2017; Galhotra et al. 2017; Tramèr et al. 2017; Udeshi et al. 2018]) and recent work on fairness targeting feed-forward neural networks (e.g., [Manisha and Gujar 2020; Yurochkin et al. 2020]).

We pre-processed these datasets both to make them fair with respect to a certain sensitive input feature as well as to seed bias. This way, we eliminate as many sources of uncontrolled bias as possible, i.e., bias originally present in the dataset as well as bias potentially introduced by the training process due to under-representation in the dataset (as discussed in the Introduction). We describe how we seeded bias in each particular dataset later on in this section.

Our methodology for making the data fair was common across datasets. Specifically, given an original dataset and a sensitive feature (say, race), we selected the largest population with a particular value for this feature (say, Caucasian) from the dataset (and discarded all others). We removed any duplicate or inconsistent entries from this population. We then duplicated the population for every other value of the sensitive feature (say, Asian and Hispanic). For example, assuming the largest population was 500 Caucasians, we created 500 Asians and 500 Hispanics, and any two of these populations differ only in the value of race. Consequently, the new dataset is fair because there do not exist two inputs k and k' that differ only in the value of the sensitive feature for which the classification outcomes are different. A similar methodology is used by Yurochkin et al. in recent related work [Yurochkin et al. 2020].

We define the *unfairness score* of a dataset as the percentage of inputs k in the dataset for which there exists another input k' that differs from k only in the value of the sensitive feature and the classification outcome. Our fair datasets have an unfairness score of 0%.

All datasets used in our experiments are open source as part of LIBRA.

11.2 Setup

Since neural-network training is non-deterministic, we typically train eight neural networks on each dataset, unless stated otherwise. The model sizes range from 2 hidden layers with 5 nodes each to 32 hidden layers with 40 nodes each. All models were trained with Keras, using the RMSprop optimizer with the default learning rate, and categorical crossentropy as the loss function. Each model was trained for 50 iterations. With these settings, we generally obtain lower accuracy values than those reported in the literature (e.g., [Manisha and Gujar 2020]). We believe that this is largely due to the fact that we modified the original datasets to make them fair or to seed bias and, more importantly, the fact that we did not invest into hyperparameter optimizations. However, we remark and demonstrate below that training neural networks with higher accuracy is not needed to show that our analysis works (i.e., is indeed able to certify fairness or detect bias) and to evaluate its performance. All models used in our experiments are open source as part of LIBRA. For each model, we assume a uniform distribution of the input space.

We performed all experiments on a 12-core Intel ® Xeon ® X5650 CPU @ 2.67GHz machine with 48GB of memory, running Debian GNU/Linux 9.6 (stretch).

11.3 Results

In the following, we present our experimental results for each of the above research questions.

RQ1: Detecting Seeded Bias. This research question focuses on detecting seeded bias by comparing the analysis results for models trained with fair versus biased data.

Table 1. Analysis of Models Trained on Fair and {Age, Credit > 1000}-Biased Data (German Credit Data)

CREDIT	BOXES				SYMBOLIC				DEEPPOLY				
	FAIR DATA		BIASED DATA		FAIR DATA		BIASED DATA		FAIR DATA		BIASED DATA		
	BIAS	TIME	BIAS	TIME	BIAS	TIME	BIAS	TIME	BIAS	TIME	BIAS	TIME	
≤ 1000	0.09%	47s	0.09%	2m 17s	0.09%	13s	0.09%	1m 10s	0.09%	**10s**	0.09%	**39s**	MIN
	0.19%	5m 46s	0.45%	13m 2s	0.19%	**1m 5s**	0.45%	2m 41s	0.19%	1m 12s	0.45%	**1m 46s**	MEDIAN
	0.33%	30m 59s	0.95%	1h 56m 57s	0.33%	**4m 8s**	0.95%	**13m 16s**	0.33%	5m 45s	0.95%	18m 18s	MAX
> 1000	2.21%	1m 42s	4.52%	21m 11s	2.21%	**38s**	4.52%	**3m 7s**	2.21%	39s	4.52%	4m 44s	MIN
	6.72%	31m 42s	23.41%	1h 36m 51s	6.72%	8m 59s	23.41%	41m 44s	6.63%	**4m 58s**	23.41%	**15m 39s**	MEDIAN
	14.96%	7h 7m 12s	33.19%	16h 50m 48s	14.96%	4h 16m 52s	33.19%	8h 5m 14s	14.96%	**1h 9m 45s**	31.17%	**6h 51m 50s**	MAX

For this experiment, we used the German Credit dataset[5]. This dataset classifies creditworthiness into two categories, "good" and "bad". An input feature is age, which we consider sensitive to bias. (Recall that this could also be an input feature that the user considers indirectly sensitive to bias.) We seeded bias in the fair dataset by randomly assigning a bad credit score to people of age 60 and above who request a credit amount of more than EUR 1 000 until we reached a 20% unfairness score of the dataset. The median classification accuracy of the models (17 inputs and 4 hidden layers with 5 nodes each) trained on fair and biased data was 71% and 65%, respectively.

To analyze these models, we set L = 0 to be sure to complete the analysis on 100% of the input space. The drawback with this is that the pre-analysis might end up splitting input partitions endlessly. To counteract, for each model, we chose the smallest upper bound U that did not cause this issue (i.e., we used values for U between 1 and 16). Table 1 shows the analysis results for the different choices of domain used for the forward pre-analysis. In particular, it shows whether the models are biased with respect to age for credit requests of 1 000 or less as well as for credit requests of over 1 000. Columns BIAS and TIME show the detected bias (in percentage of the entire input space) and the analysis running time. We show minimum, median, and maximum bias percentage and running time for each credit request group. For each line in Table 1, we highlighted the choice of the abstract domain that entailed the shortest analysis time.

In this case, we expect the analysis to detect the highest bias percentage for credit requests of over 1 000 and for the models trained on biased data. This is indeed what we obtain: the analysis finds $3.5x$ median bias for high credit amounts compared to models trained on fair data. This demonstrates that *our approach is able to effectively detect seeded bias.*

For models trained on fair data, we observe a maybe unexpected difference in the bias found for small credit amounts compared to large credit amounts. This is in part due to the fact that bias is given in percentage of the entire input space and not scaled with respect to the *analyzed* input space (small credit amounts correspond to a mere 4% of the input space). When scaling the bias percentage with respect to the analyzed input space, the difference is less marked: the median bias is 0.19% / 4% = 4.75% for small credit amounts and 6.72% / 96% = 7% (or 6.63% / 96% = 6.9% for the DEEPPOLY domain) for large credit amounts. The remaining difference indicates that the models contain bias that does not necessarily depend on the credit amount. The bias is introduced by the training process itself (as explained in the Introduction) and is not due to imprecision of our analysis. Recall that our approach is exact, and imprecision is only introduced when estimating the bias percentage (cf. Section 10). Similar considerations justify the difference in the bias found for small credit amounts for models trained on fair data compared to those trained on biased data.

Finally, for comparison, the analysis of models trained on the original dataset (with median accuracy of 74%) found 0.28% bias for small credit amounts and 17.7% bias for large credit amounts.

RQ2: Answering Bias Queries. To further evaluate the precision of our approach, we created queries concerning bias within specific groups of people, each corresponding to a subset of the

[5] https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

Table 2. Queries on Models Trained on Fair and Race-Biased Data (ProPublica's COMPAS Data)

QUERY	BOXES				SYMBOLIC				DEEPPOLY				
	FAIR DATA		BIASED DATA		FAIR DATA		BIASED DATA		FAIR DATA		BIASED DATA		
	BIAS	TIME	BIAS	TIME	BIAS	TIME	BIAS	TIME	BIAS	TIME	BIAS	TIME	
AGE < 25 RACE BIAS?	0.22%	24m 32s	0.12%	14m 53s	0.22%	11m 34s	0.12%	**7m 14s**	0.22%	**5m 18s**	0.12%	8m 46s	MIN
	0.31%	1h 54m 48s	0.99%	57m 33s	0.32%	**36m 0s**	0.99%	20m 43s	0.32%	47m 16s	0.99%	**16m 38s**	MEDIAN
	2.46%	2h 44m 11s	8.33%	5h 29m 19s	2.46%	2h 17m 3s	8.50%	3h 34m 50s	2.12%	**1h 11m 43s**	6.48%	**2h 5m 5s**	MAX
MALE AGE BIAS?	2.60%	24m 14s	4.51%	34m 23s	2.64%	25m 13s	5.20%	29m 19s	2.70%	**19m 47s**	5.22%	**20m 51s**	MIN
	6.08%	1h 49m 42s	6.95%	2h 3m 39s	6.77%	**1h 1m 51s**	7.02%	1h 2m 26s	6.77%	1h 13m 31s	7.00%	**47m 28s**	MEDIAN
	8.00%	5h 56m 6s	12.56%	8h 26m 55s	8.40%	**2h 2m 22s**	12.71%	4h 55m 35s	8.84%	2h 20m 23s	12.88%	**3h 25m 21s**	MAX
CAUCASIAN PRIORS BIAS?	2.18%	2h 54m 18s	2.92%	46m 53s	2.18%	1h 20m 41s	2.92%	30m 23s	2.18%	**18m 26s**	2.92%	**15m 29s**	MIN
	2.95%	6h 56m 44s	4.21%	3h 50m 38s	2.95%	4h 12m 28s	4.21%	3h 32m 52s	2.95%	**2h 36m 1s**	4.21%	**1h 34m 7s**	MEDIAN
	5.36%	**45h 2m 12s**	6.98%	70h 50m 10s	5.36%	60h 53m 6s	6.98%	49h 51m 42s	5.36%	52h 10m 2s	6.95%	**17h 48m 22s**	MAX

entire input space. We used the COMPAS dataset[6] from ProPublica for this experiment. The data assigns a three-valued recidivism-risk score (high, medium, and low) indicating how likely criminals are to re-offend. The data includes both personal attributes (e.g., age and race) as well as criminal history (e.g., number of priors and violent crimes). As for RQ1, we trained models both on fair and biased data. Here, we considered race as the sensitive feature. We seeded bias in the fair data by randomly assigning high recidivism risk to African Americans until we reached a 20% unfairness score of the dataset. The median classification accuracy of the 3-class models (19 inputs and 4 hidden layers with 5 nodes each) trained on fair and biased data was 55% and 56%, respectively.

To analyze these models, we used a lower bound L of 0, and an upper bound U between 7 and 19. Table 2 shows the results of our analysis (i.e., columns shown as in Table 1) for three queries, each representing a different choice for the set K of sensitive features:

Q_A: Is there bias with respect to *race* for people younger than 25?
Q_B: Is there bias with respect to *age* for males?
Q_C: Is there bias with respect to the number of priors for Caucasians?

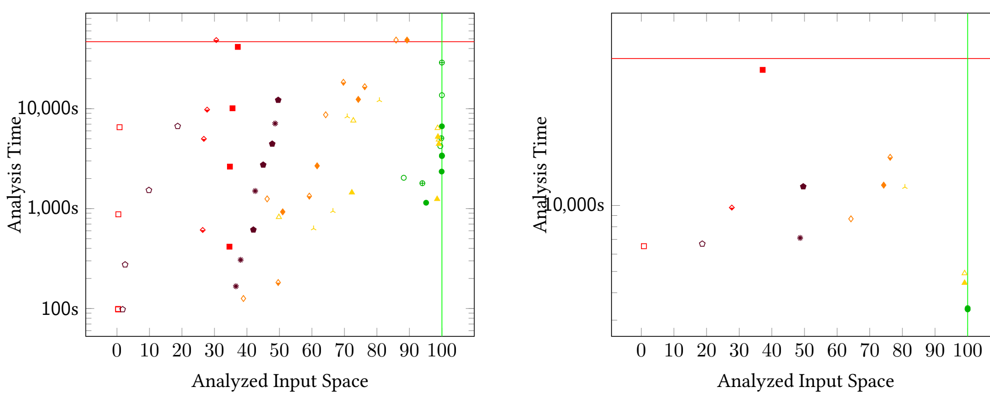
In this case, we expect the result of the analysis to change coherently according to the choice of K. Specifically, Q_A is expected to show a significant difference in the bias detected for models trained on fair data compared to models trained on biased data, while Q_B and Q_C should not show marked differences between fair and biased models. However, bias with respect to number of priors seems natural in the context of the dataset (i.e., recidivism risk increases with higher number of priors) and, therefore, we expect the bias percentage obtained for Q_C to be higher than that found for Q_B across both sets of fair and biased neural-network models.

The results in Table 2 meet our expectations. For Q_A, the analysis detects about three times as much median bias for the models trained on biased data compared to those trained on fair data. In contrast, for Q_B, the analysis finds a comparable amount of age bias across both sets of models. This becomes more evident when scaling the median bias with respect to the queried input space (males correspond to 50% of the input space): the smallest median bias for the models trained on fair data is 12.16% (for the BOXES domain) and the largest median bias for the models trained on biased data is 14.04% (for the SYMBOLIC domain). Finally, for Q_C, the analysis detects significant bias across both sets of models with respect to the number of priors. When considering the queried input space (Caucasians represent $1/6$ of the entire input space), this translates to 17.7% median bias for the models trained on fair data and 25.26% for the models trained on biased data. Overall, these results *demonstrate the effectiveness of our analysis in answering specific bias queries.* For comparison, the analysis of neural-network models trained on the original dataset (with median accuracy of 63%) found 1.21% median bias for Q_A, 5.34% median bias for Q_B, and 5.86% median bias for Q_C.

[6] https://www.propublica.org/datastore/dataset/compas-recidivism-risk-score-data-and-analysis

Table 3. Comparison of Different Model Structures (Adult Census Data)

|M|	U	BOXES INPUT	|C|	|F|		TIME	SYMBOLIC INPUT	|C|	|F|		TIME	DEEPPOLY INPUT	|C|	|F|		TIME
10 ○ ● ⊕	4	88.26%	1482	77	1136	33m 55s	95.14%	1132	65	686	19m 5s	93.99%	1894	77	992	29m 55s
	6	99.51%	769	51	723	1h 10m 25s	99.93%	578	47	447	39m 8s	99.83%	1620	54	1042	1h 24m 24s
	8	100.00%	152	19	143	3h 47m 23s	100.00%	174	18	146	1h 51m 2s	100.00%	1170	26	824	8h 2m 27s
	10	100.00%	1	1	1	**55m 58s**	100.00%	1	1	1	56m 8s	100.00%	1	1	1	56m 43s
12 △ ▲ ⅄	4	49.83%	719	9	329	13m 43s	72.29%	1177	11	559	24m 9s	60.52%	1498	14	423	10m 32s
	6	72.74%	1197	15	929	2h 6m 49s	98.54%	333	7	195	20m 46s	66.46%	1653	17	594	15m 44s
	8	98.68%	342	9	284	1h 46m 43s	98.78%	323	9	190	1h 27m 18s	70.87%	1764	18	724	2h 19m 11s
	10	99.06%	313	7	260	1h 21m 47s	99.06%	307	5	182	**1h 13m 55s**	80.76%	1639	18	1007	3h 22m 11s
20 ◇ ♦ ⬙	4	38.92%	1044	18	39	2m 6s	51.01%	933	31	92	15m 28s	49.62%	1081	34	79	3m 2s
	6	46.22%	1123	62	255	20m 51s	61.60%	916	67	405	44m 40s	59.20%	1335	90	356	22m 13s
	8	64.24%	1111	96	792	2h 24m 51s	74.27%	1125	78	780	3h 26m 20s	69.69%	1574	127	652	5h 6m 7s
	10	85.90%	1390	71	1339	>13h	89.27%	1435	60	1157	>13h	76.25%	1711	148	839	**4h 36m 23s**
40 □ ■ ⬙	4	0.35%	10	0	0	1m 39s	34.62%	768	1	1	6m 56s	26.39%	648	2	3	10m 11s
	6	0.35%	10	0	0	1m 38s	34.76%	817	4	5	43m 53s	26.74%	592	8	10	1h 23m 11s
	8	0.42%	12	1	2	14m 37s	35.56%	840	21	28	2h 48m 15s	27.74%	686	32	42	2h 43m 2s
	10	0.80%	23	10	13	1h 48m 43s	37.19%	880	50	75	**11h 32m 21s**	30.56%	699	83	121	>13h
45 ⬠ ⬟ ✳	4	1.74%	50	0	0	1m 38s	41.98%	891	14	49	10m 14s	36.60%	805	6	8	2m 47s
	6	2.50%	72	3	22	4m 35s	45.00%	822	32	143	45m 42s	38.06%	847	25	50	5m 7s
	8	9.83%	282	25	234	25m 30s	47.78%	651	46	229	1h 14m 5s	42.53%	975	74	180	25m 1s
	10	18.68%	522	33	488	1h 51m 24s	49.62%	714	51	294	**3h 23m 20s**	48.68%	1087	110	373	1h 58m 34s



(a) (b) Zoom on Best U-Configurations

Fig. 3. Comparison of Different Model Structures (Adult Census Data)

For each line in Table 2, we highlighted the choice of abstract domain that entailed the shortest analysis time. We observe that DEEPPOLY seems generally the better choice. The difference in performance becomes more striking as the analyzed input space becomes smaller, i.e., for Q_C. This is because DEEPPOLY is specifically designed for proving *local* robustness of neural networks. Thus, our input partitioning, in addition to allowing for parallelism, is also enabling analyses designed for local properties to prove global properties, like dependency fairness.

RQ3: Effect of Model Structure on Scalability. To evaluate the effect of the model structure on the scalability of our analysis, we trained models on the Adult Census dataset[7] by varying the number of layers and nodes per layer. The dataset assigns a yearly income (> or ≤ USD 50K) based on personal attributes such as gender, race, and occupation. All models (with 23 inputs) were trained on a fair dataset with respect to gender and had minimum classification accuracy of 78%.

Table 3 shows the results. The first column (|M|) shows the total number of hidden nodes and introduces the marker symbols used in the scatter plot of Figure 3 (to identify the domain used for

[7] https://archive.ics.uci.edu/ml/datasets/adult

the forward pre-analysis: left, center, and right symbols respectively refer to the BOXES, SYMBOLIC, and DEEPPOLY domains). The models have the following number of hidden layers and nodes per layer (from top to bottom): 2 and 5; 4 and 3; 4 and 5; 4 and 10; 9 and 5.

Column U shows the chosen upper bound for the analysis. For each model, we tried four different choices of U. Column INPUT shows the input-space coverage, i.e., the percentage of the input space that was completed by the analysis. Column |C| shows the total number of analyzed (i.e., completed) input space partitions. Column |F| shows the total number of abstract activation patterns (left) and feasible input partitions (right) that the backward analysis had to explore. The difference between |C| and the number of partitions shown in |F| are the input partitions that the pre-analysis found to be already fair (i.e., uniquely classified). Finally, column TIME shows the analysis running time. We used a lower bound L of 0.5 and a time limit of 13h. For each model in Table 3, we highlighted the configuration (i.e., domain used for the pre-analysis and chosen U) that achieved the highest input-space coverage (the analysis running time being decisive in case of equality or timeout).

The scatter plot of Figure 3a visualizes the input coverage and analysis running time. We zoom in on the best U-configurations for each pre-analysis domain (i.e., the chosen U) in Figure 3b.

Overall, we observe that *coverage decreases for larger model structures*, and the more precise SYMBOLIC and DEEPPOLY domains result in a significant coverage boost, especially for larger structures. We also note that, as in this case we are analyzing the entire input space, DEEPPOLY generally performs worse than the SYMBOLIC domain. In particular, for larger structures, *the SYMBOLIC domain often yields a higher input coverage in a shorter analysis running time.* Interestingly, the input coverage is higher for the largest model with 45 hidden nodes (i.e., 9 hidden layers with 5 nodes each) than that with 40 hidden nodes (i.e., 4 hidden layers with 10 nodes each). In fact, in neural-network models with more layers but fewer nodes per layer, the activation status of a node fixes the activation status of more nodes in subsequent layers. As a consequence, fewer activation patterns are actually needed to cover a larger portion of the input space. For instance with the SYMBOLIC domain only 14 patterns are enough to cover 41.98% of the input space for the model with 45 hidden nodes with U = 4, while for the model with 40 hidden nodes already 50 patterns are needed to cover only 37.19% of the input space with U = 10. Finally, we observe that *increasing the upper bound* U *tends to increase coverage independently of the specific model structure.* However, interestingly, this does not always come at the expense of an increased running time. In fact, such a change often results in decreasing the number of partitions that the expensive backward analysis needs to analyze (cf. columns |F|) and, in turn, this reduces the overall running time.

RQ4: Effect of Analyzed Input Space on Scalability. As said above, the analysis of the models considered in Table 3 is conducted *on the entire input space*. In practice, as already mentioned, one might be interested in just a portion of the input space, e.g., depending on the probability distribution. More generally, **we argue that the size of the analyzed input space (rather than the size of the analyzed neural network) is the most important factor that affects the performance of the analysis**. To support this claim, we trained even larger models and analyzed them with respect to queries exercising different input space sizes. Table 4 shows the results. The first column again shows the total number of hidden nodes for each trained model. In particular, the models we analyzed have the following number of hidden layers and nodes per layer (from top to bottom): 4 and 5; 8 and 10; 16 and 20; 32 and 40. Column QUERY shows the query used for the analysis and the corresponding queried input space size. Specifically, the queries identify people with the following characteristics:

*A***:** TRUE queried input space: 100.0%

Table 4. Comparison of Different Input Space Sizes and Model Structures (Adult Census Data)

|M|	QUERY	BOXES INPUT	BOXES |C|	BOXES |F|		BOXES TIME	SYMBOLIC INPUT	SYMBOLIC |C|	SYMBOLIC |F|		SYMBOLIC TIME	DEEPPOLY INPUT	DEEPPOLY |C|	DEEPPOLY |F|		DEEPPOLY TIME
20	F 0.009%	100.000% 0.009%	9	2	3	3m 3s	100.000% 0.009%	5	1	2	3m 5s	100.000% 0.009%	3	1	1	**2m 33s**
	E 0.104%	99.996% 0.104%	83	9	39	3m 13s	100.000% 0.104%	26	3	9	3m 8s	100.000% 0.104%	22	3	9	**2m 38s**
	D 1.042%	99.978% 1.042%	457	13	176	5m	100.000% 1.042%	292	9	63	**4m 50s**	100.000% 1.042%	287	6	65	5m 14s
	C 8.333%	99.696% 8.308%	3173	20	1211	36m 12s	100.000% 8.333%	2668	13	417	**17m 40s**	100.000% 8.333%	2887	10	519	29m 52s
	B 50%	97.318% 48.659%	15415	61	5646	1h 39m 36s	99.991% 49.996%	12617	34	2112	**1h 1m 19s**	99.978% 49.989%	13973	24	2405	1h 14m 19s
	A 100%	94.032% 94.032%	18642	70	8700	2h 30m 46s	99.935% 99.935%	15445	40	3481	**1h 29m**	99.896% 99.896%	17784	39	4076	1h 47m 7s
80	F 0.009%	99.931% 0.009%	11	0	0	3m 5s	99.961% 0.009%	17	0	0	**3m 2s**	99.957% 0.009%	10	0	0	2m 36s
	E 0.104%	99.583% 0.104%	61	0	0	3m 6s	99.783% 0.104%	89	0	0	**3m 10s**	99.753% 0.104%	74	0	0	2m 44s
	D 1.042%	97.917% 1.020%	151	0	0	2m 56s	99.258% 1.034%	297	0	0	**3m 41s**	98.984% 1.031%	477	0	0	2m 58s
	C 8.333%	83.503% 6.958%	506	2	3	2h 1m	95.482% 7.956%	885	25	34	>13h	93.225% 7.768%	1145	23	33	**12h 57m 37s**
	B 50%	25.634% 12.817%	5516	7	11	**1h 28m 6s**	76.563% 38.281%	4917	123	182	>13h	63.906% 31.953%	7139	117	152	>13h
	A 100%	0.052% 0.052%	12	0	0	25m 51s	61.385% 61.385%	5156	73	102	**10h 25m 2s**	43.698% 43.698%	4757	68	88	>13h
320	F 0.009%	99.931% 0.009%	6	0	0	3m 15s	99.944% 0.009%	9	0	0	**3m 35s**	99.931% 0.009%	6	0	0	3m 30s
	E 0.104%	99.583% 0.104%	121	0	0	3m 39s	99.627% 0.104%	120	0	0	**6m 34s**	99.583% 0.104%	31	0	0	4m 22s
	D 1.042%	97.917% 1.020%	151	0	0	6m 18s	98.247% 1.024%	597	0	0	**21m 9s**	97.917% 1.020%	301	0	0	9m 35s
	C 8.333%	83.333% 6.944%	120	0	0	30m 37s	88.294% 7.358%	755	0	0	**1h 36m 35s**	83.342% 6.945%	483	0	0	52m 29s
	B 50%	25.000% 12.500%	5744	0	0	2h 24m 36s	46.063% 23.032%	4676	0	0	**7h 25m 57s**	25.074% 12.537%	5762	4	4	>13h
	A 100%	0.000% 0.000%	0	0	0	2h 54m 25s	24.258% 24.258%	2436	0	0	**9h 41m 36s**	0.017% 0.017%	4	0	0	5h 3m 33s
1280	F 0.009%	99.931% 0.009%	11	0	0	7m 35s	99.948% 0.009%	10	0	0	**24m 42s**	99.931% 0.009%	6	0	0	7m 6s
	E 0.104%	99.583% 0.104%	31	0	0	15m 49s	99.674% 0.104%	71	0	0	**51m 52s**	99.583% 0.104%	31	0	0	15m 14s
	D 1.042%	97.917% 1.020%	151	0	0	1h 49s	98.668% 1.028%	557	0	0	**3h 31m 45s**	97.917% 1.020%	301	0	0	1h 3m 33s
	C 8.333%	83.333% 6.944%	481	0	0	**7h 11m 39s**	–	–	–	–	>13h	83.333% 6.944%	481	0	0	7h 12m 57s
	B 50%	–	–	–	–	>13h	–	–	–	–	>13h	–	–	–	–	>13h
	A 100%	–	–	–	–	>13h	–	–	–	–	>13h	–	–	–	–	>13h

B: $A \wedge \text{age}^{8} \leq 53.5$ queried input space: 50.00%
C: $B \wedge \text{race} = \text{white}$ queried input space: 8.333% (3 race choices)
D: $C \wedge \text{work class} = \text{private}$ queried input space: 1.043% (4 work class choices)
E: $D \wedge \text{marital status} = \text{single}$ queried input space: 0.104% (5 marital status choices)
F: $E \wedge \text{occupation} = \text{blue-collar}$ queried input space: 0.009% (6 occupation choices)

For the analysis budget, we used L = 0.25, U = 0.1 ∗ |M|, and a time limit of 13h. Column INPUT shows, for each domain used for the forward pre-analysis, the coverage of the queried input space (i.e., the percentage of the input space that satisfies the query and was completed by the analysis) and the corresponding input-space coverage (i.e., the same percentage but this time scaled to the entire input space). Columns U, |C|, |F|, and TIME are as before. Where a timeout is indicated (i.e., TIME > 13h) and the values for the INPUT, |C|, and |F| columns are missing, it means that the timeout occurred during the pre-analysis; otherwise, it happened during the backward analysis. For each model and query, we highlighted the configuration (i.e., the abstract domain used for the

[8] This corresponds to *age* ≤ 0.5 with min-max scaling between 0 and 1.

Table 5. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 12 CPUs

L	U	⬟ BOXES					▲ SYMBOLIC					⋆ DEEPPOLY				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
0.5	4	15.28%	37	0	0	8s	58.33%	79	8	20	1m 26s	69.79%	115	10	39	3m 18s
	6	17.01%	39	6	6	51s	69.10%	129	22	61	5m 41s	80.56%	104	23	51	7m 53s
	8	51.39%	90	28	85	12m 2s	82.64%	88	31	67	12m 35s	91.32%	84	27	56	19m 33s
	10	79.86%	89	34	89	34m 15s	93.06%	98	40	83	42m 32s	96.88%	83	29	58	43m 39s
0.25	4	59.09%	1115	20	415	54m 32s	95.94%	884	39	484	54m 31s	98.26%	540	65	293	14m 29s
	6	83.77%	1404	79	944	37m 19s	98.68%	634	66	376	23m 31s	99.70%	322	79	205	13m 25s
	8	96.07%	869	140	761	1h 7m 29s	99.72%	310	67	247	1h 3m 33s	99.98%	247	69	177	22m 52s
	10	99.54%	409	93	403	1h 35m 20s	99.98%	195	52	176	1h 2m 13s	100.00%	111	47	87	34m 56s
0.125	4	97.13%	12449	200	9519	3h 33m 48s	99.99%	1101	60	685	47m 46s	99.99%	768	81	415	19m 1s
	6	99.83%	5919	276	4460	3h 23m	100.00%	988	77	606	26m 47s	100.00%	489	80	298	16m 54s
	8	99.98%	1926	203	1568	2h 14m 25s	100.00%	404	73	309	46m 31s	100.00%	175	57	129	20m 11s
	10	100.00%	428	95	427	1h 39m 31s	100.00%	151	53	141	57m 32s	100.00%	80	39	62	28m 33s
0	4	100.00%	19299	295	15446	6h 13m 24s	100.00%	1397	60	885	40m 5s	100.00%	766	87	425	**16m 41s**
	6	100.00%	4843	280	3679	2h 24m 7s	100.00%	763	66	446	35m 24s	100.00%	401	81	242	32m 29s
	8	100.00%	1919	208	1567	2h 9m 59s	100.00%	404	73	309	45m 48s	100.00%	193	68	144	24m 16s
	10	100.00%	486	102	475	1h 41m 3s	100.00%	217	55	192	1h 2m 11s	100.00%	121	50	91	30m 53s

pre-analysis) that achieved the highest input-space coverage with the shortest analysis running time. Note that, where the |F| column only contains zeros, it means that the backward analysis had no activation patterns to explore; this implies that the entire covered input space (i.e., the percentage shown in the INPUT column) was already certified to be fair by the forward analysis.

Overall, we observe that *whenever the analyzed input space is small enough* (i.e., queries $D - F$), *the size of the neural network has little influence on the input space coverage* and slightly impacts the analysis running time, independently of the domain used for the forward pre-analysis. Instead, for larger analyzed input spaces (i.e., queries $A - C$) performance degrades quickly for larger neural networks. These results thus support our claim. In fact, these considerations generalize to other research areas in the verification of neural networks, e.g., in the certification of local robustness against adversarial examples: the size of the perturbation is the most important factor that affects the performance of the verification [Tran et al. 2020]. Finally, again, we observe that the SYMBOLIC domain generally is the better choice for the forward pre-analysis, in particular for queries exercising a larger input space or larger neural networks.

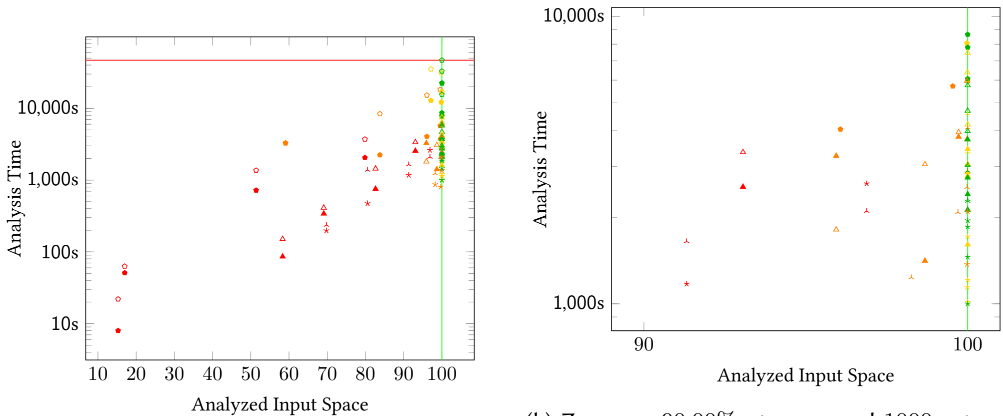
RQ5: Scalability-vs-Precision Tradeoff. To evaluate the effect of the analysis budget (bounds L and U), we analyzed a model using different budget configurations. For this experiment, we used the Japanese Credit Screening[9] dataset, which we made fair with respect to gender. Our 2-class model (17 inputs and 4 hidden layers with 5 nodes each) had a classification accuracy of 86%.

Table 5 shows the results of the analysis for different budget configurations and choices for the domain used for the forward pre-analysis. The best configuration in terms of input-space coverage and analysis running time is highlighted. The symbol next to each domain name introduces the marker used in the scatter plot of Figure 4a, which visualizes the coverage and running time. Figure 4b zooms on $90.00\% \leq$ INPUT and $1000s \leq$ TIME $\leq 1000s$.

Overall, we observe that *the more precise* SYMBOLIC *and* DEEPPOLY *domains boost input coverage*, most noticeably for configurations with a larger L. This additional precision does not always result in longer running times. In fact, a more precise pre-analysis often reduces the overall running time. This is because the pre-analysis is able to prove that more partitions are already fair without requiring them to go through the backward analysis (cf. columns |F|).

Independently of the chosen domain for the forward pre-analysis, as expected, *a larger* U *or a smaller* L *increase precision.* Increasing U or L typically reduces the number of completed partitions

[9] https://archive.ics.uci.edu/ml/datasets/Japanese+Credit+Screening



(a)

(b) Zoom on $90.00\% \leq$ INPUT and $1000s \leq$ TIME $\leq 1000s$

Fig. 4. Comparison of Different Analysis Configurations (Japanese Credit Screening)

Table 6. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 4 CPUs

L	U	⬠ BOXES					△ SYMBOLIC					⅄ DEEPPOLY				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
0.5	4	15.28%	44	0	0	22s	58.33%	96	8	26	2m 31s	69.79%	85	9	30	3m 57s
	6	17.01%	40	5	5	1m 3s	69.10%	97	18	45	6m 52s	80.56%	131	26	63	23m 6s
	8	51.39%	96	29	88	22m 47s	82.64%	128	34	87	24m 5s	91.32%	117	34	78	27m 28s
	10	79.86%	109	36	107	1h 1m 54s	93.06%	104	36	92	56m 8s	96.88%	69	31	50	35m 2s
0.25	4	59.09%	1147	22	405	54m 51s	95.94%	715	43	407	30m 12s	98.26%	488	65	272	20m 35s
	6	83.77%	1757	80	1149	2h 19m 50s	98.68%	693	73	400	50m 57s	99.70%	322	79	205	34m 42s
	8	96.07%	1129	136	950	4h 13m 49s	99.72%	289	62	232	1h 5m 53s	99.98%	153	56	113	42m 25s
	10	99.54%	510	92	497	5h 3m 34s	99.98%	158	57	150	1h 39m 14s	100.00%	109	46	85	1h 8m 18s
0.125	4	97.13%	12398	200	9491	9h 46m	99.99%	1864	58	1188	1h 46m 25s	99.99%	1257	92	670	51m 19s
	6	99.83%	5919	273	4460	8h 40m 11s	100.00%	697	71	404	50m 58s	100.00%	465	95	287	47m 53s
	8	99.98%	1331	212	1158	4h 39m 58s	100.00%	293	71	233	1h 10m 5s	100.00%	201	67	151	56m 12s
	10	100.00%	428	94	427	4h 45m 30s	100.00%	211	55	188	2h 4m 27s	100.00%	121	50	91	1h 16m 29s
0	4	100.00%	20631	296	16611	>13h	100.00%	1424	58	885	1h 6m 30s	100.00%	911	92	502	**37m 58s**
	6	100.00%	6093	296	4563	9h 8m 47s	100.00%	632	72	371	50m 37s	100.00%	403	85	247	38m 26s
	8	100.00%	1919	211	1567	6h 15m 29s	100.00%	378	79	287	1h 18m 16s	100.00%	174	65	128	48m 20s
	10	100.00%	402	93	401	4h 19m 3s	100.00%	180	56	154	1h 35m 56s	100.00%	82	38	63	50m 51s

(cf. columns |C|). Consequently, partitions tend to be more complex, requiring both forward and backward analyses. Since the backward analysis tends to dominate the running time, more partitions generally increase the running time (when comparing configurations with similar coverage). Based on our experience, the optimal budget largely depends on the analyzed model.

RQ6: Leveraging Multiple CPUs. To evaluate the effect of parallelizing the analysis using multiple cores, we re-ran the analyses of RQ5 on 4 CPU cores instead of 12. Table 6 shows these results. We observe the most significant increase in running time for 4 cores for the BOXES domain. On average, the running time increases by a factor of 2.6. On the other hand, for the SYMBOLIC and DEEPPOLY domains, the running time with 4 cores increases less drastically, on average by a factor of 1.6 and 2, respectively. This is again explained by the increased precision of the forward analysis; fewer partitions require a backward pass, where parallelization is most effective.

Note that the differences between the columns |C| and |F| of Table 5 and Table 6 are due to random choices that happen during the analysis, i.e., mainly the order in which (continuous) non-sensitive features are listed, which in turn dictates the order in which the partitioning happens

Table 7. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 24 vCPUs

L	U	BOXES					SYMBOLIC					DEEPPOLY				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
0.5	4	15.28%	36	0	0	7s	58.33%	120	7	34	3m 32s	69.79%	75	10	27	2m 43s
	6	17.01%	39	6	7	49s	69.10%	80	21	40	4m 19s	80.56%	138	26	65	12m 27s
	8	51.39%	92	30	86	12m 27s	82.64%	96	32	76	14m 13s	91.32%	89	36	61	13m 33s
	10	79.86%	89	34	89	29m 41s	93.06%	91	37	83	47m 1s	96.88%	73	33	52	30m
0.25	4	59.09%	1320	21	433	57m 33s	95.94%	656	42	340	32m 38s	98.26%	488	65	272	14m 11s
	6	83.77%	1600	80	1070	1h 6m 58s	98.68%	516	61	287	18m 6s	99.70%	286	77	182	13m 14s
	8	96.07%	1148	141	969	2h 41m 1s	99.72%	260	58	207	28m 57s	99.98%	241	70	175	29m 27s
	10	99.54%	409	93	403	1h 38m 38s	99.98%	213	50	189	1h 16m 11s	100.00%	88	42	68	20m 25s
0.125	4	97.13%	12449	203	9519	3h 59m 27s	99.99%	1101	59	685	1h 2m 58s	99.99%	892	86	493	18m 4s
	6	99.83%	4198	266	3234	2h 31m 54s	100.00%	759	73	461	51m 28s	100.00%	563	108	344	40m 35s
	8	99.98%	1741	217	1488	2h 16m 27s	100.00%	308	67	242	33m 14s	100.00%	230	67	167	22m 36s
	10	100.00%	582	97	564	2h 16m 13s	100.00%	180	56	154	1h 5m 59s	100.00%	80	39	62	30m 18s
0	4	100.00%	16018	288	12964	5h 3m 18s	100.00%	1883	63	1196	1h 52m 25s	100.00%	804	90	442	**19m 47s**
	6	100.00%	4675	279	3503	3h 2m 30s	100.00%	632	71	371	38m 3s	100.00%	302	75	189	19m 51s
	8	100.00%	1609	217	1382	2h 7m 9s	100.00%	326	67	252	1h 12s	100.00%	194	68	148	26m 9s
	10	100.00%	463	99	460	2h 12m 12s	100.00%	217	55	192	1h 13m 55s	100.00%	130	48	98	50m 10s

during the analysis. As a consequence, in some cases the analysis finds fewer (resp. more) activation patterns to analyze and thus might complete a little faster (resp. slower). Finding a good criterion for identifying an optimal partitioning strategy for the analysis is left for future work.

Table 7 shows the results of the same experiment on 24 vCPUs.

12 RELATED WORK

Significant progress has been made on testing and verifying machine-learning models. We focus on fairness, safety, and robustness properties in the following, especially of deep neural networks.

Fairness Criteria. There are countless fairness definitions in the literature. In this paper, we focus on dependency fairness (originally called causal fairness [Galhotra et al. 2017]) and compare here with the most popular and related fairness notions.

Demographic parity [Feldman et al. 2015] (or group fairness) is the most common non-causal notion of fairness. It states that individuals with different values of sensitive features, hence belonging to different groups, should have the same probability of being predicted to the positive class. For example, a loan system satisfies group fairness with respect to gender if male and female applicants have equal probability of getting loans. If unsatisfied, this notion is also referred to as *disparate impact*. Our notion of fairness is stronger, as it imposes fairness on every pair of individuals that differ only in sensitive features. A classifier that satisfies group fairness does not necessarily satisfy dependency fairness, because there may still exist pairs of individuals on which the classifier exhibits bias.

Another group-based notion of fairness is *equality of opportunity* [Hardt et al. 2016]. It states that *qualified* individuals with different values of sensitive features should have equal probability of being predicted to the positive class. For a loan system, this means that male and female applicants who are qualified to receive loans should have an equal chance of being approved. By imposing fairness on every qualified pair of individuals that differ only in sensitive features, we can generalize dependency fairness to also concern both prediction and actual results. We can then adapt our technique to consider only the part of the input space that includes qualified individuals.

Other causal notions of fairness [Chiappa 2019; Kilbertus et al. 2017; Kusner et al. 2017; Nabi and Shpitser 2018, etc.] require additional knowledge in the form of a *causal model* [Pearl 2009]. A causal model can drive the choice of the sensitive input(s) for our analysis.

Testing and Verifying Fairness. Galhotra et al. [Galhotra et al. 2017] proposed an approach, Themis, that allows efficient fairness testing of software. Udeshi et al. [Udeshi et al. 2018] designed an automated and directed testing technique to generate discriminatory inputs for machine-learning models. Tramer et al. [Tramèr et al. 2017] introduced the unwarranted-associations framework and instantiated it in FairTest. In contrast, our technique provides formal fairness guarantees.

Bastani et al. [Bastani et al. 2019] used adaptive concentration inequalities to design a scalable sampling technique for providing probabilistic fairness guarantees for machine-learning models. As mentioned in the Introduction, our approach differs in that it gives definite (instead of probabilistic) guarantees. However, it might exclude partitions for which the analysis is not exact.

Albarghouthi et al. [Albarghouthi et al. 2017b] encoded fairness problems as probabilistic program properties and developed an SMT-based technique for verifying fairness of decision-making programs. As discussed in the Introduction, this technique has been shown to scale only up to neural networks with at most 3 inputs and a single hidden layer with at most 2 nodes. In contrast, our approach is designed to be perfectly parallel, and thus, is significantly more scalable and can analyze neural networks with hundreds (or, in some cases, even thousands) of hidden nodes.

A recent technique [Ruoss et al. 2020] certifies individual fairness of neural networks, which is a local property that coincides with robustness within a particular distance metric. In particular, individual fairness dictates that similar individuals should be treated similarly. Our approach, however, targets certification of neural networks for the global property of dependency fairness.

For certain biased decision-making programs, the program repair technique proposed by Albarghouthi et al. [Albarghouthi et al. 2017a] can be used to repair their bias. Albarghouthi and Vinitsky [Albarghouthi and Vinitsky 2019] further introduced fairness-aware programming, where programmers can specify fairness properties in their code for runtime checking.

Robustness of Deep Neural Networks. Robustness is a desirable property for traditional software [Chaudhuri et al. 2012; Goubault and Putot 2013; Majumdar and Saha 2009], especially control systems. Deep neural networks are also expected to be robust. However, research has shown that deep neural networks are not robust to small perturbations of their inputs [Szegedy et al. 2014] and can even be easily fooled [Nguyen et al. 2015]. Subtle imperceptible perturbations of inputs, known as adversarial examples, can change their prediction results. Various algorithms [Carlini and Wagner 2017b; Goodfellow et al. 2015; Madry et al. 2018; Tabacof and Valle 2016; Zhang et al. 2020] have been proposed that can effectively find adversarial examples. Research on developing defense mechanisms against adversarial examples [Athalye et al. 2018; Carlini and Wagner 2016, 2017a,b; Cornelius 2019; Engstrom et al. 2018; Goodfellow et al. 2015; Huang et al. 2015; Mirman et al. 2018, 2019] is also active. Dependency fairness is a special form of robustness in the sense that neural networks are expected to be *globally* robust with respect to their sensitive features.

Testing Deep Learning Systems. Multiple frameworks have been proposed to test the robustness of deep learning systems. Pei et al. [Pei et al. 2017] proposed the first whitebox framework for testing such systems. They used neuron coverage to measure the adequacy of test inputs. Sun et al. [Sun et al. 2018] presented the first concolic-testing [Godefroid et al. 2005; Sen et al. 2005] approach for neural networks. Tian et al. [Tian et al. 2018] and Zhang et al. [Zhang et al. 2018] proposed frameworks for testing autonomous driving systems. Gopinath et al. [Gopinath et al. 2018] used symbolic execution [Clarke 1976; King 1976]. Odena et al. [Odena et al. 2019] were the first to develop coverage-guided fuzzing for neural networks. Zhang et al. [Zhang et al. 2020] proposed a blackbox-fuzzing technique to test their robustness.

Formal Verification of Deep Neural Networks. Formal verification of deep neural networks has mainly focused on safety properties. However, the scalability of such techniques for verifying large real-world neural networks is limited. Early work [Pulina and Tacchella 2010] applied abstract interpretation to verify a neural network with six neurons. Recent work [Gehr et al. 2018; Huang et al. 2017; Katz et al. 2017; Singh et al. 2019; Wang et al. 2018] significantly improves scalability. Huang et al. [Huang et al. 2017] proposed a framework that can verify local robustness of neural networks based on SMT techniques [Barrett and Tinelli 2018]. Katz et al. [Katz et al. 2017] developed an efficient SMT solver for neural networks with ReLU activation functions. Gehr et al. [Gehr et al. 2018] traded precision for scalability and proposed a sound abstract interpreter that can prove local robustness of realistic deep neural networks. Singh et al. [Singh et al. 2019] proposed the deeppoly domain for certifying robustness of neural networks. Wang et al. [Wang et al. 2018] are the first to use symbolic interval arithmetic to prove security properties of neural networks.

13 CONCLUSION AND FUTURE WORK

We have presented a *novel, automated, perfectly parallel static analysis* for certifying fairness of feed-forward neural networks used for classification of tabular data — a problem of ever-increasing real-world importance nowadays. Our approach provides *definite fairness guarantees* for the analyzed input space and it is the first in this area that is *configurable in terms of scalability and precision*. The analysis can thus support a wide range of use cases throughout the development lifecycle of neural networks: ranging from quick sanity checks during development to formal fairness audits before deployments. We have rigorously formalized the approach and proved its soundness, and we demonstrated its effectiveness in practice with an extensive experimental evaluation.

In future work, we plan on integrating strategies in libra for making the parameter configuration of U and L automatic during the analysis. For instance, by starting the analysis with a low U (resp. large L) and gradually increasing (resp. decreasing) it only on the input space partitions on which it is necessary. We also plan on supporting other encodings for categorical features than one-hot encoding, such as entity embeddings [Guo and Berkhahn 2016], which have become more and more popular in recent years. Our approach will also automatically benefit from future advances in the design of abstract domains for analyzing neural networks. Vice versa, we believe and hope that future work can also build on our approach. For instance, other tools could feed on the results of our analysis and, say, provide weaker fairness guarantees (e.g., probabilistic) for the input partitions that could not be certified, or repair the analyzed neural network models by eliminating bias. More generally, we believe that the design of the approach and its underlying ideas, such as that of leveraging abstract activation patterns, are potentially useful in other verification settings, e.g., proving functional properties of neural networks [Katz et al. 2017; Wang et al. 2018, etc.].

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive feedback. This work was supported by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science).

REFERENCES

Aws Albarghouthi, Loris D'Antoni, and Samuel Drews. 2017a. Repairing Decision-Making Programs Under Uncertainty. In *CAV*. 181–200. https://doi.org/10.1007/978-3-319-63387-9_9

Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017b. FairSquare: Probabilistic Verification of Program Fairness. *PACMPL* 1, OOPSLA (2017), 80:1–80:30. https://doi.org/10.1145/3133904

Aws Albarghouthi and Samuel Vinitsky. 2019. Fairness-Aware Programming. In *FAT**. 211–219. https://doi.org/10.1145/3287560.3287588

Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *ICML (PMLR, Vol. 80)*. PMLR, 274–283.

Solon Barocas and Andrew D. Selbst. 2016. Big Data's Disparate Impact. *California Law Review* 104, 3 (2016), 671–732.

Clark W. Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*. Springer, 305–343.

Alexander I. Barvinok. 1994. A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension is Fixed. *Mathematics of Operations Research* 19, 4 (1994), 769–779. https://doi.org/10.1287/moor.19.4.769

Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. 2019. Probabilistic Verification of Fairness Properties via Concentration. *PACMPL* 3, OOPSLA (2019), 118:1–118:27.

Joy Buolamwini and Timnit Gebru. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *FAT (PMLR, Vol. 81)*. PMLR, 77–91.

Nicholas Carlini and David A. Wagner. 2016. Defensive Distillation is Not Robust to Adversarial Examples. *CoRR* abs/1607.04311 (2016).

Nicholas Carlini and David A. Wagner. 2017a. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *AISec@CCS*. ACM, 3–14.

Nicholas Carlini and David A. Wagner. 2017b. Towards Evaluating the Robustness of Neural Networks. In *S&P*. IEEE Computer Society, 39–57.

Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. 2012. Continuity and Robustness of Programs. *Commun. ACM* 55, 8 (2012), 107–115. https://doi.org/10.1145/2240236.2240262

Liqian Chen, Antoine Miné, and Patrick Cousot. 2008. A Sound Floating-Point Polyhedra Abstract Domain. In *APLAS*. 3–18. https://doi.org/10.1007/978-3-540-89330-1_2

Silvia Chiappa. 2019. Path-Specific Counterfactual Fairness. In *AAAI*. 7801–7808. https://doi.org/10.1609/aaai.v33i01.33017801

Lori A. Clarke. 1976. A System to Generate Test Data and Symbolically Execute Programs. *TSE* 2 (1976), 215–222. Issue 3.

Cory Cornelius. 2019. The Efficacy of SHIELD Under Different Threat Models. *CoRR* abs/1902.00541 (2019).

Patrick Cousot. 2002. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science* 277, 1-2 (2002), 47–103. https://doi.org/10.1016/S0304-3975(00)00313-3

Patrick Cousot and Radhia Cousot. 1976. Static Determination of Dynamic Properties of Programs. In *Second International Symposium on Programming*. 106–130.

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*. 238–252. https://doi.org/10.1145/512950.512973

Patrick Cousot and Radhia Cousot. 1979. Systematic Design of Program Analysis Frameworks. In *POPL*. 269–282. https://doi.org/10.1145/567752.567778

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*. 84–96. https://doi.org/10.1145/512760.512770

Anupam Datta, Matthew Fredrikson, Gihyuk Ko, Piotr Mardziel, and Shayak Sen. 2017. Use Privacy in Data-Driven Systems: Theory and Experiments with Machine Learnt Programs. In *CCS*. 1193–1210. https://doi.org/10.1145/3133956.3134097

Logan Engstrom, Andrew Ilyas, and Anish Athalye. 2018. Evaluating and Understanding the Robustness of Adversarial Logit Pairing. *CoRR* abs/1807.10272 (2018).

Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *KDD*. ACM, 259–268.

Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness Testing: Testing Software for Discrimination. In *FSE*. 498–510. https://doi.org/10.1145/3106237.3106277

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *S & P*. 3–18. https://doi.org/10.1109/SP.2018.00058

Khalil Ghorbal, Eric Goubault, and Sylvie Putot. 2009. The Zonotope Abstract Domain Taylor1+. In *CAV*. 627–633. https://doi.org/10.1007/978-3-642-02658-4_47

Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed Automated Random Testing. In *PLDI*. ACM, 213–223.

Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*. http://arxiv.org/abs/1412.6572

Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S. Pasareanu, and Sarfraz Khurshid. 2018. Symbolic Execution for Deep Neural Networks. *CoRR* abs/1807.10439 (2018).

Eric Goubault and Sylvie Putot. 2013. Robustness Analysis of Finite Precision Implementations. In *APLAS*. 50–57. https://doi.org/10.1007/978-3-319-03542-0_4

Nina Grgić-Hlača, Muhammad Bilal Zafar, Krishna P. Gummadi, and Adrian Weller. 2016. The Case for Process Fairness in Learning: Feature Selection for Fair Decision Making. In *NIPS 2016 ML and the Law*.

Cheng Guo and Felix Berkhahn. 2016. Entity Embeddings of Categorical Variables. *CoRR* abs/1604.06737 (2016).

Boris Hanin and David Rolnick. 2019. Deep ReLU Networks Have Surprisingly Few Activation Patterns. In *NIPS*. Curran Associates, Inc., 359–368. http://papers.nips.cc/paper/8328-deep-relu-networks-have-surprisingly-few-activation-

patterns.pdf
Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *NIPS*. 3315–3323.
Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. 2015. Learning with a Strong Adversary. *CoRR* abs/1511.03034 (2015). http://arxiv.org/abs/1511.03034
Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety Verification of Deep Neural Networks. In *CAV*. 3–29. https://doi.org/10.1007/978-3-319-63387-9_1
Bertrand Jeannet and Antoine Miné. 2009. APRON: A Library of Numerical Abstract Domains for Static Analysis. In *CAV*. 661–667. https://doi.org/10.1007/978-3-642-02658-4_52
Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV*. 97–117. https://doi.org/10.1007/978-3-319-63387-9_5
Matthew Kay, Cynthia Matuszek, and Sean A. Munson. 2015. Unequal Representation and Gender Stereotypes in Image Search Results for Occupations. In *CHI*. ACM, 3819–3828.
Niki Kilbertus, Mateo Rojas-Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, and Bernhard Schölkopf. 2017. Avoiding Discrimination Through Causal Reasoning. In *NIPS*. 656–666.
James C. King. 1976. Symbolic Execution and Program Testing. *CACM* 19 (1976), 385–394. Issue 7.
Matt Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual Fairness. In *NIPS*. 4069–4079.
Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. 2016. How We Analyzed the COMPAS Recidivism Algorithm. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm.
Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. 2019. Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification. In *SAS*. 296–319. https://doi.org/10.1007/978-3-030-32304-2_15
Kristian Lum and William Isaac. 2016. To Predict and Serve? *Significance* 13 (2016), 14–19. Issue 5.
Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR*. OpenReview.net.
Rupak Majumdar and Indranil Saha. 2009. Symbolic Robustness Analysis. In *RTSS*. 355–363. https://doi.org/10.1109/RTSS.2009.17
Padala Manisha and Sujit Gujar. 2020. FNNC: Achieving Fairness Through Neural Networks. In *IJCAI*. 2277–2283.
Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2019. A Survey on Bias and Fairness in Machine Learning. *CoRR* abs/1908.09635 (2019).
Antoine Miné. 2004. Relational Abstract Domains for the Detection of Floating-Point Run-Time Errors. In *ESOP*. 3–17. https://doi.org/10.1007/978-3-540-24725-8_2
Antoine Miné. 2006a. Symbolic Methods to Enhance the Precision of Numerical Abstract Domains. In *VMCAI*. 348–363. https://doi.org/10.1007/11609773_23
Antoine Miné. 2006b. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation* 19, 1 (2006), 31–100. https://doi.org/10.1007/s10990-006-8609-1
Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML*. 3575–3583.
Matthew Mirman, Gagandeep Singh, and Martin T. Vechev. 2019. A Provable Defense for Deep Residual Networks. *CoRR* abs/1903.12519 (2019).
Razieh Nabi and Ilya Shpitser. 2018. Fair Inference on Outcomes. In *AAAI*. AAAI Press.
Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*. 807–814.
Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *CVPR*. 427–436. https://doi.org/10.1109/CVPR.2015.7298640
Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. 2019. Dissecting Racial Bias in an Algorithm Used to Manage the Health of Populations. *Science* 366 (2019), 447–453. Issue 6464.
Augustus Odena, Catherine Olsson, David Andersen, and Ian J. Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *ICML (PMLR, Vol. 97)*. PMLR, 4901–4911.
Judea Pearl. 2009. *Causality: Models, Reasoning and Inference*. Cambridge University Press.
Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *SOSP*. 1–18. https://doi.org/10.1145/3132747.3132785
Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *CAV*. 243–257. https://doi.org/10.1007/978-3-642-14295-6_24
Anian Ruoss, Mislav Balunovic, Marc Fischer, and Martin T. Vechev. 2020. Learning Certified Individually Fair Representations. *CoRR* abs/2002.10312 (2020).
Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: A Concolic Unit Testing Engine for C. In *ESEC/FSE*. ACM, 263–272.
Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. An Abstract Domain for Certifying Neural Networks. *PACMPL* 3, POPL (2019), 41:1–41:30. https://doi.org/10.1145/3290354

Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In *ASE*. ACM, 109–119.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. In *ICLR*. http://arxiv.org/abs/1312.6199

Pedro Tabacof and Eduardo Valle. 2016. Exploring the Space of Adversarial Images. In *IJCNN*. 426–433. https://doi.org/10.1109/IJCNN.2016.7727230

Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *ICSE*. ACM, 303–314.

Florian Tramèr, Vaggelis Atlidakis, Roxana Geambasu, Daniel J. Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. 2017. FairTest: Discovering Unwarranted Associations in Data-Driven Applications. In *EuroS&P*. IEEE, 401–416.

Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *CAV*. 18–42.

Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated Directed Fairness Testing. In *ASE*. ACM, 98–108.

Caterina Urban and Peter Müller. 2018. An Abstract Interpretation Framework for Input Data Usage. In *ESOP*. 683–710. https://doi.org/10.1007/978-3-319-89884-1_24

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks Using Symbolic Intervals. In *Security*. USENIX, 1599–1614.

Mikhail Yurochkin, Amanda Bower, and Yuekai Sun. 2020. Training Individually Fair ML Models with Sensitive Subspace Robustness. In *ICLR*.

Fuyuan Zhang, Sankalan Pal Chowdhury, and Maria Christakis. 2020. DeepSearch: A Simple and Effective Blackbox Attack for Deep Neural Networks. In *ESEC/FSE*. ACM. To appear.

Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *ASE*. ACM, 132–142.