Perfectly Parallel Fairness Certification of Neural Networks

Caterina Urban, Maria Christakis, Valentin Wüstholz, Fuyuan Zhang

Perfectly Parallel Fairness Certification of Neural Networks

Caterina Urban
INRIA
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 Diligence
Germany
valentin.wustholz@consensys.net

Fuyuan Zhang
MPI-SWS
Germany
fuyuan@mpi-sws.org

Abstract

Recently, there is growing concern that machine-learning models, which currently assist or even automate decision making, reproduce, and in the worst case reinforce, bias of the training data. The development of tools and techniques for certifying fairness of these models or describing their biased behavior is, therefore, critical. In this paper, we propose a *perfectly parallel* static analysis for certifying *causal fairness* of feed-forward neural networks used for classification tasks. When certification succeeds, our approach provides definite guarantees, otherwise, it describes and quantifies the biased behavior. 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 and demonstrate its effectiveness on models trained with popular datasets.

1 Introduction

Due to the tremendous advances in machine learning and the vast amounts of available data, software systems, and neural networks in particular, are of ever-increasing importance in our everyday decisions, whether by assisting them or by autonomously making them. 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, therefore, 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 [39]. 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 [52]. 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 [36]. Commercial facial recognition algorithms, which are increasingly used in law enforcement, are less effective for women and darker skin types [8].

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

Causal fairness. To meet these needs, we have designed a static analysis framework for certifying *causal fairness* [38] of feed-forward neural networks. Specifically, given input features that are sensitive to bias, like race or gender, a neural network is causally fair if the output classification is not affected by different values of the sensitive features.

Of course, the most obvious approach to avoid such bias is to remove any sensitive feature from the training data. However, this does not work for three main reasons. First, neural networks learn from latent variables (e.g., [41, 63]). For instance, a credit-screening algorithm might not use gender as an explicit input but still be biased with respect to it, say, because most individuals whose first name ends in 'a' are denied credit in the training data. 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 inaccurate. For example, if Asians are underrepresented in the training data, facial recognition is less likely to be accurate for these people. Third, the information provided by a sensitive feature might be necessary, for instance, to introduce intended bias in a certain input region. Assume a credit-screening algorithm that should not discriminate with respect to age unless it is 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.

Our approach. Our approach certifies causal fairness of neural networks used for *classification* 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) sensitive feature(s). In the end, our approach reports for which regions of the input space the neural network is proved 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. For instance, 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 analyzes.

Related work. In the literature, most work on verifying fairness of machine-learning models has focused on providing probabilistic guarantees (e.g., [2, 7]). In contrast, our approach gives definite guarantees for those input partitions that satisfy the analysis configuration. Moreover, our approach is *exact* for these partitions. In other words, the tradeoff in comparison to related work is that we might exclude partitions for which our analysis is not exact. In this paper, we investigate how far we can push such an exact analysis in the context of fairness certification of neural networks.

Contributions. We make the following contributions:

1. We propose a perfectly parallel static analysis approach for certifying causal fairness of feed-forward neural networks. If certification fails, our approach can describe and quantify the biased input space region(s).
2. We show that our approach 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 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.

2 Overview

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

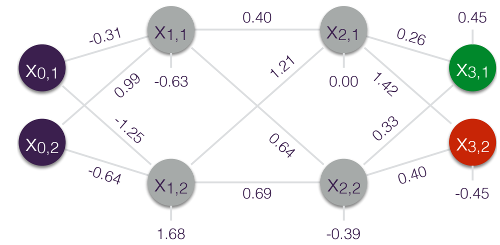

Figure 1. Small, constructed example of trained feed-forward neural network for credit approval.

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 does not discriminate with respect to age for small credit amounts. There is also no bias for younger people with respect to the requested credit. 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 the age for large credit amounts. Note that this bias may be intended or accidental — our analysis does not aim to address this question. Below, we present on a high level how our approach achieves these results.

Naïve approach. In theory, the simplest way to certify fairness with respect to a given sensitive input 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 (smaller than 10 000), some of whom receive the credit while others do not.

This approach, however, is not practical. Specifically, for a neural network using the popular 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, a backward 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* [31]. 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 [17]) 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). 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 four, 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 ($x_{0,1} : [0.5, 1].x_{0,2} : [0, 1]$). Note 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., all three 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 $I_{2,2}$ still 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 the analysis.

Feasible input partitions (within bounds L and U) are then 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 the 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 [19, 20], 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. 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.

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., two 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 *sensitive* to bias (e.g., $x_{0,2}$ in Figure 1) and $\overline{\mathrm{K}} \stackrel{\text{def}}{=} \mathrm{X}_0 \setminus \mathrm{K}$ to denote the input nodes not deemed sensitive to bias.

The value of each hidden and output node $\mathrm{x}_{i,j}$ is computed by an *activation function* f applied to a linear combination of the values of all nodes in the preceding layer [27], i.e., $\mathrm{x}_{i,j} = f\left(\sum_k^{|\mathrm{L}_{i-1}|} \mathrm{w}_{j,k}^i \cdot \mathrm{x}_{i-1,k} + \mathrm{B}_{i,j}\right)$, where $\mathrm{w}_{j,k}^i$ and $\mathrm{B}_{i,j}$ are weight and bias coefficients in W_i and B_i, respectively. In a *fully-connected neural network*, all $\mathrm{w}_{j,k}^i$ 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) [50]: $\mathrm{ReLU}(x) = \max(x, 0)$. 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 [27]. We discuss other activation functions in Section 9.

4 Trace Semantics

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 (potentially infinite) set of states and the *acyclic* transition relation $\tau \subseteq \Sigma \times \Sigma$ describes the possible transitions between states [16, 18].

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 $\Sigma_{|Y} \stackrel{\text{def}}{=} \left\{ s_{|Y} \;\middle|\; s \in \Sigma \right\}$ 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, that is, $\langle s, s' \rangle \in \tau$ 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 [16]:

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

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

The trace semantics fully describes the behavior of M. However, reasoning about a particular 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 about M's behavior. In the following sections, we formally define our property of interest, causal fairness, and systematically derive, using *abstract interpretation* [18], a semantics tailored to reasoning about this property.

5 Causal Fairness

A *property* is specified by its extension, that is, by the set of elements having such a property [18, 19]. 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 *causal 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 $(\mathrm{x}_{0,i}, \mathrm{x}_{0,j})$ one-hot encoding, say, gender information, $\mathbb{V} = \{(1,0), (0,1)\}$; for $x_{0,k}$ encoding continuous data, say, in the range $[0, 1]$, a possibility is $\mathbb{V} = \{[0, 0.25], [0.25, 0.75], [0.75, 1]\}$. 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 $T \in \mathcal{P}(\Sigma^+)$ [64]

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

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). In other words, 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 causal-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 $\mathcal{F}_\mathrm{K}[Y]$.

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 $\mathcal{F}_\mathrm{K}[Y]$ (and in particular, from the definition of $\textsc{unused}_\mathrm{K}$, cf. Equation 4), we have the following result:

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

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

In particular, this means that in order to determine whether a neural-network model M satisfies causal fairness, we can independently verify, for each of its possible target classes $\mathrm{O} \in \mathbb{O}$, 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 $\bullet \stackrel{\text{def}}{=} \{\Sigma^+_\mathrm{O} \mid \mathrm{O} \in \mathbb{O}\}$ be a trace partition with respect to outcome. We have the following Galois connection

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

where $\alpha_\bullet(S) \stackrel{\text{def}}{=} \{T_\mathrm{O} \mid T \in S \wedge \mathrm{O} \in \mathbb{O}\}$. The order $\mathrel{\dot\subseteq}$ is the pointwise ordering between sets of traces with the same outcome, i.e., $A \mathrel{\dot\subseteq} B \stackrel{\text{def}}{=} \bigwedge_{\mathrm{O} \in \mathbb{O}} \dot{A}_\mathrm{O} \subseteq \dot{B}_\mathrm{O}$, 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:

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

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

6.2 Dependency Semantics

We observe that, to reason about causal 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[1]

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

where $\alpha_\rightsquigarrow(S) \stackrel{\text{def}}{=} \{\{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in T\} \mid T \in S\}$ [64] abstracts away all intermediate states of any trace. We finally derive the *dependency semantics* $\Lambda_\rightsquigarrow \in \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma))$:

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

In the following, let $(\!|\mathrm{M}|\!)_\rightsquigarrow$ denote the dependency semantics of a particular neural-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 $S^Y \stackrel{\text{def}}{=} \{R^Y \mid R \in S\}$ 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 causal fairness:

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

[1] Note that here and in the following, for convenience, we abuse notation and reuse the order symbol $\mathrel{\dot\subseteq}$, 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 $\llparenthesis \mathrm{M} \rrparenthesis^{Y} \subseteq \mathcal{F}_{\mathrm{K}}[Y]$. Thus, from the Galois connections in Equation 6 and 8, we have $\alpha_{\rightsquigarrow}(\alpha_{\bullet}(\llparenthesis \mathrm{M} \rrparenthesis^{Y})) \sqsubseteq \alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y]))$. From the definition of $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow}$ (cf. Equation 9), we can then conclude that $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow} \sqsubseteq \alpha_{\rightsquigarrow}(\alpha_{\bullet}(\mathcal{F}_{\mathrm{K}}[Y]))$. □

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

Proof. The proofs 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 $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow} \subseteq \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$ by checking if the dependency semantics $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow}$ 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. $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y] \Leftrightarrow$
$\forall A, B \in \llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow} \colon (A_\omega \neq B_\omega \Rightarrow A_{0|\overline{\mathrm{K}}} \cap B_{0|\overline{\mathrm{K}}} = \emptyset)$

Proof. Let $\mathrm{M} \models \mathcal{F}_{\mathrm{K}}[Y]$. From Corollary 6.3, we have that $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow} \subseteq \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$. Thus, from the definition of $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow}$ (cf. Equation 9), we have $\forall \mathrm{O} \in \mathbb{O} \colon \alpha_{\rightsquigarrow}(\llbracket \mathrm{M} \rrbracket^{Y}_{\mathrm{O}}) \in \alpha_{\rightsquigarrow}(\mathcal{F}_{\mathrm{K}}[Y])$. In particular, from the definition of $\alpha_{\rightsquigarrow}$ and $\mathcal{F}_{\mathrm{K}}[Y]$ (cf. Equation 5), we have that $\textsc{unused}_{\mathrm{K}}(\llbracket \mathrm{M} \rrbracket^{Y}_{\mathrm{O}})$ for each $\mathrm{O} \in \mathbb{O}$. From the definition of $\textsc{unused}_{\mathrm{K}}$ (cf. Equation 4), for each pair of *non-empty* $\llbracket \mathrm{M} \rrbracket^{Y}_{\mathrm{O}_1}$ and $\llbracket \mathrm{M} \rrbracket^{Y}_{\mathrm{O}_2}$ 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 $\forall A, B \in \llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow} \colon (A_\omega \neq B_\omega \Rightarrow A_{0|\overline{\mathrm{K}}} \cap B_{0|\overline{\mathrm{K}}} = \emptyset)$. □

7 Naïve Causal-Fairness Analysis

In this section, we present a first static analysis for causal 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 backward 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).

Algorithm 1 : A Naïve Backward Analysis

1: **function** BACKWARD(M, A, x)
2: a $\leftarrow \textsc{outcome}_{\mathrm{A}}\llbracket \mathrm{x} \rrbracket(\textsc{new}_{\mathrm{A}})$
3: **for** $i \leftarrow \mathrm{N} - 1$ **down to** 0 **do**
4: **for** $j \leftarrow |\mathrm{L}_i|$ **down to** 0 **do**
5: a $\leftarrow \overleftarrow{\textsc{assign}}_{\mathrm{A}}\llbracket \mathrm{x}_{i,j} \rrbracket(\overleftarrow{\textsc{relu}}_{\mathrm{A}}\llbracket \mathrm{x}_{i,j} \rrbracket \mathrm{a})$
6: **return** a
7: **function** CHECK(O)
8: $\mathrm{B} \leftarrow \emptyset$ ▷ B: biased
9: **for all** $\mathrm{o}_1, \mathrm{a}_1 \in \mathrm{O}$ **do**
10: **for all** $\mathrm{o}_2 \neq \mathrm{o}_1, \mathrm{a}_2 \in \mathrm{O}$ **do**
11: **if** $\mathrm{a}_1 \sqcap_{\mathrm{A}_2} \mathrm{a}_2 \neq \bot_{\mathrm{A}_2}$ **then**
12: $\mathrm{B} \leftarrow \mathrm{B} \cup \{\mathrm{a}_1 \sqcap_{\mathrm{A}_2} \mathrm{a}_2\}$
13: **return** B
14: **function** ANALYZE(M, K, Y, A)
15: $\mathrm{O} \leftarrow \dot{\emptyset}$
16: **for** $j \leftarrow 0$ **up to** $|\mathrm{L}_\mathrm{N}|$ **do** ▷ perfectly parallelizable
17: a $\leftarrow$ BACKWARD(M, A, $\mathrm{x}_{\mathrm{N},j}$)
18: $\mathrm{O} \leftarrow \mathrm{O} \cup \left\{\mathrm{x}_{\mathrm{N},j} \mapsto (\textsc{assume}_{\mathrm{A}}\llbracket Y \rrbracket \mathrm{a})_{|\overline{\mathrm{K}}}\right\}$
19: $\mathrm{B} \leftarrow$ CHECK(O)
20: **return** $\mathrm{B} = \emptyset$, B ▷ fair: $\mathrm{B} = \emptyset$, maybe biased: $\mathrm{B} \neq \emptyset$

More specifically, the transfer function $\textsc{outcome}_{\mathrm{A}}\llbracket \mathrm{x} \rrbracket$ (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 $\overleftarrow{\textsc{relu}}_{\mathrm{A}}\llbracket \mathrm{x}_{i,j} \rrbracket$ and $\overleftarrow{\textsc{assign}}_{\mathrm{A}}\llbracket \mathrm{x}_{i,j} \rrbracket$ (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 causal 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 causal 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 ANALYZE(M, K, Y, A) of Algorithm 1 returns TRUE, $\emptyset$ then M satisfies $\mathcal{F}_{\mathrm{K}}[Y]$.

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 $\mathrm{x}_{\mathrm{N},j}$ (cf. Line 17). Thus, it actually computes an over-approximation $\llparenthesis \mathrm{M} \rrparenthesis^{Y\natural}_{\rightsquigarrow}$ of the dependency semantics $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow}$, i.e., $\llparenthesis \mathrm{M} \rrparenthesis^{Y}_{\rightsquigarrow} \sqsubseteq \llparenthesis \mathrm{M} \rrparenthesis^{Y\natural}_{\rightsquigarrow}$. Thus, if $\llparenthesis \mathrm{M} \rrparenthesis^{Y\natural}_{\rightsquigarrow}$ satisfies $\mathcal{F}_{\mathrm{K}}[Y]$, i.e., $\forall A, B \in \llparenthesis \mathrm{M} \rrparenthesis^{Y\natural}_{\rightsquigarrow} \colon (A_\omega \neq B_\omega \Rightarrow A_{0|\overline{\mathrm{K}}} \cap B_{0|\overline{\mathrm{K}}} = \emptyset)$ (according to Lemma 6.4, cf. Line 19), then by transitivity we can conclude that also $\llparenthesis \mathrm{M} \rrparenthesis^{Y\natural}_{\rightsquigarrow}$ necessarily satisfies $\mathcal{F}_{\mathrm{K}}[Y]$. □

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 [20] would make for a rather fast but too imprecise analysis. This is because they are not able to precisely handle constraints like $\max \mathrm{X_N} = \mathrm{x}$, which are introduced by $\textsc{outcome}_{\mathrm{A}}[\![\mathrm{x}]\!]$ to partition with respect to outcome.

Furthermore, even polyhedra would not be precise enough in general. Indeed, each $\overleftarrow{\textsc{relu}}_{\mathrm{A}}[\![\mathrm{x}_{i,j}]\!]$ would over-approximate what effectively is a conditional branch. Let $|\mathrm{M}| \overset{\text{def}}{=} |\mathrm{L}_1| + \cdots + |\mathrm{L}_{\mathrm{N}-1}|$ 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 [19] 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. Lines 16), each of the $|\mathrm{L_N}|$ processes would have to effectively explore $2^{|\mathrm{M}|}$ 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.

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: $\forall \mathrm{I} \in \mathbb{I}, \mathrm{V} \in \mathbb{V}\colon \exists s \in \mathbb{I}\colon s(\mathrm{K}) = \mathrm{V}$. For instance, $\mathbb{I} = \{T_0, T_0'\}$, with T and T' in Example 5.1 is a fair input partition of $Y = \{s \mid s(\mathrm{x}_{0,1}) = 0.5 \vee s(\mathrm{x}_{0,1}) = 0.75\}$.

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

$$\langle \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma)), \Subset \rangle \underset{\alpha_{\mathbb{I}}}{\overset{\gamma_{\mathbb{I}}}{\leftrightarrows}} \langle \mathcal{P}(\mathcal{P}(\Sigma \times \Sigma)), \Subset_{\mathbb{I}} \rangle, \tag{10}$$

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

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

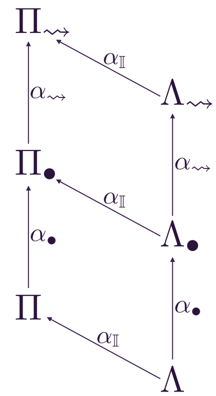

Figure 2. Hierarchy of semantics.

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

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

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 proofs follows trivially from the definition of $\Subset_{\mathbb{I}}$ (cf. Equation 6 and 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 Causal-Fairness Analysis

In this section, we build on the parallel semantics to design our novel *perfectly parallel* static analysis for causal 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., $\Pi^{\mathbb{I}}_{\rightsquigarrow} \Subset_{\mathbb{I}} \Pi^{\mathbb{I}^\natural}_{\rightsquigarrow}$.

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 causal-fairness analysis of a neural-network model M with respect to its

Algorithm 2 : Our Analysis Based on Activation Patterns

1: **function** FORWARD(M, A, I)
2: $\quad$ a, p $\leftarrow$ $\text{ASSUME}_\text{A}[\![\text{I}]\!](\text{NEW}_\text{A})$, ϵ
3: $\quad$ **for** $i \leftarrow 1$ **up to** N **do**
4: $\quad\quad$ **for** $j \leftarrow 0$ **up to** $|\text{L}_i|$ **do**
5: $\quad\quad\quad$ a, p $\leftarrow$ $\overrightarrow{\text{RELU}}^{\text{p}}_{\text{A}}[\![\text{x}_{i,j}]\!](\overrightarrow{\text{ASSIGN}}_{\text{A}}[\![\text{x}_{i,j}]\!]\text{a})$
6: $\quad$ **return** a, p
7: **function** BACKWARD(M, A, O, p)
8: $\quad$ a $\leftarrow$ $\text{OUTCOME}_\text{A}[\![\text{O}]\!](\text{NEW}_\text{A})$
9: $\quad$ **for** $i \leftarrow \text{N} - 1$ **down to** 0 **do**
10: $\quad\quad$ **for** $j \leftarrow |\text{L}_i|$ **down to** 0 **do**
11: $\quad\quad\quad$ a $\leftarrow$ $\overleftarrow{\text{ASSIGN}}_{\text{A}}[\![\text{x}_{i,j}]\!](\overleftarrow{\text{RELU}}^{\text{p}}_{\text{A}}[\![\text{x}_{i,j}]\!]\text{a})$
12: $\quad$ **return** a
13: **function** ANALYZE(M, K, Y, A_1, A_2, L, U)
14: $\quad$ F, E, C $\leftarrow \dot{\emptyset}, \dot{\emptyset}, \emptyset$ $\quad$ ▷ F: feasible, E: excluded, C: completed
15: $\quad$ $\mathbb{I} \leftarrow \{Y\}$
16: $\quad$ **while** $\mathbb{I} \neq \emptyset$ **do** $\quad$ ▷ perfectly parallelizable
17: $\quad\quad$ I $\leftarrow$ $\mathbb{I}$.GET()
18: $\quad\quad$ a, p $\leftarrow$ FORWARD(M, A_1, I)
19: $\quad\quad$ **if** UNIQUELY-CLASSIFIED(a) **then** $\quad$ ▷ I is already fair
20: $\quad\quad\quad$ C $\leftarrow$ C $\cup$ {I}
21: $\quad\quad$ **else if** $|\text{M}| - |\text{p}| \leq \text{U}$ **then** $\quad$ ▷ I is feasible
22: $\quad\quad\quad$ F $\leftarrow$ F $\uplus$ {p $\mapsto$ I}
23: $\quad\quad$ **else if** $|\text{I}| \leq \text{L}$ **then** $\quad$ ▷ I is excluded
24: $\quad\quad\quad$ E $\leftarrow$ E $\uplus$ {p $\mapsto$ I}
25: $\quad\quad$ **else** $\quad$ ▷ I must be partitioned further
26: $\quad\quad\quad$ $\mathbb{I} \leftarrow \mathbb{I} \cup \text{PARTITION}_{\overline{\text{K}}}(\text{I})$
27: $\quad$ B $\leftarrow \emptyset$ $\quad$ ▷ B: biased
28: $\quad$ **for all** p, $\mathbb{I} \in$ F **do** $\quad$ ▷ perfectly parallelizable
29: $\quad\quad$ O $\leftarrow \dot{\emptyset}$
30: $\quad\quad$ **for** $j \leftarrow 0$ **up to** $|\text{L}_\text{N}|$ **do**
31: $\quad\quad\quad$ a $\leftarrow$ BACKWARD(M, A_2, $\text{x}_{\text{N},j}$, p)
32: $\quad\quad\quad$ O $\leftarrow$ O $\cup \left\{\text{x}_{\text{N},j} \mapsto \text{a}\right\}$
33: $\quad\quad$ **for all** I $\in \mathbb{I}$ **do**
34: $\quad\quad\quad$ O' $\leftarrow \dot{\emptyset}$
35: $\quad\quad\quad$ **for all** o, a $\in$ O **do**
36: $\quad\quad\quad\quad$ O' $\leftarrow$ O' $\cup \left\{\text{o} \mapsto (\text{ASSUME}_{\text{A}_2}[\![\text{I}]\!]\text{a})_{|\overline{\text{K}}}\right\}$
37: $\quad\quad\quad$ B $\leftarrow$ B $\cup$ CHECK(O')
38: $\quad\quad\quad$ C $\leftarrow$ C $\cup$ {I}
39: $\quad$ **return** C, B = $\emptyset$, B, E $\quad$ ▷ fair: B = $\emptyset$, maybe biased: B $\neq \emptyset$

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 $2^{|\text{M}|}$ 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 $\text{p}_{i,j}$ represents the activation status of the ReLU operation used to compute the value of hidden node $\text{x}_{i,j}$. If $\text{p}_{i,j}$ is $\text{x}_{i,j}$, the ReLU is always active, otherwise the ReLU is always inactive and $\text{p}_{i,j}$ is $\overline{\text{x}_{i,j}}$.

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 $\overrightarrow{\text{RELU}}^{\text{p}}_{\text{A}}[\![\text{x}_{i,j}]\!]$ 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 causal 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.

Otherwise, I needs to be partitioned further, with respect to $\overline{\text{K}}$ (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 $\max \text{X}_\text{N} = \text{x}$, 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 $\overleftarrow{\text{RELU}}^{\text{p}}_{\text{A}}[\![\text{x}_{i,j}]\!]$ uses p to choose which path(s) to explore at each ReLU operation, i.e., only the active (resp. inactive) path if $\text{x}_{i,j}$ (resp. $\overline{\text{x}_{i,j}}$) appears in p, or both if the activation status of the ReLU corresponding to the hidden node $\text{x}_{i,j}$ 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 causal 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 neural-network model M satisfies causal fairness with respect to K and Y over C.

Theorem 9.1. If function $\textsc{analyze}(\mathrm{M}, \mathrm{K}, Y, \mathrm{A}_1, \mathrm{A}_2, \mathrm{L}, \mathrm{U})$ in Algorithm 2 returns C, $\textsc{true}$, $\emptyset$, then M satisfies $\mathcal{F}_{\mathrm{K}}[Y]$ over the input-space fraction C.

Proof (Sketch). $\textsc{analyze}(\mathrm{M}, \mathrm{K}, Y, \mathrm{A}_1, \mathrm{A}_2, \mathrm{L}, \mathrm{U})$ 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 $\mathrm{x}_{\mathrm{N},j}$ (cf. Line 31). Thus, it actually computes an over-approximation $\{\!|\mathrm{M}|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$ of the parallel semantics $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow}$, i.e., $\{\!|\mathrm{M}|\!\}^{\mathbb{I}}_{\rightsquigarrow} \mathrel{\dot\subseteq} \{\!|\mathrm{M}|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$. Thus, if $\{\!|\mathrm{M}|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$ satisfies $\mathcal{F}_{\mathrm{K}}[Y]$, i.e., $\forall \mathrm{I} \in \mathbb{I} \colon \forall A, B \in \{\!|\mathrm{M}|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow} \colon (A^{\mathrm{I}}_{\omega} \neq B^{\mathrm{I}}_{\omega} \Rightarrow A^{\mathrm{I}}_{0}|_{\overline{\mathrm{K}}} \cap B^{\mathrm{I}}_{0}|_{\overline{\mathrm{K}}} = \emptyset)$ (according to Lemma 8.4, cf. Lines 33-37), then by transitivity we can conclude that also $\{\!|\mathrm{M}|\!\}^{\mathbb{I}^\natural}_{\rightsquigarrow}$ necessarily satisfies $\mathcal{F}_{\mathrm{K}}[Y]$. □

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 [17, 20, 25, 47, 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 [13, 45, 57].

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 $\overrightarrow{\textsc{relu}}^{\mathrm{p}}_{\mathrm{A}}[\![\mathrm{x}_{i,j}]\!]$ (cf. Line 5) and $\overleftarrow{\textsc{relu}}^{\mathrm{p}}_{\mathrm{A}}[\![\mathrm{x}_{i,j}]\!]$ (cf. Line 11), which would have to be replaced with the transfer functions corresponding to the considered activation function. Piecewise-linear activation functions, like $\textsc{Leaky ReLU}(x) = \max(x, k \cdot x)$ or $\textsc{Hard TanH}(x) = \max(-1, \min(x, 1))$, can be treated analogously to ReLUs. Other functions, e.g., $\textsc{Sigmoid}(x) = \frac{1}{1+e^{-x}}$, can be soundly over-approximated [57].

10 Implementation

We implemented our causal-fairness analysis described in the previous section in a tool called LIBRA. The implementation is written in PYTHON and is open-source[2].

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.

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

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 (i.e., BOXES in the following) or a combination of boxes and symbolic constant propagation [40, 46] (i.e., BOXES+SYMBOLIC in the following). As previously mentioned, we use disjunctive polyhedra for the backward analysis. All abstract domains are built on top of the APRON abstract-domain library [34].

Parallelization. Both 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. To obtain the latter, for simplicity, we just use the size of a box wrapped around each biased region. More precise but also costlier solutions exist [6].

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 analysis budget affect the scalability-vs-precision tradeoff?
- **RQ5:** Can our analysis effectively leverage multiple CPUs?

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 [44] or used by related techniques (e.g., [1–3, 7, 21, 23, 62, 63]).

We pre-processed these datasets both to make them fair with respect to a certain sensitive input feature as well as to seed bias. We describe how we seeded bias in each particular dataset later in this section.

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

CREDIT	FAIR DATA						BIASED DATA					
	U	BIAS	|C|	|F|		TIME	U	BIAS	|C|	|F|		TIME
≤ 1000	12	0.33%	138	32	32	52s	12	0.79%	196	56	56	7m 50s
	12	0.17%	165	23	23	4m 16s	12	0.31%	141	26	26	1m 11s
	12	0.09%	140	10	10	29s	12	0.90%	198	59	59	14m 27s
	12	0.15%	159	22	22	2m 3s	12	0.42%	189	37	37	3m 42s
	12	0.23%	157	25	25	1m 56s	12	0.00%	130	13	13	22s
	12	0.30%	166	32	32	1m 11s	12	0.41%	176	37	37	2m 56s
	12	0.20%	135	25	25	1m 4s	12	0.48%	181	39	39	1m 20s
	12	0.16%	168	14	14	17s	12	0.09%	196	10	10	1m 42s
MIN		0.09%				17s		0.00%				22s
MEDIAN		0.19%				1m 8s		0.41%				2m 19s
MAX		0.33%				4m 16s		0.90%				14m 27s
> 1000	12	12, 20%	202	101	101	32m 9s	15	27.59%	310	264	265	7h 21m 1s
	15	7.43%	215	103	103	2h 51m 10s	12	30.77%	252	182	184	42m 56s
	12	2.21%	155	22	22	1m 23s	16	33.19%	273	233	236	12h 50m 6s
	12	4.29%	185	39	39	10m 51s	12	16.45%	236	189	189	1h 50m 57s
	12	9.73%	172	84	84	23m 13s	12	0.00%	165	5	5	17s
	12	14.96%	234	173	176	4h 25m 47s	12	17.24%	246	171	172	1h 16m 31s
	12	6.00%	199	67	67	27m 17s	16	19.23%	206	138	138	3h 39m 57s
	12	4.61%	200	48	48	23m 37s	12	4.52%	224	94	94	1h 5m 13s
MIN		2.21%				1m 23s		0.00%				17s
MEDIAN		6.72%				25m 27s		18.24%				1h 33m 44s
MAX		14.96%				4h 25m 47s		33.19%				12h 50m 6s

Our methodology for making the data fair was common across datasets. In particular, 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 causally 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.

We define the *causal-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%.

11.2 Setup

Since neural-network training is non-deterministic, we typically train eight neural networks (with four hidden layers with five nodes) on each dataset, unless stated otherwise.

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). All datasets and models we used are also open-source as part of LIBRA.

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.

For this experiment, we used the German Credit Data[3]. This dataset classifies creditworthiness into two categories, "good" and "bad". An input feature is age, which we consider 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% causal-unfairness score of the dataset. The median classification accuracy of the models trained on fair and biased data was 71% and 65%, respectively.

Table 1 shows the analysis results for all models. For the forward pre-analysis, we used the BOXES+SYMBOLIC domain. 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 that did not cause this issue. Column U shows the chosen upper bound for each model. Column |C| shows

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

Table 2. Queries on Neural Networks Trained on Fair and Race-Biased Data (ProPublica's COMPAS Data) — Full Table

QUERY	FAIR DATA						BIASED DATA					
	U	BIAS	$\lvert C\rvert$	$\lvert F\rvert$		TIME	U	BIAS	$\lvert C\rvert$	$\lvert F\rvert$		TIME
AGE < 25 RACE BIAS?	10	0.23%	52	22	23	1h 49m 12s	10	0.83%	26	24	24	2h 32m 53s
	10	0.83%	44	18	18	26m 23s	10	8.79%	60	33	34	19m 2s
	10	0.22%	42	16	16	44m 59s	10	1.15%	24	14	14	16m 51s
	10	0.24%	175	33	33	45m 36s	10	0.42%	17	16	16	8m 27s
	10	0.30%	151	63	63	51m 6s	10	0.12%	32	14	14	22m 47s
	10	0.33%	67	19	19	54m 45s	10	1.59%	33	27	27	1h 59m 57s
	10	1.19%	27	24	24	12m 33s	10	3.34%	162	122	122	39m 38s
	10	2.46%	17	16	16	16m 35s	10	0.18%	17	16	16	11m 36s
MIN		0.22%				12m 33s		0.12%				8m 27s
MEDIAN		0.32%				45m 17s		0.99%				20m 54s
MAX		2.46%				1h 49m 12s		8.79%				2h 32m 53s
MALE AGE BIAS?	10	0.00%	335	147	335	38m 58s	10	0.00%	343	164	343	1h 32m 10s
	10	0.00%	306	124	191	44m 33s	10	0.00%	730	265	730	1h 10m 7s
	10	0.00%	258	75	258	33m 38s	10	0.00%	268	119	268	25m 23s
	10	0.00%	1443	211	395	45m 43s	10	0.00%	103	73	103	45m 55s
	10	0.00%	1298	414	714	51m 44s	10	0.00%	408	131	263	32m 14s
	10	0.00%	517	266	517	1h 39m 27s	10	0.00%	305	123	279	1h 55m 16s
	10	0.00%	504	138	353	17m 28s	10	0.00%	681	319	414	35m 15s
	10	0.00%	403	222	381	46m 16s	10	0.00%	391	280	391	57m 36s
MIN		0.00%				17m 28s		0.00%				25m 23s
MEDIAN		0.00%				45m 8s		0.00%				51m 45s
MAX		0.00%				1h 39m 27s		0.00%				1h 55m 16s
CAUCASIAN PRIORS BIAS?	12	2.18%	46	39	39	8h 20m 48s	15	2.92%	44	43	43	9h 15m 19s
	12	3.66%	68	57	57	2h 1m 43s	15	6.98%	45	41	41	1h 24m 13s
	15	2.73%	46	43	43	3h 45m 15s	12	4.43%	45	39	39	31m 38s
	19	2.19%	47	46	46	28h 48m 46s	12	3.40%	42	41	41	36m 10s
	19	3.17%	212	212	212	156h 56m 42s	15	3.09%	39	38	38	2h 34m 28s
	12	2.45%	57	43	43	6h 21m 40s	15	5.79%	54	52	53	4h 35m 30s
	15	3.94%	48	45	45	3h 29m 22s	19	5.10%	49	48	48	52h 11m 13s
	15	5.36%	47	46	46	7h 3m 25s	17	3.99%	46	44	44	13h 1m 5s
MIN		2.18%				2h 1m 43s		2.92%				31m 38s
MEDIAN		2.95%				6h 42m 32s		4.21%				3h 34m 59s
MAX		5.36%				156h 56m 42s		6.98%				52h 11m 13s

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). Columns BIAS and TIME show the detected bias (in percentage of the entire input space) and the analysis running time, respectively. In particular, the table 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. We also report minimum, median, and maximum values for both bias and analysis running time.

We observe that, for models trained on fair data, age bias for credit amounts $\leq$ 1 000 is very small in comparison to larger amounts. This is because small credit amounts correspond to a mere 4% of the input space. When only considering the input space of amounts $\leq$ 1 000, the median bias is 0.19% / 4% = 4.75%, whereas when only considering larger amounts, the median bias is 6.72% / 96% = 7%. This shows 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).

For the models trained on biased data, the analysis finds significantly more bias for larger credit amounts in comparison to the models trained on the fair dataset. As expected, it also finds similar bias across the different models for smaller credit amounts. This demonstrates that *our approach is able to effectively detect seeded bias.* It is interesting to point out that the model on the fourth row of the table does not pick up the bias introduced in the dataset, which of course only corresponds to a small sample of the input space.

RQ2: Answering specific bias queries. To further evaluate the precision of our approach, we created queries concerning

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

|M|	U	BOXES					BOXES+SYMBOLIC				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
10 ○●	4	86.81%	1447	230	1142	28m 2s	93.61%	1110	227	699	16m 57s
	6	99.51%	786	255	739	59m 15s	99.93%	581	231	450	39m 16s
	8	100.00%	152	118	143	4h 55m 57s	100.00%	174	133	146	3h 24m 42s
	10	100.00%	1	1	1	56m 18s	100.00%	1	1	1	56m 22s
12 △▲	4	49.76%	712	26	334	12m 26s	72.22%	1176	39	558	21m 48s
	6	72.67%	1191	60	926	2h 2m 57s	98.54%	331	36	193	20m 38s
	8	98.68%	342	56	284	1h 38m 31s	98.78%	323	41	190	41m 0s
	10	99.06%	313	65	260	1h 25m 42s	99.06%	307	47	182	1h 12m 5s
20 ◇◆	4	22.01%	625	24	39	2m 6s	44.06%	845	48	92	14m 26s
	6	45.24%	1111	123	260	21m 30s	60.03%	895	166	406	42m 22s
	8	64.17%	1108	299	795	2h 46m 48s	74.10%	1122	305	779	2h 8m 25s
	10	85.87%	1376	387	1329	>13h	89.24%	1425	376	1150	>13h
40 □■	4	0.00%	0	0	0	1m 5s	0.69%	20	1	1	3m 33s
	6	0.00%	0	0	0	1m 5s	3.19%	92	5	5	40m 40s
	8	0.14%	4	1	2	13m 58s	9.48%	258	28	28	2h 40m 43s
	10	0.63%	18	12	13	1h 48m 43s	19.62%	544	74	75	12h 25m 43s
45 ⬠⬟	4	0.00%	0	0	0	1m 9s	27.26%	697	25	49	8m 24s
	6	0.83%	24	3	22	3m 44s	39.65%	771	84	147	24m 1s
	8	9.41%	270	58	234	22m 49s	47.47%	712	141	238	55m 30s
	10	18.68%	522	150	488	1h 39m 33s	49.62%	651	168	283	3h 24m 15s

bias within specific groups of people, each corresponding to a subset of the entire input space.

We used the COMPAS dataset[4] from ProPublica for this experiment. The data assigns a recidivism-risk score (high, medium, and low) indicating whether criminals are likely 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% causal-unfairness score of the dataset. The median classification accuracy of the models trained on fair and biased data was 55% and 56%, respectively.

To analyze these models, we used the BOXES+SYMBOLIC domain for the forward pre-analysis, a lower bound L of 0, and an upper bound U between 10 and 19. Table 2 summarizes the results of our analysis (i.e., all columns are shown as in Table 1) for three queries:

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

The analysis is able to complete 100% of the input space for each query. For Q_A, the analysis detects only a small percentage of bias in the fair models, but as expected, the *bias is found to be significantly higher (ca. 3X) for the biased models*. In contrast, for Q_B, the analysis is able to *verify that there is no bias for males* in both sets of models. Finally, for Q_C, *the analysis detects significant bias with respect to the number of priors*. Note that the bias percentages are always with respect to the entire input space; not just with respect to Caucasians (for Q_c) representing 1/6 of the input space. Also, note that we did not introduce any bias with respect to the number of priors, so this bias is intended and present in the original data. As one would expect, recidivism risk differs for different numbers of priors. Overall, these results *demonstrate the effectiveness of our analysis in answering specific bias queries by detecting bias or verifying its absence.*

RQ3: Effect of model structure on scalability. This research question evaluates the effect of the model structure on the scalability of our analysis. To answer it, we trained models on the Adult Census Data[5] 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. We trained all models on a fair dataset with respect to gender and ensured that each model reached a minimum classification accuracy of 78%.

Table 3 summarizes the results for all models. The first column shows the total number of hidden nodes and introduces the marker symbols used in the scatter plot of Figure 3 (the left symbol refers to the BOXES domain, whereas the right one refers to the BOXES+SYMBOLIC domain used by the forward pre-analysis). The models use the following number

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

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

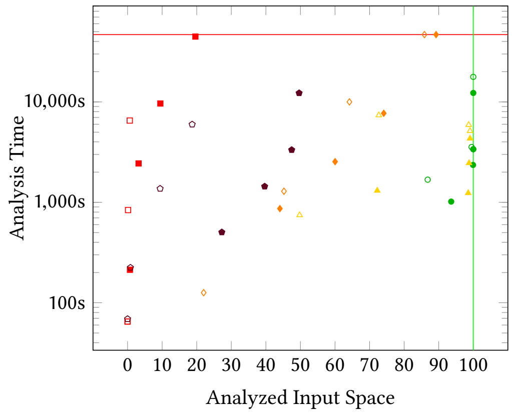

Figure 3. Comparison of Different Model Structures (Adult Census Data)

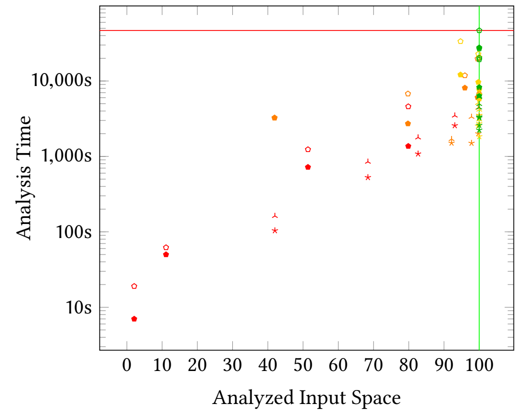

Figure 4. Comparison of Different Analysis Configurations (Japanese Credit Screening)

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 upper bound chosen for each model, while the INPUT and TIME columns show the input-space coverage (i.e., the percentage of the input space that was completed by the analysis) and the running time. As before, column |C| shows the number of completed input space partitions, and |F| shows the number of abstract activation patterns (left) and feasible input partitions (right) explored by the backward analysis. We used a lower bound L of 0.5 and a total-time limit of 13h.

The scatter plot of Figure 3 visualizes the input coverage and analysis running time. Overall, *coverage decreases for more complex model structures* and *the more precise (but expensive) BOXES+SYMBOLIC domain results in a significant coverage boost, especially for more complex structures*.

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, as we will explain in RQ4, such a change tends to help the forward pre-analysis in already proving certain partitions fair. This results in decreasing the number of partitions that the expensive backward analysis needs to analyze as well as the overall running time.

RQ4: 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[6] dataset, which we made fair with respect to gender. Our model had a classification accuracy of 86%.

Table 4 shows the results for different analysis configurations and domains of the forward pre-analysis. Note that the symbol next to each domain introduces the marker used in the scatter plot of Figure 4, which visualizes the coverage and running time.

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

Overall, we observe that *the more precise BOXES+SYMBOLIC domain boosts input coverage (most noticeably for configurations with a larger* L*)*. Surprisingly, this additional precision does not always result in longer running times. In fact, *for long-running analyses, BOXES+SYMBOLIC typically reduces the running time.* This is because the classification within more partitions is proved fair already by the pre-analysis without requiring the backward analysis.

As expected, *a larger* U *or a smaller* L *increase precision.* Increasing U or L typically reduces the number of partitions. 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.

RQ5: Leveraging multiple CPU cores. To evaluate the effect of parallelizing the analysis using multiple cores, we re-ran the analyses of RQ4 on 4 CPU cores instead of 12. Table 5 shows these results. *For the BOXES domain, we observe a significant increase in running time for 4 cores*, especially for configurations that achieve high coverage. For instance, the running time increases by a factor of 3.4 for L = 0 and U = 10. On the other hand, *for the BOXES+SYMBOLIC domain, the running time with 4 cores typically increases less drastically.* This is again explained by the increased precision of the forward analysis; fewer partitions require a backward pass, where parallelization is most effective.

Finally, Table 6 shows 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.

Table 4. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 12 CPUs

L	U	BOXES					BOXES+SYMBOLIC				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
0.5	4	2.08%	6	0	0	7s	42.01%	71	13	23	1m 44s
	6	11.11%	28	7	7	50s	68.40%	96	34	43	8m 47s
	8	51.39%	98	55	87	12m 1s	82.64%	103	72	80	18m 3s
	10	79.86%	83	67	83	22m 48s	93.06%	91	79	82	42m 49s
0.25	4	41.91%	1225	41	415	54m 8s	92.14%	955	120	407	25m 8s
	6	79.77%	1470	214	957	45m 22s	97.81%	507	163	278	25m 2s
	8	95.92%	1159	476	969	2h 15m 14s	99.72%	389	220	294	33m 26s
	10	99.54%	437	294	434	1h 40m 52s	99.98%	174	143	162	58m 35s
0.125	4	94.68%	16348	671	9191	3h 22m 2s	99.64%	2167	194	727	42m 22s
	6	99.74%	6219	951	3955	2h 41m 2s	99.99%	1115	264	537	45m 31s
	8	99.98%	1775	786	1450	2h 8m 22s	100.00%	293	192	233	30m 38s
	10	100.00%	399	287	399	1h 36m 48s	100.00%	155	137	145	58m 36s
0	4	94.68%	47380	1133	18005	7h 41m 41s	99.64%	3780	196	730	43m 16s
	6	99.74%	5369	938	3414	2h 17m 26s	99.99%	783	204	349	54m 21s
	8	99.98%	1531	751	1273	1h 48m 38s	100.00%	360	217	275	37m 23s
	10	100.00%	512	354	506	1h 47m 54s	100.00%	163	142	152	56m 1s

Table 5. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 4 CPUs

L	U	BOXES					BOXES+SYMBOLIC				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
0.5	4	2.08%	6	0	0	19s	42.01%	79	17	29	2m 42s
	6	11.11%	30	6	7	1m 2s	68.40%	124	40	55	14m 8s
	8	51.39%	90	57	85	20m 37s	82.64%	137	82	93	29m 40s
	10	79.86%	128	108	123	1h 16m 28s	93.06%	108	86	95	57m 54s
0.25	4	41.91%	1159	42	379	54m 16s	92.14%	1010	120	364	28m 14s
	6	79.77%	1456	210	969	1h 53m 1s	97.81%	776	216	429	55m 41s
	8	95.92%	926	407	804	3h 17m 18s	99.72%	296	192	234	1h 13m 32s
	10	99.54%	519	342	506	5h 28m 27s	99.98%	204	156	180	2h 0m 17s
0.125	4	94.68%	15993	681	8739	9h 19m 36s	99.64%	3470	231	1120	1h 24m 57s
	6	99.74%	4951	851	3257	6h 19m 56s	99.99%	786	208	371	51m 24s
	8	99.98%	1548	745	1317	5h 43m 12s	100.00%	303	189	232	1h 9m 21s
	10	100.00%	506	344	500	5h 42m 11s	100.00%	168	138	157	1h 56m 41s
0	4	94.68%	36165	1076	14877	>13h	99.64%	6700	235	1245	1h 41m 58s
	6	99.74%	5802	955	3592	7h 27m 41s	99.99%	1156	264	537	1h 14m 6s
	8	99.98%	1552	751	1297	5h 21m 11s	100.00%	360	217	275	1h 22m 23s
	10	100.00%	528	373	521	5h 37m 28s	100.00%	199	152	179	1h 44m 3s

Testing and verifying fairness. Galhotra et al. [23] proposed an approach, Themis, that allows efficient fairness testing of software. Udeshi et al. [63] designed an automated and directed testing technique to generate discriminatory inputs for machine-learning models. Tramer et al. [62] introduced the unwarranted-associations framework and instantiated it in FairTest. In contrast, our technique provides formal fairness guarantees.

Bastani et al. [7] used adaptive concentration inequalities to design a scalable technique for verifying fairness of machine-learning models. Albarghouthi et al. [2] encoded fairness problems as probabilistic program properties and developed an SMT-based technique for verifying fairness of decision-making programs. For certain biased decision-making programs, the program repair technique proposed by Albarghouthi et al. [1] can be used to repair their bias. Albarghouthi et al. [3] further introduced fairness-aware programming, where programmers can specify fairness properties in their code for runtime checking. 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.

Table 6. Comparison of Different Analysis Configurations (Japanese Credit Screening) — 24 vCPUs

L	U	BOXES					BOXES+SYMBOLIC				
		INPUT	|C|	|F|		TIME	INPUT	|C|	|F|		TIME
0.5	4	2.08%	6	0	0	6s	42.01%	71	13	23	1m 33s
	6	11.11%	30	6	7	44s	68.40%	102	40	53	4m 43s
	8	51.39%	125	65	101	16m 44s	82.64%	102	68	77	21m 19s
	10	79.86%	101	81	98	29m 43s	93.06%	104	84	92	45m 6s
0.25	4	41.91%	1211	41	381	38m 40s	92.14%	936	126	349	38m 4s
	6	79.77%	1423	207	944	43m 21s	97.81%	541	178	287	23m 38s
	8	95.92%	978	432	835	1h 12m 1s	99.72%	362	210	284	52m 32s
	10	99.54%	409	295	403	1h 37m 51s	99.98%	197	150	177	1h 10m 28s
0.125	4	94.68%	21388	678	11490	4h 46m 44s	99.64%	3619	227	1199	59m 56s
	6	99.74%	6124	961	3956	2h 32m 2s	99.99%	910	232	433	33m 7s
	8	99.98%	1513	729	1267	1h 43m 53s	100.00%	348	203	266	49m 59s
	10	100.00%	596	392	578	2h 43m 43s	100.00%	176	137	156	1h 32m 9s
0	4	94.68%	48195	1119	18148	8h 29m 51s	99.64%	5171	226	1019	1h 14m 7s
	6	99.74%	7484	1093	4629	3h 10m 35s	99.99%	837	221	388	31m 36s
	8	99.98%	1439	728	1235	1h 41m 27s	100.00%	319	198	248	38m 39s
	10	100.00%	483	353	472	1h 39m 44s	100.00%	160	138	150	1h 1m 23s

Robustness of deep neural networks. Robustness is a desirable property for traditional software [12, 30, 43], 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 [59] and can even be easily fooled [51]. Subtle imperceptible perturbations of inputs, known as adversarial examples, can change their prediction results. Various algorithms [11, 28, 42, 60, 66] have been proposed that can effectively find adversarial examples. Research on developing defense mechanisms against adversarial examples [4, 9–11, 15, 22, 28, 32, 48, 49] is also active. Causal fairness of neural networks 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. [54] proposed the first whitebox framework for testing such systems. They used neuron coverage to measure the adequacy of test inputs. Sun et al. [58] presented the first concolic-testing [26, 56] approach for neural networks. Tian et al. [61] and Zhang et al. [67] proposed frameworks for testing autonomous driving systems. Gopinath et al. [29] used symbolic execution [14, 37]. Odena et al. [53] were the first to develop coverage-guided fuzzing for neural networks. Zhang et al. [66] 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 [55] applied abstract interpretation to verify a neural network with six neurons. Recent work [24, 33, 35, 57, 65] significantly improves scalability. Huang et al.[33] proposed a framework that can verify local robustness of neural networks based on SMT techniques [5]. Katz et al. [35] developed an efficient SMT solver for neural networks with ReLU activation functions. Gehr et al. [24] traded precision for scalability and proposed a sound abstract interpreter that can prove local robustness of realistic deep neural networks. Singh et al. [57] proposed a new abstract domain for certifying robustness of neural networks. Their abstract domain could also be used in our setting to certify fairness properties. Wang et al. [65] are the first to use symbolic interval arithmetic to prove security properties of neural networks.

13 Conclusion and Future Work

We have presented an automated, perfectly parallel analysis for certifying fairness of neural networks. The analysis is configurable to support a wide range of use cases throughout the development lifecycle of neural networks: ranging from short sanity checks during development to formal fairness audits before deployments.

In future work, we plan to extend our technique in various ways. For instance, by automatically tuning parameters (such as the upper bound U) during the analysis or by feeding analysis results to other tools. Such tools may be used to provide probabilistic fairness guarantees for partitions that could not be certified or repair networks by eliminating bias.

References

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

[2] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *PACMPL* 1, OOPSLA (2017), 80:1–80:30. https://doi.org/10.1145/3133904
[3] Aws Albarghouthi and Samuel Vinitsky. 2019. Fairness-Aware Programming. In *FAT**. 211–219. https://doi.org/10.1145/3287560.3287588
[4] 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.
[5] Clark W. Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*. Springer, 305–343.
[6] 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
[7] Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. 2019. Probabilistic verification of fairness properties via concentration. *PACMPL* 3, OOPSLA (2019), 118:1–118:27.
[8] Joy Buolamwini and Timnit Gebru. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *FAT (PMLR)*, Vol. 81. PMLR, 77–91.
[9] Nicholas Carlini and David A. Wagner. 2016. Defensive Distillation is Not Robust to Adversarial Examples. *CoRR* abs/1607.04311 (2016).
[10] Nicholas Carlini and David A. Wagner. 2017. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *AISec@CCS*. ACM, 3–14.
[11] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *S&P*. IEEE Computer Society, 39–57.
[12] 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
[13] 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
[14] Lori A. Clarke. 1976. A System to Generate Test Data and Symbolically Execute Programs. *TSE* 2 (1976), 215–222. Issue 3.
[15] Cory Cornelius. 2019. The Efficacy of SHIELD under Different Threat Models. *CoRR* abs/1902.00541 (2019).
[16] 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
[17] Patrick Cousot and Radhia Cousot. 1976. Static Determination of Dynamic Properties of Programs. In *Second International Symposium on Programming*. 106–130.
[18] 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
[19] Patrick Cousot and Radhia Cousot. 1979. Systematic Design of Program Analysis Frameworks. In *POPL*. 269–282. https://doi.org/10.1145/567752.567778
[20] 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
[21] 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
[22] Logan Engstrom, Andrew Ilyas, and Anish Athalye. 2018. Evaluating and Understanding the Robustness of Adversarial Logit Pairing. *CoRR* abs/1807.10272 (2018).
[23] 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
[24] 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
[25] 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
[26] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed Automated Random Testing. In *PLDI*. ACM, 213–223.
[27] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press.
[28] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*. http://arxiv.org/abs/1412.6572
[29] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S. Pasareanu, and Sarfraz Khurshid. 2018. Symbolic Execution for Deep Neural Networks. *CoRR* abs/1807.10439 (2018).
[30] 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
[31] 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
[32] 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
[33] 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
[34] 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
[35] 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
[36] 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.
[37] James C. King. 1976. Symbolic Execution and Program Testing. *CACM* 19 (1976), 385–394. Issue 7.
[38] Matt Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual Fairness. In *NIPS*. 4069–4079.
[39] 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.
[40] 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
[41] Kristian Lum and William Isaac. 2016. To Predict and Serve? *Significance* 13 (2016), 14–19. Issue 5.
[42] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR*. OpenReview.net.
[43] Rupak Majumdar and Indranil Saha. 2009. Symbolic Robustness Analysis. In *RTSS*. 355–363. https://doi.org/10.1109/RTSS.2009.17

[44] 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).

[45] 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

[46] Antoine Miné. 2006. Symbolic Methods to Enhance the Precision of Numerical Abstract Domains. In *VMCAI*. 348–363. https://doi.org/10.1007/11609773_23

[47] Antoine Miné. 2006. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation* 19, 1 (2006), 31–100. https://doi.org/10.1007/s10990-006-8609-1

[48] Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML*. 3575–3583.

[49] Matthew Mirman, Gagandeep Singh, and Martin T. Vechev. 2019. A Provable Defense for Deep Residual Networks. *CoRR* abs/1903.12519 (2019).

[50] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*. 807–814.

[51] 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

[52] 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.

[53] 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.

[54] 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

[55] 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

[56] Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: A Concolic Unit Testing Engine for C. In *ESEC/FSE*. ACM, 263–272.

[57] 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

[58] 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.

[59] 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

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

[61] 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.

[62] 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.

[63] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated directed fairness testing. In *ASE*. ACM, 98–108.

[64] 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

[65] 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.

[66] Fuyuan Zhang, Sankalan Pal Chowdhury, and Maria Christakis. 2019. DeepSearch: Simple and Effective Blackbox Fuzzing of Deep Neural Networks. *CoRR* abs/1910.06296 (2019).

[67] 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.