Quantitative Input Usage Static Analysis

Denis Mazzucato, Marco Campion, Caterina Urban

Quantitative Input Usage Static Analysis

Denis Mazzucato, Marco Campion, and Caterina Urban

INRIA & ENS | PSL, {denis.mazzucato,marco.campion,caterina.urban}@inria.fr

Abstract. Programming errors in software applications may produce plausible yet erroneous results, without providing a clear indication of failure. This happens, for instance, when certain inputs have a disproportionate impact on the program result. To address this issue, we propose a novel quantitative static analysis for determining the impact of inputs on the program computations, parametrized in the definition of impact. This static analysis employs an underlying abstract backward analyzer and computes a sound over-approximation of the impact of program inputs, providing valuable insights into how the analyzed program handles them. We implement a proof-of-concept static analyzer to demonstrate potential applications.

1 Introduction

Disastrous outcomes may result from programming errors in safety-critical settings, 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 has disproportionate impact on the program computations compared to the developers' expectations. A notable example is the Reinhart and Rogoff article "Growth in a Time of Debt" [15], which was heavily cited to justify austerity measures around the world in the following years, and was later discovered to be flawed [12]. Notably, one of the several programming and methodological errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964, compromising the authors' conclusion. Hence, it is important to employ techniques that enhance the confidence in the usage of input variables.

In this direction, Barowy et al. [1] 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 qualitative properties about input data usage, e.g., only addressing whether an input variable is used or not [16, 17].

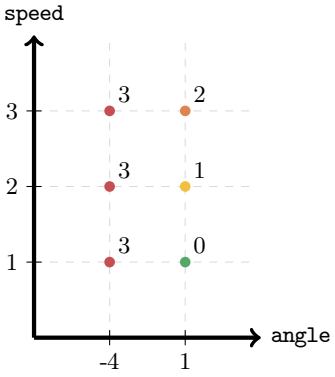
In this work, we present a novel quantitative input usage framework to discriminate between input variables with different impact on the outcome of a program. Such knowledge could either certify intended behavior or reveal potential flaws, by matching the developers' intuition on the expected impact of their input with the actual result of the quantitative study. We characterize the impact of an input variable with a notion of dependency between variables and outputs. Compared to other quantitative notions of dependency, e.g., quantitative information flow [8, 10], there are some key differences as the information we measure or the granularity of input contributions. Our framework is parametric in

Input preconditions:

$\texttt{angle} \in \{-4,1\}$
$\texttt{speed} \in \{1,2,3\}$

```
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: Program computing landing risk during the airstrip approach.

the choice of impact definition to better fit several factors, such as the program structure, the environment, the expertise of the developer, and the intuition of the researcher.

We propose an automatic always-terminating sound static analysis leveraging a backward analyzer to compute an over-approximation of the program semantics. In particular, this last component takes as input sets of program outputs, called *output buckets*, and computes an over-approximation of the input states leading to these buckets. Then, the end-user chooses the impact definition that best fits their needs, and our analysis applies such definition on the result of the previous phase. This approach, parametrized on the impact definition, ensures a more targeted and customizable analysis. We demonstrate the potential applications of our approach, by evaluating a proof-of-concept tool of our static analysis against a set of use cases.

Contributions We make the following contributions:

1. In Section 3, we develop a theoretical framework by abstract interpretation [6] to quantify the impact of input variables by considering two instances of impact: Outcomes and Range.
2. In Section 4, we present our static analysis and a possible abstract implementation of the impact instances.
3. Finally, Section 5 evaluates our proof-of-concept against three use cases (six more in Appendix B): a simplified program from the Reinhart and Rogoff article, a program extracted from the recent OpenAI keynote, and one from termination analysis.

2 Overview

In this section we present an overview of our quantitative analysis using a simple 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`, for the aircraft-airstrip alignment angle and the aircraft speed, respectively. A value of 1 represents a good alignment while -4 a non-aligned angle, whereas 1, 2, 3 denote low, medium,

and high speed[1]. A safer approach is indicated by lower speed. The landing risk coefficient combines the absolute landing angle and speed. The output variable `risk` is the danger level with possible values {0,1,2,3}, where 0 represents low danger and 3 high danger.

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. Our goal is to develop a static analysis capable of quantifying the contribution of each input variable to the computation of the output variable `risk`. Given the connection between qualitative input usage and information flow analyses [16], it comes natural to first explore *quantitative* information flow analyses, formerly introduced by the pioneering work of Denning [8] and Gray [10]. Such analyses measure information leakage about a secret through the concept of *entropy*, based on observations of the program's output values. In particular, we focus on the work proposed by Köpf and Rybalchenko [13] which quantifies an upper bound of the entropy of a program's input variables by computing an over-approximation of the set of input-output observations, sometimes called equivalence classes. They employ Shannon entropy, min-entropy, and other entropies through the enumeration of these equivalence classes and their respective sizes. The equivalence classes are partitions of the input space in which two input assignments belong to the same class whenever the program produces an equivalent output. For example, the equivalence classes of the program of Figure 1a are $\Pi(\mathrm{P}) = \{\{\langle x,y\rangle \mid \mathrm{P}(x,y) = z\} \mid z \in \{0,1,2,3\}\} = \{\{\langle 1,1\rangle\},\{\langle 1,2\rangle\},\{\langle 1,3\rangle\},\{\langle -4,1\rangle,\langle -4,2\rangle,\langle -4,3\rangle\}\}$. We develop our impact definitions by adapting their approach to our needs in three successive attempts.

First Attempt. The Shannon-entropy H computes the average uncertainty of input values based on observations of the program's outcomes, while min-entropy H_∞ computes the worst-case uncertainty. We consider min-entropy as closer to our needs since our aim is to discover the worst-case impact, i.e., the case in which a variable contributes the most.

$$H_\infty(\mathrm{P}) \stackrel{\text{def}}{=} \log_2 \frac{|\textsc{Input}_\mathrm{P}|}{|\Pi(\mathrm{P})|}$$

where $\textsc{Input}_\mathrm{P}$ is the set of all input values. Computing min-entropy on the program P of Figure 1a, we obtain $H_\infty(\mathrm{P}) = 0.58$, indicating that the input is highly guessable. Indeed, when the risk level is 3, the potential values of input variables are $\texttt{angle} = -4$ and $\texttt{speed} \in 1,2,3$; for all other output values, the input values are completely determined. Unfortunately, min-entropy lacks granularity and measures the uncertainty of the input variables collectively. Instead, our aim is to quantify the individual contributions.

Second Attempt. To address the previous issue, we exploit low and high labels for input variables, where the former are considered as public, available to the attacker, and the latter as secret. To assess the impact of each input variable, we prioritize one high variable at a time, considering all others as low variables. Subsequently, we compute the min-entropy of the labelled program to quantify the extent of the impact.

We define $\mathrm{P}^{\texttt{angle}}(x) \stackrel{\text{def}}{=} \langle \mathrm{P}(x,1),\mathrm{P}(x,2),\mathrm{P}(x,3)\rangle$ which represents the sequence of programs where `angle` is high and `speed` is low. Similarly, $\mathrm{P}^{\texttt{speed}}(y) \stackrel{\text{def}}{=} \langle \mathrm{P}(-4,y),\mathrm{P}(1,y)\rangle$ where `speed` is high and `angle` is low.

[1] We initially focus on discrete values to simplify the example and convey the concept. We expand to continuous inputs during the use case evaluation in Appendix B.1

Computing $H_\infty(\mathrm{P}^{\texttt{angle}})$ and $H_\infty(\mathrm{P}^{\texttt{speed}})$ yields 0 on both because all equivalence classes consist of singletons, meaning the number of inputs equals the number of outputs. Thus, there's no uncertainty in the value of `angle` given outputs of $\mathrm{P}^{\texttt{angle}}$, or in the value of `speed` given outputs of $\mathrm{P}^{\texttt{speed}}$. Indeed, observing the output $\langle 3,3,3\rangle$ from the program $\mathrm{P}^{\texttt{angle}}$ implies that `angle` is -4, while observing $\langle 0,1,2\rangle$ implies that `angle` is 1. The same applies to $\mathrm{P}^{\texttt{speed}}$ where observing $\langle 3,0\rangle$ implies `speed`$=1$, $\langle 3,1\rangle$ implies `speed`$=2$, and $\langle 3,2\rangle$ implies `speed`$=3$.

However, this approach does not isolate the contributions of high variables; these outcomes are combined into a tuple of values through the return statement and thus evaluated together. Consequently, min-entropy cannot distinguish the contribution of each input variable independently.

Third Attempt. An immediate solution is to develop a similar approach to the one used for the high-low variables, but instead of using min-entropy for the derived programs ($\mathrm{P}^{\texttt{angle}}$ and $\mathrm{P}^{\texttt{speed}}$), we count the number of outcomes of the partially-applied programs, cf. programs $\mathrm{P}(x,1)$,$\mathrm{P}(x,2)$, and $\mathrm{P}(x,3)$ for $\mathrm{P}^{\texttt{angle}}$; $\mathrm{P}(-4,y)$ and $\mathrm{P}(1,y)$ for $\mathrm{P}^{\texttt{speed}}$. These programs are referred to as $\mathrm{P}^{\texttt{angle}}_y$ and $\mathrm{P}^{\texttt{speed}}_x$ respectively. Therefore, applied to our example, we define the *outcomes* impact definition as $H_{\mathrm{O}}(\mathrm{P}^{\texttt{angle}}) \stackrel{\text{def}}{=} \max\{|\Pi(\mathrm{P}^{\texttt{angle}}_y)|\,|\, y\in\{1,2,3\}\}$ and $H_{\mathrm{O}}(\mathrm{P}^{\texttt{speed}}) \stackrel{\text{def}}{=} \max\{|\Pi(\mathrm{P}^{\texttt{speed}}_x)|\,|\, x\in\{-4,1\}\}$, we retain the maximum to obtain the worst-case scenario. We obtain $H_{\mathrm{O}}(\mathrm{P}^{\texttt{angle}})=2$ and $H_{\mathrm{O}}(\mathrm{P}^{\texttt{speed}})=3$. This means that variations of the value of `angle` result in at most 2 different outputs, while variations of the value of `speed` result in at most 3. Effectively, this is the first notion of impact capable of discriminating the contribution of each input variable on the program computation, exploiting the number of reachable outcomes from variations of the value of the input variable under consideration.

Impact Analysis. Overall, the approach proposed by Köpf and Rybalchenko [13] can be adapted to our needs. Nevertheless, their analysis grows exponentially with the number of low variables, which in our adaptation corresponds to the number of inputs, minus one. To address this limitation, we leverage an over-approximation of input-output observations of the program, focusing solely on the low variables. By doing so, we obtain the set of input configurations that lead to the same output value by variation on the value of high variables. As a result, our approach performs the program analysis only one time. A similar technique could also be used to mitigate such explosion in their work.

The impact definition H_{O} exploits *the number of* reachable outcomes resulting from variations in the input variable under consideration. We propose also a second definition that focuses on *the length of* extreme reachable values. Both definitions give us different insights on the program of Figure 1a. In particular, the first quantity indicates which variation reaches a greater number of output values, while the second tells us which variation in the values of input variables results in larger differences between output values. Below, we provide the intuition of both impact definitions.

First Impact Definition (Outcomes). Firstly, we show $\textsc{Outcomes}_i(\mathrm{P})$ (derived from H_{O}), where i is the input variable of interest and P the program under evaluation. This impact definition counts the number of different outputs reachable through variations in the input values. Table 1 shows the steps of the impact definition Outcomes. The intuition behind is that, for each input configuration $\langle x,y\rangle$ (column $\textsc{Input}_{\mathrm{P}}$), we gather together the set of input configurations $\langle x',y'\rangle$ resulting from variations of $\langle x,y\rangle$ on the

Table 1: Impact of for OUTCOMES(P) and RANGE(P) definitions for both `angle` and `speed` variables. Computational features are highlighted in blue.

VARIABLE	INPUT$_P$	RELEVANT TRACES	OUTPUTS	OUTCOMES	RANGE
`angle`	$\langle -4,1\rangle$	$\langle -4,1\rangle \to \langle 3\rangle, \langle 1,1\rangle \to \langle 0\rangle$	{3,0}	2	3
	$\langle -4,2\rangle$	$\langle -4,2\rangle \to \langle 3\rangle, \langle 1,2\rangle \to \langle 1\rangle$	{3,1}		
	$\langle -4,3\rangle$	$\langle -4,3\rangle \to \langle 3\rangle, \langle 1,3\rangle \to \langle 2\rangle$	{3,2}		
	$\langle 1,1\rangle$	$\langle 1,1\rangle \to \langle 0\rangle, \langle -4,1\rangle \to \langle 3\rangle$	{0,3}		
	$\langle 1,2\rangle$	$\langle 1,2\rangle \to \langle 1\rangle, \langle -4,2\rangle \to \langle 3\rangle$	{1,3}		
	$\langle 1,3\rangle$	$\langle 1,3\rangle \to \langle 2\rangle, \langle -4,3\rangle \to \langle 3\rangle$	{2,3}		
`speed`	$\langle -4,1\rangle$	$\langle -4,1\rangle \to \langle 3\rangle, \langle -4,2\rangle \to \langle 3\rangle, \langle -4,3\rangle \to \langle 3\rangle$	{3}	3	2
	$\langle -4,2\rangle$	$\langle -4,1\rangle \to \langle 3\rangle, \langle -4,2\rangle \to \langle 3\rangle, \langle -4,3\rangle \to \langle 3\rangle$	{3}		
	$\langle -4,3\rangle$	$\langle -4,1\rangle \to \langle 3\rangle, \langle -4,2\rangle \to \langle 3\rangle, \langle -4,3\rangle \to \langle 3\rangle$	{3}		
	$\langle 1,1\rangle$	$\langle 1,1\rangle \to \langle 0\rangle, \langle 1,2\rangle \to \langle 1\rangle, \langle 1,3\rangle \to \langle 2\rangle$	{0,1,2}		
	$\langle 1,2\rangle$	$\langle 1,1\rangle \to \langle 0\rangle, \langle 1,2\rangle \to \langle 1\rangle, \langle 1,3\rangle \to \langle 2\rangle$	{0,1,2}		
	$\langle 1,3\rangle$	$\langle 1,1\rangle \to \langle 0\rangle, \langle 1,2\rangle \to \langle 1\rangle, \langle 1,3\rangle \to \langle 2\rangle$	{0,1,2}		

input variable i alone, column RELEVANT TRACES. 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 $\langle x',y'\rangle$, column OUTPUTS. Therefore, among all the sets, we return the maximum size of these sets of output values, column OUTCOMES. As seen before, we obtain $\text{OUTCOMES}_{\texttt{angle}}(\text{P}) = 2$ and $\text{OUTCOMES}_{\texttt{speed}}(\text{P}) = 3$. 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`.

Second Impact Definition (RANGE). The intuition behind this second quantity RANGE_i is that, instead of counting how many reachable outputs are from variations of the input variable i, we yield the difference between maximum and minimum of reachable outputs. Following again Table 1, we can see that the range of reachable outputs from variations of `angle` is, at most, the interval [0,3], with a length of 3. Instead, the range of reachable outputs from variations of `speed` is, at most, the interval [0,2], with a length of 2. Therefore, we obtain $\text{RANGE}_{\texttt{angle}}(\text{P}) = 3$ and $\text{RANGE}_{\texttt{speed}}(\text{P}) = 2$, column RANGE. Hence, we gain the insight that varying the angle of approach might drastically alter the landing risk, whereas the speed has less influence. In contrast to the conclusion of OUTCOMES where `speed` has a greater impact than `angle`. 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).

Abstract Analysis. 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.

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.

3 Quantitative Input Data Usage

In this section we present some preliminaries on program computations, then we introduce our quantitative framework with the formal definitions of RANGE and OUTCOMES.

Program Semantics. The *semantics* of a program is a mathematical characterization of its behavior for 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 [6, 5]. 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.

Let $\Sigma^n \stackrel{\text{def}}{=} \{s_0 ... s_{n-1} \mid \forall i < n.\ s_i \in \Sigma\}$ be the set of all sequences of 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, $\Sigma^\infty \stackrel{\text{def}}{=} \{s_0 ... \mid \forall i \in \mathbb{N}.\ s_i \in \Sigma\}$ 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, let $\Sigma^\perp \stackrel{\text{def}}{=} \Sigma \cup \{\perp\}$. 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 $\sigma_\omega = \perp$ when $\sigma \in \Sigma^\infty$. To concatenate two sequences of states $\sigma, \sigma' \in \Sigma^{+\infty}$, we write $\sigma \cdot \sigma'$. It holds that $\sigma \cdot \epsilon = \epsilon \cdot \sigma = \sigma$ and $\sigma \cdot \sigma' = \sigma$ whenever $\sigma \in \Sigma^\infty$. To merge two sets of sequences $T \subseteq \Sigma^+$ and $T' \subseteq \Sigma^{+\infty}$, we write $T\ ;\ T' \stackrel{\text{def}}{=} \{\sigma \cdot s \cdot \sigma' \mid s \in \Sigma \wedge \sigma \cdot s \in T \wedge s \cdot \sigma' \in T'\}$ when a finite sequence in T terminates with the initial state of a sequence in T'.

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* is 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 [5]:

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

We write $\Lambda[\![\mathrm{P}]\!]$ to denote the trace semantics of a particular program P. 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 is facilitated by the design of a semantics that abstracts away from irrelevant details about program executions. In our work, we focus on *extensional* properties, namely, properties based on the observation of input-output relations of $\Lambda[\![\mathrm{P}]\!]$. Therefore, we employ the dependency semantics $\Lambda^{\rightsquigarrow} \in \wp(\Sigma \times \Sigma^\perp)$ [16] as an abstraction of the trace semantics removing intermediate steps, i.e., $\Lambda^{\rightsquigarrow} \stackrel{\text{def}}{=} \{\langle \sigma_0, \sigma_\omega \rangle \mid \sigma \in \Lambda[\![\mathrm{P}]\!]\}$. Starting from the dependency semantics, we define our property of interest – quantitative input data usage – and use abstract interpretation to systematically derive a semantics tailored to reason about this property.

Property. A *property* is specified by its extension, that is, the set of elements that manifest such a property [6]. We consider properties of programs, with dependency semantics in $\wp(\Sigma \times \Sigma^\perp)$, which are sets of sets of dependencies in $\wp(\wp(\Sigma \times \Sigma^\perp))$.

The strongest property of the dependency semantics $\Lambda^{\rightsquigarrow}$ is the standard *collecting semantics* $\Lambda^{\mathbb{C}} \in \wp(\wp(\Sigma \times \Sigma^\perp))$, defined as $\Lambda^{\mathbb{C}} \stackrel{\text{def}}{=} \{\ \Lambda^{\rightsquigarrow}\ \}$, which is satisfied only and exactly by $\Lambda^{\rightsquigarrow}$. Therefore, a program P satisfies a given property $F \in \wp(\wp(\Sigma \times \Sigma^\perp))$, written $\mathrm{P} \models F$, if and only if P belongs to F, or equivalently, its collecting semantics $\Lambda^{\mathbb{C}}$ is a subset of F, formally

$$\mathrm{P} \models F \ \Leftrightarrow\ \Lambda^{\mathbb{C}}[\![\mathrm{P}]\!] \subseteq F \tag{1}$$

Our goal is to quantify the impact of a specific input variable on the computation of the program. To this end, we introduce the notion of impact, denoted by the function $\textsc{Impact}_i \in \wp(\Sigma \times \Sigma^\perp) \to \mathbb{I}^{+\infty}_{\geq 0}$, which maps program semantics to a non-negative domain of quantities, where i represents the input variable of interest in the program under analysis. We implicitly assume the use of an *output descriptor* $\phi \in \Sigma^\perp \to \mathbb{I}^{\pm\infty}$ to determine the desired output of a program by observations on program states[2]. The output descriptor ϕ is generic enough to cover plenty of use cases, providing the end-user the flexibility to choose the interpretation and meaning of program outputs.

Example 1. Consider the landing alarm system presented in Figure 1. 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

[2] The option of returning $\pm\infty$ from the output descriptor is to deal with infinite traces, which do not have a final state ($\sigma_\omega = \perp$ for any $\sigma \in \Sigma^\infty$).

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

$$\phi(x) \stackrel{\text{def}}{=} \begin{cases} d & \text{if } x = \langle a,b,c,d \rangle \\ +\infty & \text{otherwise} \end{cases}$$

In other words, we are interested in the value of `risk` for terminating traces.

Given an impact definition of interest, we define the *k-bounded impact property* $\mathscr{B}_i^{\leq k} \in \wp(\wp(\Sigma \times \Sigma^\perp))$ as the set of dependency semantics with impact with respect to the input variable i below the threshold $k \in \mathbb{I}_{\geq 0}^{+\infty}$. Formally,

$$\mathscr{B}_i^{\leq k} \stackrel{\text{def}}{=} \{\Lambda^{\rightsquigarrow} \in \wp(\Sigma \times \Sigma^\perp) \mid \textsc{Impact}_i(\Lambda^{\rightsquigarrow}) \leq k\} \tag{2}$$

We require $\textsc{Impact}_i$ to be monotonic, i.e., for any $S,S' \in \wp(\Sigma \times \Sigma^\perp)$, it holds that if $S \subseteq S'$ then $\textsc{Impact}_i(S) \leq \textsc{Impact}_i(S')$. Intuitively, this ensures that an impact applied to an over-approximation of the program semantics can only produce a higher quantity, allowing for a sound k-bounded impact verification.

Next, we formalize the already introduced impact metrics Outcomes and Range. Given a program P and its variables $\mathbb{V}$, we assume program states are maps from variables to a numerical domain, i.e., $\Sigma = \mathbb{V} \to \mathbb{I}$. The set $\Delta \subseteq \mathbb{V}$ is the set of input variables. We write $\Sigma|_K = K \to \mathbb{I}$ for the program states reduced to the subset of variables $K \subseteq \mathbb{V}$. For instance $\Sigma|_\Delta$ is the set of states restricted to the input variables. The predicate $s =_K s'$ indicates that the two states $s,s' \in \Sigma^\perp\big|_K$, agree on the values of the variables in $K \subseteq \mathbb{V}$, or they are both $\perp$, formally $s =_K s' \Leftrightarrow (s \neq \perp \wedge s' \neq \perp \wedge \forall v \in K.\ s(v) = s'(v)) \vee s = s' = \perp$.

Outcomes. Formally $\textsc{Outcomes}_i \in \wp(\Sigma \times \Sigma^\perp) \to \mathbb{N}^{+\infty}$ counts the number of different output values reachable by varying the input variable $i \in \Delta$. Intuitively, for any possible input configuration $s \in \Sigma|_\Delta$, we gather the set $S \in \wp(\Sigma \times \Sigma^\perp)$ of all input-output state dependencies with an input configuration that is a variation of s on the input variable i, i.e., $\{\langle s_0,s_\omega \rangle \in S \mid s_0 =_{\Delta \setminus \{i\}} s\}$. Then, $\textsc{Outcomes}_i$ is the maximal cardinality of the output values $\{\phi(s_\omega) \mid \langle s_0,s_\omega \rangle \in S \wedge s_0 =_{\Delta \setminus \{i\}} s\}$. Formally,

$$\textsc{Outcomes}_i(S) \stackrel{\text{def}}{=} \sup_{s \in \Sigma|_\Delta} \left| \{\phi(s_\omega) \mid \langle s_0,s_\omega \rangle \in S \wedge s_0 =_{\Delta \setminus \{i\}} s\} \right| \tag{3}$$

where $|\cdot|$ is the cardinality operator, and $\sup(X)$ is the supremum operator, i.e., the smallest q such that $q \geq x$ for all $x \in X$. From the definition above, it is easy to note that $\textsc{Outcomes}_i(S)$ is monotone in the amount of dependencies S. That is, the more dependencies in input, the higher the impact as only more dependencies can satisfy the condition of Eq. (3) and hence increase the number of outcomes.

Range. The quantity $\textsc{Range}_i \in \wp(\Sigma \times \Sigma^\perp) \to \mathbb{R}_{\geq 0}^{+\infty}$ determines the length of the range of output values from all the possible variations in the input variable $i \in \Delta$. This definition employs the auxiliary function $\textsc{Length} \in \wp(\mathbb{I}^{\pm\infty}) \to \mathbb{I}_{\geq 0}^{+\infty}$, defined as follows: $\textsc{Length}(X) \stackrel{\text{def}}{=} \sup X - \inf X$ if $X \neq \emptyset$, where sup and inf are the supremum and infimum operators, while $\textsc{Length}(X) \stackrel{\text{def}}{=} 0$ otherwise. Formally,

$$\textsc{Range}_i(S) \stackrel{\text{def}}{=} \sup_{s \in \Sigma|_\Delta} \textsc{Length}(\{\phi(s_\omega) \mid \langle s_0,s_\omega \rangle \in S \wedge s_0 =_{\Delta \setminus \{i\}} s\})$$

Similarly to Outcomes, Range is monotone in the amount of dependencies S.

4 A Static Analysis for Quantitative Input Data Usage

In this section, we introduce a sound computable static analysis to determine an upper bound on the impact of an input variable i. The soundness of the approach leverages two elements: (1) an underlying abstract semantics $\Lambda^{\leftarrow}$ to compute an over-approximation of the dependency semantics $\Lambda^{\rightsquigarrow}$; and (2) a sound computable implementation of $\textsc{Impact}_i$, written $\texttt{Impact}_i^\sharp$, used in the property $\mathscr{B}_i^{\leq k}$. All proofs are in Appendix A.

To quantify the usage of an input variable, we need to determine the input configurations leading to specific output values. As our impact definitions $\textsc{Outcomes}_i$ and $\textsc{Range}_i$ measure over the different output values (i.e., $\phi(s_\omega)$) our underlying abstract semantics will be a *backward* (co-)reachability semantics starting from *disjoint* abstract post-conditions, over-approximating the (concrete) output values of the dependency semantics. Specifically, we abstract the concrete output values with an indexed set $B^\sharp \in \mathbb{D}^{\sharp n}$ of n disjoint *output buckets*, where $\langle \mathbb{D}^\sharp, \sqsubseteq, \bot^\sharp, \top^\sharp, \sqcup, \sqcap \rangle$ is an abstract state domain with concretization function $\gamma^\sharp \in \mathbb{D}^\sharp \to \wp(\Sigma^\bot)$. The choice of these output buckets is essential for obtaining a precise and meaningful analysis result.

For each output bucket $B_j^\sharp \in \mathbb{D}^\sharp$, our analysis computes an over-approximation of the dependency semantics restricted to the input configurations leading to $\gamma^\sharp(B_j^\sharp)$. More formally, let $\Lambda^{\rightsquigarrow}|_X \stackrel{\text{def}}{=} \{\langle s_0, s_\omega\rangle \in \Lambda^{\rightsquigarrow} \mid s_\omega \in X\}$ be the reduction of the dependency semantics $\Lambda^{\rightsquigarrow}$ to the dependencies with final states in X. Our static analysis is parametrized by an underlying backward abstract family[3] of semantics $\Lambda^{\leftarrow}[\![\mathrm{P}]\!] \in \mathbb{D}^\sharp \to \mathbb{D}^\sharp$ which computes the backward semantics $\Lambda^{\leftarrow}[\![\mathrm{P}]\!]B_j^\sharp$ from a given output bucket $B_j^\sharp \in \mathbb{D}^\sharp$. The concretization function $\gamma^{\leftarrow} \in (\mathbb{D}^\sharp \to \mathbb{D}^\sharp) \to \mathbb{D}^\sharp \to \wp(\Sigma \times \Sigma^\bot)$ employs $\gamma^\sharp$ to restore all possible input-output dependencies, i.e., $\gamma^{\leftarrow}(\Lambda^{\leftarrow}[\![\mathrm{P}]\!])B_j^\sharp \stackrel{\text{def}}{=} \{\langle s_0, s_\omega\rangle \mid s_0 \in \gamma^\sharp(\Lambda^{\leftarrow}[\![\mathrm{P}]\!]B_j^\sharp) \wedge s_\omega \in \gamma^\sharp(B_j^\sharp)\}$. We can thus define the soundness condition for the backward semantics with respect to the reduction of the dependency semantics.

Definition 1 (Sound Over-Approximation for $\Lambda^{\leftarrow}$). *For all programs* P*, and output bucket $B_j^\sharp \in \mathbb{D}^\sharp$, the family of semantics $\Lambda^{\leftarrow}$ is a* sound over-approximation *of the dependency semantics $\Lambda^{\rightsquigarrow}$ reduced with $\gamma^\sharp(B_j^\sharp)$, when it holds that:*

$$\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!]|_{\gamma^\sharp(B_j^\sharp)} \subseteq \gamma^{\leftarrow}(\Lambda^{\leftarrow}[\![\mathrm{P}]\!])B_j^\sharp$$

We define $\Lambda^\times \in \mathbb{D}^{\sharp n} \to \mathbb{D}^{\sharp n}$ as the backward semantics repeated on a set of output buckets $B^\sharp \in \mathbb{D}^{\sharp n}$, that is, $\Lambda^\times[\![\mathrm{P}]\!]B^\sharp \stackrel{\text{def}}{=} (\Lambda^{\leftarrow}[\![\mathrm{P}]\!]B_j^\sharp)_{j \leq n}$. Again, the concretization function $\gamma^\times \in (\mathbb{D}^{\sharp n} \to \mathbb{D}^{\sharp n}) \to \mathbb{D}^{\sharp n} \to \wp(\Sigma \times \Sigma^\bot)$ employs the abstract concretization $\gamma^\sharp$ to restore all possible input-output dependencies over all the output buckets, i.e., $\gamma^\times(\Lambda^\times[\![\mathrm{P}]\!])B^\sharp \stackrel{\text{def}}{=} \bigcup_{j \leq n}\{\langle s_0, s_\omega\rangle \mid s_0 \in \gamma^\sharp((\Lambda^\times[\![\mathrm{P}]\!]B^\sharp)_j) \wedge s_\omega \in \gamma^\sharp(B_j^\sharp)\}$.

Lemma 1 (Sound Over-Approximation for $\Lambda^\times$). *For all programs* P*, output buckets $B^\sharp \in \mathbb{D}^{\sharp n}$, and a family of semantics $\Lambda^{\leftarrow}$, the semantics $\Lambda^\times$ is a* sound over-approximation *of the dependency semantics $\Lambda^{\rightsquigarrow}$ reduced to $\bigcup_{j \leq n}\gamma^\sharp(B_j^\sharp)$:*

$$\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!]|_{\bigcup_{j \leq n}\gamma^\sharp(B_j^\sharp)} \subseteq \gamma^\times(\Lambda^\times[\![\mathrm{P}]\!])B^\sharp$$

Whenever the output buckets *cover* the whole output space, $\Lambda^\times$ is a sound over-approximation of $\Lambda^{\rightsquigarrow}$. The concept of covering for output buckets ensures that no final states of the dependency semantics, i.e. $\Omega^{\rightsquigarrow} \stackrel{\text{def}}{=} \{s_\omega \mid \langle s_0, s_\omega\rangle \in \Lambda^{\rightsquigarrow}\}$, are missed from the analysis.

[3] A family of semantics is a set of program semantics parametrized by an initialization.

Definition 2 (Covering). *We say that the output buckets* $B^\sharp \in \mathbb{D}^{\sharp^n}$ *cover the whole output space whenever* $\Omega^{\rightsquigarrow} \subseteq \bigcup_{j \leq n} \gamma^\sharp(B^\sharp_j)$.

Next, we expect a sound implementation $\texttt{Impact}^\sharp_i \in \mathbb{D}^{\sharp^n} \times \mathbb{D}^{\sharp^n} \to \mathbb{I}^{\pm\infty}$ to return a bound on the impact which is always higher than the concrete counterpart $\textsc{Impact}_i$.

Definition 3 (Sound Implementation). *For all output buckets* $B^\sharp$ *and family of semantics* $\Lambda^{\leftarrow}$, $\texttt{Impact}^\sharp_i$ *is a* sound implementation *of* $\textsc{Impact}$, *whenever*

$$\textsc{Impact}_i(\gamma^\times(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp)) \;\leq\; \texttt{Impact}^\sharp_i(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp, B^\sharp)$$

The next result shows that our static analysis is sound when employed to verify the property of interest $\mathscr{B}^{\leq k}_{\bar{i}}$ for the program P. That is, if $\texttt{Impact}^\sharp_i$ returns the bound k', and $k' \leq k$, then the program P satisfies the property $\mathscr{B}^{\leq k}_{\bar{i}}$, cf. $\mathrm{P} \models \mathscr{B}^{\leq k}_{\bar{i}}$.

Theorem 1 (Soundness). *Let* $\mathscr{B}^{\leq k}_{\bar{i}}$ *be the property of interest we want to verify for the program* P *and the input variable* $i \in \Delta$. *Whenever,*

(i) $\Lambda^{\leftarrow}$ *is sound with respect to* $\Lambda^{\rightsquigarrow}$, *cf. Def.* (1), *and*
(ii) $B^\sharp$ *covers the whole output space, cf. Def.* (2), *and*
(iii) $\texttt{Impact}^\sharp_i$ *is a sound implementation of* $\textsc{Impact}_i$, *cf. Def.* (3),

the following implication holds:

$$\texttt{Impact}^\sharp_i(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp, B^\sharp) = k' \;\wedge\; k' \leq k \;\Rightarrow\; \mathrm{P} \models \mathscr{B}^{\leq k}_{\bar{i}}$$

Finally, we define $\texttt{Range}^\sharp_i$ and $\texttt{Outcomes}^\sharp_i$ as possible implementations for $\textsc{Range}_i$ and $\textsc{Outcomes}_i$, respectively. We assume the underlying abstract state domain $\mathbb{D}^\sharp$ is equipped with an operator $\texttt{Project}^\sharp_i \in \mathbb{D}^\sharp \to \mathbb{D}^\sharp$ to project away the input variable i.

The definition of $\texttt{Outcomes}^\sharp_i$ first projects away the input variable i from all the given abstract values, then it collects all intersecting abstract values via the meet operator $\sqcap$. These intersections represent potential concrete input configurations where variations on the value of i lead to changes of program outcome, from a bucket to another. We return the maximum number of abstract values that intersects after projections:

$$\texttt{Outcomes}^\sharp_i(X^\sharp, B^\sharp) \stackrel{\text{def}}{=} \max\{|J| \mid J \in \texttt{IntersectAll}((\texttt{Project}^\sharp_i(X^\sharp_j))_{j \leq n})\} \quad (4)$$

Note the use of max instead of sup as in the concrete counterpart (Eq. (3)) since the number of intersecting abstract values is bounded by n, i.e., the number of output buckets. The function `IntersectAll` takes as input an indexed set of abstract values and returns the set of indices of abstract values that intersect together, defined as follows:

$$\texttt{IntersectAll}(X^\sharp \in \mathbb{D}^{\sharp^n}) \stackrel{\text{def}}{=} \{J \mid J \subseteq \mathbb{N} \wedge \forall j \leq n, p \leq n.\; j \in J \wedge p \in J \;\wedge\; X^\sharp_j \sqcap X^\sharp_p\}$$

Finding all the indices of intersecting abstract values is equivalent to find cliques in a graph, where each node represents an abstract value and an edge exists between two nodes if and only if the corresponding abstract values intersect. Therefore, `IntersectAll` can be efficiently implemented based on the graph algorithm by **?**].

Similarly, we define $\texttt{Range}^\sharp_i$ as the maximum length of the range of the extreme values of the buckets represented by intersecting abstract values after projections. In such case, we assume $\mathbb{D}^\sharp$ is equipped with an additional abstract operator $\texttt{Length}^\sharp \in \mathbb{D}^\sharp \to \mathbb{I}^{+\infty}_{\geq 0}$, which returns the length of the given abstract element, otherwise $+\infty$ if the abstract element is unbounded or represents multiple variables.

$$\texttt{Range}^\sharp_i(X^\sharp, B^\sharp) \stackrel{\text{def}}{=} \max\{\texttt{Length}^\sharp(K) \mid K \in I\} \quad (5)$$

$$\text{where } I \;=\; \{\sqcup\{B^\sharp_j \mid j \in J\} \mid J \in \texttt{IntersectAll}((\texttt{Project}^\sharp_i(X^\sharp_j))_{j \leq n})\}$$

In Appendix A.2, we prove that the abstract counterparts $\texttt{Range}^\sharp_i$ and $\texttt{Outcomes}^\sharp_i$ are sound over-approximations of the concrete counterparts $\textsc{Range}_i$ and $\textsc{Outcomes}_i$.

5 Experimental Results

The goal of this section is to highlight the potential of our static analysis for quantitative input data usage. We implemented a proof-of-concept tool in Python 3 that employs the `Interproc`[4] abstract interpreter to perform the backward analysis. Then, we exploited this tool to automatically derive a sound input data usage of three different scenarios. More use cases are shown in the Appendix B. As each impact result must be interpreted with respect to what the program computes, we analyze each scenario separately.

Growth in a Time of Debt. Reinhart and Rogoff article "Growth in a Time of Debt" [15] proposed a correlation between high levels of public debt and low economic growth, and was heavily cited to justify austerity measures around the world. One of the several errors discovered in the article is the incorrect usage of the input value relative to Norway's economic growth in 1964. The data used in the article is publicly available but not the spreadsheet file. We reconstructed this simplified example based on the technical critique by Herndon et al. [12], and an online discussion[5].

The program 1.1 below computes the cross-country mean growth for the public debt-to-GDP $60-90\%$ category, key point to the article's conclusions. The input data is the average growth rate for each country within this public dept-to-GDP category. The problem with this computation is that Norway has only one observation in such category, which alone could disrupt the mean computation among all the countries. Indeed, the year that Norway appears in the $60-90\%$ category achieved a growth rate of 10.2%, while the average growth rate for the other countries is 2.7%. With such high rate, the mean growth rate raised to 3.4%, altering the article's conclusions.

```
def mean_growth_rate_60_90(
    portugal1, portugal2, portugal3,
    norway1,
    uk1, uk2, uk3, uk4,
    us1, us2, us3):
  portugal_avg = (portugal1 + portugal2 + portugal3) / 3
  norway_avg = norway1
  uk_avg = (uk1 + uk2 + uk3 + uk4) / 4
  us_avg = (us1 + us2 + us3) / 3
  avg = (portugal_avg + norway_avg + uk_avg + us_avg) / 4
```

Listing 1.1: Program computing the mean growth rate in the $60-90\%$ category.

We assume growth rate values between -20% and 20% for all countries, consequentially, the output ranges are between these bounds as well. We instrumented the output buckets to cover the full output space in buckets of size 1, i.e., $\{t \leq \texttt{avg} < t+1 \mid -20 \leq t \leq 20\}$. Results for both Outcomes and Range are shown in Table 2. The analysis discovers that the Norway's only observation for this category `norway1` has the biggest impact on the output, as perturbations on its value are capable of reaching 10 different outcomes (cf. column `norway1`), while the other countries only have 5, 2, and 3, respectively for Portugal, UK, and US. The same applies to Range as the output buckets have size 1 and all the input perturbations are only capable of reaching contiguous buckets. Hence, we obtain the same exact results.

[4] `https://github.com/jogiet/interproc`
[5] `https://economics.stackexchange.com/q/18553`

Table 2: Quantitative input usage for the program 1.1 from Reinhart and Rogoff's article.

Impact	portugal1	portugal2	portugal3	norway1	uk1	uk2	uk3	uk4	us1	us2	us3
Outcomes	5	5	5	10	2	2	2	2	3	3	3
Range	5	5	5	10	2	2	2	2	3	3	3

Our analysis is able to discover the disproportionate impact of Norway's only observation in the mean computation, which would have prevented one of the several programming errors found in the article. From a review of the program 1.1, it is clear that Norway's only observation has a greater contribution to the computation, as it does not need to be averaged with other observations first. However, such methodological error is less evident when dealing with a higher number of input observations (1175 observations in the original work) and the computation is hidden behind a spreadsheet.

GPT-4 Turbo. The second use case we present is drawn from Sam Altman's recent OpenAI keynote in September 2023[6], where he presented the GPT-4 Turbo. This new version of the GPT-4 language model brings the ability to write and interpret code directly without the need of human interaction. Hence, as showcased in the keynote, the user could prompt multiple information to the model, such as related to the organization of a holiday trip with friends in Paris, and the model automatically generates the code to compute the share of the total cost of the trip and run it in background. In this environment, users are unable to directly view the code unless they access the backend console. This limitation makes it challenging for them to evaluate whether the function has been implemented correctly or not, assuming users have the capability to do so.

From the keynote, we extracted the function `share_division` which computes the user's share of the total cost of a holiday trip to Paris, given the total cost of the Airbnb, the flight cost, and the number of friends going on the trip.

```
def share_division(
    airbnb_total_cost_eur,
    flight_cost_usd,
    number_of_friends):
  share_airbnb = airbnb_total_cost_eur / number_of_friends
  usd_to_eur = 0.92
  flight_cost_eur = flight_cost_usd * usd_to_eur
  total_cost_eur = share_airbnb + flight_cost_eur
```

Listing 1.2: Program computing share division for holiday planning among friends.

Regarding the input bounds of `share_division`, users are willing to spend between 500 and 2000 for the Airbnb, between 50 and 1000 for the flight, and travel with between 2 and 10 friends. As a result, they expect their share, variable `total_cost_eur`, to be between 90 and 1900. To compute the impact of the input variables we choose the output buckets to cover the expected output space in buckets of size 100, i.e., $\{100t+90 \leq \texttt{total\_cost\_eur} < \min\{100(t+1)+90, 1900\} \mid 0 \leq t \leq 19\}$. The findings are

[6] https://www.youtube.com/live/U9mJuUkhUzk?si=HOzuH3-gr_kTdhCt&t=2330

Table 3: Results of the quantitative analysis.

Impact	airbnb_total_cost_eur	flight_cost_usd	number_of_friends
Outcomes	10	17	9
Range	1099	1709	999

(a) Program 1.2 to compute the share division.

	x	y
Outcomes	50	10
Range	499	99

(b) Program 1.3.

similar for both the Outcomes and Range analysis, see Table 3a. The input variable `flight_cost_usd` has the biggest impact on the output, as perturbations on its value are capable of reaching 17 different output buckets (resp. a range of 1709 output values), while the other two, `airbnb_total_cost_eur` and `number_of_friends`, only reach 10 and 9 output buckets (resp. have ranges of size 1099 and 999), respectively. These results confirm the user expectations about the proposed program from ChatGPT: the flight cost yields the biggest impact as it cannot be shared among friends.

Termination Analysis. This use case comes from the termination category of the software verification competition SV-COMP[7]. Program 1.3 is adapted from the work of Gopan and Reps [9, Fig. 1a]. Assuming both input positives, $\texttt{x},\texttt{y} \geq 0$, this program terminates in $\texttt{x}+1$ iterations if $\texttt{y} > 50$, otherwise it terminates in $\texttt{x}-2\texttt{y}+103$ iterations.

```
def example(x, y):
  counter = 0
  while x >= 0:
    if y <= 50:
      x += 1
    else
      x -= 1
    y += 1
    counter += 1
```

Listing 1.3: Timing analysis.

We define `counter` as the output variable, with output buckets defined as $\{10k \leq \texttt{counter} < 10(k+1) \mid 0 \leq k < 50\}$ and $\{\texttt{counter} \geq 500\}$. These output buckets represent cumulative ranges of iterations required for termination. The analysis results are illustrated in Table 3b, they show that the input variable `x` has the biggest impact. Modifying the value of `x` can result in the program terminating within any of the other 50 iteration ranges. On the other hand, perturbations on `y` can only result in the program terminating within 10 different iteration ranges. Such difference is motivated by the fact that `y` is only used to determine the number of iterations in the case where `y` is greater than 50, otherwise it is not used at all. Therefore, two values of `y`, e.g., y_0 and y_1, only result in two different ranges of iterations required to make the program terminate if either both of them are below 50 or $y_0 < 50 \wedge y_1 \geq 50$ or $y_0 \geq 50 \wedge y_1 < 50$, not in all the cases.

[7] https://sv-comp.sosy-lab.org/

The given results can be interpreted as follows: the speed of termination of this loop is highly dependent on the value of `x`, while `y` has a much smaller impact.

6 Conclusion

We presented an automated and sound analysis to statically quantify the usage of input variables based on a given impact definition. Our research is inspired by the works of Barowy et al. [1], which explores the data usage with a stochastic approach for debugging spreadsheet applications, and Urban and Müller [16], who introduce the qualitative property of "input data usage". In our work, we further advanced their work by considering quantitative properties of input data usage. While the qualitative property provides insights into the usage or not of program inputs, our work offers more flexibility providing a quantification of such a usage. We demonstrate potential applications with a proof-of-concept tool against a set of use cases.

As a future work, we plan to develop a modular tool to support the analysis in a solid and extensible way. We also plan to introduce heuristics able to automatically (or iteratively) infer the output buckets, since the choice of the starting buckets is essential to our quantitative analysis. Future directions could extend fairness certification studies on neural network models [17, 14]. Our quantitative notion introduces a quantitative fairness measure. Another promising direction is the exploration of new impact definitions capable of handling non-determinism at the transition system level. This could leverage probabilistic abstract interpretation [7]. Recent developments in quantitative information flow offer multiple ideas for novel impact definition: Zhang and Kaminski [19] developed a calculus based on strongest-postcondition-style allowing quantitative reasoning of information flow, and Henzinger et al. [11] generalized the hierarchy of safety and liveness properties to quantitative safety and liveness. It could also be interesting to exploit an impact definition to analyze the impact of abstract domains in static program analyzers, e.g., by using pre-metrics as defined in [2, 3]. Developing new relational abstract domains to discover specific non-linear variable relations could drastically improve the analysis precision, additionally taking into account input distributions. Further investigations of our analysis could also reveal new perspectives in the context of timing side-channel attacks [18], broadening the practical applications of our research.

References

[1] D. W. Barowy, D. Gochev, and E. D. Berger. Checkcell: data debugging for spreadsheets. *OOPSLA 2014*. https://doi.org/10.1145/2660193.2660207.
[2] M. Campion, M. Dalla Preda, and R. Giacobazzi. Partial (in)completeness in abstract interpretation: limiting the imprecision in program analysis. *POPL 2022*, . https://doi.org/10.1145/3498721.
[3] M. Campion, C. Urban, M. Dalla Preda, and R. Giacobazzi. A formal framework to measure the incompleteness of abstract interpretations. *SAS 2023*, . https://doi.org/10.1007/978-3-031-44245-2_7.
[4] H. Y. Chen, S. Flur, and S. Mukhopadhyay. Termination proofs for linear simple loops. *SAS 2012*. https://doi.org/10.1007/978-3-642-33125-1_28.
[5] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci. 2002*. https://doi.org/10.1016/S0304-3975(00)00313-3.

[6] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. *POPL 1977*. https://doi.org/10.1145/512950.512973.
[7] P. Cousot and M. Monerau. Probabilistic abstract interpretation. *ESOP 2012*. https://doi.org/10.1007/978-3-642-28869-2_9.
[8] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
[9] D. Gopan and T. W. Reps. Lookahead widening. *CAV 2006*. https://doi.org/10.1007/11817963_41.
[10] J. W. Gray. Toward a mathematical foundation for information flow security. *IEEE Computer Society 1991*. https://doi.org/10.1109/RISP.1991.130769.
[11] T. A. Henzinger, N. Mazzocchi, and N. E. Saraç. Quantitative safety and liveness. *FoSSaCS 2023*. https://doi.org/10.1007/978-3-031-30829-1_17.
[12] T. Herndon, M. Ash, and R. Pollin. Does high public debt consistently stifle economic growth? a critique of reinhart and rogoff. *Cambridge Journal of Economics 2014*. https://doi.org/10.1093/cje/bet075.
[13] B. Köpf and A. Rybalchenko. Automation of quantitative information-flow analysis. *SFM 2013*. https://doi.org/10.1007/978-3-642-38874-3_1.
[14] D. Mazzucato and C. Urban. Reduced products of abstract domains for fairness certification of neural networks. *SAS 2021*. https://doi.org/10.1007/978-3-030-88806-0_15.
[15] C. M. Reinhart and K. S. Rogoff. Growth in a time of debt. *American Economic Review 2010*. https://doi.org/10.1257/AER.100.2.573.
[16] C. Urban and P. Müller. An abstract interpretation framework for input data usage. *ESOP 2018*. https://doi.org/10.1007/978-3-319-89884-1_24.
[17] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang. Perfectly parallel fairness certification of neural networks. *OOPSLA 2020*. https://doi.org/10.1145/3428253.
[18] W. H. Wong. Timing attacks on RSA: revealing your secrets through the fourth dimension. *ACM Crossroads 2005*. https://doi.org/10.1145/1144396.1144401.
[19] L. Zhang and B. L. Kaminski. Quantitative strongest post: a calculus for reasoning about the flow of quantitative information. *OOPSLA 2022*. https://doi.org/10.1145/3527331.

A Proofs and Additional Definitions

This section contains the proofs of theorems, lemmas, and additional definitions of the material presented in Section 4.

A.1 Abstract semantics

When possible, we omit the program letter P in the notation.

Proof (Lemma (1), Sound Over-Approximation for $\Lambda^{\times}$). For all output buckets $B^{\sharp} \in \mathbb{D}^{\sharp^n}$, and a family of semantics $\Lambda^{\leftarrow}$, we prove that the semantics $\Lambda^{\times}$ is a sound over-approximation of the dependency semantics $\Lambda^{\rightsquigarrow}$ reduced to $\bigcup_{j\leq n}\gamma^{\sharp}(B_j^{\sharp})$.

$$
\begin{aligned}
&\gamma^{\times}(\Lambda^{\times})B^{\sharp} && \text{by definition of } \gamma^{\times}\\
&= \bigcup_{j\leq n}\{\langle s_0,s_\omega\rangle \mid s_0 \in \gamma^{\sharp}((\Lambda^{\times}(B^{\sharp}))_j) \wedge s_\omega \in \gamma^{\sharp}(B_j^{\sharp})\} && \text{by definition of } \Lambda^{\times}\\
&= \bigcup_{j\leq n}\{\langle s_0,s_\omega\rangle \mid s_0 \in \gamma^{\sharp}(((\Lambda^{\leftarrow}(B_t^{\sharp}))_{t\leq n})_j) \wedge s_\omega \in \gamma^{\sharp}(B_j^{\sharp})\} && \text{by indexed set property}\\
&= \bigcup_{j\leq n}\{\langle s_0,s_\omega\rangle \mid s_0 \in \gamma^{\sharp}(\Lambda^{\leftarrow}(B_j^{\sharp})) \wedge s_\omega \in \gamma^{\sharp}(B_j^{\sharp})\} && \text{by definition of } \gamma^{\leftarrow}\\
&= \bigcup_{j\leq n}\gamma^{\leftarrow}(\Lambda^{\leftarrow})B_j^{\sharp}
\end{aligned}
$$

From Def. (1), we obtain that $\forall j \leq n.\ \Lambda^{\rightsquigarrow}|_{\gamma^{\sharp}(B_j^{\sharp})} \subseteq \gamma^{\leftarrow}(\Lambda^{\leftarrow}(B_j^{\sharp}))$. Thus, by monotonicity of the union operator over set inclusion, it holds that $\bigcup_{j\leq n}\Lambda^{\rightsquigarrow}|_{\gamma^{\sharp}(B_j^{\sharp})} \subseteq \bigcup_{j\leq n}\gamma^{\leftarrow}(\Lambda^{\leftarrow}(B_j^{\sharp}))$. We conclude by:

$$
\begin{aligned}
\bigcup_{j\leq n}\Lambda^{\rightsquigarrow}|_{\gamma^{\sharp}(B_j^{\sharp})} &= \bigcup_{j\leq n}\{\langle s_0,s_\omega\rangle \in \Lambda^{\rightsquigarrow} \mid s_\omega \in \gamma^{\sharp}(B_j^{\sharp})\} && \text{by definition of } \Lambda^{\rightsquigarrow}|_X\\
&= \{\langle s_0,s_\omega\rangle \in \Lambda^{\rightsquigarrow} \mid s_\omega \in \bigcup_{j\leq n}\gamma^{\sharp}(B_j^{\sharp})\} && \text{by set definition}\\
&= \Lambda^{\rightsquigarrow}|_{\bigcup_{j\leq n}\gamma^{\sharp}(B_j^{\sharp})} && \text{by definition of } \Lambda^{\rightsquigarrow}|_X
\end{aligned}
$$

□

Proof ($\Lambda^{\times}$ is a Sound Over-Approximation of $\Lambda^{\rightsquigarrow}$ without reduction). For all output buckets $B^{\sharp} \in \mathbb{D}^{\sharp^n}$ covering the whole output space (cf. Def. (2)), and a family of semantics $\Lambda^{\leftarrow}$, we prove that the semantics $\Lambda^{\times}$ is a sound over-approximation of the dependency semantics $\Lambda^{\rightsquigarrow}$ without reduction.

$$
\begin{aligned}
\Lambda^{\rightsquigarrow} &= \{\langle s_0,s_\omega\rangle \in \Lambda^{\rightsquigarrow} \mid s_\omega \in \Omega^{\rightsquigarrow}\} && \text{by set definition}\\
&\subseteq \{\langle s_0,s_\omega\rangle \in \Lambda^{\rightsquigarrow} \mid s_\omega \in \bigcup_{j\leq n}\gamma^{\sharp}(B_j^{\sharp})\} && \text{by covering, Def. (2)}\\
&= \Lambda^{\rightsquigarrow}|_{\bigcup_{j\leq n}\gamma^{\sharp}(B_j^{\sharp})} && \text{by definition of } \Lambda^{\rightsquigarrow}|_X\\
&\subseteq \gamma^{\times}(\Lambda^{\times})B^{\sharp} && \text{by Lemma (1)}
\end{aligned}
$$

□

Proof (Theorem (1)*, Soundness).* Whenever,

(i) $\Lambda^{\leftarrow}$ is sound with respect to $\Lambda^{\rightsquigarrow}$, cf. Def. (1), and
(ii) $B^\sharp$ covers the whole output space, cf. Def. (2), and
(iii) $\texttt{Impact}_i^\sharp$ is a sound implementation of $\textsc{Impact}_i$, cf. Def. (3),

we prove that $\texttt{Impact}_i^\sharp(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp,B^\sharp)=k' \wedge k'\leq k \Rightarrow \mathrm{P}\models\mathscr{B}_i^{\leq k}$.

$$\begin{aligned} k\geq k' = \texttt{Impact}_i^\sharp(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp,B^\sharp) && \text{by hypothesis}\\ \geq \textsc{Impact}_i(\gamma^\times(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp)) && \text{by (iii)}\end{aligned}$$

By (i), (ii), and Lemma (1) we obtain that $\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!]\subseteq\gamma^\times(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp)$. Thus, from the monotonicity of $\textsc{Impact}_i$ it follows that $\textsc{Impact}_i(\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!])\leq\textsc{Impact}_i(\gamma^\times(\Lambda^\times[\![\mathrm{P}]\!]B^\sharp))$.

It follows that $\textsc{Impact}_i(\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!])\leq k'$. Therefore, by definition of $\mathscr{B}_i^{\leq k}$, cf. Eq. (2), it holds that $\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!]\in\mathscr{B}_i^{\leq k}$. From the definition of the collecting semantics $\Lambda^{\mathbb{C}}[\![\mathrm{P}]\!]$, it follows that $\{\Lambda^{\rightsquigarrow}[\![\mathrm{P}]\!]\}\subseteq\mathscr{B}_i^{\leq k}$. We conclude that $\mathrm{P}\models\mathscr{B}_i^{\leq k}$ by Eq. (1) applied to $\mathscr{B}_i^{\leq k}$ as F. □

A.2 Abstract Impact Definitions

In order to prove that the impact implementations (cf. Eq. (5) and Eq. (4)) are sound, we require the output buckets $B^\sharp$ to be *compatible* with the output descriptor ϕ. Intuitively, compatibility to ensure that counting buckets in the abstract instead of output values does not miss any concrete output value. This condition is necessary to prove that actual impact implementations is sound, cf. Def. (3).

Definition 4 (Compatibility). *Given the output buckets $B^\sharp\in\mathbb{D}^{\sharp n}$ and the output descriptor $\phi\in\Sigma^\perp\to\mathbb{I}^{\pm\infty}$, we say that $B^\sharp$ is* compatible *with ϕ, whenever it holds:*

$$\forall s_j\in\gamma^\sharp(B_j^\sharp),s_p\in\gamma^\sharp(B_p^\sharp).\ \phi(s_j)\neq\phi(s_p)\ \Rightarrow\ B_j^\sharp\neq B_p^\sharp$$

Furthermore, we assume a soundness condition on the project away operator to ensure that $\texttt{Project}_i^\sharp(s^\sharp)$ represents all the concrete states result of perturbations on the variable i from a state represented by $s^\sharp$.

Definition 5 (Soundness of $\texttt{Project}_i^\sharp$). *Given an abstract value $s^\sharp\in\mathbb{D}^\sharp$, for all $s\in\gamma^\sharp(s^\sharp)$, whenever it exists a state s' such that $s=_{\Delta\setminus\{i\}}s'$, then it holds that $s'\in\gamma^\sharp(\texttt{Project}_i^\sharp(s^\sharp))$.*

The above condition ensures that no intersection is missed, potentially spurious ones are allowed by the abstraction.

Before proceeding to prove that $\texttt{Outcomes}_i^\sharp$ is a sound implementation of $\textsc{Outcomes}_i$, we show that $\textsc{Outcomes}_i$ is bounded by the number of buckets when the conditions of covering and compatibility hold for the output buckets.

Lemma 2 ($\textsc{Outcomes}_i$ Upper Bound). *When the buckets $B^\sharp$ are compatible, cf. Def.* (4)*, and cover the whole output space, cf. Def.* (2)*, then $\textsc{Outcomes}_i(\Lambda^{\rightsquigarrow})\leq n$.*

Proof. We notice that $\textsc{Outcomes}_i(\Lambda^{\rightsquigarrow})\leq|\{\phi(s_\omega)\,|\,s_\omega\in\Omega^{\rightsquigarrow}\}|$ as the set of outputs for the dependency semantics is always bigger than any set of outputs. It is easy to note that the cardinality of $\{\phi(s_\omega)\,|\,s_\omega\in\Omega^{\rightsquigarrow}\}$ is upper bounded by n since any two output states s_ω,s'_ω that produce different output readings, i.e., $\phi(s_\omega)\neq\phi(s'_\omega)$, belong to different buckets, i.e., $s_\omega\in\gamma^\sharp(B_j^\sharp)\wedge s'_\omega\in\gamma^\sharp(B_p^\sharp)\wedge B_j^\sharp\neq B_p^\sharp$ (by compatibility, cf. Def. (4)*). Where the existence of the two buckets is guaranteed by covering (cf. Def.* (2)*). Therefore, there are at most n different output readings.* □

Lemma 3 ($\texttt{Outcomes}_i^\sharp$ is a Sound Implementation of $\textsc{Outcomes}_i$). *Let $i \in \mathbb{V}$ the input variable of interest, $\mathbb{D}^\sharp$ the abstract domain, $\Lambda^{\leftarrow}$ the family of semantics, and $B^\sharp \in \mathbb{D}^{\sharp n}$ the starting output buckets. Whenever the following conditions hold:*

(i) $\Lambda^{\leftarrow}$ is sound with respect to $\Lambda^{\rightsquigarrow}$, cf. Def. (1), and
(ii) $B^\sharp$ covers the whole output space, cf. Def. (2),
(iii) $B^\sharp$ is compatible with ϕ, cf. Def. (4), and
(iv) $\texttt{Project}_i^\sharp$ eliminates the variable i from the abstract value, cf. Def. (5),

we show that $\texttt{Outcomes}_i^\sharp$ is a sound implementation of $\textsc{Outcomes}_i$.

Proof. From (i), (ii), and the fact that $\textsc{Outcomes}_i$ is monotone, we obtain that $\textsc{Outcomes}_i(\Lambda^{\rightsquigarrow}) \leq \textsc{Outcomes}_i(\gamma^\times(\Lambda^\times)B^\sharp)$. By definitions of $\texttt{Outcomes}_i^\sharp$, cf. Eq. (4), and $\textsc{Outcomes}_i$, cf. Eq. (3), we need to show that:

$$\textsc{Outcomes}_i(\Lambda^{\rightsquigarrow}) = \sup_{s \in \Sigma|_\Delta} \mid \{\phi(s_\omega) \mid \langle s_0, s_\omega \rangle \in \Lambda^{\rightsquigarrow} \wedge s_0 =_{\Delta \setminus \{i\}} s\} \mid$$

$$\leq$$

$$\texttt{Outcomes}_i^\sharp(X^\sharp, B^\sharp) = \max \{|J| \mid J \in \texttt{IntersectAll}((\texttt{Project}_i^\sharp(X_j^\sharp))_{j \leq n})\}$$

where $X^\sharp = \Lambda^\times(B^\sharp)$. First, from (iii) and Lemma (2) we know that $\textsc{Outcomes}_i$ is limited by the number of buckets n. Notably, $\textsc{Outcomes}_i$ cannot be unbounded, but it has to be a number, at most n. Thus, it exists an initial state $\overline{s} \in \Sigma|_\Delta$ such that $\textsc{Outcomes}_i(\Lambda^{\rightsquigarrow}) = |S_{\overline{s}}|$, where $S_{\overline{s}} = \{\phi(s_\omega) \mid \langle s_0, s_\omega \rangle \in \Lambda^{\rightsquigarrow} \wedge s_0 =_{\Delta \setminus \{i\}} \overline{s}\}$. We conclude in case this cardinality is 0 as anything returned by $\texttt{Outcomes}_i^\sharp$ would be greater. In the other case, by covering (cf. (ii)), for all dependencies in $S_{\overline{s}}$ it exists a bucket $B_j^\sharp$ such that $s'_\omega \in \gamma^\sharp(B_j^\sharp)$.

Furthermore, by compatibility (cf. (iii)), for any pair of dependencies $\langle s_0, s_\omega \rangle, \langle s'_0, s'_\omega \rangle \in S_{\overline{s}}$ leading to two different outcomes $\phi(s_\omega) \neq \phi(s'_\omega)$, we have two different buckets $B_j^\sharp, B_p^\sharp$ such that $s_\omega \in \gamma^\sharp(B_j^\sharp)$ and $s'_\omega \in \gamma^\sharp(B_p^\sharp)$. Note that, by definition of $S_{\overline{s}}$ it holds that $s_0 =_{\Delta \setminus \{i\}} \overline{s} =_{\Delta \setminus \{i\}} s'_0$, by transitivity $s_0 =_{\Delta \setminus \{i\}} s'_0$.

Let us call $X_j^\sharp$ and $X_p^\sharp$ the corresponding abstract values from the backward analysis applied to the buckets $B_j^\sharp$ and $B_p^\sharp$, respectively. From the fact that $\Lambda^{\leftarrow}$ is sound with respect to $\Lambda^{\rightsquigarrow}$, cf. (i), it holds that $s_0 \in \gamma^\sharp(X_j^\sharp)$ and $s'_0 \in \gamma^\sharp(X_p^\sharp)$. By the soundness condition of $\texttt{Project}_i^\sharp$, cf. (iv), we obtain that both states s_0 and s'_0 belong to each other projection, i.e., $s_0 \in \gamma^\sharp(\texttt{Project}_i^\sharp(X_j^\sharp))$ and $s'_0 \in \gamma^\sharp(\texttt{Project}_i^\sharp(X_p^\sharp))$.

Finally, the function $\texttt{IntersectAll}$ applied to the projected preconditions $X_j^\sharp$ and $X_p^\sharp$ finds an intersection between the indices j and p as $\texttt{Project}_i^\sharp(X_j^\sharp) \sqcap \texttt{Project}_i^\sharp(X_p^\sharp)$ definitely holds since they share concrete states. Therefore, whenever it exists an intersection in the concrete, the two indices representing the respective precondition discovered by the backward analysis belong to the set J in Eq. (4). As a consequence, the maximum cardinality of J takes into account all the possible intersections in $S_{\overline{s}}$, hence $\texttt{Outcomes}_i^\sharp(X^\sharp, B^\sharp) \geq |S_{\overline{s}}|$. □

The approach is based on a similar reasoning for the abstract range implementation $\texttt{Range}_i^\sharp$ with the additional soundness condition for $\texttt{Length}^\sharp$. Such condition ensures that the abstract length is always greater than the concrete one, i.e., $\texttt{Length}^\sharp(s^\sharp) \geq \textsc{Length}(\{\phi(s) \mid s \in \gamma^\sharp(s^\sharp)\})$.

B Full Experimental Overview

As continuation of Section 5, we present a full experimental overview of our quantitative framework with additional use cases. First, we present the analysis for the program in Figure 1a described in Section 2. Then, we analyze a program extracted from the work of Barowy et al. [1] on Excel spreadsheets, regarding the computation of a student final grade. Afterwards, we analyze programs from the literature on termination analysis as SV-Comp, with two examples proposed by Chen et al. [4]. Finally, we analyze a synthetic program with loops (with non-linear invariants) to compute a linear expression. The experimental setup is the same as for the experiments presented in Section 5.

B.1 Landing Risk System

We start with the program described in Figure 1a, computing the landing risk of an aircraft, given the angle of approach and the aircraft speed. We are interested in studing what are the differences in their usage with our quantitative framework.

```
def landing_risk(angle, speed):
  landing_coeff = abs(angle) + speed
  if landing_coeff < 2 then
    risk = 0
  else if landing_coeff > 5 then
    risk = 3
  else
    risk = floor(landing_coeff - 2)
```

Listing 1.4: Program computing the landing risk of an aircraft.

We apply our quantitative framework to the `landing-risk` function with the input bounds of Figure 1a, $(\texttt{angle} = -1 \vee \texttt{angle} = 4) \wedge (\texttt{speed} = 1 \vee \texttt{speed} = 2 \vee \texttt{speed} = 3)$. Each output risk corresponds to an output bucket, $\{\texttt{risk} = n \mid n \in \{0,1,2,3\}\}$. The analysis results are presented in Table 4 and validate the manual computations presented in the overview. Variations in value of the input `angle` only result in a single change to the output, while variations in the `speed` input can lead to two output modifications (see column Outcomes). In terms of the length of ranges (column Range), modifications to the `angle` input cover the entire spectrum of output values, whereas to the `speed` input only span a range of 2.

One could also be interested in the input bounds to cover the full continuous input space for the aircraft angle of approach, where $(-4 \leq \texttt{angle} \leq 4) \wedge (1 \leq \texttt{speed} \leq 3)$, see Figure 2. In this instance, starting from the same output buckets, the precision of the abstraction drastically decreases as it only employs convex abstract domains and thus not able to capture the full input space symmetric characteristics around 0. Indeed, the analysis result, Figure 3e, is unable to reveal any difference in the input usage of both input variables. The abstract backward computation of each bucket results in an abstract region that intersects with any other (after projections). As a consequence, Outcomes and Range are unable to provide any meaningful information, second row of Table 4.

A possible approach to overcome the non-convexity of the input space is to split the input space into two subspaces (as a bounded set of disjunctive polyhedra), $-4 \leq \texttt{angle} \leq 0$ and $0 \leq \texttt{angle} \leq 4$, third and fourth row of Table 4. In the first subset $-4 \leq \texttt{angle} \leq 0$, we are able to perfectly captures the input regions that lead

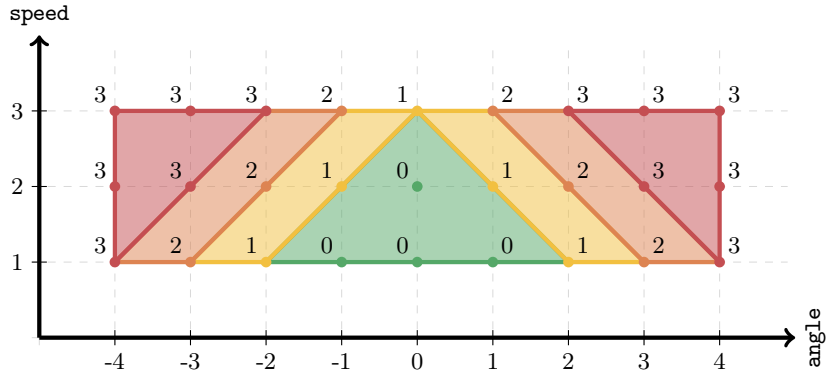
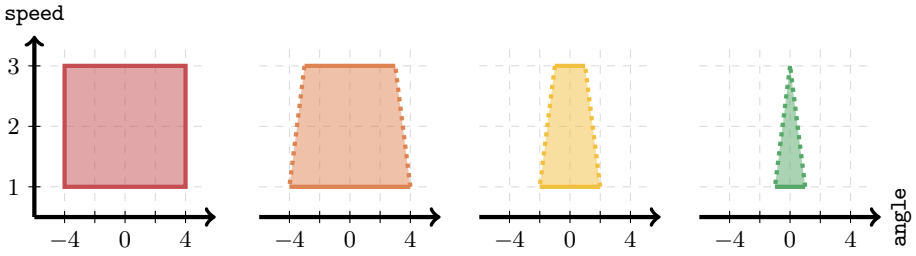
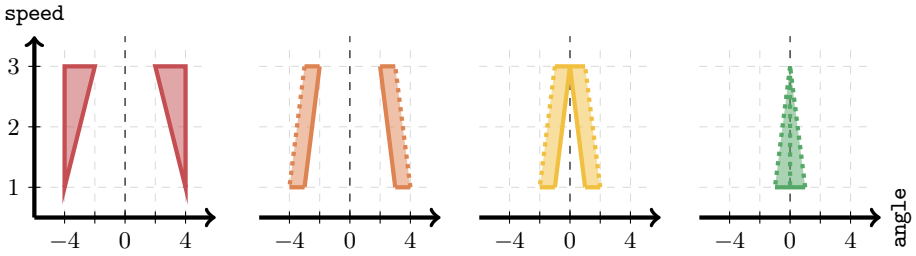


Fig. 2: Input space composition with continuous input values, where $-4 \leq \texttt{angle} \leq 4$ and $1 \leq \texttt{speed} \leq 3$



(a) {risk=3} (b) {risk=2} (c) {risk=1} (d) {risk=0}

(e) Analysis result using the polyhedra domain.



(f) {risk=3} (g) {risk=2} (h) {risk=1} (i) {risk=0}

(j) Analysis result after splitting the input space into two subspaces around angle=0.

Fig. 3: Analysis results using Interproc.

to each output bucket with our abstract analyzer. Therefore, we are able to recover the information that the input configurations from the bucket {risk=3} do not intersect with the ones from the bucket {risk=0} after projecting away the axis speed. As the end, our analysis notices that variations in the value of the input angle results in three possible output values, while variations in the speed input lead to two. Similarly, regarding the range of values, variations in the angle input cover the entire spectrum of output values, whereas to the speed input only span a range of 2 since it exists no input

value such that modifications in the `speed` value could obtain a range of output values bigger than 2. The same reasoning applies to the other subspace with $0 \leq \texttt{angle} \leq 4$.

Input Bounds		Buckets	Outcomes		Range	
			angle	speed	angle	speed
$\texttt{angle} = -1 \vee \texttt{angle} = 4$	$1 \leq \texttt{speed} \leq 3$	$\{\texttt{risk} = n \mid n \in \{0,1,2,3\}\}$	1	2	3	2
$-4 \leq \texttt{angle} \leq 4$			3	3	3	3
$-4 \leq \texttt{angle} \leq 0$			3	2	3	2
$0 \leq \texttt{angle} \leq 4$			3	2	3	2

Table 4: Quantitative input usage for the landing risk program 1.4.

We present the program 1.5, computing the weighted average for the final grade of a student given the grades of the homework assignments, quizzes and exams. This use case is extracted from an Excel spreadsheet found in the CheckCell benchmark suite [1].

```
def final_grade(
    HW1, HW2, HW3, HW4,
    QZ1, QZ2, QZ3, QZ4,
    EX1, EX2):
  HW_coeff = 0.2
  QZ_coeff = 0.3
  EX_coeff = 0.5
  HW_avg = HW_coeff * (HW1 + HW2 + HW3 + HW4) / 4
  QZ_avg = QZ_coeff * (QZ1 + QZ2 + QZ3 + QZ4) / 4
  EX_avg = EX_coeff * (EX1 + EX2) / 2
  avg = HW_avg + QZ_avg + EX_avg
```

Listing 1.5: Program computing the landing risk of an aircraft.

In the original work, Barowy et al. [1] proposed a stochastic analysis to discover possible transposition errors in this spreadsheet function, errors where digits are swapped within cells. The student grades are provided in advance for this analysis, [84,77,92,93,87,90,85,91,84,78]. We can encode such property by defining the input bounds as $v_i = x_{i,0}x_{i,1} \vee v_i = x_{i,1}x_{i,0}$, where v_i is the input variable and $x_{i,0}$ and $x_{i,1}$ are the two digits of the value of v_i. Furthermore, a student is considered to pass the course if the final grade is greater than or equal to 85. The output buckets to encode this property are $\{\texttt{avg} \geq 85\}$ and $\{\texttt{avg} < 85\}$, respectively the student passes or fails the course.

In the encoding presented above, the analysis result shows that (single) transposition errors could affect the student's graduation if they occur in the homework assignments `HW1`, `HW2`, or `QZ1`. Note that performing the Range analysis would have no meaning as output buckets are not limited in size.

Furthermore, additional information on the flow of information can be obtained by considering different output buckets. For instance, consider the output buckets to convert grades from [0,100] to [0,10]: $\{0 \leq \texttt{avg} < 15\}$,$\{15 \leq \texttt{avg} < 25\}$,...,$\{95 \leq \texttt{avg} \leq 100\}$, one bucket each final grade rounded up. The analysis shows that (single) transposition errors are able to affect, at most, one grade point in the final grade for homework assignments and quizzes. Instead, errors in exam grades are far more impactful, as they can affect up to 3 grade points in the final grade. As one can notice, with the current input bounds,

we allow only two values for each input variable, thus one expects a maximum possible influence of one point, but instead the analysis returns up to 3. This imprecision is caused again by the non-convex input space, therefore the abstraction considers $v_i = x_{i,0}x_{i,1} \vee v_i = x_{i,1}x_{i,0}$ as a convex region, $\min\{x_{i,0}x_{i,1}, x_{i,1}x_{i,0}\} \leq v_i \leq \max\{x_{i,0}x_{i,1}, x_{i,1}x_{i,0}\}$.

The last experiment on this use case is to consider the input bounds to cover the full input space for the student grades, $0 \leq v_i \leq 100$, and the output buckets to encode the student final grade rounded up. Surprisingly, even with the larger input bounds, the analysis results are identical to the previous experiments in terms of the OUTCOMES analysis, meaning that each homework assignment and quiz can alter one grade only, compared to quizzes which they alter three. The RANGE analysis shows that changes in the homework assignments and quizzes can affect the final grade up to 24 units, while changes in the exam grades can affect the final grade up to 44 units (considering the final grade units between 0 and 100).

The takeaway of the analysis results is that the final grade is partially affected by homework assignments and quizzes, while exam grades are far more impactful.

B.2 Termination

This use case comes from the software verification competition SV-Comp[8], where the goal is to verify the termination of a program. Program `termination_a` 1.6 and `termination_b` 1.7 have originally been proposed by Chen et al. [4], respectively these are Example (2.16) and Example (2.21) of such work.

```
def termination_a(x, y):
  while x > 0:
    x = y
    y = y - 1
  result = x + y
  return result
```

Listing 1.6: Program Ex2.16 from software verification competition SV-Comp.

Program `termination_a` returns the value of `y` whenever `x` $= 0$, otherwise it returns -1. We assume both input variables are positive up to 1000, $0 \leq$ `x` ≤ 1000 and $0 \leq$ `y` ≤ 1000. Regarding such a function, it is interesting to study its behaviors around 0, thus the output bucket are {`result < 0`},{`result` $= 0$}, and {`result > 0`}. With the above parameters, the analysis OUTCOMES returns 1 for both input variables. Such result is not too interesting, but by looking at the internal stages of the analysis we notice that perturbations on the value of the variable `x` may be able to produce from an output negative value to zero or a positive one (and viceversa). While perturbations on the value of the variable `y` are only able to produce from zero to positive (and viceversa).

As a second experiments, we consider the buckets from -1 to 19, {`result` $= n \mid -1 \leq n \leq 19$}, and we notice that the analysis OUTCOMES returns 1 for the input variable `x` and 19 for `y`, meaning that the variable `y` is able to affect far more output values than `x`. However, combing the results of the previous experiment, only the variable `x` is able to affect the negative output values.

```
def termination_b(x, y):
  while x > 0:
```

[8] https://sv-comp.sosy-lab.org/

```
    x = x + y
    y = -y - 1
  result = x + y
  return result
```

Listing 1.7: Program Ex2.21 from software verification competition SV-Comp.

This second program `termination_b` returns the value of `y` whenever $\texttt{x}=0$, otherwise it returns -1. Unfortunately, the backward analysis does not capture a precise loop invariant, thus both the analyses, OUTCOMES and RANGE, are inconclusive in such case. The key takeaway is that our analysis is highly dependent on the precision of the underlying backward analysis.

As a conclusion, even though SV-Comp proposes challenging benchmarks for termination, reachability, and safety analyses, they are not amenable for information flow analysis. Most of the time, their examples involve loops with complex invariant, but as input-output relations, the variables involved are just zeroed out after the loop. Drawing examples from their dataset is less appealing to our work.

B.3 Linear Loops

The last use case, program `linear_expression` depicted in 1.8, computes the linear expression $(5x+2y)$ via repeated additions. Note that the invariant of the loop is indeed non-linear ($\texttt{result}=\texttt{i}*\texttt{x}$ and $\texttt{result}=\texttt{result}'+\texttt{i}*\texttt{y}$ respectively for the first and second loop, where $\texttt{result}'$ is the value of `result` before entering the second loop), but the loop is executed a fixed number of times, thus the analysis is able to compute the exact output buckets through loop unrolling.

```
def linear_expression(x, y):
  result = 0
  i = 0
  while i < 5:
    result = result + x
    i += 1
  i = 0
  while i < 2:
    result = result + y
    i += 1
```

Listing 1.8: Program computing the linear expression $(5x+2y)$ via repeated additions.

For the analysis the input bounds are $0 \leq \texttt{x} \leq 1000$ and $0 \leq \texttt{y} \leq 1000$, while the output buckets are $\{n*100 \leq \texttt{result} < (n+1)*100 \mid n \leq 70\}$. Both analyses, OUTCOMES and RANGE, show that `x` has an impact $\frac{5}{2}$ times bigger than `y` on the output. Thus, the impact quantity provides insight about the termination speed. Indeed, the loop for `x` is executed 5 times, while the one for `y` only 2.