A Refactoring for Data Minimisation Using Formal Verification*

Florian Lanzinger[0000−0001−8560−6324], Mattias Ulbrich[0000−0002−2350−1831], and Alexander Weigl[0000−0001−8446−4598]

Karlsruhe Institute of Technology, Karlsruhe, Germany
`<surname>@kit.edu`

Abstract. Concerns about protecting private user data grow as automated data processing by software becomes more ubiquitous in our society. One important principle is *data minimisation*, which requires that all collected personal data must be limited to what is necessary for the respective declared purpose. For existing software systems, it can be difficult to determine which input data are indeed necessary to compute the result and which data can (in certain cases) be left out. We present a new approach that can be used to adapt an existing program to data minimisation requirements. In this framework a user transmits a set of facts about their data instead of the full data. We formally define the approach and introduce a variety of minimisation notions for it. We have implemented the approach and demonstrate its feasibility on a few selected examples.

1 Introduction

Motivation. In 2016, the European Union enacted the General Data Protection Regulation (GDPR) [3] to restrict the storage and use of personal data. Among other regulations, it defines the principle of *data minimisation*, which states that collected "[p]ersonal data shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed" [3, §5(1)(c)].

```
tax_rate(age, income) {
  if (age<18) return 0.0;
  if (age<25 && !(income>1000))
    return 0.1;
  if (income<=1000)
    return 0.2;
  return 0.3; }
```

Fig. 1. A simple program computes the tax rate based on age and income.

When one has to retrofit an existing piece of software to follow this principle, it may be difficult to determine what information from the personal data is necessary to compute the result and what can and should be left out.

Consider the example program in Figure 1, taken from [9], which is run on a tax agency's server. The server receives an applicant's age and income as input

* This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs and based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action.

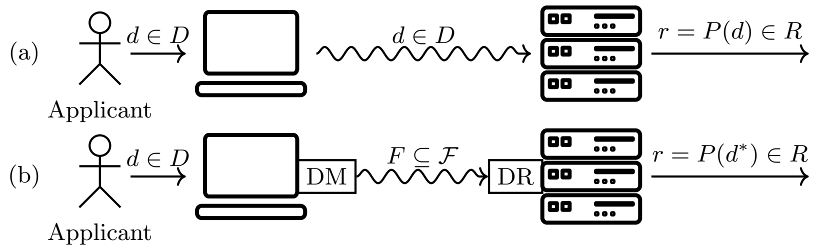

Fig. 2. Sketch of the scenario. In (a), the applicants transmit their individual personal data directly to the server of the agency. Using our approach in (b), the applicants can disguise their personal data d by transmitting a set of facts F.

and computes their tax rate. However, it would never be necessary to submit the exact age to determine the rate: it suffices to know if the age is below 18, between 18 and 24, or above 24. For the income, all that can be relevant is if it surpasses 1000 units. Moreover, if the applicant's age is below 18, it is not necessary to submit any information about the income, as the tax rate is always 0 below the age of 18. Thus, which data is needed to compute the result not only depends on the computed function, but also on the individual applicant's data themselves. With our approach, we enable applicants to selectively disclose their personal data. Instead of transmitting the full personal data, the applicant selects and transmits a set of true facts about their data which are sufficient to infer the result to be computed. These facts are predicates over the input variables of the program, and describe a set of data points in which the actual data point must lie. In our example, a 20-year-old applicant with an income of 2000 can transmit $\{age \geq 18, income \geq 1001\}$ and hence needs not disclose their exact age nor income. The server at the tax agency must then reconstruct an input value from the received facts and use it to compute the tax rate. We will show when and how this is possible.

Contribution. The definition of the minimisation problem which we use and a sketch for a heuristic solution were first introduced by Lanzinger and Weigl [9]. In this paper, we present a detailed formalisation of the problem and of the solution that we propose. We examine three different formal notions of minimisation, namely model-theoretic minimisation, information-theoretic minimisation and vulnerability minimisation based on different notions of entropy. We then present and compare several ways of algorithmically obtaining such (approximate) minimisations and show how they fit into the formal framework. We have implemented the approach based on existing formal analysis tools to obtain new software components to be added to the sending and receiving end of a distributed process without the original software needing to be changed.

Figure 2 shows how the presented approach adapts an existing pipeline. Figure 2(a) shows how the result (the tax rate in the example) is originally computed: The applicant uses a front end – like a web form – to submit a data point $d \in D$ (D denotes the set of personal data) containing their personal data to the back end server of the agency, which then runs a program P on d to compute the result r. Our data-minimisation approach is shown in Figure 2(b): Instead of transmitting the data point d directly, the front end applies a data-minimising routine DM projecting d onto a fact set $F \subseteq \mathcal{F}$ that intuitively can be seen as a set of statements (logical formulas over the input variables of D) taken from a fact base of possible statements $\mathcal{F}$. This minimises the transmitted information in the sense that the agency gains less information about the applicant from F than it would gain by receiving d. The agency then uses the second component in our approach, the data restoration unit DR, to construct an arbitrary, but representative data point d^* which satisfies all facts in F. The data points d^* and d can, but need not be, the same. If F was well-chosen, the agency can compute the original result $r = P(d^*) = P(d)$ without having to modify their program and without knowing the exact value of d.

2 Background on Quantitative Information Flow

We will use some notions from quantitative information flow [14] (QIF) to quantify the amount of personal data sent to the agency. QIF is based on Shannon's information theory and different notions of entropy and conditional entropy, which we will briefly introduce here. QIF is used to measure the information flow from the inputs of a program to its outputs. Often, one quantifies the information that a system leaks, i.e., the amount of confidential information an attacker can learn by observing the output.

Let $\mathtt{D}$ and $\mathtt{O}$ be random variables that stand for the confidential input data ($\mathtt{D}$) and the output ($\mathtt{O}$) of a program that is observed by an adversary. The a-priori probability $\Pr(\mathtt{O} = o \mid \mathtt{D} = i)$ describes the probability that the observable output value o will be produced for a given input i. If the analysed program is deterministic, this is either 1 or 0. The a-posteriori probability $\Pr(\mathtt{D} = i \mid \mathtt{O} = o) = \frac{\Pr(\mathtt{D}=i)}{\Pr(\mathtt{O}=o)} \cdot \Pr(\mathtt{O} = o \mid \mathtt{D} = i)$ describes the probability of the input having been i under the observation of o. This can be used to talk about the amount of information that the adversary learns about the input by observing the output.

The gain of information by an adversarial attacker upon inspecting the output can be measured using different metrics. Two of them are the Shannon entropy and the min-entropy.

The Shannon entropy is usually understood as the uncertainty about an information source from which the observation comes, measured as the expected number of bits required to encode the data under an ideal encoding. The conditional Shannon entropy $H(\mathtt{D} \mid \mathtt{O})$ describes the number of bits required to encode information in $\mathtt{D}$ under the observation of $\mathtt{O}$:

$$\begin{aligned} H(\mathtt{D} \mid \mathtt{O}) &= \sum_o \Pr(\mathtt{O} = o) \cdot H(\mathtt{D} \mid \mathtt{O} = o) \\ &= \sum_o \Pr(\mathtt{O} = o) \cdot \left(\sum_i \Pr(\mathtt{D} = i \mid \mathtt{O} = o) \cdot \log_2 \frac{1}{\Pr(\mathtt{D} = i \mid \mathtt{O} = o)} \right) \end{aligned}$$

Alternatively, the vulnerability of a random variable X can be defined as the maximum probability $V(X) = \max_x \Pr(X = x)$ among the possible values of X. The idea is that a high vulnerability indicates that the correct data point can be guessed easily: The min-entropy of a random variable X is defined using the vulnerability as $H_\infty(X) = -\log_2 V(X)$. Additionally, the conditional vulnerability $V(\mathtt{D} \mid \mathtt{O} = o) = \max_i \Pr(\mathtt{D} = i \mid \mathtt{O} = o)$ measures the success rate when the attacker tries to guess the secret under a given observation o.

In the context of QIF, we are interested in the conditional entropy $H_\infty(\mathtt{D} \mid \mathtt{O})$ that describes the expected amount of information about the input that one receives upon observing the output. Formally, it is defined as follows.

$$\begin{aligned} H_\infty(\mathtt{D} \mid \mathtt{O}) &= \sum_o \Pr(\mathtt{O} = o) \cdot V(\mathtt{D} \mid \mathtt{O} = o) \\ &= -\log_2 \left(\sum_o \Pr(\mathtt{O} = o) \cdot (\max_i \Pr(\mathtt{D} = i \mid \mathtt{O} = o)) \right) \end{aligned}$$

3 Formalisation

This section formalises the notions of facts and information-theoretic minimality. We will see how we can apply this formalisation to implement our approach in the following sections.

3.1 Programs and Fact Bases

We consider a fixed domain D of possible personal data points and R the set of possible results. In the introductory example in Figure 1, $D = \mathbb{N} \times \mathbb{N}$ and $R = [0.0, 1.0] \subseteq \mathbb{R}$. We consider the program to be data-minimised as a total and deterministic function $P \colon D \to R$ that maps a user's data point $d \in D$ to a result $r = P(d) \in R$.

A *fact* is a predicate $f \in 2^D$ on the set of data points D, characterised by those points that satisfy the predicate. Usually a predicate is syntactically represented as a constraint on the variables standing for the individual values within the data point. For example, we use the constraint $\mathtt{age} > 18$ to denote the set of data points $\{(\mathtt{age}, \mathtt{income}) \in D = \mathbb{N} \times \mathbb{N} \mid \mathtt{age} > 18\}$. We denote the set of all conceivable facts on the domain as $\mathbb{F} = 2^D$. We write $d \models f$ to indicate that the data point d satisfies the predicate $f \in \mathbb{F}$ and $d \models F$ to indicate that d satisfies all predicates in $F \subseteq \mathbb{F}$. In the following, we will generally not draw facts from $\mathbb{F}$. Instead, we assume that facts are taken from a *fact base*, a subset of predicates $\mathcal{F} \subseteq \mathbb{F}$. Not every set is suited as a fact base. For a given program P, we demand that for all possible data points d, we have a set of facts that uniquely identifies the result $P(d)$.

Definition 1 (honest fact sets and projections). *Given a point $d \in D$, we call a fact set F* honest *about d if $d \models F$. We call a projection $\pi : D \to 2^{\mathcal{F}}$* honest *if $d \models \pi(d)$ for every $d \in D$.*

Definition 2 (sufficiently precise fact sets and fact bases). *Let $P\colon D \to R$ be a program.*

A fact base $\mathcal{F} \subseteq \mathbb{F}$ is called sufficiently precise for P *if for all $d \in D$ there exists a sufficiently precise and honest set $F \subseteq \mathcal{F}$.*

A fact set $F \subseteq \mathcal{F}$ is called sufficiently precise *if there exists an $r \in R$ such that for all $d \in D$ with $d \models F$, we have $P(d) = r$.*

This property will be useful for the following decomposition of the computation of P. With reference to Figure 2b, we can define two functions π_0 and ω that take the roles of the data minimisation DM and data restoration DR in the sketch. The *canonical projection* $\pi_0 : D \to 2^{\mathcal{F}}$ maps points $d \in D$ from the domain to all facts that are satisfied by d, i.e., $\pi_0(d) := \{f \in \mathcal{F} \mid d \models f\}$. The partial *witness* function $\omega : 2^{\mathbb{F}} \nrightarrow D$ produces for a fact set $F \subseteq \mathbb{F}$ a witness $\omega(F) \in D$ with $\omega(F) \models F$ iff such a value exists.

Observation 1 *Let $\mathcal{F}$ be a sufficiently precise fact base, then: $P = P \circ \omega \circ \pi_0$. The following diagram commutes:*

$$\begin{tikzcd}
D \arrow[rr, "P"] \arrow[dr, "\pi_0"'] & & R \\
& 2^{\mathcal{F}} \xrightarrow{\omega} D \arrow[ur, "P"] &
\end{tikzcd}$$

This follows from the following observation: If $\mathcal{F}$ is sufficiently precise, then the projection $\pi_0(d)$ is an honest, sufficiently precise fact set for every $d \in D$. This implies that $d' = \omega(\pi_0(d))$ returns a value with $P(d) = P(d')$.

Thus, it is possible on the front end to encode a point d in form of a set of predicates $\pi_0(d)$ and transmit that instead of d. On the back end, a witness $\omega(\pi_0(d))$ can be chosen to complete the computation by running the original program $P(\omega(\pi_0(d)))$ on the witness yielding $P(d)$. Note that it is not necessary that $\omega(\pi_0(d)) = d$, i.e., the back end need not be able to reconstruct the input data, but the witness will have the same result under P as d[1].

Where does the data minimisation happen here? It is in the function π_0 that maps to the facts that we lose information on the input data. In general, a set of constraints can have many satisfying assignments (models), π_0 is not an injection such that information is reduced here. At the same time, while the information is reduced, all transmitted facts must be valid statements on d.

In case there is doubt about the applicant having submitted an honest fact set, certificates proving the facts can be requested. For this, we assume that a set of certifiable facts $\mathcal{C} \subseteq F$ is given. Intuitively, a certificate $c \in \mathcal{C}$ is a fact which trusted third parties can attest (e.g., by inspecting legal documents), and this attestation can be audited (e.g., by digital signature). Of course, a certificate has to be honest, i.e., $d \models c$. For example, the applicant's electronic passport might can certify that `age` > 18. We say that c is a certificate for a fact $f \in \mathbb{F}$ if $c \to f$.

[1] d will be in the same equivalence class in the kernel of P.

A large part of the remainder of the paper will address the challenges of designing suitable precise fact bases and identifying clever projection functions that can be used instead of the canonical projection π_0 while maintaining the property of Observation 1. Any honest projection π must obey $\pi(d) \subseteq \pi_0(d)$.

However, two corner-case fact bases are worth mentioning here ($\cong$ meaning that two sets are isomorphic):

$$\mathcal{F}_D = \{\{d\} \mid d \in D\} \cong D \qquad \mathcal{F}_R = \{\{d \mid P(d) = r\} \mid r \in R\} \cong P(D) \subseteq R$$

In the trivial fact base $\mathcal{F}_D$, every data point is captured by precisely one predicate (like $\mathit{income} = 1234 \wedge \mathit{age} = 37$). The canonical projection only wraps the concrete data in a unique predicate and the witness function unwraps them. This fact base covers the comparison 'base line' without data minimisation. On the other extreme, the fact base $\mathcal{F}_R$ loses as much information as possible on the front end and effectively transmits to the back end only the result of the computation P. This is the other end of the spectrum: It minimises the data most (according to any minimisation definition). In Section 4.1, we see an algorithm using the weakest precondition transformer that can compute $\mathcal{F}_R$.

Although this seems like an obvious solution to our data minimisation problem, it is also impractical. While we want to minimise the personal data that are initially transmitted, we also want to be able to certify the facts later. Using $\mathcal{F}_R$ as the fact base, all relevant knowledge about the applicant's data point d is encoded into a single fact, which is determined solely by the result $r = P(d)$. For certification by a third party, the applicant needs to fully disclose d to the third party that verifies that the result $r = P(d)$ was correctly computed. Using a fact base other than $\mathcal{F}_R$, the applicant may release more information initially, but less information later when certificates are requested.

3.2 Data Minimisation

After having set up the formal framework describing our approach, we can now focus on the actual question of data minimisation within the approach: A data point $d \in D$ is projected onto a set $\pi(d) \subseteq \mathbb{F}$ of facts. But since there are many possible fact bases and different possible projections that preserve data reconstructability, we need means to differentiate. The interesting question remains to be able to compare different sets of facts with respect to the information they divulge to the agency. We will discuss three notions in the following: (1) model-theoretic, (2) information-theoretic based on Shannon entropy and (3) vulnerability-driven based on min-entropy. For each notion, we define a different partial order $\preceq\colon 2^{\mathbb{F}} \times 2^{\mathbb{F}}$. We write $F \preceq F'$ if a fact set $F \subseteq \mathbb{F}$ carries less information (as defined by the picked notion) than another fact set $F' \subseteq \mathbb{F}$.

Model-theoretic Minimisation The first notion uses the sets of satisfying instances as a basis for an order on fact sets. For a fact set $F \subseteq \mathbb{F}$, the set of models (i.e., satisfying instances) $\mathit{models}(F) := \{d \in D \mid d \models F\}$ is the set of data points that satisfy all facts in F. A fact set with more satisfying data points in the model set sends less information about the actual value within this set to the agency. Intuitively, if there are more satisfying data points, the agency has a lower chance of guessing the correct one. Since, at least for now, we do not incorporate probability distributions over the data points, we will only compare sets by their model sets if one is a subset of the other. The more information a fact set conveys, the fewer models it has.

Definition 3 (Model-theoretic minimisation $\preceq_{mt}$). *Given two fact sets $F, F' \subseteq \mathbb{F}$, we say F carries model-theoretically at most as much information as F' and write $F \preceq_{mt} F'$ if all instances satisfying F' also satisfy F, i.e., if $models(F') \subseteq models(F)$.*

With this model-theoretic notion of 'more models implies less information' at hand, we can derive a syntactic notion which allows us to judge syntactic representations w.r.t. minimality.

Observation 2 (Syntactic minimisation) *Given two fact sets $F, F' \subseteq \mathbb{F}$ that are represented syntactically by two sets of constraints C, C' respectively, if $C \subseteq C'$, then $F \preceq_{mt} F'$.*

The opposite direction needs not hold, since the same fact may be expressed by syntactically incomparable constraints. Moreover, even if $C \subset C'$ is a strict subset, the conveyed information may remain the same if the constraints in $C' \setminus C$ are implied by C. For example, the fact set $\{age > 18\}$ contains less information than $\{age > 18, income > 1000\}$ but as much information as $\{age > 18, age > 17\}$.

For instance, if a fact set has only one model, it identifies the corresponding confidential data directly. If it has two models (and all data points are equally likely), then there is a $50:50$ chance to guess the correct confidential data point.

Minimisation by Shannon-Entropy Model-theoretic minimisation is a good measurement if the compared fact sets are in a subset relationship. Another approach allows us to assign a number to fact sets and thus make the comparison a total relation.

We achieve this by applying the notions of quantified information flow (QIF) [14] (introduced in Section 2) to our problem. In the scenario of data minimisation, the attacker is the agency that learns information about the confidential applicant data d by observing the transmitted facts $F = \pi(d) \subseteq \mathcal{F}$. In terms of Section 2, the input variable $\mathtt{D}$ takes values in D and the observation (the data transmitted to the agency) $\mathtt{O}$ takes values in $2^{\mathcal{F}}$.

A fact set F partitions the domain D into those data points that satisfy F and those that do not. We assume a given a-priori probability distribution[2] $\Pr(\mathtt{D} = d)$ on the input D that models how likely a particular data point $d \in D$ occurs amongst all applicants.

[2] We assume that D is finite and discrete, although we are confident that this can be generalised.

One way to measure the (information-theoretic) amount of information that the agency receives is via the probability mass of those data points which are consistent with the made observation, i.e., $\Pr_{\mathbb{F}}(F) := \sum_{\substack{d\in D\\ d\models F}} \Pr(\mathtt{D} = d)$. Intuitively, this is the probability for a randomly chosen applicant that F applies to them. The more applicants are consistent with F, the less information it leaks. $\Pr_{\mathbb{F}}$ is in general not a distribution over $2^{\mathbb{F}}$. We define an information-theoretic notion of minimality:

Definition 4 (Information-theoretic minimisation $\preceq_{it}$). *Given two fact sets $F, F' \subseteq \mathbb{F}$, we say F carries* information-theoretically less information *than F' and write $F \preceq_{it} F'$ if $\Pr_{\mathbb{F}}(F) \geq \Pr_{\mathbb{F}}(F')$.*

The important difference between Definitions 3 and 4 is that the former does not take probabilities into account and thus allows for a comparison only if one fact set is a subset of the other. Since the latter notion is based on probabilities, it can express more cases.

Observation 3 *For all fact sets $F, F' \subseteq \mathbb{F}$:*
- $F \preceq_{mt} F' \implies F \preceq_{it} F'$
- $F \preceq_{mt} F' \Leftrightarrow F \preceq_{it} F'$ *if $F \subseteq F'$ or $F' \subseteq F$*
- $F \preceq_{it} F' \Leftrightarrow |models(F)| \leq |models(F')|$ *(i.e., F has fewer models than F') if D is finite and $\Pr(\mathtt{D} = d)$ is uniformly distributed.*

In information theory, the information content of random variables is usually computed in bits by taking its logarithm. We get the Shannon entropy if we compute the expectation value of the information of the input data. Since we can use the observations to learn about the data point, we are actually interested in the conditional Shannon-entropy $H(\mathtt{D} \mid \mathtt{O})$ (as defined in Section 2). This, in turn, allows us to compare different projection functions w.r.t. the transmitted information.

Observation 4 *Consider two different projection functions $\pi_1, \pi_2 : D \to 2^{\mathcal{F}}$ inducing the random observation variables $\mathtt{O}_1, \mathtt{O}_2$ respectively with $H(\mathtt{D} \mid \mathtt{O}_1) \geq H(\mathtt{D} \mid \mathtt{O}_2)$. In the long run, the agency will learn less[3] about the applicants if the projection π_1 is used by all applicants (in comparison to when π_2 would be used).*

Example 1. Let us revisit the trivial fact bases $\mathcal{F}_D$ and $\mathcal{F}_R$ from Section 3.1. For a data point d, the single-element fact $\{d\}$ is the maximum w.r.t. model- and information-theoretic minimisation. Nothing more can be said than the full data if the fact set is to remain honest. The fact $\{d' | P(d) = P(d')\}$ characterising all data points with the same result of P, on the other hand, is the minimum w.r.t. both notions. There is no way to convey less information about the data point if the fact set is to remain sufficiently precise:

$$\{d\} \quad \preceq_{mt/it} \quad F \quad \preceq_{mt/it} \quad \{d' \mid P(d) = P(d')\}$$

for any sufficiently precise and honest fact set F for d.

[3] i.e., will gather less information in the sense of Shannon information-theory

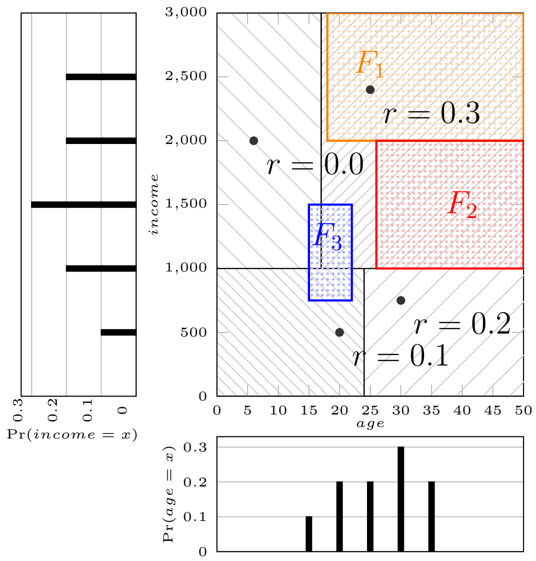

Fig. 3. The input space of the introductory example Figure 1 partitioned by the different program outputs $R = \{0, \dots, 0.3\}$. F_1, F_2 and F_3 are the regions of different fact sets. The probabilities for age and income are given on the margin.

Minimisation by Min-Entropy In Section 2, we also introduce the (conditional) min-entropy $H_\infty(\mathtt{D} \mid \mathtt{O})$. In contrast to the Shannon entropy, the min-entropy is driven by an attacker model. It captures the success of an attacker to guess the data point in one try [14]. This guessing is modelled by the vulnerabilities $V(\mathtt{D})$ and $V(\mathtt{D} \mid \mathtt{O})$.

Observation 5 *An applicant with the data point $d \in D$ should use a fact set F for which d is unlikely among all other models.*

The attacker observes the transmitted facts F, and tries to guess the data point i. After the definition of vulnerability $V(\mathtt{D} \mid \mathtt{O} = F)$, the attacker selects the data point i that has the highest probability $\Pr(\mathtt{D} = i \mid \mathtt{O} = F)$. As a consequence, if the applicant can choose between two fact sets F and F', they should select F iff $\Pr(\mathtt{D} = d \mid \mathtt{O} = F) \leq \Pr(\mathtt{D} = d \mid \mathtt{O} = F')$, for their individual data point d. As $\Pr(\mathtt{D} = d \mid \mathtt{O} = F) = \frac{\Pr(\mathtt{D}=d, \mathtt{O}=F)}{\Pr(\mathtt{O}=F)}$, we can also observe that the applicants can achieve this by selecting likely (often selected) set of facts or unlikely combinations of their data points and the transmitted facts.

Definition 5 (Vulnerability minimisation $\preceq_V$). *Given two fact sets $F, F' \subseteq \mathbb{F}$, we say F is less* vulnerable *than F', and write $F \preceq_V F'$ if $V(\mathtt{D} \mid \mathtt{O} = F) \geq V(\mathtt{D} \mid \mathtt{O} = F')$.*

Example 2. Let us consider our introductory example with $(\mathit{age}, \mathit{income}) \in D$, where $0 \leq \mathit{age} \leq 50$, $0 \leq \mathit{income} \leq 3000$. Figure 3 shows the input space, the partitioning based on the output of Figure 1, and the assumed margin probability $\Pr(\texttt{age})$ and $\Pr(\texttt{income})$. We assume that $\Pr(\texttt{age})$ and $\Pr(\texttt{income})$ are independent, hence $\Pr(\texttt{age}, \texttt{income}) = \Pr(\texttt{age}) \cdot \Pr(\texttt{income})$. Hence, the data point with the maximum overall probability is $j = (30, 1500)$ as the combination of the maximum probability of age and income. For convenience, we keep the probabilities discrete, sparsely populated with a small support domain. Let us consider the following fact sets:

$$F_1 = \{age > 18, income \geq 2000\}$$
$$F_2 = \{age > 25, income > 1000, income \leq 2000\}$$
$$F_3 = \{15 < age \leq 23, income \geq 750, income \leq 1500\}$$

Their corresponding subsets in the input space are also drawn in Figure 3.

First, we attest that F_1, F_2, and F_3 are not in a subset relationship to each other (cf. Observation 2). None of them is a subset of the other. Moreover, F_1 and F_2 are sufficiently precise to guarantee the result $r = 0.3$. On the other hand, F_3 is not precise enough to guarantee a single result because its region overlaps with multiple output partitions. Hence, let us concentrate on F_1 and F_2. The fact sets are also incomparable using subset relations on their models, e.g., $\text{models}(F_1) \not\subset \text{models}(F_2)$ and vice versa. Using the number of models, we see that F_1 allows more data points: $|\text{models}(F_1)| > |\text{models}(F_2)|$. But both are smaller than the outer output partition for $r = 0.3$.

The sum of the probabilities results into the following probabilities for the fact sets: $\Pr_{\mathbb{F}}(F_1) = 0.36$ and $\Pr_{\mathbb{F}}(F_2) = 0.40$. We see that $F_2 \preceq_{it} F_1$ as F_2 is more probable. Using the vulnerabilities for $V(F_1) = 0.06$ and $V(F_2) = 0.09$, we would rather select F_1. For the data point $i = (30, 2000)$, which lies in F_1 and F_2, we would use F_2 as $P(\texttt{D} = i \mid \texttt{O} = F_1) = 0.17$ against $\Pr(\texttt{D} = i \mid \texttt{O} = F_2) = 0.15$. Note, data point j also lies in F_2, hence it makes F_2 interesting to select for all data points besides j.

3.3 Limitations

In order to check if a fact set is sufficiently precise, the applicant needs access to the program. Otherwise, it is impossible for the applicant to know if a fact set has a unique result value or not. Naturally, this restriction limits the application scenarios for our approach since not all agencies are willing to disclose in all detail how they come to their decisions. A bank, for instance, need not fully disclose their algorithm to decide the interest rate for a loan to an applicant. However, there are cases where the availability of the computed function is a sensible assumption: The introductory example of the income tax rate computation is such a situation. It is definitely reasonable to assume that the tax law is publicly known. The same applies to other calculations described by law and regulations.

Our approach can be adapted if the sources are not available, but naturally becomes less transparent. A solution that does not require access to the source code is for the applicant to always submit the set of all facts which hold for their data point (i.e., they apply π_0). Alternatively, the company can build a projection function into their front ends. In that case, the applicant can neither check if their transmitted data is sufficiently precise nor can they know anything about the minimality of the data provided. The design and analysis of the fact base and projection can then only happen inside the agency: The base must be sufficiently precise and statements about its minimising effects can only be made within the agency. It would be possible to have the minimisation design audited by a third party to increase trust by applicants into the scheme.

Our approach can identify minimal fact sets needed to compute a result, thus identifying the minimal set of 'adequate, relevant, and limited to what is necessary' [3] facts for the given computation. However the method can only judge the transmitted facts based on their being needed in the program, it cannot judge the adequacy of the program's purpose in the first place. For example, if an insurance company chooses to discriminate against green-eyed people by marking up all prices for them, our approach will consider a person's eye colour to be a relevant fact. Moreover, different parameters within a data point can be correlated and the agency may still gain knowledge even if problematic parameters are excluded from facts (e.g., the *address* might be correlated with the *income*, and facts on *address* may leak information on *income* even if the income is not in the fact set).

4 Data Minimisation Using Formal Verification

In this section, we refine the approach formally defined in the last section towards an implementation that makes use of existing functionalities in formal verification approaches and tools. Figure 4 refines the sketched pipeline from Figure 2 by listing the tasks that are needed in the data minimisation (which corresponds to projection π) layer in the front end and the data restoration layer (which corresponds to witness function ω) in the back end.

On the client side, the applicant submits their data point $d \in D$ which is then subject to a projection onto facts from the fact base $\mathcal{F}$. In general, each applicant may select a different fact set for d or even submit a different fact set every time they run the algorithm. The approach solely relies on two properties that $F = \pi(d)$ must have: (1) F must be honest and (2) F must be sufficiently precise. F is then transmitted to the back end (say, via a network connection). The back end receives a fact set F and tries to compute $P(d)$ from that. First, it should check if the received fact set is indeed sufficiently precise – there is no need to trust the client on that point. Checking the honesty of the set on the other hand is not possible, unless the agency requires that facts be certified (e.g., signed by trusted parties); but this is outside the scope of this paper. Then, the back end reconstructs a witness $d^* := \omega(F)$ that satisfies all constraints of F. The result is then computed as $P(d^*)$ and guaranteed to be equal to $P(d)$. In the following, we will outline

1. how a sufficiently precise fact base can be extracted fully automatically from the source code of P,

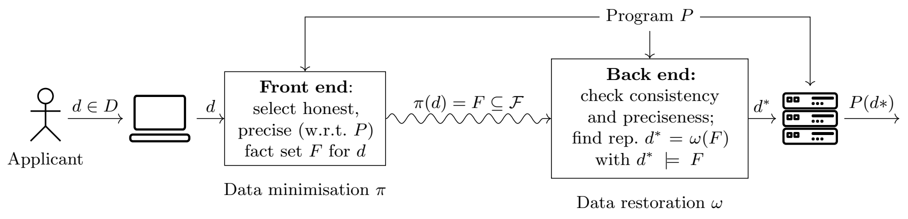
Program P
Applicant
$d \in D$
d
Front end: select honest, precise (w.r.t. P) fact set F for d
Data minimisation π
$\pi(d) = F \subseteq \mathcal{F}$
Back end: check consistency and preciseness; find rep. $d^* = \omega(F)$ with $d^* \models F$
Data restoration ω
d^*
$P(d*)$

Fig. 4. Refined sketch of the pipeline, showing the tasks in the front and back end.

2. how a model-theoretically minimal, yet sufficiently precise fact set F consistent with d can be automatically computed,
3. how a witness d^* can be obtained from F that computes $P(d)$, and
4. how it can be verified that there is no witness for another result value (i.e., that F is sufficiently precise).

4.1 Front End: Finding Facts

In earlier considerations, we did not make any assumptions about the fact base $\mathcal{F}$. In general, a fact base could be handcrafted specifically for its target program by an expert because of the knowledge they have about the domain and its data minimality principles. For an automatic data minimisation refactoring of a process, it is helpful to have a possibility to fully automatically extract a fact base from the code. Luckily, as we will see, a sufficiently precise fact set can be extracted automatically from the program code.

The program determines the outcome, and hence partitions the input space into regions of equal output. The conditions that describe these regions can be found in control and data flow expressions of the program, and a fact base can be derived by a static analysis.

The first candidate is the *weakest precondition calculus* (wp-calculus) [4], in which the formula $wp(P, c)$ represents the weakest precondition s.t. the post-condition c holds after the execution of the program P. If we assume that P is loop-free, the formula $WP(d) := wp(P, o = P(d))$ exactly describes all data points with the same output as $d \in D$. The facts resulting from the wp-calculus thus produce the fact base $\mathcal{F}_R$ introduced in Section 3.1.

Example 3. Consider our introductory example in Figure 1. The wp-calculus returns the following facts to achieve the outcome of 0.1.

$$wp(\texttt{tax\_rate}, \texttt{o} = 0.1) = \{\neg(\texttt{age} < 18) \land \texttt{age} < 25 \land \neg(\texttt{income} > 1000)\}$$

A drawback of the wp-calculus is the need for specification. In particular, to preserve minimality, all unbounded loops must be specified with a sufficiently strong loop invariant. However, there are no unbounded loops in the examples and case study in this paper. The wp-calculus thus finds a fact set F under the fact base $\mathcal{F}_R$ (see Section 3.1). Even though the wp-calculus does not explicitly model any probabilities, its use of $\mathcal{F}_R$ guarantees that F is minimal under all three notions of minimality discussed in Section 3.2 because $\mathcal{F}_R \overset{\sim}{=} P(D) \subseteq R$.

A more practical idea is to capture the path conditions during the execution of $P(d)$. In addition to the normal evaluation of each expression to a concrete value, we store each path condition symbolically, using concrete values for non-input variables. When we capture these symbolic expressions, we obtain a set of (path) expressions over the input variables. These only capture the control flow of the program, but not the data flow. To capture the data flow, we add a fact for each output variable o that asserts equality between the symbolic value of o and its concrete at the end of the execution of $P(d)$. Formally, we add the fact $\texttt{seval}(o) = \texttt{eval}(o)$ where $\texttt{seval}(o)$ returns a (symbolic) expression that describes the computation of o by using the input variables, and $\texttt{eval}(o)$ which evaluates to the concrete value of o.

We call the predicate transformer $\mathit{fwd}(P, \sigma, \varsigma)$ because of the forward application (in contrast to the wp-calculus, which is applied backwards). P denotes the program, σ the current concrete state, and ς the symbolic state over the input variables. $\emptyset$ denotes the state with an empty variable assignment. $\sigma[v \leftarrow e]$ denotes an update of the state σ in which the variable v is assigned to the concrete or symbolic value e. We use $\texttt{eval}(e)$ to denote the concrete evaluation of an expression e, and $\texttt{seval}(e)$ for the symbolical evaluation.

$$\begin{aligned}
\mathit{fwd}(v := e \;;\; P, \sigma, \varsigma) &\rightsquigarrow \mathit{fwd}(P, \sigma[v \leftarrow \mathit{eval}(e)], \varsigma[v \leftarrow \texttt{seval}(e)]) \\
\mathit{fwd}(\texttt{if}(c)\; b_1 \;\texttt{else}\; b_2 \;;\; P, \sigma, \varsigma) &\rightsquigarrow \{\texttt{seval}(c)\} \cup \mathit{fwd}(P, \sigma, \varsigma) \quad \text{ if } \texttt{eval}(c) \\
\mathit{fwd}(\texttt{if}(c)\; b_1 \;\texttt{else}\; b_2 \;;\; P, \sigma, \varsigma) &\rightsquigarrow \{\neg\texttt{seval}(c)\} \cup \mathit{fwd}(P, \sigma, \varsigma) \quad \text{ if } \neg\texttt{eval}(c) \\
\mathit{fwd}(\epsilon, \sigma, \varsigma) &\rightsquigarrow \bigcup_{o \in OutVar} \texttt{seval(o)=eval}(o)
\end{aligned} \tag{1}$$

Example 4. Re-consider Figure 1; the tax rate computation for the data point $(\mathit{age} = 20, \mathit{income} = 800)$ results in 0.1. A sufficiently precise fact set is

$$\begin{aligned}
&\mathit{fwd}(\texttt{tax\_rate}, \emptyset[\mathit{age} \leftarrow 20][\mathit{income} \leftarrow 800], \emptyset) \\
&\qquad\qquad = \{\neg(\texttt{age} < 18), \texttt{age} < 25, \neg(\texttt{income} > 1000)\} \;.
\end{aligned}$$

This program contains only control flow, but no data flow. In every control-flow path, the return value is a constant independent of the input. To see why the last rule (1) is required, consider the program `out = x MOD 4` which returns the last two bits of the input `x`. This program only consists of a data flow. For instance, we obtain the fact $x \text{ MOD } 4 = 1$ for the data point $x = 13$ by (1).

Observation 6 *The fwd-calculus returns a sufficiently precise and honest fact set for a data point $d \in D$.*

While the fwd-calculus can guarantee these two properties, it does not guarantee minimality of the resulting fact set. It is also vulnerable against syntactical manipulation in the sense that a programmer can exploit the fwd-calculus to ensure chosen facts are always included in the result by mentioning them in the program code although they are not semantically required. For example, consider the program `if (x) out = 0 else out = 0`, which always returns 0, but the fwd-calculus includes the fact x.

4.2 Front End: Minimising Fact Sets

The fwd-calculus does not guarantee any notion of minimality. For example, consider two nested if-statements "`if (a) if (b) ...`", for which the fwd-calculus includes facts a and b, and misses the suffice and more minimal fact $a \wedge b$ in contrast to the wp-calculus. Nonetheless, we can use the resulting fact set F of the fwd-calculus as a starting point to extract a set $F' \subseteq F$ that is minimal w.r.t. $\preceq_{mt}$. We introduce a technique to find such a F' by using minimal unsatisfiability cores for proofs which are reported by verification tools.

If a formula set C is unsatisfiable, modern propositional satisfiability (SAT) or satisfiability modulo theories (SMT) solvers can compute a minimal unsat-core $C' \subseteq C$ such that C' is unsatisfiable and there is no unsatisfiable set $C'' \subset C'$. In general, the minimal unsat-core is not unique. For a given $d \in D$, with $r = P(d)$ the following formula encodes a check for preciseness of a consistent fact set F:

$$\textsc{NotSufPrec}(F) := wp(P, \texttt{out} \neq r) \wedge \bigwedge F \tag{2}$$

If $\textsc{NotSufPrec}(F)$ is unsatisfiable, we can obtain a minimal unsat-core F' from the verification tool to obtain a fact set which implies that the result of P is r. Using the minimal unsat-core, we minimise a set of facts F' minimal w.r.t. the minimisation $\preceq_{mt}$ according to Observation 2. If $\textsc{NotSufPrec}(F)$ is satisfiable, then F is not sufficiently precise. We will exploit this in the next section for the back end computations.

4.3 Back end: Computing the Result from a Fact Set

The back end needs to re-establish the result $r = P(d)$ from a received fact set F. In the formalisation in Section 3.1, we introduce a partial witness function $\omega : 2^{\mathbb{F}} \nrightarrow D$ that produces a witness such that $\omega(F) \models F$ (if $F \in dom(\omega)$). We can rely here on another feature of modern SAT and SMT solvers: They can produce a model in case they detect that an input is satisfiable. Hence, we encode $\bigwedge F$ as an SMT input. If it is satisfiable, the solver delivers $d^* := \omega(F)$. The back end can feed d^* into P and obtain the result $r = P(d^*)$. If $\bigwedge F$ is unsatisfiable, the fact set is inconsistent, which is the front end's fault.

Proving Sufficient Preciseness. As mentioned, the transmitted fact set F has to be honest and sufficiently precise. While honesty cannot be checked without requiring certificates for the facts, we *can* check preciseness. We again use the formula defined in (2). If $\textsc{NotSufPrec}(F)$ is satisfiable, then there is a data point $d^! \in D$ with $P(d^!) \neq P(d^*)$ which means that there is no unique result value. The back end can once again blame the front end.

However, the back end need not fail at this point. Instead it could follow a policy that if a fact set is not sufficiently precise, the back end chooses any satisfying data point, in particular, that with the worst possible outcome (for the applicant). In our scenario: If the applicant chooses not to disclose the income, the tax agency assumes the highest tax rate consistent with the facts. Technically, this can be implemented by a repeated satisfiability test. If the applicant wants to have a low tax rate, the agency can repeatedly use the postcondition $out > r$ to check if there is a data point for F which has a worse tax rate.

Of course, for these ideas to work, we must have SMT formulas that solvers can decide. For many practical applications, we can limit formula expressions in P and in F to SMT formulas from decidable theories only, e.g., by encoding variables as finite bit vectors.

Example 5. For example, let P again be the `tax_rate` function from Figure 1, and F the set of facts represented by the predicates $\{\texttt{age} < 18, \texttt{income} > 1000\}$ which has been computed using the fwd-calculus. The formula from (2) reads $wp(\texttt{tax\_rate}, out \neq 0.0) \wedge \bigwedge F$ and is unsatisfiable. The minimum-unsat core provided by an SMT solver presented with this input shows that the set F' represented by $\{age < 18\}$ is a sufficiently precise subset. This is sent to the back end which finds a witness $\omega(F') = (12, 200)$ and computes a tax rate of 0.0 from this data point. The formula $wp(\texttt{tax\_rate}, out \neq 0.0) \wedge age < 18$ is unsatisfiable, so the back end knows it received a sufficiently precise result.

5 Implementation and Experiments

In this section, we give an overview over the prototypical implementation and explain its use on a more sophisticated example. Both are publicly available[4].

5.1 Prototypical Implementation

For our experiments, we implemented the functionalities of the front and back end on top of CBMC [2] and Z3 [13]. Z3 is a state-of-the-art SMT solver and CBMC is a bounded model-checker for C programs, which is why the implementation cannot handle unbounded loops. The implementation is written in Python and orchestrates the underlying tools. In general, the Python script takes an augmentable program and a YAML file containing meta-data. An augmentable program contains markers that allow injecting program code at the marked positions in the source code. Figure 5 shows a minimal skeleton with markers. The required meta-data contain the name of the input and output variables of the program along with the variable assignments of the data point $d \in D$, and the calculated output $r = P(d)$. In the following, we assume that all facts are Boolean expression in the C language.

[4] `https://github.com/wadoon/data-minimization`

As CBMC provides a way to check explicit assertions in C programs, we only have to encode our proof obligations into `assert`-statements. Also we exploit CBMC for the generation of SMT input and DIMACS (a SAT input format) encoding of the given program and assertions.

```
#ifndef NOHEADER
// include files
#endif
int main() {
  //!INPUT
  // calculation
  //!OUTPUT
}
```

Fig. 5. Skeleton of an augmentable program. Marker comments for code injection.

Front end: Execution of the Program. The first step on the client side is the execution of the program with input d. We use the marker `//!INPUT` to set the values of the input variables to the data point, and the marker `//!OUTPUT` to add statements to print out the value of the output variables. The augmented program is compiled and executed. Its output is parsed to obtain $r = P(d)$.

Front end: Finding Facts. Despite the exact solution (wp-calculus) being known, the extraction of facts is not sufficient solved. The main issue is scalability. Therefore, we implemented several lightweight techniques to come up with a fact set: First, we implemented the fwd-calculus from Section 4.1; second, we implemented an approach with symbolic execution based on the single static assignment (SSA) form with a similar goal. The construction of the SSA-form scales well, but the extraction of facts does not, i.e., it is only applicable if we expand the computed expression of the program variables only to a limited depth. The third heuristic approach tries to find a fact set by using expressions and constants from the program and the assignments of input variables.

Whichever technique is used to obtain the fact set; in the end, we can check whether the fact set is consistent and sufficiently precise.

Front end: Minimising the Fact Set. To obtain a fact set $F' \subseteq F$ which is minimised w.r.t. $\preceq_{mt}$ as explained in Sec. 4.2, we use CBMC and Z3. In the skeleton, the marker `//!INPUT` is replaced by a list of `assume(f)` statements for each fact $f \in F$. The `//!OUTPUT` is replaced by the assertion `assert(o ==` r`)` in which $r = P(d)$ is the computed output and `o` is the variable holding the output. CBMC is then called to produce an SMT input file to which we add annotations and commands that request and control the generation of a minimal unsat core. We pass it on to Z3 and parse its output. If the assumptions imply the equality, we can read out an unsat-core of that proof.

Back end: Calculation of the Outcome. The main task on the back-end side is to compute the final result $r = P(d)$ from a fact set F. Once computed, r will also be used check the consistency and preciseness of F. The computation of the outcome requires the symbolical execution of the program under the assumption that the (symbolic) input adheres to F. Luckily, we can exploit CBMC to achieve this: We use the same query on CBMC as in the minimisation step above: The `//!INPUT` covers the assumptions of the facts, and `//!OUTPUT` is replaced by `assert(false);`. If CBMC proves this program correct, then F is inconsistent, and it will be rejected. If this program is not correct, we obtain a counterexample variable assignment d^* whose input satisfies F and whose output is $r = P(d^*)$.

Back end: Checking Sufficient Preciseness. To check sufficient preciseness of a given fact set, we use the exact same setup as for minimising the fact set (replacement of `//!INPUT` and `//!OUTPUT` marker). But since we are not interested in an unsat-core here, we directly ask CBMC to verify that the assumptions always lead to the same (previously) computed output.

5.2 Example: Account Charges

Figure 6 shows a program computing the monthly charges for a bank account. As input variables, it takes the age of the account holder, their monthly cash receipt on the bank account, and the reward points of the customer loyalty program. The program privileges young customers, customers with a high cash receipt and loyal customers. Note that the given loop is bounded, as it terminates after at most 10 iterations.

```
int charge(int age, int income, int reward) {
    if (age < 18) return 0;
    if (age < 27) {
        if(income <= 1500) { return 0; }
        else { return 5 - 2 * reward; } }
    int g = 10 - 2 * reward;
    do { g = g - 1; income = income - 500; }
    while(income >= 500 && g > 0);
    return g; }
```

Fig. 6. Example program for the calculation of account administration charges.

Let's consider an account holder with `age` = 35, `income` = 1250, and `reward` = 2.[5] After executing the program with these inputs, it calculates the charge of 4. By inspecting the program, the account holder might come up with the following fact set

$$\{\underbrace{age \geq 27}_{f_1},\quad \underbrace{1250 \leq income < 1400}_{f_2},\quad \underbrace{reward = 2}_{f_3},\quad \underbrace{income/500 = 2}_{f_4}\}\ .$$

Note that the arithmetic operations have C semantics, hence the division is on integers. The selected facts are consistent and sufficiently precise, and the considered data point satisfies them.

The fact minimisation step using the unsat-core method tells us that f_4 can be omitted. Indeed, f_2 could also have been omitted; the minimisation is known to be ambiguous. As $f_2 \to f_4$, the account holder would select f_4 to increase the number of models. Hence, $\{f_1, f_3, f_4\}$ are transmitted to the agency. The agency starts with testing the consistency, symbolically executes the program, and checks for preciseness of the computed output. The verification of all proof obligations takes a negligible run time (ca. 170-230 ms), but an additional hint for the loop's upper bound is required for CBMC.

6 Related Work

Predicate Abstraction. Our approach has similarities to predicate abstraction [8]. Predicate abstraction is a verification technique in which the state space over

[5] See `examples/run.account_charges.1.sh` in the linked repository.

the program variables is projected on a state space over predicates. Each state in the space over predicates describes a set of concrete states. The goal of predicate abstraction is to select the smallest number of predicates such that the required properties are provable. In our case, we are interested in facts (predicates) which abstract the input space of the program and also result in the determined program outcome (property).

Data Minimisation. Goldsteen et al. [6] introduce an approach for data minimisation in machine learning models. They generalise the input based on certain features, removing those features that influence the program's result the least. Thus, personal data are still needed for training.

Biega et al. [1] examine data minimisation in recommendation systems. They identify two definitions for minimisation: *Global* data minimisation minimises the total data collected about all users while ensuring some minimum average quality level for all recommendations. *Per-user* data minimisation considers each user individually and minimises the data collected about them while ensuring some minimum quality level for their individual recommendations. In this dichotomy, our approach performs per-user minimisation. The alternative without access to the source code discussed in Section 3.3 falls back on global minimisation.

Unlike us, both of these papers consider a scenario (machine learning or product recommendations) in which the result based on the minimised data does not have to be exactly equal to the result based on the original data. In our approach, this could be modelled by not requiring the fact set to identify one unique result, but a set of results which only differ by some predefined distance. Alternatively, we could group the possible results into equivalence classes and require a set of facts to uniquely identify an equivalence class.

Formalisation of Privacy Properties. Mödersheim et al. [12] introduce the notion of (α, β)-privacy: Given a first-order formula α which models the intentionally released information and another formula β which models the information known by the intruder, (α, β)-privacy is achieved if the intruder cannot derive a statement from β which cannot also be derived from α. In our approach, minimality is achieved if the amount of information that can be derived from the transmitted data, but cannot be derived from the computation result, is minimal.

Differential privacy [5] is a notion of privacy in data sets. It is achieved when the conclusions drawn from the data set are not affected by the presence or absence of any individual datum. Local differential privacy is an approach of achieving differential privacy by randomising each participant's data before collecting it. Hao et al. [7] apply this approach to software execution traces; they define a way of modifying the traces to achieve differential privacy without compromising the validity of the data. Instead of falsifying the transmitted data, we minimise the amount of non-falsified information that is submitted. Like Hao et al.'s approach, ours also depends on the applicant's honesty; but in our case, this can be somewhat mitigated by choosing facts that can be certified later.

k-anonymity [15] is a related notion of privacy in data sets: It is achieved by replacing concrete data in a set by equivalence classes such that the data of any one individual becomes equivalent to that of at least $k-1$ other individuals. Since k-anonymity does not impose any restriction on the diversity within one equivalence class, knowing what class someone belongs to might still reveal some information about their exact data. Thus, improved versions of k-anonymity like l-diversity [11] and t-closeness [10] were introduced. Our approach is similar in that it partitions users into classes. However, instead of having a fixed equivalence relation, we choose one of several possible relations induced by the fact base, based on how much information the relation reveals about the user. Restrictions similar to those imposed by t-closeness could be useful in judging the usefulness of a fact base; e.g., a base which only includes facts fulfilled by most possible users will likely force the user to reveal more information.

Ziller et al. [16] offer a definition of privacy based on information flow. According to them, privacy is the ability of a sender to upper-bound the amount of information that can be computed from their message, independent of any prior knowledge of the receiver. This definition is closely linked, but not identical, to differential privacy. One important difference is that differential privacy is context-independent and only considers what conclusions can be drawn from the current data set. Our approach has a similar problem. The applicant cannot necessarily control or even know how much prior knowledge the agency has about them and what it does with that knowledge. They can however upper-bound the information content of the message they send, and (by inspecting the source code) ensure that only the information they sent is considered for the decision at hand. In addition, the information contained in the decision's result serves as a lower bound for how much they have to send.

7 Conclusion

We present an approach to minimise the transmitted personal information of the client to the agency. To achieve this, the clients send facts which describe their personal data instead of their concrete data. These facts need to be consistent with the original data and precise enough to guarantee the original outcome.

Our current implementations for fact extraction are very limited. Hence, we are considering a refinement process in which precise (but maybe too strong) fact sets are weakened, or imprecise sets of facts are strengthened by adding new facts. Beside the technical requirements, the extracted facts should also be comprehensible to the applicant and the agency.

References

1. Biega, A.J., Potash, P., III, H.D., Diaz, F., Finck, M.: Operationalizing the legal principle of data minimization for personalization. In: Huang, J., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., Liu, Y. (eds.) SIGIR 2020, Proc. pp. 399–408. ACM (2020). https://doi.org/10.1145/3397271.3401034
2. Clarke, E.M., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004, Proc. Lecture Notes in Computer

Science, vol. 2988, pp. 168–176. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15
3. Council of the European Union: General Data Protection Regulation (2016), `https://eur-lex.europa.eu/eli/reg/2016/679`
4. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Inc. (1976)
5. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**(3-4), 211–407 (2014). https://doi.org/10.1561/0400000042
6. Goldsteen, A., Ezov, G., Shmelkin, R., Moffie, M., Farkash, A.: Data minimization for GDPR compliance in machine learning models. CoRR (2020), `http://arxiv.org/abs/2008.04113`
7. Hao, Y., Latif, S., Zhang, H., Bassily, R., Rountev, A.: Differential Privacy for Coverage Analysis of Software Traces. In: Møller, A., Sridharan, M. (eds.) ECOOP 2021. LIPIcs, vol. 194, pp. 8:1–8:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.ECOOP.2021.8
8. Jhala, R., Podelski, A., Rybalchenko, A.: Predicate abstraction for program verification. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 447–491. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_15
9. Lanzinger, F., Weigl, A.: Towards a formal approach for data minimization in programs. In: Garcia-Alfaro, J., Muñoz-Tapia, J.L., Navarro-Arribas, G., Soriano, M. (eds.) Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 161–169. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-93944-1_11
10. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: 2007 IEEE 23rd International Conference on Data Engineering. pp. 106–115 (2007). https://doi.org/10.1109/ICDE.2007.367856
11. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data **1**(1), 3–es (mar 2007). https://doi.org/10.1145/1217299.1217302
12. Mödersheim, S., Viganò, L.: Alpha-beta privacy. ACM Trans. Priv. Secur. **22**(1) (Jan 2019). https://doi.org/10.1145/3289255
13. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008,Proc. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
14. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009, Proc. Lecture Notes in Computer Science, vol. 5504, pp. 288–302. Springer (2009). https://doi.org/10.1007/978-3-642-00596-1_21
15. Sweeney, L.: K-anonymity: A model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. **10**(5), 557–570 (oct 2002). https://doi.org/10.1142/S0218488502001648
16. Ziller, A., Mueller, T., Braren, R., Rueckert, D., Kaissis, G.: Privacy: An axiomatic approach (2022). https://doi.org/10.48550/ARXIV.2203.11586