Quantitative Input Usage Static Analysis

DENIS MAZZUCATO, MARCO CAMPION, and CATERINA URBAN, INRIA & ENS | PSL, France

Data science is increasingly being used to assist decision-making even in a variety of high-stakes and safety-critical domains, such as finance, transportation, and healthcare. However, programming errors within these domains pose significant risks. In fact, errors in data science software often produce plausible yet erroneous results, without giving a clear indication of failure. One potential source of such errors is the unexpected impact of certain inputs with respect to the program result. To address this issue, we propose a novel quantitative framework for determining the impact of inputs on the program computations. Building on this framework, we introduce an abstract interpretation-based backward static analysis, parametric in the definition of impact. This analysis computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program utilizes them. We implement a proof-of-concept static analyzer which demonstrates the practical applicability of our framework in the context of neural network analysis. An empirical comparison against stochastic feature importance metrics shows that our approach provides information on the usage of program inputs that aligns closely with the impact definition of interest, offering a new mathematically-proven basis to enhance confidence and reliability in data science software.

Additional Key Words and Phrases: Quantitative Verification, Abstract Interpretation, Data Science

1 INTRODUCTION

Nowadays, data science plays a crucial role by helping organizations and individuals make critical decisions based on data-driven insights in a variety of fields, including healthcare, finance, and avionics (e.g., [Kohli and Chadha 2020; Yang et al. 2019]). Therefore, disastrous outcomes may result from programming errors in these safety-critical settings. As our reliance on data science grows, so does our vulnerability to errors, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input variable of an application has an unexpected impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff [2010] article "Growth in a Time of Debt", which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [Herndon et al. 2014]. Notably, one of the several programming errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, with an excessive effect on the article conclusion. This flaw indicates a key methodological problem that compromised the authors' claims. In general, data science applications, which involve long pipeline of layers that filter, merge, and manipulate data, are prone to programming errors that cause some input variables to be more, or less, influent than what the application expects. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables for data science software.

In this direction, Barowy et al. [2014] proposed a stochastic approach specific for spreadsheet applications. Such approach is able to estimate the impact of input cells. However, the lack of mathematical guarantees precludes the employment of such technique in safety-critical contexts. On the other hand, existing formal methods-based approached only target properties about input data usage, only addressing whether an input variable is used or not [Urban and Müller 2018].

In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a program. This framework can be used to identify variables that have a disproportionate impact. Such knowledge could either certify intended behavior or reveal potential flaws, by matching the developers' intuition on the expected impact of their input with the actual result of the quantitative

Authors' address: Denis Mazzucato, denis.mazzucato@inria.fr; Marco Campion, marco.campion@inria.fr; Caterina Urban, caterina.urban@inria.fr, INRIA & ENS | PSL, Paris, France.

study. We characterize the impact of an input variable with a notion of dependency between variables and the outcome of the program. Compared to other quantitative notions of dependency, e.g., quantitative information flow [Denning 1982; Gray 1991], the key difference is about the information we measure. Our quantitative definition quantifies the effect of the value of input variables on the program outcome while quantitative information flow counts, for example, how many of information are used to compute the result.

In general, determining how an input variable influences a program outcome depends on several factors, such as the structure of the program, the environment, the expertise of the developer, and the intuition of the researcher. Consequently, there is no single impact definition that fits all requirements. For this reason, our framework is parametric in the impact definition of choice.

Building on this framework, we introduce a backward static analysis based on abstract interpretation [Cousot and Cousot 1977] parametric in the definition of impact, which automatically infers a sound over-approximation of the impact of input variables when analyzing neural network models. We derive our static analysis by a hierarchy of consecutive abstractions discarding the unnecessary details for deciding our property of interest. This design principle ensures that our abstract semantics soundly approximates our property, by returning an upper-bound on the impact quantity. The inputs of our static analysis consist of a set of cumulative ranges representing program outputs, called output buckets. Then, the analysis backward computes the input states leading to these buckets and applies a computable implementation of the impact on the result of the backward reachability analysis. This approach allows end-users to choose the impact that best fits their needs, ensuring a more targeted and customizable analysis.

Finally, we provide an analysis prototype based on the (backward) disjunctive polyhedra domain which demonstrates the practical applicability of our approach. We compare our prototype with feature importance metrics [Breiman 2001] and show that our sound technique produces useful input usage information with respect to the specific parametric definition provided by the end-user.

Contributions. We make the following contributions:

(1) we develop a theoretical framework by abstract interpretation to quantify the impact of input variables based on input-output observations;
(2) we introduce Range, Outcomes, and Changes as three different impact instances;
(3) we implement a proof-of-concept prototype of our static analysis tailored for neural networks that implements a sound analysis for the impact definition Changes. This prototype has been developed in an open-source tool called QStat;
(4) we evaluate our prototype, specialized with the Changes impact definition, by comparing its results against two stochastic metrics from the machine learning community.

The rest of the paper is organized as follows: Section 2 provides an overview of impact quantification applied to a small example together with an intuition of our static analysis approach. Then, Section 3 presents the necessary background on program semantics. Section 4 formalizes our property of interest and provides three impact instances, followed by Section 5 explaining our concrete semantics. Afterwards, Section 6 outlines our static analysis and Section 7 presents a first prototype of such analysis. By concluding, Sections 8 and 9 discuss related and future work.

2 OVERVIEW

In this section we present an overview of our quantitative analysis using a small example.

The program depicted in Figure 1a is a prototype of an aircraft landing alarm system. The goal of the program is to inform the pilot about the level of risk associated with the landing approach. It takes two input variables, denoted as `angle` and `speed`. The input `angle` indicates the angle of alignment between the airplane and the airstrip. A value of 1 represents a good alignment, whereas a value of -4 a non-aligned angle. The input `speed`

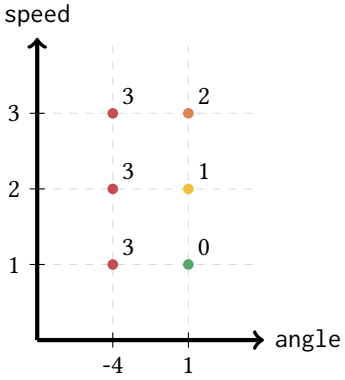
Input preconditions:

angle ∈ {−4, 1}

speed ∈ {1, 2, 3}

```
1  landing_coeff = abs(angle) + speed
2  if landing_coeff < 2 then
3    risk = 0
4  else if landing_coeff > 5 then
5    risk = 3
6  else
7    risk = floor(landing_coeff) - 2
```

(a) Program for the landing alarm system.



(b) Input space composition.

Fig. 1. Example of a program computing the landing risk during the approach of the airstrip given the alignment angle and speed of the aircraft.

denotes the speed of the aircraft, with 1, 2, 3 denoting respectively low, medium, and high speed[1]. A lower speed implies a safer approach. In our settings, an input configuration is a pair of values $\langle x, y \rangle$ assigned to the input variables at the beginning of the program execution. Specifically, the first component x corresponds to the value of `angle` and the second y to the value of `speed`.

The program computes first the landing risk coefficient, denoted as `landing_coeff`, at Line 1. This coefficient is obtained by taking the absolute value of the landing angle (accounting for both negative and positive approach angles) and adding it to the speed. Afterwards, Line 2 and 4 restrict `landing_coeff` to range between values 2 and 5. Values below 2 signify low danger, while values above 5 indicate high danger. Respectively, we assign to the output variable `risk` the value of 0 for low danger and the value of 3 for high danger. The medium degree of dangerousness is computed at Line 7 and ranges between 1 and 2, which assigns to the output variable `risk` the largest integer less than, or equal, to `landing_coeff` − 2. Note that, the possible output values of `risk` are $\{0, 1, 2, 3\}$.

Figure 1b shows the input space composition of this system, where the label near each input represents the degree of risk assigned to the corresponding input configuration. It is easy to note that a nonaligned angle of approach corresponds to a considerably higher level of risk, whereas the risk with a correct angle depends mostly on the aircraft speed.

Impact Analysis. Understanding the effects of input variables on program executions allows us to reveal potential bugs or certify intended behavior. Therefore we present a framework to quantify the impact of input variables on the outcome of a program. Intuitively, the definition of impact of a variable depends on various factors such as program structure, environment, developer expertise, and researcher intuition. Different notions of impact leverage different traits of the input-output relations in computation. As a result, it is not possible to formalize a definition that fits all requirements.

In general, all possible definitions of impact establish some relationship between variations of the input and the output values. In this section, we introduce two impact notions. The first impact definition focuses on *the size of* extreme reachable values resulting from variations in the input variable. The second exploits *the number of*

[1]We initially focus on discrete values to simplify the example and convey the concept. We plan to expand our analysis to include continuous inputs from Section 4.3

Table 1. Impact of the input variable angle using the $\textsc{Range}_{\texttt{angle}}(\mathrm{P})$ definition.

Input Configuration	Perturbed Traces	Output Values	$\textsc{Range}_{\texttt{angle}}(\mathrm{P})$
$\langle -4, 1\rangle$	$\langle -4, 1\rangle \to \langle 3\rangle, \langle 1, 1\rangle \to \langle 0\rangle$	$\{3, 0\}$	3
$\langle -4, 2\rangle$	$\langle -4, 2\rangle \to \langle 3\rangle, \langle 1, 2\rangle \to \langle 1\rangle$	$\{3, 1\}$	2
$\langle -4, 3\rangle$	$\langle -4, 3\rangle \to \langle 3\rangle, \langle 1, 3\rangle \to \langle 2\rangle$	$\{3, 2\}$	1
$\langle 1, 1\rangle$	$\langle 1, 1\rangle \to \langle 0\rangle, \langle -4, 1\rangle \to \langle 3\rangle$	$\{0, 3\}$	3
$\langle 1, 2\rangle$	$\langle 1, 2\rangle \to \langle 1\rangle, \langle -4, 2\rangle \to \langle 3\rangle$	$\{1, 3\}$	2
$\langle 1, 3\rangle$	$\langle 1, 3\rangle \to \langle 2\rangle, \langle -4, 3\rangle \to \langle 3\rangle$	$\{2, 3\}$	1

Table 2. Impact of the input variable speed using the $\textsc{Range}_{\texttt{speed}}(\mathrm{P})$ definition.

Input Configuration	Perturbed Traces	Output Values	$\textsc{Range}_{\texttt{speed}}(\mathrm{P})$
$\langle -4, 1\rangle$	$\langle -4, 1\rangle \to \langle 3\rangle, \langle -4, 2\rangle \to \langle 3\rangle, \langle -4, 3\rangle \to \langle 3\rangle$	$\{3\}$	0
...	...	...	...
$\langle 1, 1\rangle$	$\langle 1, 1\rangle \to \langle 0\rangle, \langle 1, 2\rangle \to \langle 1\rangle, \langle 1, 3\rangle \to \langle 2\rangle$	$\{0, 1, 2\}$	2
...	...	...	...

reachable outcomes. Both definitions give us different insights on the program of Figure 1a. In particular, the first quantity tells us which variation in the values of input variables results in larger differences between output values, while the second indicates which variation reaches a greater number of output values.

First Impact Definition (Range). First, we define $\textsc{Range}_i(\mathrm{P})$, where i is the input variable under consideration and P is a program. The intuition behind the quantity $\textsc{Range}_i$ is that, for each input configuration $\langle x, y\rangle$, we gather together the set of input configurations $\langle x', y'\rangle$ resulting from variations of $\langle x, y\rangle$ on the input variable i. Then, we collect the set of reachable outputs from the set of input configurations $\langle x', y'\rangle$ through the program P. As a result, for each input configuration $\langle x, y\rangle$, we obtain the set of output values reachable from $\langle x, y\rangle$ or a variation $\langle x', y'\rangle$. Therefore, among all the sets, we return the maximum size of the range of these sets of output values.

Let us show how to quantify the impact of angle in Figure 1a. For each input configuration $\langle x, y\rangle$, we perturb $\langle x, y\rangle$ to obtain similar input configurations $\langle x', y\rangle$ that may differ in the first component (corresponding to the input variable angle), but coincide on the other. For example, let us consider the input configuration $\langle -4, 1\rangle$. Modifying $\langle -4, 1\rangle$ in the value of angle yields the set $\{\langle -4, 1\rangle, \langle 1, 1\rangle\}$. Consequently, we collect all traces originating from input configurations in $\{\langle -4, 1\rangle, \langle 1, 1\rangle\}$, i.e., $\{\langle -4, 1\rangle \to \langle 3\rangle, \langle 1, 1\rangle \to \langle 0\rangle\}$. To quantify the impact, we focus solely on the output values of these traces $\{3, 0\}$. The size of the range of these values is $\textsc{size}(\{3, 0\}) = \max\{3, 0\} - \min\{3, 0\} = 3 - 0 = 3$. In summary, the impact of angle on the output variable risk from the input configuration $\langle -4, 1\rangle$ is 3. Table 1 provides the quantity of impact regarding the input variable angle across all possible input configurations. The impact value is shown on the last column on the right. Our primary interest lies in identifying the highest impact among all possible input configurations, which in our case is 3. On the other hand, Table 2 illustrates case for the other input variable, speed, where the dotted rows are equivalent to others in terms of reachable output values, and thus already taken into consideration. To summarize,

Table 3. Impact of the input variable Outcomes(P) definition for both angle and speed variables.

Input Data	Input Configuration	Perturbed Traces	Output Values	Outcomes(P)
angle	$\langle -4, 1\rangle$	$\langle -4, 1\rangle \rightarrow \langle 3\rangle, \langle 1, 1\rangle \rightarrow \langle 0\rangle$	$\{3, 0\}$	2
	$\langle -4, 2\rangle$	$\langle -4, 2\rangle \rightarrow \langle 3\rangle, \langle 1, 2\rangle \rightarrow \langle 1\rangle$	$\{3, 1\}$	2
	$\langle -4, 3\rangle$	$\langle -4, 3\rangle \rightarrow \langle 3\rangle, \langle 1, 3\rangle \rightarrow \langle 2\rangle$	$\{3, 2\}$	2
	...	...	...	...
speed	$\langle -4, 1\rangle$	$\langle -4, 1\rangle \rightarrow \langle 3\rangle, \langle -4, 2\rangle \rightarrow \langle 3\rangle,$ $\langle -4, 3\rangle \rightarrow \langle 3\rangle$	$\{3\}$	1
	$\langle 1, 1\rangle$	$\langle 1, 1\rangle \rightarrow \langle 0\rangle, \langle 1, 2\rangle \rightarrow \langle 1\rangle,$ $\langle 1, 3\rangle \rightarrow \langle 2\rangle$	$\{0, 1, 2\}$	3
	...	...	...	...

we can outline the impact definition Range as follows:

$$\begin{aligned} \textsc{Range}_{\texttt{angle}}(P) &\stackrel{\text{def}}{=} \max\{\textsc{size}(\{P(x', y') \mid \langle x', y'\rangle \in \textsc{Input} \wedge y = y'\}) \mid \langle x, y\rangle \in \textsc{Input}\} \\ \textsc{Range}_{\texttt{speed}}(P) &\stackrel{\text{def}}{=} \max\{\textsc{size}(\{P(x', y') \mid \langle x', y'\rangle \in \textsc{Input} \wedge x = x'\}) \mid \langle x, y\rangle \in \textsc{Input}\} \end{aligned} \tag{1}$$

where Input is the set of all input configurations. According to this definition, we obtain $\textsc{Range}_{\texttt{angle}}(\mathrm{P}) = 3$ and $\textsc{Range}_{\texttt{speed}}(\mathrm{P}) = 2$ where P is the program of Figure 1a. Hence, we gain the insight that varying the angle of approach might drastically alter the landing risk, whereas the speed has less influence.

Second Impact Definition (Outcomes). The second definition we explore counts the number of different outputs reachable through variations in the input values, rather than considering their size. Using the same formalization, we denote this notion as:

$$\begin{aligned} \textsc{Outcomes}_{\texttt{angle}}(P) &\stackrel{\text{def}}{=} \max\{|\{P(x', y') \mid \langle x', y'\rangle \in \textsc{Input} \wedge y = y'\}| \mid \langle x, y\rangle \in \textsc{Input}\} \\ \textsc{Outcomes}_{\texttt{speed}}(P) &\stackrel{\text{def}}{=} \max\{|\{P(x', y') \mid \langle x', y'\rangle \in \textsc{Input} \wedge x = x'\}| \mid \langle x, y\rangle \in \textsc{Input}\} \end{aligned}$$

Note the use of the cardinality function $|\cdot|$ instead of the function size. Table 3 shows the steps of the impact definition Outcomes. Accordingly, we obtain $\textsc{Outcomes}_{\texttt{angle}}(\mathrm{P}) = 2$ and $\textsc{Outcomes}_{\texttt{speed}}(\mathrm{P}) = 3$ The underlying idea is that a higher number of reachable outputs indicates a greater influence of the input variable under consideration. The conclusion from Outcomes is that speed has a greater influence than angle, in contrast to the conclusion of Range where angle has a greater impact than speed. Although it may seem counterintuitive at first, the difference between the two impact instances is due to the different program traits they explore. Range quantifies over the variance in the extreme values of the set of output values, while Outcomes quantifies over the variance in the number of unique output values. Consequently, changes in angle yield a bigger variation in the degree of risk compared to speed, while changes in speed reach far more risk levels compared to angle.

Note that, enumerating all possible input configurations is not computationally practical. Specifically, when dealing with more complex input space compositions, this approach is highly inefficient or even infeasible (as in the case of continuous input spaces).

Our Approach. To quantify the impact of a program, one can rely solely on the input-output observations of the program. Thus, our approach is based on an abstraction of input-output relations, which allows us to automatically infer a sound upper-bound on the program's impact.

The analysis starts with a set of abstractions called *output buckets*. A bucket is an abstract element representing a set of output states. While this abstraction may limit the ability to precisely reason about the impact of output values within the same bucket, it permits automatic reasoning across different buckets. Afterwards, an abstract interpretation based static analyzer propagates each output bucket backward through the program under consideration. The analyzer returns an abstract element for each output bucket, representing an over-approximation of the set of input configurations that lead to the output values inside the starting bucket. This result contains also spurious input configurations that may not lead to a value inside the output bucket. Based on the chosen impact definition Impact (e.g., Range or Outcomes), we perform computations and comparisons on the abstract elements returned by the analysis to obtain an upper-bound k'. By construction, $k \leq k'$, where k is the real (concrete) impact quantity obtained by the definition Impact.

Assuming Range as our impact of interest, given the set of abstract elements returned from the backward analysis (one for each bucket), we first project away the input variable i under consideration. This means that our abstract elements now represent input configurations without the variable i. To be precise, we obtain *partial input configurations* taking into account all possible permutations of the input i. After the projection, we check for intersections among the abstract elements. Any intersection indicates two input configurations that only differ in the value of the variable i, leading to two different output buckets.

Continuing our landing alarm system example, we start with three output buckets: $b_0 = \{-1, 0\}$, $b_1 = \{1, 2\}$, and $b_2 = \{3, 4\}$ for instance. Note that, only $\{0, 1, 2, 3\}$ are reachable `risk` values, but we may not have the exact post-condition of a program. Therefore, for each of these three buckets, we run the backward analysis. The backward analysis is parametric in the choice of the abstract domain, for instance, the disjunctive polyhedra domain based on the convex polyhedra domain [Cousot and Halbwachs 1978]. Without showing the details of the analysis, starting with the bucket b_0, only the first branch of the if statement can be processed, Line 3. Therefore, `landing_coeff` should be smaller than 2 at the end of Line 1. Consequently, the sum of the absolute values of `angle` and `speed` should be smaller than 2. Thus, our analysis discovers that only the input configuration $\langle 1, 1\rangle$ leads to the first bucket. Similarly, we discover that from the bucket b_1, we reach the input configurations $\langle 1, 2\rangle$ and $\langle 1, 3\rangle$. Lastly, from the bucket b_2, we reach the input configurations $\langle -4, 1\rangle$, $\langle -4, 2\rangle$, and $\langle -4, 3\rangle$.

Regarding the case in which we quantify the impact of `angle`, we remove `angle` from each of the abstract elements returned by the analysis. For example, from the bucket b_0, we have the abstract input configuration $\langle 1\rangle$; from the bucket b_1, we have $\langle 2\rangle$ and $\langle 3\rangle$; and from the bucket b_2, we have $\langle 1\rangle$, $\langle 2\rangle$, and $\langle 3\rangle$. By checking the intersections, we find that bucket b_0 intersects with bucket b_2 since they have $\langle 1\rangle$ in common. Bucket b_2 also intersects with bucket b_1 since they have both $\langle 2\rangle$ and $\langle 3\rangle$ in common. Understanding the meaning of intersections is crucial. For example, the first intersection between buckets b_0 and b_2 means that there exist two input configurations (namely $\langle 1, 1\rangle$ and $\langle -4, 1\rangle$) that only differ in the value of `angle` (from `angle` $= 1$ in the first configuration $\langle 1, 1\rangle$ to `angle` $= -4$ in the second configuration $\langle -4, 1\rangle$), where the first configuration leads to an output in bucket b_0 and the second configuration leads to an output in the other bucket b_2. Therefore, we are able to assign a range to this variation by taking the minimum of bucket b_0 and the maximum of bucket b_2, resulting in the range $[-1, 4]$ with a size of 5. Similarly, we obtain the range $[1, 4]$ with a size of 3 from the other intersection between buckets b_1 and b_2. Finally, we return the maximum among all possible sizes, which is 5. This result is an over-approximation of the impact value for the function $\textsc{Range}_{\texttt{angle}}(\mathrm{P})$.

Similarly, from the result of the backward analysis, we repeat the steps to quantify the impact regarding the other input variable `speed`. Therefore, the analysis results is an impact of 3 since the only buckets intersecting after projecting away `speed` are buckets b_0 and b_1, with abstract input configurations of $\langle 1\rangle$ for both, while bucket b_2 contains only the value $\langle -4\rangle$.

Regarding the impact definition Outcomes, whenever we discover intersections among different buckets, we count the total number of different outcome values they carry. Consequently, we find that the two intersections for both variables `angle` and `speed` hold four different outcome values in total. In such case, our analysis would

conclude that the two variables hold the same impact. Indeed, the choice of the starting output buckets is critical. For instance, with $b_0 = \{0\}$, $b_1 = \{1\}$, $b_2 = \{2\}$, and $b_3 = \{3\}$ one would recover maximum precision and obtain a useful analysis quantification.

In conclusion, the sound upper-bound discovered by our analysis is always higher than the concrete one by construction of the theoretical framework. The precision of our analysis is the distance between these two bounds, it mostly depends on the choice of output buckets and approximation of the backward analysis. Nevertheless, our analysis is completely automatic and always terminates.

3 TRACE SEMANTICS

This section introduces the background needed for reasoning about the behavior of programs, i.e., program semantics. We begin by considering the *semantics* of a program, a mathematical characterization of its behavior across all possible input data. We model the operational semantics of a program as a *transition system* $\langle \Sigma, \tau \rangle$ where Σ is a (potentially infinite) set of program states and the transition relation $\tau \subseteq \Sigma \times \Sigma$ describes the feasible transitions between states [Cousot 2002; Cousot and Cousot 1977]. The set $\Omega \stackrel{\text{def}}{=} \{s \in \Sigma \mid \forall s' \in \Sigma.\ \langle s, s' \rangle \notin \tau\}$ represents the *final states* of the program.

In the following, we define several definitions related to sequences of program states. Let $\Sigma^n \stackrel{\text{def}}{=} \{s_0 \dots s_{n-1} \mid \forall i < n.\ s_i \in \Sigma\}$ be the set of all sequences containing exactly n program states. We write ϵ to denote the empty sequence, i.e., $\Sigma^0 \stackrel{\text{def}}{=} \{\epsilon\}$. We define $\Sigma^\star \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \Sigma^n$ as the set of all finite sequences, $\Sigma^+ \stackrel{\text{def}}{=} \Sigma^\star \setminus \Sigma^0$ as the set of all non-empty finite sequences, $\Sigma^\infty \stackrel{\text{def}}{=} \{s_0 \dots \mid \forall i \in \mathbb{N}.\ s_i \in \Sigma\}$ as the set of all infinite sequences, and $\Sigma^{+\infty} \stackrel{\text{def}}{=} \Sigma^+ \cup \Sigma^\infty$ as the set of all non-empty finite or infinite sequences. Additionally, we write $\Sigma^\perp \stackrel{\text{def}}{=} \Sigma \cup \{\perp^\Sigma\}$ for the state Σ extended with $\perp^\Sigma$ designating nontermination [Scott and Strachey 1971]. Given a sequence $\sigma \in \Sigma^{+\infty}$, we write $\sigma_0 \in \Sigma$ to denote the initial state of σ and $\sigma_\omega \in \Sigma^\perp$ to denote the final state of σ when $\sigma \in \Sigma^+$, otherwise $\perp^\Sigma$ when $\sigma \in \Sigma^\infty$. In the rest of the paper, $\mathbb{I} \in \{\mathbb{N}, \mathbb{Z}, \mathbb{R}\}$ represents a set of numerical values. We write $\mathbb{I}^{\pm\infty}$ to denote $\mathbb{I}$ extended with the symbols $+\infty$ and $-\infty$. The set $\mathbb{I}_{\geq 0} \stackrel{\text{def}}{=} \{n \in \mathbb{I} \mid n \geq 0\}$ denotes non-negative numbers. Similarly, we can use other predicates, for instance, $\mathbb{I}_{\leq m} \stackrel{\text{def}}{=} \{n \in \mathbb{I} \mid n \leq m\}$ denotes the set of numbers below or equal $m \in \mathbb{I}$.

Given a transition system $\langle \Sigma, \tau \rangle$, a *trace* refers to a non-empty sequence of program states that respects the transition relation τ, i.e., for every pair of consecutive states $s, s' \in \Sigma$ in the trace, it holds that $\langle s, s' \rangle \in \tau$. The *trace semantics* $\Lambda \in \wp(\Sigma^{+\infty})$ generated by a transition system $\langle \Sigma, \tau \rangle$ is the union between all finite traces that are terminating in a final state in Ω, and all non-terminating infinite traces [Cousot 2002]:

$$\begin{aligned} \Lambda \stackrel{\text{def}}{=} & \bigcup_{n \in \mathbb{N}_{\geq 0}} \{s_0 \dots s_{n-1} \in \Sigma^n \mid \forall i < n-1.\ \langle s_i, s_{i+1} \rangle \in \tau \wedge s_{n-1} \in \Omega\} \\ & \cup \{s_0 \dots \in \Sigma^\infty \mid \forall i \in \mathbb{N}.\ \langle s_i, s_{i+1} \rangle \in \tau\} \end{aligned}$$

We write Λ to denote the semantics generated by a transition systems, otherwise written with the semantics brackets $\Lambda[\![\mathrm{P}]\!]$ when the program P is given explicitly. The same applies for other semantics defined in the rest of paper.

The trace semantics fully describes the behavior of a program. However, reasoning about a particular property of a program does not need to consider all aspects of its behavior. In fact, reasoning is facilitated by the design of a semantics that abstracts away from irrelevant details about program executions. In the next sections, we define our property of interest–quantitative input data usage–and use abstract interpretation to systematically derive, by successive abstractions, a semantics tailored to reason about this property.

4 QUANTITATIVE INPUT DATA USAGE

A *property* is specified by its extension, which refers to the set of elements that manifest such a property [Cousot and Cousot 1977]. We consider properties of programs, with trace semantics in $\wp(\Sigma^{+\infty})$, which are sets of sets of

traces in $\wp(\wp(\Sigma^{+\infty}))$. This is in contrast to properties of traces, which are sets of traces in $\wp(\Sigma^{+\infty})$. Notably, program properties that involve relationship between different traces cannot be expressed as trace properties [Clarkson and Schneider 2010].

The strongest property of the trace semantics Λ is the standard *collecting semantics* $\Lambda^{\mathbb{C}} \in \wp(\wp(\Sigma^{+\infty}))$, defined as $\Lambda^{\mathbb{C}} \stackrel{\text{def}}{=} \{ \Lambda \}$, which is satisfied by Λ and nothing else. Therefore, a program P satisfies a given property $\mathrm{F} \in \wp(\wp(\Sigma^{+\infty}))$, written $\mathrm{P} \models \mathrm{F}$, if and only if P belongs to F, or equivalently, its collecting semantics $\Lambda^{\mathbb{C}}$ is a subset of F, formally

$$\mathrm{P} \models \mathrm{F} \;\Leftrightarrow\; \Lambda^{\mathbb{C}}[\![\mathrm{P}]\!] \subseteq \mathrm{F} \tag{2}$$

Our goal is to quantify the impact of a specific input variable on the computation of the program. We focus on understanding the influence of inputs with respect to an input-output view of the program. We introduce the notion of impact, denoted as $\textsc{Impact}_i^{\langle m,\mathcal{F}\rangle}$, which maps program semantics in $\wp(\Sigma^{+\infty})$ to a given domain of quantities (e.g., $\mathbb{N}$), where i represents the input variable of interest in the program under analysis and $\langle m, \mathcal{F}\rangle$ is the *output descriptor* to determine the desired output of a program by observations on program states $\Sigma^{\perp}$.

Specifically, $m \in \Sigma^{\perp} \to \mathbb{M}$ selects the output of interest from a given state and returns its corresponding value within the domain $\mathbb{M}$. This domain $\mathbb{M}$ is parametric to our framework and represents the domain of program outputs. Additionally, $\mathcal{F} \subseteq \mathbb{M} \cap \mathbb{I}^{\pm\infty}$ filters output states and selectively engages a subset of the potential outcomes. Through this filtering mechanism, undesired outcomes are directly excluded and a numerical value is ensured.

Definition 4.1 (Output Descriptor). Let $\mathbb{M}$ be a domain. Given $m \in \Sigma^{\perp} \to \mathbb{M}$ and $\mathcal{F} \subseteq \mathbb{M} \cap \mathbb{I}^{\pm\infty}$, the tuple $\langle m, \mathcal{F}\rangle$ is called an output descriptor.

The above output characterization $\langle m, \mathcal{F}\rangle$ is generic enough to cover plenty of use cases. We leverage this output descriptor to provide the end user of the framework the flexibility to choose the interpretation and meaning of program outputs, without establishing it beforehand. Below, we instantiate the output descriptor using the example of Figure 1.

Example 4.2. Regarding our example of the landing alarm system, the program states are $\Sigma = \{\langle a, b, c, d\rangle \mid a \in \{-4, 1\} \wedge b \in \{1, 2, 3\} \wedge c \in \mathbb{N} \wedge d \in \{0, 1, 2, 3\}\}$, where a is the value of `angle`, b of `speed`, c of `landing_coeff`, and d of `risk`. Here, we abuse the notation and use Σ as set of tuples instead of a map between variables and values, the two views are equivalent. The output descriptor is instantiated with $\mathbb{M} = \{0, 1, 2, 3, *\}$ where $*$ is a placeholder element for non-termination, $\mathcal{F} = \{0, 1, 2, 3\}$, and $m = \textsc{select}$ where

$$\textsc{select}(x) \stackrel{\text{def}}{=} \begin{cases} d & \text{if } x = \langle a, b, c, d\rangle \\ * & \text{if } x = \perp^{\Sigma} \end{cases} \tag{3}$$

Note that, the program of Figure 1a contains no infinite traces. This output descriptor considers infinite traces as $*$ in the output reader m, then $\mathcal{F}$ filters out the unneeded $*$. Accordingly, $\langle \textsc{select}, \mathbb{N}_{\leq 3}\rangle$ identifies the variable `risk`, fourth element of the tuple, as output variable of the program and explicitly excludes infinite traces. However, the end-user of the analysis may be interested in only a subset of the possible outcomes of the program. For instance, only about the risk levels in $\{0, 1, 2\}$, forgetting about the value 3. It is crucial that our impact definitions remain sound to the user assumption on post-conditions, even when it is under-approximating the exact one. Thus the filter specifies this information by $\mathcal{F} = \{0, 1, 2\}$, which is a subset of all the possible values of the output variable `risk`.

For other contexts, rather than considering a single output variable one may be interested in a custom operation. For example, the output of a neural network classifier is the index of the output neuron holding the highest value. Hence, for a network with n output neurons, we could define $\mathbb{M} = \mathbb{N}_{\leq n}$, $m(x_0, \ldots, x_{m+n-1}, x_{m+n}) = \arg\max_{0 \leq j \leq n} x_{m+n-j}$, and $\mathcal{F} = \mathbb{N}_{\leq n}$, where the function $\arg\max_j X_j$ returns the *argument* j of the value holding the *maximum* among the indexed family X_j.

We can now define our property of interest, the *k-bounded impact property* $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$. By extension, $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ is the set of trace semantics such that the impact of the set of traces allowed by the output descriptor $\langle m, \mathcal{F}\rangle$, w.r.t. the input variable i, is below the threshold k. Formally,

$$\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \stackrel{\text{def}}{=} \{\Lambda \in \wp(\Sigma^{+\infty}) \mid \textsc{Impact}^{\langle m,\mathcal{F}\rangle}_{i}(\{\sigma \in \Lambda \mid m(\sigma_\omega) \in \mathcal{F}\}) \leq k\} \tag{4}$$

Following the definition of $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$, our validation framework, Eq. (2), is instantiated as

$$\mathrm{P} \models \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \;\Leftrightarrow\; \Lambda^{\mathbb{C}}[\![\mathrm{P}]\!] \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \tag{5}$$

We are interested in properties $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ that are *extensional*, namely, properties based on the observation of input-output relations of program states. This means that, if $\Lambda[\![\mathrm{P}]\!] \in \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ and $\Lambda[\![\mathrm{P}']\!]$ consider the same set of input-output observations of $\Lambda[\![\mathrm{P}]\!]$, then $\Lambda[\![\mathrm{P}']\!] \in \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$. As a consequence, the impact is not affected by intermediate states, $\textsc{Impact}^{\langle m,\mathcal{F}\rangle}_{i}(\Lambda[\![\mathrm{P}]\!]) = \textsc{Impact}^{\langle m,\mathcal{F}\rangle}_{i}(\Lambda[\![\mathrm{P}']\!])$. Furthermore, we require $\textsc{Impact}^{\langle m,\mathcal{F}\rangle}_{i}$ to be monotonic, i.e., for any $X, Y \in \wp(\Sigma^{+\infty})$, it holds that $X \subseteq Y$ if and only if $\textsc{Impact}^{\langle m,\mathcal{F}\rangle}_{i}(X) \leq \textsc{Impact}^{\langle m,\mathcal{F}\rangle}_{i}(Y)$. Intuitively, this ensures that an impact applied to an over-approximation of the program semantics can only produce a higher quantity, enabling the definition of a sound terminating static analysis.

Next, we formalize the already introduced impact metrics Range and Outcomes, and we define a new impact, called Changes, which will be used as a proof of concept of our static analysis (Section 7).

In the following, given a program P, we assume its program states are maps from variables $\mathbb{V}$ to a numerical domain $\mathbb{I}$, i.e., $\Sigma = \mathbb{V} \to \mathbb{I}$, where $\mathbb{V}$ represents the set of variables used in the program P. The set $\Delta \subseteq \mathbb{V}$ is the set of input variables. We write $\Sigma|_K \in K \to \mathbb{I}$ for the program states reduced to the subset of variables $K \subseteq \mathbb{V}$. For instance $\Sigma = \Sigma|_{\mathbb{V}}$ is the set of states without reduction, and $\Sigma|_\Delta$ is the set of states restricted to the input variables. The predicate $s =_K s'$ indicates that the two states $s, s' \in \Sigma^{\perp}|_K$, agree on all values of the set of variables $K \subseteq \mathbb{V}$, or they are both $\perp^\Sigma$, formally $s =_K s' \Leftrightarrow (s \neq \perp^\Sigma \wedge s' \neq \perp^\Sigma \wedge \forall v \in K.\, s(v) = s'(v)) \vee s = s' = \perp^\Sigma$.

4.1 Range

We begin with the definition of $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_{i} \in \wp(\Sigma^{+\infty}) \to \mathbb{R}^{+\infty}_{\geq 0}$ (see Eq. (1) where we used it to quantify the impact of the program depicted in Figure 1a). The quantity $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_{i}$ determines the size of the range of the set of output values described by the output descriptor $\langle m, \mathcal{F}\rangle$ of all the possible variations in the input variable i. First, we define step-by-step the quantity $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_{i}$, followed by the instantiation of this quantity within the context of example of Figure 1. We present the formal definition at the end.

Intuitively, for any possible input configuration $x \in \Sigma|_\Delta$, we collect all the traces that are starting from an input configuration that is a variation of x on the input variable i, i.e., $\{\sigma \in S \mid \sigma_0 =_{\Delta\setminus\{i\}} x\}$, where $S \in \wp(\Sigma^{+\infty})$. Then, we collect the output values of this set of traces by means of the output descriptor $\langle m, \mathcal{F}\rangle$, i.e., $\{m(\sigma_\omega) \mid \sigma \in S \wedge \sigma_0 =_{\Delta\setminus\{i\}} x\}$. Specifically, this set contains all the output readings performed by m. Afterwards, we extract the size of the range of these output values using the function size. Formally, the function $\textsc{size} \in \wp(\mathcal{F}) \to \mathbb{R}^{+\infty}_{\geq 0}$ returns the size of the range of the given set of output values. That is, for any $Y \subseteq \mathcal{F}$, $\textsc{size}(Y) \stackrel{\text{def}}{=} \max Y - \min Y$. We define $\max Y \stackrel{\text{def}}{=} q$ (resp., $\min Y \stackrel{\text{def}}{=} q$) such that $q \in \mathbb{R}^{\pm\infty}$ is the smallest value bigger (resp., biggest value smaller) than any value in Y (assuming $\max \emptyset = -\infty$, $\min \emptyset = +\infty$ and $-\infty + \infty = 0$). Finally, we iterate through each input configuration x and return the maximum value to ensure the greatest impact is preserved.

Example 4.3 (Landing Alarm System). Let us revisit the example of the landing alarm system, with program states $\Sigma = \{\langle a, b, c, d\rangle \mid a \in \{-4, 1\} \wedge b \in \{1, 2, 3\} \wedge c \in \mathbb{N} \wedge d \in \mathbb{N}_{\leq 3}\}$ and output descriptor $\langle \textsc{select}, \mathbb{N}_{\leq 3}\rangle$, Eq. (3). The input variables are $\Delta = \{\texttt{angle}, \texttt{speed}\}$, consequently the input configurations are $\Sigma|_\Delta = \{\langle -4, 1\rangle, \langle -4, 2\rangle, \langle -4, 3\rangle, \langle 1, 1\rangle, \langle 1, 2\rangle, \langle 1, 3\rangle\}$. We begin by considering $i = \texttt{angle}$ and $\langle -4, 1\rangle$ as the

first input configuration to be explored. Hence, we collect all traces that are starting from an input configuration that is a variation of $\langle -4, 1\rangle$, i.e., $\{\sigma \in \Lambda[\![\mathrm{P}]\!] \mid \sigma_0 =_{\Delta\setminus\{\texttt{angle}\}} \langle -4, 1\rangle\}$, where $\Delta \setminus \{\texttt{angle}\} = \{\texttt{speed}\}$ and consequently $\sigma_0 =_{\{\texttt{speed}\}} \langle -4, 1\rangle$ holds whenever the initial state of σ has `speed` = 1. A possible trace of this set is $\langle 1, 1, 0, 0\rangle \longrightarrow \langle 1, 1, 2, 0\rangle \longrightarrow \langle 1, 1, 2, 0\rangle$ where, at the beginning, we randomly assign `landing_coeff` = 0 and `risk` = 0, respectively the third and fourth component of the initial state. We collect the output values of this set of traces, $\{\textsc{select}(\sigma_\omega) \mid \sigma \in \Lambda[\![\mathrm{P}]\!] \wedge \sigma_0(\texttt{speed}) = 1\}$. As a result, we obtain the set of output values $\{0, 3\}$. For instance, the output value 0 is the result of the trace we exhibited previously, where the last state is $\langle 1, 1, 2, 0\rangle$ and thus the `risk` variable of this trace is the last component with value 0. Finally, applying the size operator returns the value 3, $\textsc{size}(\{0, 3\}) = 3$. By doing so for all possible input configurations in $\Sigma|_\Delta$, we obtain $\textsc{Range}^{\langle \textsc{select},\mathbb{N}_{\leq 3}\rangle}_{\texttt{angle}}(\Lambda[\![\mathrm{P}]\!]) = 3$. Table 1 and 2 in overview (Section 2) illustrate the steps for $\textsc{Range}^{\langle \textsc{select},\mathbb{N}_{\leq 3}\rangle}_{\texttt{angle}}(\Lambda[\![\mathrm{P}]\!])$ and $\textsc{Range}^{\langle \textsc{select},\mathbb{N}_{\leq 3}\rangle}_{\texttt{speed}}(\Lambda[\![\mathrm{P}]\!])$ respectively.

*Definition 4.4 (*Range*).* Given an input variable $i \in \Delta$, and an output descriptor $\langle m, \mathcal{F}\rangle$, the quantity $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_i \in \wp(\Sigma^{+\infty}) \longrightarrow \mathbb{R}^{+\infty}_{\geq 0}$ is defined as

$$\textsc{Range}^{\langle m,\mathcal{F}\rangle}_i(S) \stackrel{\text{def}}{=} \max_{x \in \Sigma|_\Delta} \textsc{size}(\{m(\sigma_\omega) \mid \sigma \in S \wedge \sigma_0 =_{\Delta\setminus\{i\}} x\})$$

It is easy to note that $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_i(S)$ uses only input-output states (cf., σ_0 and σ_ω) of traces $\sigma \in S$. Therefore, assuming $S, S' \in \wp(\Sigma^{+\infty})$ share the same set of input-output observations, $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_i(S) = \textsc{Range}^{\langle m,\mathcal{F}\rangle}_i(S')$ holds. This implies that, $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ is an extensional property when $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_i$ is used to instantiate $\textsc{Impact}^{\langle m,\mathcal{F}\rangle}_i$ in $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$. Furthermore, this impact definition is monotone in the amount of traces. That is, the more traces in input, the higher the impact, meaning that adding traces can only introduce more variety. Formally, for any $S, S' \in \wp(\Sigma^{+\infty})$, $S \subseteq S'$ if and only if $\textsc{Range}^{\langle m,\mathcal{F}\rangle}_i(S) \leq \textsc{Range}^{\langle m,\mathcal{F}\rangle}_i(S')$.

4.2 Outcomes

We present the definition of $\textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i \in \wp(\Sigma^{+\infty}) \longrightarrow \mathbb{N}^{+\infty}$ counting the number of different output values reachable by varying the input variable i.

Example 4.5 (Landing Alarm System). We revisit again the example of the landing alarm system. Assuming i = `angle`, $\langle -4, 1\rangle$ is the first input configuration to be explored, we collect all traces that are starting from an input configuration that is a variation of $\langle -4, 1\rangle$. As before, we obtain $\{\sigma_\omega(\texttt{risk}) \mid \sigma \in S \wedge \sigma_0(\texttt{speed}) = 1\} = \{0, 3\}$. Here, we apply the cardinality rather than the size function, hence $|\{0, 3\}| = 2$. By doing so for all possible input configurations $\Sigma|_\Delta$, we obtain $\textsc{Outcomes}^{\langle \textsc{select},\mathbb{N}_{\leq 3}\rangle}_{\texttt{angle}}(\Lambda[\![\mathrm{P}]\!]) = 2$. Table 3 illustrates the steps for both $\textsc{Outcomes}^{\langle \textsc{select},\mathbb{N}_{\leq 3}\rangle}_{\texttt{angle}}(\Lambda[\![\mathrm{P}]\!])$ and $\textsc{Outcomes}^{\langle \textsc{select},\mathbb{N}_{\leq 3}\rangle}_{\texttt{speed}}(\Lambda[\![\mathrm{P}]\!])$.

*Definition 4.6 (*Outcomes*).* Given an input variable $i \in \Delta$, and an output descriptor $\langle m, \mathcal{F}\rangle$, the quantity $\textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i \in \wp(\Sigma^{+\infty}) \longrightarrow \mathbb{N}^{+\infty}$ is defined as

$$\textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i(S) \stackrel{\text{def}}{=} \max_{x \in \Sigma|_\Delta} |\{m(\sigma_\omega) \mid \sigma \in S \wedge \sigma_0 =_{\Delta\setminus\{i\}} x\}|$$

As similarly noted for the previous definition, $\textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i(S)$ uses only input-output states (cf., σ_0 and σ_ω) of traces $\sigma \in S$. Therefore, assuming $S, S' \in \wp(\Sigma^{+\infty})$ share the same set of input-output observations, $\textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i(S) = \textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i(S')$ holds. This impact definition is monotone as well, in the amount of traces. Formally, for any $S, S' \in \wp(\Sigma^{+\infty})$, $S \subseteq S'$ if and only if $\textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i(S) \leq \textsc{Outcomes}^{\langle m,\mathcal{F}\rangle}_i(S')$.

4.3 Changes

We introduce the last impact metric, denoted as $\textsc{Changes}^{\langle m,\mathcal{F}\rangle}_i \in \wp(\Sigma^{+\infty}) \longrightarrow \mathbb{N}^{+\infty}$, counting how many times the program outcome changes by modifications in the value of the input variable i. Notably, this metric is similar to

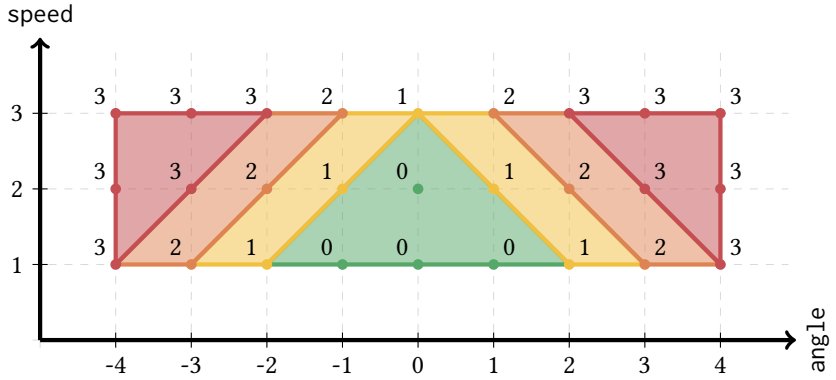


Fig. 2. Extended example, input space composition with continuous input values, angle $\in [-4, 4]$ and speed $= [1, 3]$

the previous Outcomes but considers changes in the outcome *with repetitions*, that is, if two different variations in the input variable i result in the same change in the outcome, it counts as a double change. The higher the number of changes in the outcome, the greater the influence on the program outcome. Therefore, this definition demonstrates its effectiveness when the same outcomes are reachable by multiple variations. In Section 7, we show the result of applying such impact definition in the context of neural network models, where Range and Outcomes would not be useful. This is due to the fact that all the possible variations often lead to every outcome in such context. Hence, counting the repetitions is a potential solution to define a meaningful impact definition.

Additionally, this definition $\textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}$ considers only variations in the value of input configurations that do not belong to the same *continuous region*, containing no gaps or interruptions. To this end, we first define a function $\textsc{seg}_i^m \in \wp(\Sigma^{+\infty}) \times \mathcal{F} \to \wp(\wp(\Sigma^{+\infty}))$, which takes as input a set of traces $S \in \wp(\Sigma^{+\infty})$ and an output value $z \in \mathcal{F}$. This function partitions the set of traces S into continuous subsets of S with respect to the value of i. Each subset S' satisfies two conditions: first, all traces within S' ends with output value z, by means of $\langle m, \mathcal{F}\rangle$. Second, for any two traces in S', there is no other trace with a value of i *between them* leading to an output value different from z. Formally,

$$\textsc{seg}_i^m(S, z) \stackrel{\text{def}}{=} \{S' \subseteq S \mid \forall \sigma \in S'.\ m(\sigma_\omega) = z \tag{6}$$

$$\wedge\ \forall \sigma' \in S'.\ \exists \sigma'' \in S.\ \sigma_0(i) \leq \sigma_0''(i) \leq \sigma_0'(i) \Rightarrow \sigma'' \in S'\} \tag{7}$$

where condition (6) enforces that all the traces in the subset S' lead to the same outcome z, and condition (7) imposes the continuity condition with respect to the input variable i. To better illustrate how this definition works, we introduce continuous values in the example of Figure 1 to obtain an input space with repetitions.

Example 4.7 (Landing Alarm System, extended). We extend the example of Figure 1 by allowing continuous values for the input variables angle and speed. Hence, angle $\in [-4, 4]$ and speed $\in [1, 3]$. A value close to 0 for the variable angle represents an optimal alignment while both positive and negative values indicates various degrees of misalignment. Figure 2 depicts the input space composition. The green region represents the input configurations for which the landing is safe. Yellow, orange, and red respectively reflects an increasing level of risk on both sides of the safe region. We show how $\textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}$ works step-by-step. We begin by collecting all

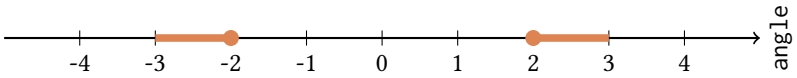


Fig. 3. Input configurations of traces belonging to $\text{SEG}^{\text{SELECT}}_{\texttt{angle}}(\{\sigma \mid \sigma_0(\texttt{speed}) = 2\}, 2)$.

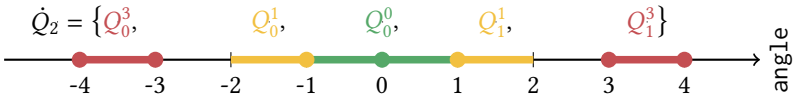


Fig. 4. Input configurations of traces belonging to $\dot{Q}_2 = \bigcup_{z\in\{0,1,3\}} \text{SEG}^{\text{SELECT}}_{\texttt{angle}}(\{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2\}, z)$.

the continuous segments starting with a value of speed = 2 leading to the output value risk = 2 (orange region):

$$\text{SEG}^{\text{SELECT}}_{\texttt{angle}}(\{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2\}, 2) = \left\{ \begin{array}{l} \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \\ \quad \wedge -3 < \sigma_0(\texttt{angle}) \leq -2\}, \\ \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \\ \quad \wedge 2 \leq \sigma_0(\texttt{angle}) < 3\} \end{array} \right\}$$

Figure 3 graphically shows the input configurations of traces belonging to the set above. Our goal is to group together continuous regions by output values that start with input configurations resulting from variations in the input variable i, and by excluding the regions that do not lead to changes in the output values. For instance, if we begin with an input configuration leading to the orange outcome, risk = 2, we obtain the set

$$\dot{Q}_2 = \bigcup_{z\in\{0,1,3\}} \text{SEG}^{\text{SELECT}}_{\texttt{angle}}(\{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2\}, z)$$
$$= \left\{ \begin{array}{l} Q^0_0 = \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \wedge -1 < \sigma_0(\texttt{angle}) < 1\}, \\ Q^1_0 = \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \wedge -2 < \sigma_0(\texttt{angle}) \leq -1\}, \\ Q^1_1 = \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \wedge 1 \leq \sigma_0(\texttt{angle}) < 2\}, \\ Q^3_0 = \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \wedge -4 \leq \sigma_0(\texttt{angle}) \leq -3\}, \\ Q^3_1 = \{\sigma \in \Lambda[\![\text{P}]\!] \mid \sigma_0(\texttt{speed}) = 2 \wedge 3 \leq \sigma_0(\texttt{angle}) \leq 4\} \end{array} \right\}$$

Figure 4 shows $\dot{Q}_2$ graphically. Note that, the dot notation over set variables (e.g., $\dot{Q}_2$ in the previous definition) indicates a set of sets of elements. Finally, the number of possible outcome changes with repetitions is the cardinality of $\dot{Q}_2$. In other words, starting from an input region (with speed = 2) leading to the output value risk = 2, by applying a variation on the value of angle, we can reach all the other regions, $\{Q^3_0, Q^1_0, Q^0_0, Q^1_1, Q^3_1\}$. Hence, we count how many regions we can reach with repetitions (Q^3_0 and Q^3_1 count as 2 regions even though they lead to the same output value 3). In total we can reach 5 other regions. In the same way, we do so by considering all the other output values 0, 1, and 3, yielding the maximum of all of them. Formally, we write $\max_{z\in\mathbb{N}_{\leq 3}} |\dot{Q}_z|$ to count how many other regions we can reach. As a result, the maximum between every input configuration is $\textsc{Changes}^{\langle \text{SELECT},\mathbb{N}_{\leq 3}\rangle}_{\texttt{angle}}(\Lambda[\![\text{P}]\!]) = 5$, since the case starting from the input with speed = 2 coincides with the case yielding the highest number of changes in outcome. Similarly, $\textsc{Changes}^{\langle \text{SELECT},\mathbb{N}_{\leq 3}\rangle}_{\texttt{speed}}(\Lambda[\![\text{P}]\!]) = 2$. Indeed, by looking at Figure 2, it is clear that angle is able to change output value far more times compared to speed.

We now formally define Changes.

Definition 4.8 (Changes). Given an input variable $i \in \Delta$, and an output descriptor $\langle m, \mathcal{F}\rangle$, the quantity $\textsc{Changes}^{\langle m,\mathcal{F}\rangle}_{i} \in \wp(\Sigma^{+\infty}) \to \mathbb{N}^{+\infty}$ is defined as the maximum, for each input configuration, of the number of

continuous regions leading to a change in output:

$$\textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}(S) \stackrel{\text{def}}{=} \max_{x\in\Sigma|_\Delta} \max_{z\in\mathcal{F}} \mid \dot{Q}_{i,x,z}^{\langle m,\mathcal{F}\rangle} \mid \tag{8}$$

where $\dot{Q}_{i,x,z}^{\langle m,\mathcal{F}\rangle} \in \wp(\wp(\Sigma^{+\infty}))$ is defined as the set of continuous regions leading to an output value $z' \neq z$.

$$\dot{Q}_{i,x,z}^{\langle m,\mathcal{F}\rangle} \stackrel{\text{def}}{=} \bigcup_{z'\in\mathcal{F}\setminus\{z\}} \textsc{seg}_i^m(\{\sigma \in S \mid \sigma_0 =_{\Delta\setminus\{i\}} x\}, z') \tag{9}$$

Assuming $S, S' \in \wp(\Sigma^{+\infty})$ share the same set of input-output observations, since we quantify over outputs $z \in \mathcal{F}$ and only access input states σ_0, it holds that $\textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}(S) = \textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}(S')$. Furthermore, we show that $\textsc{Changes}$ is monotone, that is, for any $S, S' \in \wp(\Sigma^{+\infty})$, if $S \subseteq S'$ then it holds that $\textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}(S) \leq \textsc{Changes}_i^{\langle m,\mathcal{F}\rangle}(S')$. The intuition behind is that more traces generate a higher amount of continuous regions, never a lower amount. Hence, the number of continuous regions leading to a different output (cf., $\mid \dot{Q}_{i,x,z}^{\langle m,\mathcal{F}\rangle} \mid$) can only increase with more traces.

5 HIERARCHY OF SEMANTICS

We now use abstract interpretation to systematically derive, by successive abstractions of the collecting semantics $\Lambda^{\mathbb{C}}$, two *sound and complete* semantics, $\Lambda^{\rightsquigarrow}$ and $\Lambda^{\langle m,\mathcal{F}\rangle}$, that contain only and exactly the information needed to reason about $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$. First, from the collecting semantics $\Lambda^{\mathbb{C}}$ we derive the *dependency semantics* $\Lambda^{\rightsquigarrow}$ [Urban 2020] as an abstraction that removes intermediate computational states. Then, on top of the dependency semantics we derive a semantics considering only observations that respect the output descriptor $\langle m, \mathcal{F}\rangle$, named *filter semantics*.

5.1 Dependency Semantics

Since our property of interest $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ is extensional, to quantify the usage of input variables we do not need all the information between the initial and final states of a finite trace, or after the initial state in case of an infinite trace. Thus, we can abstract the collecting semantics into a set of dependencies between initial and output states of finite traces and between initial and $\perp^{\Sigma}$ for infinite traces.

Formally, the pair of right-left adjoints $\langle \alpha^{\rightsquigarrow}, \gamma^{\rightsquigarrow}\rangle$ with $\alpha^{\rightsquigarrow} \in \wp(\wp(\Sigma^{+\infty})) \to \wp(\wp(\Sigma \times \Sigma^{\perp}))$ and $\gamma^{\rightsquigarrow} \in \wp(\wp(\Sigma \times \Sigma^{\perp})) \to \wp(\wp(\Sigma^{+\infty}))$ is defined as:

$$\alpha^{\rightsquigarrow}(\dot{X}) \stackrel{\text{def}}{=} \{\{\langle\sigma_0, \sigma_\omega\rangle \mid \sigma \in X\} \mid X \in \dot{X}\} \tag{10}$$

$$\gamma^{\rightsquigarrow}(\dot{Y}) \stackrel{\text{def}}{=} \{X \in \wp(\Sigma^{+\infty}) \mid \{\langle\sigma_0, \sigma_\omega\rangle \mid \sigma \in X\} \in \dot{Y}\} \tag{11}$$

where $\alpha^{\rightsquigarrow}$ abstracts away all the intermediate states of any trace, preserving the set-structure of $\dot{X}$, and the concretization $\gamma^{\rightsquigarrow}$ yields all the semantics that share the same input-output observations of, at least, one of the set of semantics $\dot{Y}$.

Theorem 5.1. $\langle \wp(\wp(\Sigma^{+\infty})), \subseteq\rangle \underset{\alpha^{\rightsquigarrow}}{\overset{\gamma^{\rightsquigarrow}}{\leftrightarrows}} \langle \wp(\wp(\Sigma \times \Sigma^{\perp})), \subseteq\rangle$ *is a Galois connection.*

Proof. Given a set of semantics $\dot{X} \in \wp(\wp(\Sigma^{+\infty}))$ and a set of sets of input-output observations $\dot{Y} \in \wp(\wp(\Sigma \times \Sigma^{\perp}))$ implied by the abstraction of $\dot{X}$, $\alpha^{\rightsquigarrow}(\dot{X}) \subseteq \dot{Y}$, we obtain that $\dot{X} \subseteq \gamma^{\rightsquigarrow}(\dot{Y})$ since the concretization $\gamma^{\rightsquigarrow}$ builds all the possible semantics with the same set of input-output observations of at least one of the starting semantics. Moreover, it is easy to note that $\alpha^{\rightsquigarrow}(\gamma^{\rightsquigarrow}(\dot{Y})) = \dot{Y}$ since the concretization maintains the same input-output observations and the abstraction removes only intermediate states. ∎

We can now derive the *dependency semantics* $\Lambda^{\rightsquigarrow} \in \wp(\wp(\Sigma \times \Sigma^{\perp}))$ as follows:

$$\Lambda^{\rightsquigarrow} \stackrel{\text{def}}{=} \alpha^{\rightsquigarrow}(\Lambda^{\mathbb{C}}) = \alpha^{\rightsquigarrow}(\{\ \Lambda\ \}) = \{\{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in X\} \mid X \in \{\ \Lambda\ \}\} = \{\ \{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Lambda\}\ \} \tag{12}$$

The next result shows that $\Lambda^{\rightsquigarrow}$ allows sound and complete verification for proving that the impact of the input variable i of a program P is bounded by k.

Theorem 5.2. $\Lambda^{\mathbb{C}}[\![\mathrm{P}]\!] \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \ \Leftrightarrow\ \Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!] \subseteq \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$

Proof. The implication ($\Rightarrow$) follows from the monotonicity of $\alpha^{\rightsquigarrow}$, Theorem (5.1), and the definition of $\Lambda^{\rightsquigarrow}$, Eq. (12), i.e., $\Lambda^{\mathbb{C}} \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \Rightarrow \alpha^{\rightsquigarrow}(\Lambda^{\mathbb{C}}) \subseteq \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}) \Rightarrow \Lambda^{\rightsquigarrow} \subseteq \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$. Regarding the other implication ($\Leftarrow$), from the definition of $\Lambda^{\rightsquigarrow}$ and the property of Theorem (5.1), we obtain $\Lambda^{\rightsquigarrow} \subseteq \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}) \Rightarrow \alpha^{\rightsquigarrow}(\Lambda^{\mathbb{C}}) \subseteq \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}) \Rightarrow \Lambda^{\mathbb{C}} \subseteq \gamma^{\rightsquigarrow}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}))$, which can be written as $\Lambda \in \gamma^{\rightsquigarrow}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}))$ by the definition of $\Lambda^{\mathbb{C}}$. By definition of $\gamma^{\rightsquigarrow}$, Eq. (11), it follows that $\{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Lambda\} \in \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$. Finally, by application of the definition of $\alpha^{\rightsquigarrow}$, Eq. (10), we obtain $\Lambda \in \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$. The conclusion $\Lambda^{\mathbb{C}} \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ trivially follows from the definition of ($\subseteq$). ∎

5.2 Filter Semantics

From the dependency abstractions we derive the *filter semantics*. We exploit the output descriptor $\langle m, \mathcal{F}\rangle$ to remove dependencies (tuple of input-output values) that are not relevant for our property, that is, not in $\mathcal{F}$.

Formally, the pair of right-left adjoints $\langle \alpha^{\langle m,\mathcal{F}\rangle}, \gamma^{\langle m,\mathcal{F}\rangle}\rangle$ with $\alpha^{\langle m,\mathcal{F}\rangle}, \gamma^{\langle m,\mathcal{F}\rangle} \in \wp(\wp(\Sigma \times \Sigma^{\perp})) \to \wp(\wp(\Sigma \times \Sigma^{\perp}))$ is defined as:

$$\alpha^{\langle m,\mathcal{F}\rangle}(\dot{X}) \stackrel{\text{def}}{=} \{\{\langle \sigma_0, \sigma_\omega \rangle \in X \mid m(\sigma_\omega) \in \mathcal{F}\} \mid X \in \dot{X}\}$$

$$\gamma^{\langle m,\mathcal{F}\rangle}(\dot{Y}) \stackrel{\text{def}}{=} \{Y \cup \{\langle \sigma_0, \sigma_\omega \rangle \mid m(\sigma_\omega) \in Z \wedge \sigma_0 \in I\} \mid Y \in \dot{Y} \wedge Z \subseteq \mathbb{M} \setminus \mathcal{F} \wedge I \subseteq \Sigma\}$$

where $\alpha^{\langle m,\mathcal{F}\rangle}$ abstracts away pair of states that lead to outputs that are not allowed by the output descriptor $\langle m, \mathcal{F}\rangle$ maintaining the set-structure of $\dot{X}$. The concretization $\gamma^{\langle m,\mathcal{F}\rangle}$ considers all the semantics $Y \in \dot{Y}$ extended with arbitrary observations made of discarded output values in $\mathbb{M} \setminus \mathcal{F}$. The abstract subset operator $\dot{X} \dot{\subseteq} \dot{Y}$ checks subset inclusion among sets of traces allowed by the output descriptor $\langle m, \mathcal{F}\rangle$, formally defined as:

$$\dot{X} \dot{\subseteq} \dot{Y} \Leftrightarrow \{X \in \dot{X} \mid \forall \langle s_0, s_\omega \rangle \in X.\ m(s_\omega) \in \mathcal{F}\} \subseteq \{Y \in \dot{Y} \mid \forall \langle s_0, s_\omega \rangle \in Y.\ m(s_\omega) \in \mathcal{F}\}$$

Therefore, we define the following Galois connection.

Theorem 5.3. $\langle \wp(\wp(\Sigma \times \Sigma^{\perp})), \subseteq\rangle \underset{\alpha^{\langle m,\mathcal{F}\rangle}}{\overset{\gamma^{\langle m,\mathcal{F}\rangle}}{\leftrightarrows}} \langle \wp(\wp(\Sigma \times \Sigma^{\perp})), \dot{\subseteq}\rangle$ *is a Galois connection.*

Proof. Given two set of sets of input-output observations $\dot{X}, \dot{Y} \in \wp(\wp(\Sigma \times \Sigma^{\perp}))$ such that $\alpha^{\langle m,\mathcal{F}\rangle}(\dot{X}) \dot{\subseteq} \dot{Y}$, we obtain that $\dot{X} \subseteq \gamma^{\langle m,\mathcal{F}\rangle}(\dot{Y})$ since the concretization $\gamma^{\langle m,\mathcal{F}\rangle}$ builds all the possible set of input-output observations enhanced with auxiliary observations made of output values previously discarded by $\alpha^{\langle m,\mathcal{F}\rangle}(\dot{X})$. On the other hand, we have $\dot{X} \subseteq \gamma^{\langle m,\mathcal{F}\rangle}(\dot{Y})$. By abstracting we obtain $\alpha^{\langle m,\mathcal{F}\rangle}(\dot{X})$, which filters all semantics by removing traces that do not agree with $\langle m, \mathcal{F}\rangle$. It holds that $\alpha^{\langle m,\mathcal{F}\rangle}(\dot{X}) \dot{\subseteq} \dot{Y}$ since the $\dot{\subseteq}$ operator removes, in turn, traces that do not agree with $\langle m, \mathcal{F}\rangle$. ∎

We derive the filter semantics $\Lambda^{\langle m,\mathcal{F}\rangle} \in \wp(\wp(\Sigma \times \Sigma^{\perp}))$ as follows:

$$\begin{aligned} \Lambda^{\langle m,\mathcal{F}\rangle} &\stackrel{\text{def}}{=} \alpha^{\langle m,\mathcal{F}\rangle}(\Lambda^{\rightsquigarrow}) = \alpha^{\langle m,\mathcal{F}\rangle}(\{\ \{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Lambda\}\ \}) \\ &= \{\ \{\langle \sigma_0, \sigma_\omega \rangle \in \{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Lambda\} \mid m(\sigma_\omega) \in \mathcal{F}\}\ \} \\ &= \{\ \{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Lambda \wedge m(\sigma_\omega) \in \mathcal{F}\}\ \} \end{aligned} \tag{13}$$

The next result shows that $\Lambda^{\langle m,\mathcal{F}\rangle}$ allows sound and complete verification for proving that the impact of a program is bounded by k.

Theorem 5.4. $\Lambda^{\mathbb{C}}[\![\mathrm{P}]\!] \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \;\Leftrightarrow\; \Lambda^{\langle m,\mathcal{F}\rangle}[\![\mathrm{P}]\!] \mathrel{\dot{\sqsubseteq}} \alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}))$

Proof. The implication ($\Rightarrow$) follows from Theorem (5.2), the monotonicity of $\alpha^{\langle m,\mathcal{F}\rangle}$, Theorem (5.3), and the definition of $\Lambda^{\langle m,\mathcal{F}\rangle}$, Eq. (13), i.e., $\Lambda^{\mathbb{C}} \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle} \Rightarrow \Lambda^{\rightsquigarrow} \subseteq \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}) \Rightarrow \alpha^{\langle m,\mathcal{F}\rangle}(\Lambda^{\rightsquigarrow}) \mathrel{\dot{\sqsubseteq}} \alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})) \Rightarrow \Lambda^{\langle m,\mathcal{F}\rangle} \mathrel{\dot{\sqsubseteq}} \alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}))$. Regarding the other implication ($\Leftarrow$), by the definition of $\Lambda^{\langle m,\mathcal{F}\rangle}$, $\Lambda^{\rightsquigarrow}$, and the property of Galois connections, we obtain

$$
\begin{aligned}
&\Lambda^{\langle m,\mathcal{F}\rangle} \mathrel{\dot{\sqsubseteq}} \alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})) \\
&\quad\Rightarrow \alpha^{\langle m,\mathcal{F}\rangle}(\Lambda^{\rightsquigarrow}) \mathrel{\dot{\sqsubseteq}} \alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})) && \text{(by Eq. (13))} \\
&\quad\Rightarrow \Lambda^{\rightsquigarrow} \subseteq \gamma^{\langle m,\mathcal{F}\rangle}(\alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}))) && \text{(by Theorem (5.3))} \\
&\quad\Rightarrow \{\langle\sigma_0,\sigma_\omega\rangle \mid \sigma \in \Lambda\} \in \gamma^{\langle m,\mathcal{F}\rangle}(\alpha^{\langle m,\mathcal{F}\rangle}(\alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}))) && \text{(by Eq. (12))}
\end{aligned}
$$

We split two cases depending whether $\{\langle\sigma_0,\sigma_\omega\rangle \mid \sigma \in \Lambda\}$ contains only traces allowed by $\langle m,\mathcal{F}\rangle$:

(i) $\forall\sigma \in \Lambda.\ m(\sigma_\omega) \in \mathcal{F}$ holds. In this case it is easy to note that $\{\langle\sigma_0,\sigma_\omega\rangle \mid \sigma \in \Lambda\} \in \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$ since $\gamma^{\langle m,\mathcal{F}\rangle}$ and $\alpha^{\langle m,\mathcal{F}\rangle}$ do not affect semantics that already satisfy the output descriptor.

(ii) $\exists\sigma \in \Lambda.\ m(\sigma_\omega) \notin \mathcal{F}$ holds. Given $S \subseteq \{\langle\sigma_0,\sigma_\omega\rangle \mid \sigma \in \Lambda\}$ such that any trace $\sigma \in S$ is allowed by $m(\sigma_\omega) \in \mathcal{F}$, from the previous case it is evident that $S \in \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$. By the definition of $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$, Eq. (4), it holds that any superset $S' \supseteq S$ belongs to $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ when S contains only traces allowed by the output descriptor and $S \in \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$. Hence, it holds that $\{\langle\sigma_0,\sigma_\omega\rangle \mid \sigma \in \Lambda\} \in \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$.

In both cases (i) and (ii) we obtain $\{\langle\sigma_0,\sigma_\omega\rangle \mid \sigma \in \Lambda\} \in \alpha^{\rightsquigarrow}(\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle})$. The conclusion $\Lambda^{\mathbb{C}} \subseteq \mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$ trivially follows from the definition of ($\subseteq$) and Theorem (5.2). ∎

This semantics still contains enough information to soundly and completely prove our any property $\mathscr{B}^{\leq k}_{i,\langle m,\mathcal{F}\rangle}$, therefore still undecidable.

6 A STATIC ANALYSIS FOR QUANTITATIVE INPUT DATA USAGE

In this section, we introduce a static analysis for determining a sound upper-bound on the impact of an input variable i. The soundness of the approach leverages the filter semantics $\Lambda^{\langle m,\mathcal{F}\rangle}$. As a proof of concept our static analysis is tailored specifically to feed-forward neural networks with ReLU activation functions (e.g., see the tutorial by Albarghouthi [2021] for an introduction on neural network analysis). In the conclusion, Section 9, we discuss a possible extension of our approach to programs beyond neural networks.

Neural Networks. A neural network model M can be viewed as an acyclic program that combines affine transformations with activation functions. Formally, a *feed-forward neural network* consists of neurons distributed across an input layer L_1, an output layer L_l, and $l-2$ hidden layers $\mathrm{L}_2,\ldots,\mathrm{L}_{l-1}$. Each layer L_i contains $|\mathrm{L}_i|$ neurons. We denote the set of all neurons in the network as X, with X_i representing the neurons in layer L_i, and $x_{i,j}$ referring to the j-th neuron in the i-th layer. The set of input variables is denoted as $\{x_{1,j} \mid 1 \leq j \leq |\mathrm{L}_1|\}$. Additionally, each layer $\mathrm{L}_i \in \{\mathrm{L}_2,\ldots,\mathrm{L}_l\}$ is associated with a weight matrix W_i of type $|\mathrm{L}_i| \times |\mathrm{L}_{i-1}|$ and a bias vector B_i of type $|\mathrm{L}_i|$. An input configuration for the model is represented by a $|\mathrm{L}_1|$-vector of values within the unit interval $[0,1]$. The values of hidden and output neurons are computed by applying an activation function after performing an affine transformation on the values of all neurons in the preceding layer [Goodfellow et al. 2016]. Here we consider the ReLU activation function, i.e., $\mathrm{ReLU}(x) \stackrel{\text{def}}{=} \max\{x,0\}$. Formally, the neuron value is computed as $x_{i,j} = \mathrm{ReLU}(\sum_k^{|\mathrm{L}_{i-1}|} w^i_{j,k} \cdot x_{i-1,k} + b^i_j)$ where $w^i_{j,k}$ and b^i_j denote the weight and bias coefficients

obtained from W_i and B_i, respectively. We focus on neural networks used for *classification* purposes, where the outcome of the model corresponds to the index of the neuron in the output layer L_l holding the highest value, i.e., $\arg\max_{1\le k\le|\mathrm{L}_l|} x_{l,j}$. Consequently, there are $|\mathrm{L}_l|$ possible classification targets. In this context, the output descriptor of our framework is $\langle \textsc{highest}, \mathbb{L}\rangle$ where $\textsc{highest}(\langle x_{1,1},\dots,x_{l,|\mathrm{L}_l|}\rangle) \stackrel{\text{def}}{=} (\arg\max_{1\le k\le|\mathrm{L}_l|} x_{l,k}) - 1$ returns the index of the maximum value among the neurons in the output layer, and the filter $\mathbb{L} \stackrel{\text{def}}{=} \mathbb{N}_{<|\mathrm{L}_l|}$ is the set of output indices between 0 and $|\mathrm{L}_l| - 1$. Thus, the property of interest for neural network analysis is $\mathscr{B}^{\le k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. Our goal is to automatically verify whether a neural network model M satisfies or not the property of interest $\mathscr{B}^{\le k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$, that is, whether $\mathrm{M} \models \mathscr{B}^{\le k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$ holds.

The pseudocode representing our proposed static analysis quantifying the usage of input variable i of a neural network model M is shown in Algorithm 1.

Algorithm 1 Analysis to compute Quantitative Input Data Usage

```
1:  function Backward(𝒟♮, M, ℬ_j)
2:      a ← ℬ_j
3:      for p ← l down to 1 do
4:          for q ← |L_p| down to 1 do
5:              a ← assign♮_{𝒟♮}⟦x_{p,q}⟧(relu♮_{𝒟♮}⟦x_{p,q}⟧ a)
6:      return a
7:  function Analyze(𝒟♮, M, i, ℬ, Impact♮)
8:      for j ← 0 up to |ℬ| do
9:          Ẋ_j ← Backward(𝒟♮, M, ℬ_j)
10:     k ← Impact♮(i, ℬ, Ẋ)
11:     return k
```

It begins with `Analyze` (Line 7), which combines a backward pre-analysis (Lines 8 and 9) with an impact implementation (Line 10). The backward analysis employs an abstract domain, denoted as $\mathcal{D}^\natural$, and starts from a family of abstract elements $\mathcal{B} \in \mathcal{D}^{\natural^n}$, called *output buckets*. These output buckets represent sets of output values. The pre-analysis performs a backward reachability analysis from each starting bucket $\mathcal{B}_j$, computing the set of input configurations $\dot{X}$ that lead to an output in $\mathcal{B}_j$. In other words, $\dot{X}_j$ is an over-approximation of the input configurations that can produce as output the values represented by the abstract element in $\mathcal{B}_j$. That is, whenever an input configuration $Y \in \Sigma|_\Delta$ exists such that a trace starting from Y reaches $\mathcal{B}_j$, $\dot{X}_j$ always includes Y.

To propagate an over-approximation of the concrete states backwardly through the network, the chosen abstract domain $\mathcal{D}^\natural$ should implement the necessary sound abstract transformers. These transformers, denoted as $\texttt{assign}^\natural_{\mathcal{D}^\natural}[\![x_{i,j}]\!]$ and $\texttt{relu}^\natural_{\mathcal{D}^\natural}[\![x_{i,j}]\!]$, handle affine transformations and the activation function ReLU during the backward pre-analysis. Additionally, $\mathcal{D}^\natural$ is equipped with a concretization function $\gamma^\natural \in \mathcal{D}^\natural \to \wp(\Sigma)$.

`Analyze` (Line 7) takes as input the abstract domain $\mathcal{D}^\natural$, the neural network model M, the input variable of interest $i \in \{x_{1,j} \mid 1 \le j \le |\mathrm{L}_1|\}$, n output buckets $\mathcal{B} \in \mathcal{D}^{\natural^n}$, and a sound implementation $\texttt{Impact}^\natural$ of the concrete Impact used in the property $\mathscr{B}^{\le k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. In particular, we expect a sound implementation $\texttt{Impact}^\natural$ to return a bound on the impact which is always higher than the concrete counterpart Impact, as specified by the following definition:

Definition 6.1. Let $\mathcal{B} \in \mathcal{D}^{\natural^n}$ be the output buckets, and $\dot{X} \in \mathcal{D}^{\natural^n}$ the result of the backward pre-analysis. Given a concrete Impact and an abstract implementation $\texttt{Impact}^\natural$, we say that $\texttt{Impact}^\natural$ is a *sound implementation* of

Impact for neural network analysis, if and only if,

$$\textsc{Impact}_{i}^{\langle \textsc{highest},\mathbb{L}\rangle}(S) \leq \texttt{Impact}^{\natural}(i, \mathcal{B}, \dot{X})$$

where $S = \{\langle s_0, s_\omega\rangle \mid j \leq n \wedge s_0 \in \gamma^{\natural}(\dot{X}_j) \wedge s_\omega \in \gamma^{\natural}(\mathcal{B}_j)\}$ is the set of input-output observations from the backward pre-analysis.

The next result shows that our static analysis is sound when used to verify the property of interest $\mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. That is, if the static analysis `Analyze` returns the bound k', and $k' \leq k$, then the neural network model M satisfies the property $\mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$, cf. $\mathrm{M} \models \mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. This result is possible when $\mathbb{L} \subseteq \bigcup_{j\leq n}\{\textsc{highest}(s) \mid s \in \gamma^{\natural}(\mathcal{B}_j)\}$, namely, when all the possible outcomes of a neural network are included in the concrete states belonging to the output buckets.

Lemma 6.2. *Let* M *be a neural network model,* $i \in \{x_{1,j} \mid 1 \leq j \leq |\iota_1|\}$ *the input variable of interest, and* $\mathcal{B} \in \mathcal{D}^{\natural^n}$ *the starting output buckets such that* $\mathbb{L} \subseteq \bigcup_{j\leq n}\{\textsc{highest}(s) \mid s \in \gamma^{\natural}(\mathcal{B}_j)\}$. *Whenever* `Impact`$^{\natural}$ *is a sound implementation of* $\textsc{Impact}^{\langle \textsc{highest},\mathbb{L}\rangle}$, *it holds that*

$$\texttt{Analyze}(\mathcal{D}^{\natural}, \mathrm{M}, i, \mathcal{B}, \texttt{Impact}^{\natural}) = k \implies \mathrm{M} \models \mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$$

Proof. The result of the backward pre-analysis $\dot{X}$, computed by Line 8 and 9, produces an over-approximation of the filter semantics thanks to the hypothesis that we start from an over-approximation of the filter $\mathbb{L}$, i.e., $\mathbb{L} \subseteq \bigcup_{j\leq n}\{\textsc{highest}(s) \mid s \in \gamma^{\natural}(\mathcal{B}_j)\}$. That is, when the singleton $X \in \Lambda^{\langle \textsc{highest},\mathbb{L}\rangle}[\![\mathrm{M}]\!]$ is subset of $\{\langle s_0, s_\omega\rangle \mid j \leq n \wedge s_0 \in \gamma^{\natural}(\dot{X}_j) \wedge s_\omega \in \gamma^{\natural}(\mathcal{B}_j)\}$, we refer to this latter set as S. It follows that $k = \texttt{Impact}^{\natural}(i, \mathcal{B}, \dot{X})$, Line 10.

By the soundness condition Def. (6.1), we have that $\textsc{Impact}_{i}^{\langle \textsc{highest},\mathbb{L}\rangle}(S) \leq k$. Since Impact is required to be monotone and S is an over-approximation of the filter semantics, we obtain that $\forall X \in \Lambda^{\langle \textsc{highest},\mathbb{L}\rangle}[\![\mathrm{M}]\!]$. $\textsc{Impact}_{i}^{\langle \textsc{highest},\mathbb{L}\rangle}(X) \leq k$. By the definition of $\mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$, cf. Eq. (4), it holds that $\Lambda^{\langle \textsc{highest},\mathbb{L}\rangle}[\![\mathrm{M}]\!] \dot{\sqsubseteq} \mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. Finally, by Theorem (5.4) and Eq. (5), it follows that $\mathrm{M} \models \mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. ∎

Theorem 6.3 (Soundness). *Let* $\mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$ *be the property of interest we want to verify for the neural network model* M. *The following implication holds:*

$$\texttt{Analyze}(\mathcal{D}^{\natural}, \mathrm{M}, i, \mathcal{B}, \texttt{Impact}^{\natural}) = k' \wedge k' \leq k \implies \mathrm{M} \models \mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$$

Proof. By Lemma (6.2), we have $\mathrm{M} \models \mathscr{B}^{\leq k'}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$ within the threshold k'. From Eq. (4) and the fact that $k' \leq k$, it straightforwardly follows that $\mathrm{M} \models \mathscr{B}^{\leq k}_{i,\langle \textsc{highest},\mathbb{L}\rangle}$. ∎

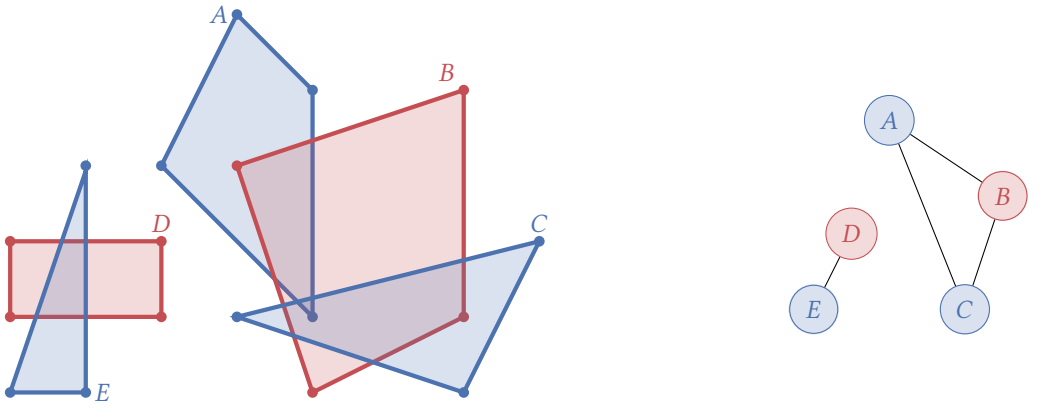
7 PROOF OF CONCEPT: Changes

The impact definitions Range and Outcomes are not well suited for the analysis of neural network models as they would return the same quantity for any input variable, so we focus on Changes. This impact definition is able to compare different inputs in the maximum number of changes in the classification of the network. In this section, we present `Changes`$^{\natural}$ as a sound implementation of Changes. First, we describe the implementation of `Changes`$^{\natural}$ (Algorithm 2), then we show the evaluation of Algorithm 1, instantiated with this implementation, in the context of neural network analysis.

The underlying idea is to group abstract elements that represent continuous regions together. To this end, our abstract domain employs a notion of disjunctive sets, which allows us to represent sets of distinct continuous regions. Specifically, we leverage the *disjunctive polyhedra abstract domain* $\mathcal{DP}$, defined as $\langle \wp_{\text{finite}}(\mathcal{P}), \subseteq^{\mathcal{DP}}\rangle$, where $\mathcal{P}$ represents the *convex polyhedra abstract domain* [Cousot and Halbwachs 1978] and $\wp_{\text{finite}}(\mathcal{P})$ is the set of all finite subset of $\mathcal{P}$. The domain $\mathcal{P}$ is also equipped with the function `project`$^{\mathcal{P}}$ computing the Fourier-Motzkin elimination algorithm [Dantzig and Eaves 1973]. Specifically, `project`$^{\mathcal{P}}$ takes as input a variable i and a polyhedron in d-dimensions, returning a polyhedron in $(d-1)$-dimensions, removing the variable i.

Algorithm 2 Algorithm for $\texttt{Changes}^{\natural}$

1: **function** $\texttt{Changes}^{\natural}(i, \mathcal{B}, \dot{X})$
2: **for** $\dot{X}_j$ **in** $\dot{X}$ **do**
3: **for** $\dot{X}_{j,k}$ **in** $\dot{X}_j$ **do**
4: $\dot{Y}_{j,k} \leftarrow \texttt{project}^{\mathcal{P}}(i, \dot{X}_{j,k})$
5: $\dot{M} \leftarrow \texttt{ConnectedComponentIndices}(\dot{Y})$
6: **return** $\max_{M \in \dot{M}} \max_{j \leq |\mathcal{B}|} |[k \in M \mid k \neq j]|$



(a) The two disjunctions of polyhedra $\{A \vee C \vee E\}$ and $\{B \vee D\}$.

(b) Equivalent graph representation.

Fig. 5. Example to demonstrate how the algorithm of the connected components works.

The function $\texttt{Changes}^{\natural}$ takes as input the variable of interest i, n output buckets $\mathcal{B} \in \mathcal{D}^{\natural n}$, and n disjunctions of polyhedra $\dot{X} \in \wp_{\text{finite}}(\mathcal{P})^n$ obtained from the backward pre-analysis. We can access each polyhedra and disjunctions of polyhedra via indexing as in a matrix-based structure, that is, $\dot{X} = \{\{P_{1,1} \vee \cdots \vee P_{1,p}\}, \ldots, \{P_{n,1} \vee \cdots \vee P_{n,q}\}\}$ where $p = |\dot{X}_1|$ and $q = |\dot{X}_n|$. For instance, $\dot{X}_j$ refers to the disjunction of polyhedra $\{P_{j,1} \vee \cdots \vee P_{j,k} \vee \ldots\}$, for $j \leq n$, and $\dot{X}_{j,k}$ refers to the polyhedra $P_{j,k} \in \mathcal{P}$.

The function iterates over each disjunction of polyhedra $\dot{X}_j \in \dot{X}$, Line 2. In turns, it iterates again over each polyhedron $\dot{X}_{j,k} \in \dot{X}_j$, Line 3. Then, it projects away the input variable i from each polyhedron $\dot{X}_{j,k}$ (Line 4). The projected polyhedra represent regions where i ranges on all possible values, considering all potential variations of this variable. The function $\texttt{ConnectedComponentIndices}$ performs a breadth-first search traversal and gathers the set of connected components, denoted as $\dot{M}$ at Line 5 (implementation details in Appendix A). In our context, a connected component represents the set of polyhedra that intersect together (after the elimination of the variable i). The underlying idea is that each connected component corresponds to the set of continuous regions reachable through variations of i. Note that, each connected component is implemented as a *multiset*, i.e., a set that allows multiple instances for each of its elements. This allows us to later exclude regions leading to the same bucket. Finally, $\texttt{Changes}^{\natural}$ determines the maximum count of changes across all connected components $\dot{M}$ and buckets $\mathcal{B}$ (Line 6). It counts the number of indices k in each connected component $M \in \dot{M}$ where k is not equal to j, thus excluding the polyhedra leading to the same output bucket j.

Example 7.1. In this small example we show how we compute the set of connected components for two disjunctions of polyhedra $\{A \vee C \vee E\}$ and $\{B \vee D\}$ (Figure 5a). We first abstract them to a graph representation, as depicted in Figure 5b, where each polyhedron corresponds to a node, and an edge is established between two polyhedra if they intersect. In the current example, A intersects with B and C, while D intersects with E. Consequently, the connected components are $\{\{A, B, C\}, \{D, E\}\}$. As an implementation detail, we do not directly return the polyhedra but the index of the disjunction they belong to. Consequently, the connected components are multisets $\{\{1, 2, 1\}, \{2, 1\}\}$ where 1 is the index of the first disjunction $\{A \vee C \vee E\}$ and 2 is the index of the second disjunction $\{B \vee D\}$.

The next result shows that our impact implementation $\mathsf{Changes}^\natural$ is sound.

Theorem 7.2. $\mathsf{Changes}^\natural$ *is a sound implementation of* Changes.

Proof (Sketch). We compare $\mathsf{Changes}^\natural$ of Algorithm 2 with Changes of Def. (4.8). The abstract implementation makes use of the disjunctive polyhedra domain. As a consequence, in the context of neural network analysis, the backward pre-analysis never abstracts two, or more, continuous regions of input configurations by the same abstract element. The projections (cf., Line 4) refers to checking that the initial state of traces is equal to x apart for the variable i, cf., $\{\sigma \mid \sigma_0 =_{\Delta\setminus\{i\}} x\}$ in Eq. (9). The connected components (Line 5) refer to gather together segments for each input configuration x and output z, that is, $\mathrm{SEG}_i^{\mathrm{HIGHEST}}(\{\sigma \mid \sigma_0 =_{\Delta\setminus\{i\}} x\}, z)$ in Eq. (9). Given a set of indices $M \in \dot{M}$ and a bucket index $j \leq |\mathcal{B}|$, the set $|\{k \in M \mid k \neq j\}|$ computed in Line 6 computes an over-approximation of $\dot{Q}_{i,x,z}^{\langle m,\mathcal{F}\rangle}$ in Eq. (8). Where the set of indices M and a bucket index j covers x and z, respectively; and the output descriptor $\langle m, \mathcal{F}\rangle$ is instantiated with $\langle \mathrm{HIGHEST}, \mathbb{L}\rangle$. The constraint $k \neq j$ ensures that only outcome values that differ from the current j (or z in the concrete) are considered in the count. We consider the maximum possible count among all $M \in \dot{M}$ and $j \leq |\mathcal{B}|$ (Line 6), ensuring we return an impact always higher than the concrete one computed by Changes. ∎

In the rest of this section, we investigate whether our prototype analysis, instantiated with $\mathsf{Changes}^\natural$ and the abstract domain $\mathcal{DP}$, successfully quantifies variations in the usage of input variables within neural networks, also called input features. Then, we evaluate the effectiveness of our framework by comparing it to two *feature importance metrics*.

Experimental setup. For our evaluation, we used public datasets from the online community platform Kaggle[2] to train several neural-network models. We focused on four datasets: "Red Wine Quality" [Cortez et al. 2009], "Prima Indians Diabetes" [Smith et al. 1988], "Rain in Australia" [Young 2019], and "Cure the Princess" [Unmoved 2023]. We pre-processed the "Rain in Australia" database and removed non-continuous input features, as our prototype does not support discrete input features yet. To preserve data consistency, since majority of daily weather observations were collected in Canberra, we eliminated the observations from other stations. As a result of this pre-processing step, we retained approximately 2000 entries, aligning with the sizes of other datasets. We trained about 700 networks per database by permuting the network structure and number of input features to obtain a uniform benchmark. The number of input features ranges from at least 3, to the number of attribute of the databases (after pre-processing), respectively, 10 attributes for "Red Wine Quality", 8 for "Prima Indians Diabetes", 17 for "Rain in Australia", and 13 for "Cure the Princess". The model size ranges from 2 to 6 hidden layers, each with 3 to 6 nodes, for a total of 6 to 36 nodes for each network model. All models were trained with Keras[3], using the Adam optimizer [Kingma and Ba 2014] with the default learning rate, and binary crossentropy as the loss function. Each model was trained for 150 iterations. The obtained neural network accuracy usually depends on the chosen subset of input features which is usually lower than the accuracy

[2] https://www.kaggle.com
[3] https://github.com/keras-team/keras

achieved in the literature. However, we remark that our study focuses on the impact analysis, therefore high accuracy is not needed in our benchmarks. All models used in our experiments are open source as part of the tool implementing our static analysis, called QSTAT. In particular, the analysis performed by our tool is $\text{QSTAT}(\text{M}, i) = \texttt{Analyze}(\mathcal{DP}, \text{M}, i, \mathcal{B}, \texttt{Changes}^\natural)$ where the output buckets represent all the target classes in the model M, one for each bucket: $\mathcal{B} = [\{ \bigwedge_{j' \leq m} x_{\text{L}l,j'} \leq x_{\text{L}l,j} \} \mid j \leq m]$.

To empirically check that our static analysis, specialized with $\texttt{Changes}^\natural$ and $\mathcal{DP}$, behaves similarly to CHANGES, we uniformly sample 1000 points in the input space of the network and then apply CHANGES to this set; we refer to this result as the *baseline*. Indeed, this approach is not sound as we could miss changes in the outcome not exploited by unsampled points, however it is overall a close-enough approximation of CHANGES. We compare the result of QSTAT with baseline employing four heuristics, called maximum common prefix length (MCPL), relaxed maximum common prefix length (RMCPL), Euclidean distance (ED), and Manhattan distance (MD), defined below.

Given two analyses F, F' (e.g., QSTAT and baseline), the heuristic (MCPL) first sorts the result of the analyses F, F' applied to all the input features, by decreasing order. The heuristic then returns the corresponding indices of the sorted results. Formally, it computes $I = \arg\text{sort}_i F(\text{M}, i)$ and $J = \arg\text{sort}_i F'(\text{M}, i)$ for the two analyses respectively, where arg sort returns the corresponding indices of the sorted list (by decreasing order), e.g., $\arg\text{sort}\langle 30, 65, 2, 60\rangle = \langle 2, 4, 1, 3\rangle$. Afterwards, (MCPL) retrieves the length of the maximal common prefix between I and J:

$$\text{MCPL}(I, J) \stackrel{\text{def}}{=} \begin{cases} 1 + \text{MCPL}(\langle i_1, \ldots, i_m\rangle, \langle j_1, \ldots, j_m\rangle) & \text{if } I = \langle i_0, \ldots, i_m\rangle \wedge J = \langle i_0, j_1, \ldots, j_m\rangle \\ 0 & \text{otherwise} \end{cases}$$

For instance, assuming $I = \langle 4, 1, 2, 3, 5\rangle$ and $J = \langle 4, 1, 2, 5, 3\rangle$, $\text{MCPL}(I, J) = 3$ since the maximum common prefix is $\langle 4, 1, 2\rangle$ of length 3. The relaxed variation (RMCPL) allows a 10% of overlap among quantity values, e.g., given $[F(\text{M}, i) \mid i \in \Delta] = \langle 0.4, 0.95, 0.6, 1\rangle$ and $[F'(\text{M}, i) \mid i \in \Delta] = \langle 30, 65, 2, 60\rangle$, we obtain $I = \arg\text{sort}\langle 0.4, 0.95, 0.6, 1\rangle = \langle 4, 2, 3, 1\rangle$ and $J = \arg\text{sort}\langle 30, 65, 2, 60\rangle = \langle 2, 4, 1, 3\rangle$, hence $\text{MCPL}(I, J)$ would return 0. However, a 10% of margin of error permits to swap the indices of values 0.95 and 1 in the first list obtaining $\langle 2, 4, 3, 1\rangle$, we say that I is equivalent to $\langle 2, 4, 3, 1\rangle$ (written $I \simeq \langle 2, 4, 3, 1\rangle$), hence $\text{MCPL}(\langle 2, 4, 3, 1\rangle, J) = 2$. Formally, we define RMCPL as the maximum of MCPL of all the possible equivalences:

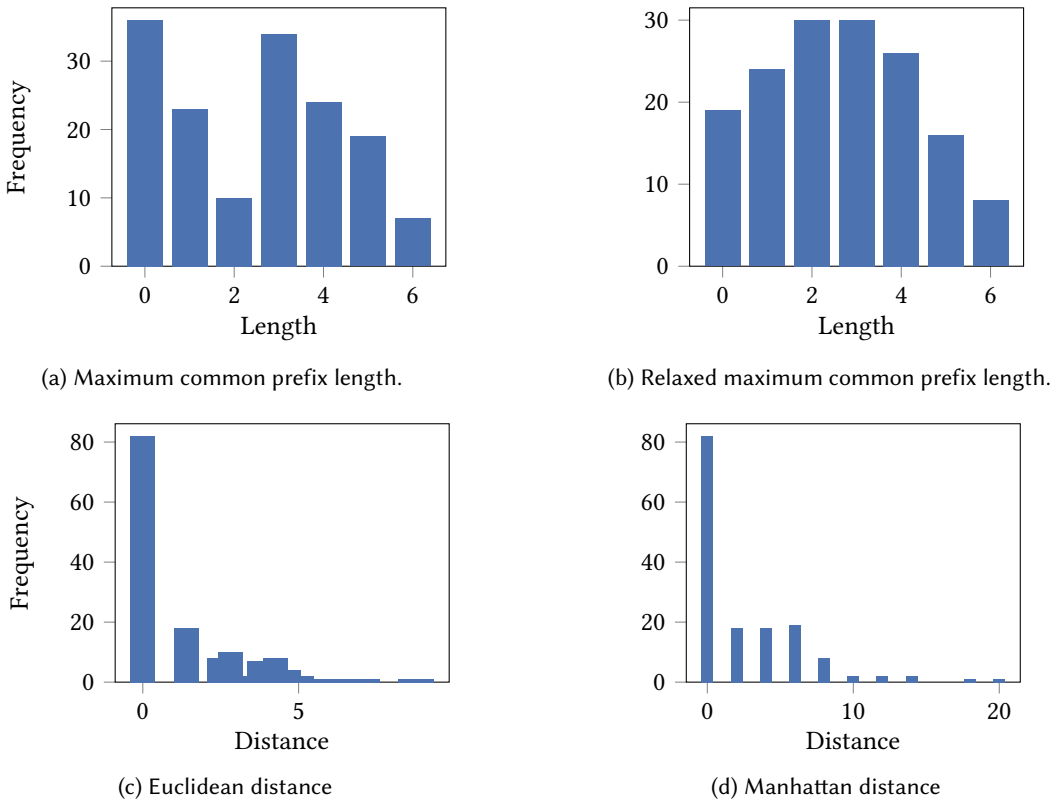
$$\text{RMCPL}(I, J) \stackrel{\text{def}}{=} \max\{\text{MCPL}(I', J') \mid I' \simeq I \wedge J' \simeq J\}$$

The Euclidean distance is defined as $\text{ED}(I, J) \stackrel{\text{def}}{=} \sqrt{\sum_k (I_k - J_k)^2}$, and the Manhattan distance as $\text{MD}(I, J) \stackrel{\text{def}}{=} \sum_k \mid I_k - J_k \mid$.

Quantifying Usage of Input Features. We first verify whether QSTAT produces a similar impact quantification with respect to CHANGES. To this end, we demonstrate QSTAT in comparison to baseline. This experiment uses the "Prima Indians Diabetes" dataset (full evaluation with all the databases in Appendix B). The evaluation, depicted in Figure 6, considers all the four heuristics.

The results confirm the similarity between QSTAT and baseline. Figure 6a and Figure 6b show the exact and relaxed maximum common prefix length results. In this context, the result of our analysis QSTAT is fairly similar to the baseline, this is even enhanced by the relaxed heuristic where close impact quantities (up to 10% difference) are softened together. In particular, Figure 6b shows that more than 100 test cases produce similar impact quantity to baseline up to, at least, the 3 most influent features.

Figure 6c and Figure 6d show the result for the Euclidean and Manhattan distance respectively. For these two heuristics, a less dense graph shows higher similarity. Low distance means that the two input vectors (or list of sorted indices as in our case) are close together. Confirming our expectancy, both Figure 6c and Figure 6d show high similitude in most of the test cases. Note that, the Euclidean distance (ED) returns a real number as distance value, thus bars do not necessarily correspond to a discrete values in the graph.



(a) Maximum common prefix length.

(b) Relaxed maximum common prefix length.

(c) Euclidean distance

(d) Manhattan distance

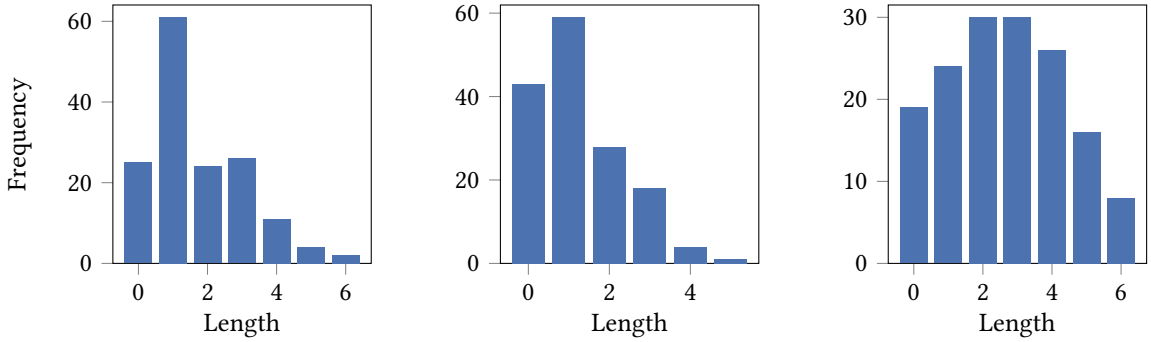
Fig. 6. "Prima Indians Diabetes" database, comparison between baseline and QStat.

Comparison with Stochastic Methods. In this second experiment, we compare to the results of our analysis with stochastic quantitative measures, known as feature importance metrics, which are used to determine the influence of input variables in machine learning models. Specifically, we compare to:

(RFE) A naïve feature importance metric that evaluates the changes in performance of a model when retrained without the feature of interest; in the following we call this metric *Retraining Feature Elimination*.

(PFI) *Permutation Feature Importance* [Breiman 2001], one of the most popular feature importance metrics, which monitors changes in performance when the values of the feature of interest are randomly shuffled.

This experiment uses the "Prima Indians Diabetes" dataset (full evaluation with all the datasets in Appendix B). Figure 7 demonstrates the comparison of baseline with, permutation feature importance (PFI), retraining feature importance (RFE), and QStat, respectively Figure 7a, Figure 7b, and Figure 7c. In this evaluation, we focus on the relaxed maximum common prefix length (rMCPL) heuristic. Like the (MCPL) approach, it highlights the most impactful features. However, the key advantage of rMCPL is the employment of a margin opf error when computing the common prefix, which allows for a clearer distinction when two analysis results are similar. In summary, we notice that our static analyzer QStat always achieves a higher similarity compared to the other two stochastic metrics considered.



(a) Baseline vs (PFI). (b) Baseline vs (RFE). (c) Baseline vs QSTAT.

Fig. 7. "Prima Indians Diabetes" database, comparison between baseline and (left to right) permutation feature importance, retraining feature importance, and QSTAT.

In conclusion, our evaluation demonstrates the effectiveness of the prototype of our static analysis in quantifying the impact of input features with respect to a formally defined impact property. Through comparison with stochastic metrics, we consistently observe a higher degree of similarity between the results produced by QSTAT and the baseline. The experiments conducted on the "Prima Indians Diabetes" dataset illustrate this similarity across various heuristics and distance measures. Appendix B contains the full overview of our experiments, we obtained similar results to the ones presented in this section. Notably, our approach stands out as the only one capable of addressing a formally defined impact property, providing a flexible framework that surpasses the hardcoded intuitions of the other two methods. Overall, these findings highlight the reliability and strength of QSTAT in assessing the significance of input features.

8 RELATED WORK

Our research draws inspiration from Barowy et al. [2014] where an empirical analysis explores the usage of data for debugging spreadsheet applications through stochastic sampling. Urban and Müller [2018] introduce a formal qualitative property, called *Input Data Usage*, to determine whether an input is used by a program or not. Their work establishes the theoretical foundations for understanding the usage of input variable. Building upon this foundations, subsequent studies [Mazzucato and Urban 2021; Urban 2020] extend the qualitative input data usage property through static analysis. These studies focused on computing the qualitative property as an indicator of bias in neural network models. Detecting bias is essential for assessing fairness and discriminatory behavior within these models. In our work, we further expand on this framework by considering quantitative properties of input data usage. While the qualitative property provides insights into the presence or absence of bias in neural networks, our work offers more flexibility. This extension, in the context of fairness verification, enables the identification of input variables more prone to bias. This knowledge facilitates a more efficient and targeted analyses, allowing researchers to leverage fairness techniques directly on the variables at a higher risk. Without our contribution, identifying biased variables would require running the qualitative analysis separately on each variable, which is computationally intensive and time-consuming. However, it is important to note that input data usage, as developed by Urban and Müller [2018] for programs beyond neural networks, is robust enough to account for the effect of non-determinism. In our prototype, specifically designed for neural network verification, we do not consider non-determinism. Therefore, if our quantitative analysis returns an impact value of 0, it holds that the input variable under consideration is definitely not used, but the other implication may

not hold. It would be interesting to explore the development of a new impact definition capable of resolving non-determinism within our generic framework. Furthermore, recent developments [Pal et al. 2022; Ranzato et al. 2021; Ranzato and Zanella 2020] are currently targeting other machine learning models, such as decision trees and support vector machines, developing formal methods to discover the usage of input variables. Quantitative analyses have the potential to enhance these techniques by leveraging a broader spectrum of possible results.

Quantitative information flow analyses, formerly introduced by the pioneering work of Denning [1982] and Gray [1991], measure the amount of leaked information about a secret through program input-output observations. The reason for such analyses is that, intuitively, safe programs are redeemed as unsafe by non-interference methods. For instance, consider the program that checks passwords: it does not exist a safe implementation since whenever the user inputs the correct password (public information), the program should prompt that the password is correct, showing to the public the secret information. Obviously, this program is understandably safe and secure but qualitative non-interference is not able to exploit it. While several effort has been already spent in the verification of the qualitative information flow properties (see Cousot [2019] for an overview) only recent interest has been raised regarding the–much more difficult to verify–quantitative counterpart. A series of papers [Clark et al. 2005a,b, 2007; Malacaria 2007] proposed static analyses to verify quantitative interference using Shannon's information theory and semantics of looping constructors, refining information-theoretic models. However, Smith [2009] argued against the extent of the Shannon entropy measure, proposing min-entropy as more accurate to measure the probability that an attacker guesses private data correctly in the first attempt. Clarkson et al. [2009] focused on non-deterministic program and proposed and variation of the Shannon entropy measure. Various static analysis techniques have been developed mainly based on abstract interpretation and symbolic execution [Assaf et al. 2017, 2016; Phan et al. 2012]. As a result, there is a need of a generic framework parametric in the definition of the quantitative measure. An unification proposed by Yasuoka and Terauchi [2014] extended previous results on inferring quantitative information flow. More recently, Zhang and Kaminski [2022] developed a calculus based on strongest-postcondition-style allowing quantitative reasoning of information flow, and Henzinger et al. [2023] generalized the hierarchy of safety and liveness properties to quantitative safety and liveness, exploring relationships among these new property classes. Conversely, our quantitative framework is derived from the qualitative input data usage property, which bears similarities, to some extent, with the qualitative information flow properties. Consequently, we drew inspiration from this field to construct our framework. More specifically, the method of assessing the impact of input variables within the threshold value of k (in the definition of our property of interest) originates from here. For instance, previous studies [Assaf et al. 2016; Phan et al. 2012] employ this approach, their analyses yield an upper bound k on the information leaks, indicating that the program leaks no more than k bits of information. The key difference among our framework and their work is the information we measure. Our analysis quantifies the effect of the input variables on the program outcome, focusing on the numerical aspect of the program, while quantitative information flow, for example, counts how many bits of information are used to compute the result [Assaf et al. 2017, 2016], or the probability of guessing the value of a non-input (private) variable [Smith 2009]. Which are orthogonal to our approach and more specific to security properties, mostly not adaptable in the context of data science software.

Input data usage and non-interference relates also with *Data Leakage* [Brownlee 2020, Chapter 4]. This property holds relevance for data science software, particularly in addressing leakage issues that arise from the training and test datasets during the process of training neural networks. The series of papers developed by Subotić et al. [Drobnjaković et al. 2022; Subotić et al. 2022, 2021] focuses on verifying data leakage within code notebooks and external libraries such as NumPy or pandas, bringing new challenges in the topic of formal verification. Within our framework, we could incorporate an adaptation of their work to encompass a notion of quantitative data leakage. Further development of this concept could be of particular interest.

9 CONCLUSION

We introduced a novel and automated approach to statically quantify the usage of input variables according to a given impact definition of interest. We focused on a prototype specifically designed for neural network analysis, demonstrating the applicability of our approach in the field of interpretability and explainability of machine learning. Additionally, we provided a rigorous formalization of the quantitative framework and formalized a static analysis, based on abstract interpretation, which provides a sound upper bound on the impact quantity.

As future work, one of our key goals is to expand our static analysis to encompass programs beyond neural networks. This extension brings new challenges, such as dealing with non-termination and non-determinism. To address these challenges, we plan to adapt our framework with the Cousot and Cousot [2012] work on non-termination.

Additionally, we recognize that non-determinism is usually tightly coupled with the probabilistic elements. Consequently, we are interested in exploring the use of *probabilistic abstract interpretation* [Cousot and Monerau 2012] within our framework. By doing so, we could potentially erase the side-effects caused by non-determinism and quantifying the true influence of input variables. This can be consider also for extending our framework to programs.

A broader comparison of our quantitative input data usage with close related properties, such as quantitative data leakage, could resort in interesting discoveries. Exploring this direction in the future would involve addressing further verification challenges posed by data science code. Especially, by the dynamic nature of code notebooks, which makes them susceptible to programming errors that are less common in other development environments. Notebooks allow users to execute code out of order or modify variables on the fly, leading to errors and inconsistencies in the analysis process. Such issues can have significant consequences, particularly in fields where data-driven decisions play a crucial role. Furthermore, data science notebooks heavily rely on external libraries like NumPy or pandas for dataset analysis and manipulation. Quantifying the impact of programs utilizing these libraries is a major challenge due to their complex and low-level implementations. Although the source code is usually available, analyzing it is a demanding task and often impractical. Therefore, employing custom abstractions of these libraries can facilitate verification by abstracting unnecessary details and focusing on the high-level logic of the libraries.

REFERENCES

Aws Albarghouthi. 2021. Introduction to Neural Network Verification. *Foundations and Trends® in Programming Languages* 7 (2021), 1–157. Issue 1–2. https://doi.org/10.1561/2500000051

Mounir Assaf, David A. Naumann, Julien Signoles, Éric Totel, and Frédéric Tronel. 2017. Hypercollecting semantics and its application to static analysis of information flow. *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 874–887. https://doi.org/10.1145/3009837.3009889

Mounir Assaf, Julien Signoles, Eric Totel, and Frédéric Tronel. 2016. The Cardinal Abstraction for Quantitative Information Flow. (6 2016). https://inria.hal.science/hal-01334604https://inria.hal.science/hal-01334604/document

Daniel W. Barowy, Dimitar Gochev, and Emery D. Berger. 2014. CheckCell: Data Debugging for Spreadsheets. *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, 507–523. https://doi.org/10.1145/2660193.2660207

Leo Breiman. 2001. Random forests. *Machine Learning* 45 (10 2001), 5–32. Issue 1. https://doi.org/10.1023/A:1010933404324

Jason Brownlee. 2020. *Data Preparation for Machine Learning*. https://books.google.fr/books?id=uAPuDwAAQBAJ

David Clark, Sebastian Hunt, and Pasquale Malacaria. 2005a. Quantified Interference for a While Language. *Electronic Notes in Theoretical Computer Science* 112 (1 2005), 149–166. Issue SPEC. ISS.. https://doi.org/10.1016/J.ENTCS.2004.01.018

David Clark, Sebastian Hunt, and Pasquale Malacaria. 2005b. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation* 15 (4 2005), 181–199. Issue 2. https://doi.org/10.1093/LOGCOM/EXI009

David Clark, Sebastian Hunt, and Pasquale Malacaria. 2007. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security* 15 (2007), 321–371. Issue 3. https://doi.org/10.3233/JCS-2007-15302

Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. 2009. Quantifying information flow with beliefs. *Journal of Computer Security* 17 (2009), 655–701. Issue 5. https://doi.org/10.3233/JCS-2009-0353

Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18 (9 2010), 1157–1210. Issue 6. https://doi.org/10.3233/JCS-2009-0393

Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47 (11 2009), 547–553. Issue 4. https://doi.org/10.1016/j.dss.2009.05.016

Patrick Cousot. 2002. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. (2002). http://www.di.ens.fr/~cousot

Patrick Cousot. 2019. Abstract Semantic Dependency. (2019), 389–410. https://doi.org/10.1007/978-3-030-32304-2_19

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: "A" unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages* Part F130756 (1 1977), 238–252. https://doi.org/10.1145/512950.512973

Patrick Cousot and Radhia Cousot. 2012. An abstract interpretation framework for termination. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages* (2012), 245–257. https://doi.org/10.1145/2103656.2103687

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages* (1 1978), 84–96. https://doi.org/10.1145/512760.512770

Patrick Cousot and Michael Monerau. 2012. Probabilistic abstract interpretation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7211 LNCS (2012), 169–193. https://doi.org/10.1007/978-3-642-28869-2_9/COVER

George B. Dantzig and B. Curtis Eaves. 1973. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory, Series A* 14 (5 1973), 288–297. Issue 3. https://doi.org/10.1016/0097-3165(73)90004-6

Dorothy Elizabeth Robling Denning. 1982. Cryptography and data security. (1982), 400.

Filip Drobnjaković, Pavle Subotić, and Caterina Urban. 2022. Abstract Interpretation-Based Data Leakage Static Analysis. (11 2022). https://arxiv.org/abs/2211.16073v1

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org

James W. Gray. 1991. Toward a mathematical foundation for information flow security. *Proceedings of the Symposium on Security and Privacy* (5 1991), 21–34. https://doi.org/10.1109/RISP.1991.130769

Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. 2023. Quantitative Safety and Liveness. (1 2023), 349–370. https://doi.org/10.1007/978-3-031-30829-1_17/COVER

T. Herndon, M. Ash, and R. Pollin. 2014. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge Journal of Economics* 38 (3 2014), 257–279. Issue 2. https://doi.org/10.1093/cje/bet075

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. (12 2014).

Puneet Kohli and Anjali Chadha. 2020. Enabling Pedestrian Safety Using Computer Vision Techniques: A Case Study of the 2018 Uber Inc. Self-driving Car Crash. , 261-279 pages. https://doi.org/10.1007/978-3-030-12388-8_19

Pasquale Malacaria. 2007. Assessing security threats of looping constructs. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages* (2007), 225–235. https://doi.org/10.1145/1190216.1190251

Denis Mazzucato and Caterina Urban. 2021. Reduced Products of Abstract Domains for Fairness Certification of Neural Networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12913 LNCS (2021), 308–322. https://doi.org/10.1007/978-3-030-88806-0_15

Abhinandan Pal, Francesco Ranzato, Caterina Urban, and Marco Zanella. 2022. Abstract Interpretation-Based Feature Importance for SVMs. (10 2022). https://arxiv.org/abs/2210.12456v1

Quoc-Sang Phan, Pasquale Malacaria, Oksana Tkachuk, and Corina S. Păsăreanu. 2012. Symbolic quantitative information flow. *ACM SIGSOFT Software Engineering Notes* 37 (11 2012), 1–5. Issue 6. https://doi.org/10.1145/2382756.2382791

Francesco Ranzato, Caterina Urban, and Marco Zanella. 2021. Fairness-Aware Training of Decision Trees by Abstract Interpretation. *International Conference on Information and Knowledge Management, Proceedings* (10 2021), 1508–1517. https://doi.org/10.1145/3459637.3482342

Francesco Ranzato and Marco Zanella. 2020. Abstract Interpretation of Decision Tree Ensemble Classifiers. *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (4 2020), 5478–5486. Issue 04. https://doi.org/10.1609/AAAI.V34I04.5998

Carmen M. Reinhart and Kenneth S. Rogoff. 2010. Growth in a Time of Debt. *American Economic Review* 100 (5 2010), 573–78. Issue 2. https://doi.org/10.1257/AER.100.2.573

Dana Scott and Christopher Strachey. 1971. Toward a mathematical semantics for computer languages. (1971).

Geoffrey Smith. 2009. On the foundations of quantitative information flow. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5504 LNCS (2009), 288–302. https://doi.org/10.1007/978-3-642-00596-1_21/COVER

Jack W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, and R.S. Johannes. 1988. Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. *Proceedings of the Annual Symposium on Computer Application in Medical Care* (11 1988), 261. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245318/

Pavle Subotić, Uroš Bojanić, and Milan Stojić. 2022. Statically Detecting Data Leakages in Data Science Code. *Proceedings of the 11th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis*, 16–22. https://doi.org/10.1145/3520313.3534657

Pavle Subotić, Lazar Milikić, and Milan Stojić. 2021. A Static Analysis Framework for Data Science Notebooks. *Proceedings - International Conference on Software Engineering* (10 2021), 13–22. https://doi.org/10.48550/arxiv.2110.08339

Unmoved. 2023. Cure The Princess. https://www.kaggle.com/datasets/unmoved/cure-the-princess

Caterina Urban. 2020. Perfectly Parallel Fairness Certification of Neural Networks. 185 (2020). https://doi.org/10.1145/3428253

Caterina Urban and Peter Müller. 2018. An abstract interpretation framework for input data usage. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10801 LNCS (2018), 683–710. https://doi.org/10.1007/978-3-319-89884-1_24/FIGURES/8

Ming Yang, Shige Wang, Joshua Bakita, Thanh Vu, F. Donelson Smith, James H. Anderson, and Jan-Michael Frahm. 2019. Re-Thinking CNN Frameworks for Time-Sensitive Autonomous-Driving Applications: Addressing an Industrial Challenge. *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 305–317. https://doi.org/10.1109/RTAS.2019.00033

Hirotoshi Yasuoka and Tachio Terauchi. 2014. Quantitative information flow as safety and liveness hyperproperties. *Theoretical Computer Science* 538 (6 2014), 167–182. Issue C. https://doi.org/10.1016/J.TCS.2013.07.031

Joe Young. 2019. Rain in Australia. https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package

Linpeng Zhang and Benjamin Lucien Kaminski. 2022. Quantitative Strongest Post A Calculus for Reasoning about the Flow of Quantitative Information. (2022). https://doi.org/10.1145/3527331

A THE ALGORITHM FOR THE INDICES OF CONNECTED COMPONENTS

Algorithm 3 describes the function `ConnectedComponentIndices`. It takes as input n disjunctions of polyhedra $\dot{Y} \in \dot{\mathcal{D}}^{\natural n}$ and returns a set of connected components. Each connected component is a multiset of indices, referring to indices of disjunction of polyhedra in $\dot{Y}$. Any two indices j and k that belong to the same connected component refer to two polyhedra that intersect, respectively inside the disjunctions $\dot{Y}_j$ and $\dot{Y}_k$.

Algorithm 3 Algorithm for `ConnectedComponentIndices`

```
1:  function ConnectedComponentIndices(Ẏ)
2:      Ṁ ← ∅                                          ▷ Set of connected components
3:      V ← [[FALSE, ...], ..., [FALSE, ...]]          ▷ Empty matrix of booleans, all set to FALSE
4:      Q ← ∅                                          ▷ Empty queue
5:      for Ẏ_j in Ẏ do
6:          for Ẏ_{j,k} in Ẏ_j do
7:              if ¬V_{j,k} then
8:                  M ← EmptyMultiset()                ▷ New component
9:                  Enqueue(Q, Ẏ_{j,k})
10:                 V_{j,k} ← TRUE
11:                 while Q ≠ ∅ do
12:                     Ẏ_{p,q} ← Dequeue(Q)
13:                     M ← M ∪ { p }
14:                     for Ẏ_w in Ẏ do
15:                         for Ẏ_{w,z} in Ẏ_w do
16:                             if ¬V_{w,z} ∧ Ẏ_{p,q} ⊓^P Ẏ_{w,z} then
17:                                 Enqueue(Q, Ẏ_{w,z})
18:                                 V_{w,z} ← TRUE
19:                 Ṁ ← Ṁ ∪ { M }
20:     return Ṁ
```

The function begins with an empty set $\dot{M}$ (Line 2), which will store the connected components, a boolean matrix V (Line 3), with all elements set to FALSE, and an empty queue Q (Line 4) to perform a breadth-first search traversal. The matrix V keeps track of visited elements to ensure efficient traversal.

Then, the function iterates through each polyhedron $\dot{Y}_{j,k}$ (Lines 5 and 6). If the polyhedron $\dot{Y}_{j,k}$ has not been visited yet ($\neg V_{j,k}$ at Line 7), a new component N is initialized as an empty multiset. The polyhedra $\dot{Y}_{j,k}$ is enqueued in Q and marked as visited ($V_{j,k} \leftarrow$ TRUE) in Lines 8,9, and 10. The breadth-first search dequeues the first element $\dot{Y}_{p,q}$ from the queue Q to the multiset M (Lines 12 and 13). Next, the function iterates through all the other polyhedra $\dot{Y}_{w,z}$. If $\dot{Y}_{w,z}$ has not been visited yet and intersects with $\dot{Y}_{p,q}$ (denoted by $\dot{Y}_{p,q} \sqcap^{\mathcal{P}} \dot{Y}_{w,z}$ at Line 16), $\dot{Y}_{w,z}$ is enqueued in Q for further exploration and marked as visited (Lines 17 and 18).

Once the breadth-first search traversal completes, the set M contains all the indices of disjunctions of polyhedra in the current connected component, thus we add M to the set of connected components $\dot{M}$ (Line 19). The function repeats this process for all unseen elements in $\dot{Y}$ until all connected components are discovered. Finally, `ConnectedComponentIndices` returns the set of sets of indices $\dot{M}$ (Line 20), representing the connected components of the polyhedra in $\dot{Y}$ based on their intersections.

B FULL EXPERIMENTAL OVERVIEW

This section contains the full overview of our experiment. All the Figures 8, 9, 10, and 11 are organized in a 4 rows by 3 columns setup, we dedicated one page for the experiments of each dataset. Each column corresponds to a different analysis. From left to right, permutation feature importance (PFI), retraining feature elimination (RFE), and QStat. Each row refers to an heuristic, namely, from top to bottom, maximum common prefix length (MCPL), relaxed maximum common prefix length (rMCPL), Euclidean distance (ED), and Manhattan distance (MD). See Section 7 for a detailed explanation on the heuristics and analyses used for out benchmarks.

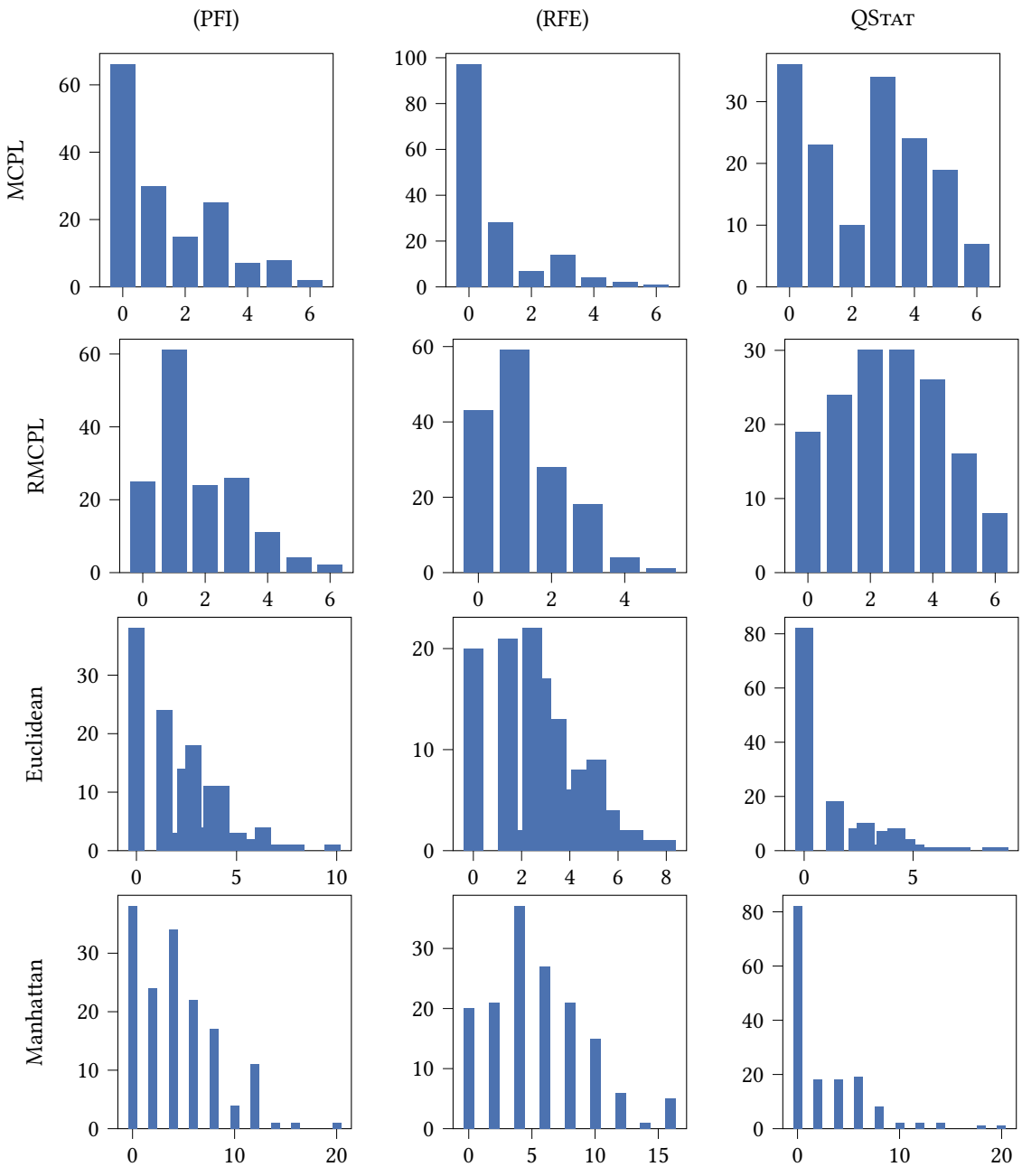


Fig. 8. Experimental overview regarding the “Prima Indians Diabetes” dataset.

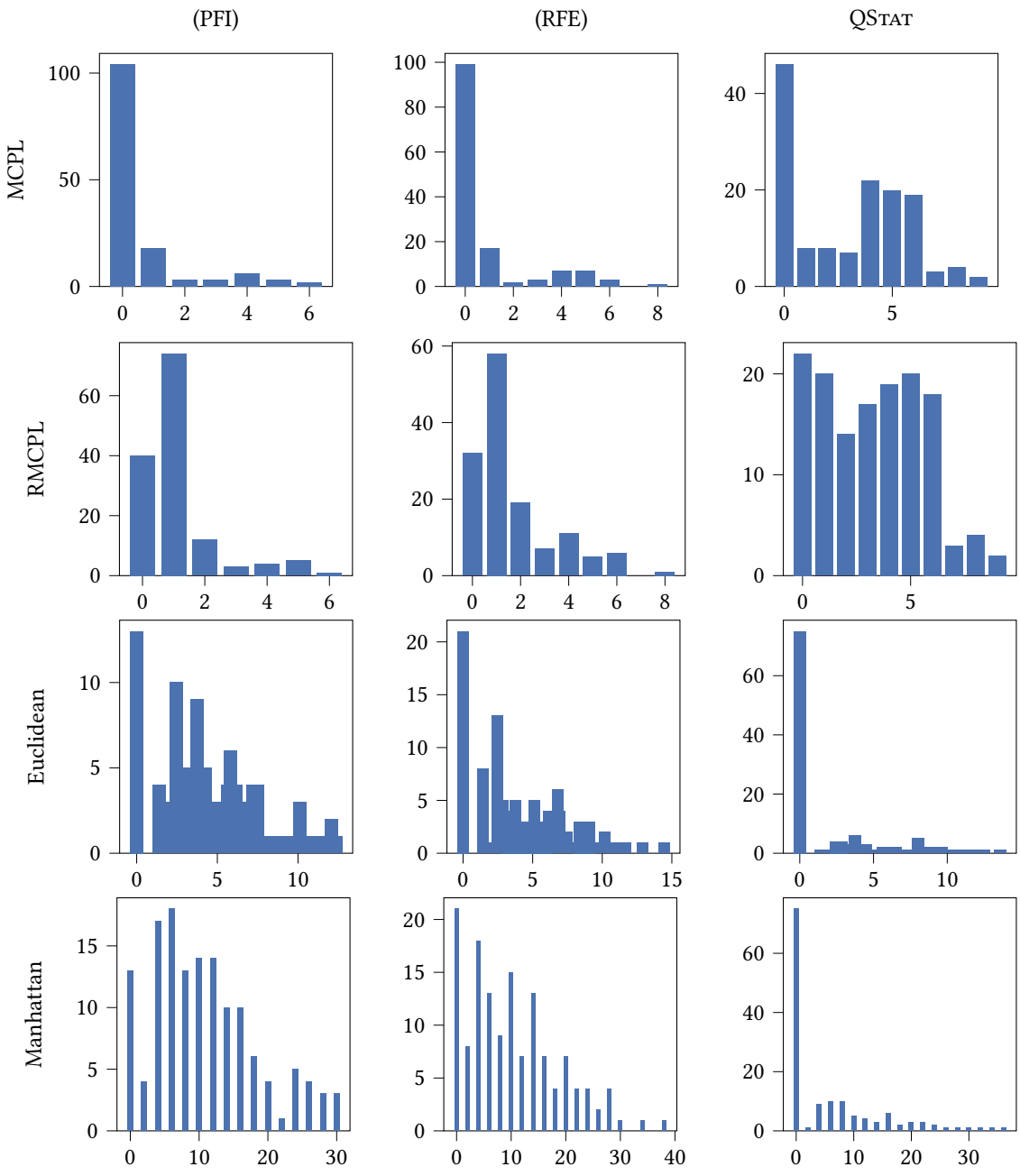


Fig. 9. Experimental overview regarding the “Red Wine Quality” dataset.

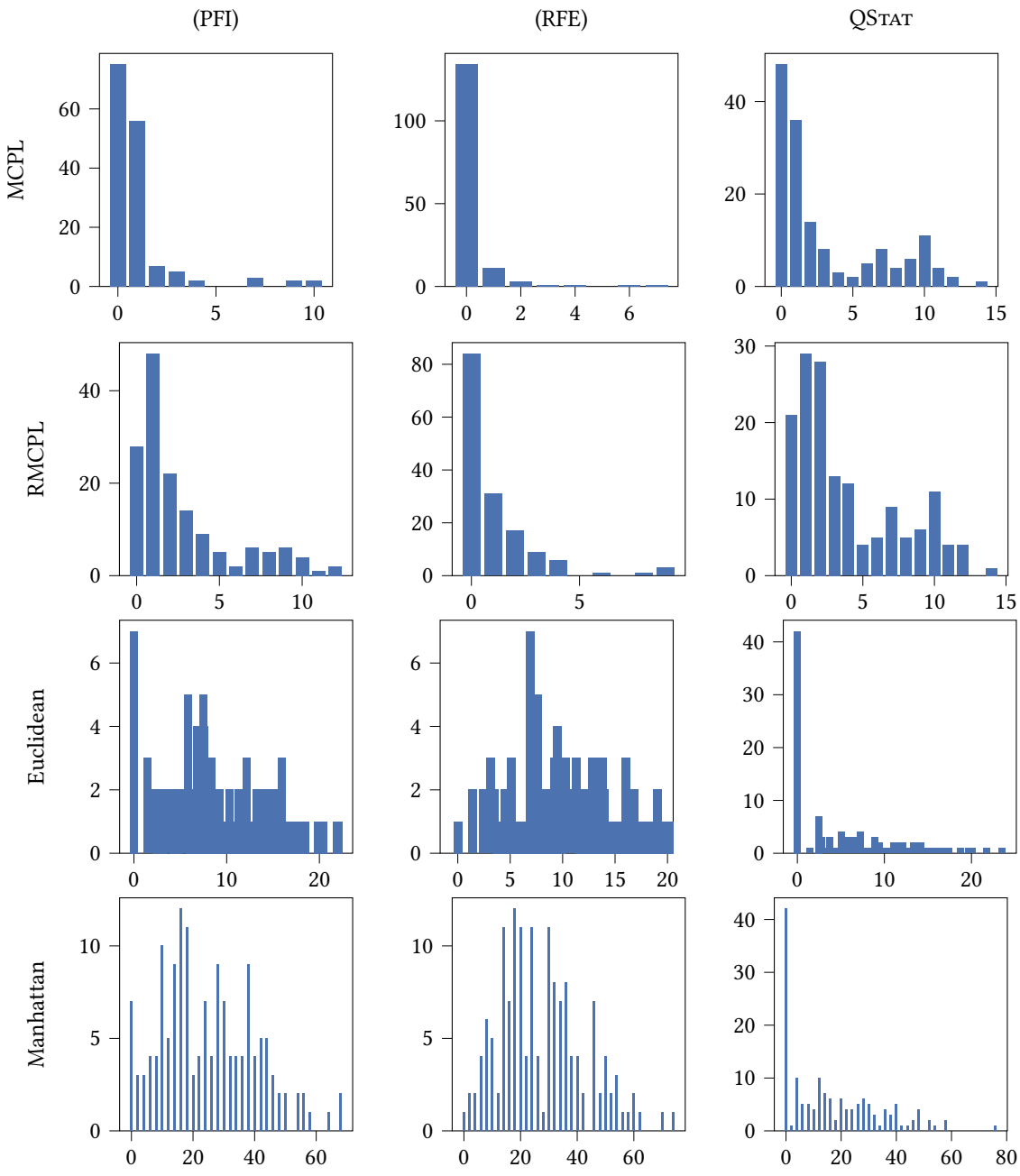


Fig. 10. Experimental overview regarding the “Rain in Australia” dataset.

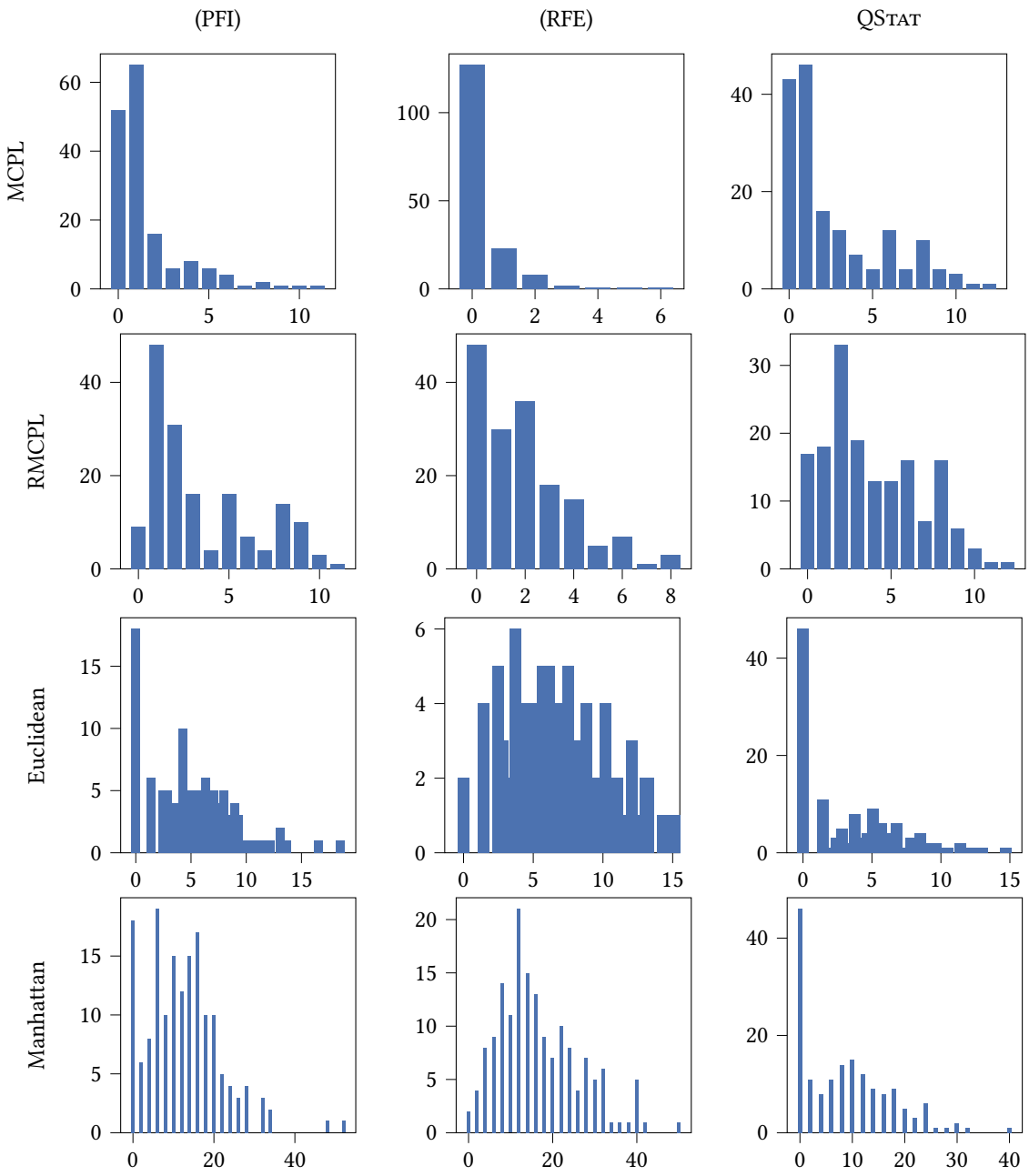


Fig. 11. Experimental overview regarding the “Cure the Princess” dataset.