CONTENTS

ABSTRACT 1

1. INTRODUCTION 1

1.1. ILLUSTRATIVE EXAMPLE 1

2. RELATED WORK 2

3. PRELIMINARIES 3

4. PRIVACY PRESERVATION 3

4.1. A FORMAL PRIVACY-PRESERVATION PROTOCOL 3

4.2. PRESERVING THE PRIVACY OF POLICY VIEWS 5

5. REPAIRING UNSAFE MAPPINGS 6

5.1. COMPUTING PARTIALLY-SAFE MAPPINGS 6

5.2. COMPUTING SAFE MAPPINGS 8

6. EXPERIMENTS 10

7. CONCLUSION 12

ACKNOWLEDGEMENTS 12

REFERENCES 12

Exchanging Data under Policy Views

Angela Bonifati
Lyon 1 University & Liris CNRS
Lyon, France
angela.bonifati@univ-lyon1.fr

Ugo Comignani
Grenoble INP Ensimag & LIG CNRS
Grenoble, France
ugo.comignani@grenoble-inp.fr

Efthymia Tsamoura
Samsung AI Research
Cambridge, UK
efi.tsamoura@samsung.com

ABSTRACT

Exchanging data between data sources is a fundamental problem in many data science and data integration tasks. In this paper, we focus on the data exchange problem in the presence of privacy constraints on the source data, which has been disregarded in the literature to date. By leveraging a logical privacy-preservation paradigm, the privacy restrictions are expressed as a set of *policy views* representing the information that is safe to expose over *all* instances of the source in order to exchange them with the target. We introduce a protocol that provides formal privacy guarantees and is *data-independent*, i.e., under certain criteria, it guarantees that the mappings leak no sensitive information independently of the instances lying in the source. Moreover, we design an algorithm for *repairing* an input mapping w.r.t. a set of policy views, in cases where the input mapping leaks sensitive information. We show that the repairing can build upon hard-coded and learning-based user preference functions and we show the trade-offs. Our empirical evaluation shows that repairing mappings is quite efficient, leading to repairing sets of 300 s-t tgds in an average time of 5s on a commodity machine. It also shows that the repairing based on learning is robust and has comparable runtimes with the hard-coded one.

KEYWORDS

privacy-preserving data integration, data exchange, mapping repairs

1 INTRODUCTION

Data exchange is a key process in data science and data integration pipelines, leading to translating data compliant with a source schema S and lying in a source database to a target database with a non-overlapping target schema T [1, 4, 17]. Data exchange is also part of metadata management operations [6], since the schema mappings between source and target also known as *source-to-target* (s-t) dependencies Σ_{st} (s-t tgds) are declarative expressions manipulating schema elements, i.e. metadata rather than data.

Despite a wealth of research on the topic, the privacy-aware variant of the data exchange problem has received little attention to date. However, recent data protection regulations such as EU GDPR or CCPA in the US bring the attention to the problem of protecting personal data when transferring data across countries and institutions, thus motivating our work. In a privacy-aware data exchange scenario (as exemplified in Figure 1), the source schema comes with a set of constraints called policy views $\mathcal{V}$ representing the data that is *safe* to expose to the target over *all instances* of the source. The policy views can be considered as user views on the data of the source and can encode possible formulations of the different purposes the data will undergo during the exchange process as in many data protection regulations.

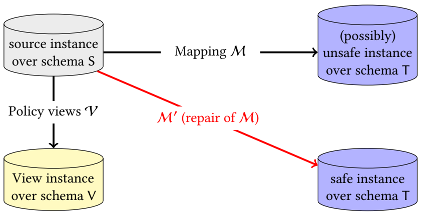

source instance over schema S
Mapping $\mathcal{M}$
(possibly) unsafe instance over schema T
Policy views $\mathcal{V}$
$\mathcal{M}'$ (repair of $\mathcal{M}$)
View instance over schema V
safe instance over schema T

Figure 1: A privacy-aware data exchange setting with mappings and policy views.

This process entails the repairing of the original mapping $\mathcal{M}$ into a mapping $\mathcal{M}'$ in order to make the exported target instance safe. However, in order to realize such a data exchange scenario, one needs to address the following issues: (1) given a set of privacy restrictions on the source schema, what would it mean for a data exchange setting to be safe under the proposed privacy restrictions?; (2) assuming that the privacy-preservation protocol is fixed, how could we assess the safety of a data exchange setting w.r.t. the privacy restrictions and provide *strong guarantees* of no privacy leakage?; finally, in case of privacy violations, (3) how could we *repair* the s-t tgds (and transform the mapping $\mathcal{M}$ into a repaired mapping $\mathcal{M}'$)?

To address the first issue, we build upon prior work on the logical foundations of privacy-preserving data integration [5, 21], and we tailor them to a data exchange setting. Hence, we define a set of s-t tgds to be safe w.r.t. the policy views if *every positive information* that is kept secret by the policy views is also kept secret by the s-t tgds. As we will see in subsequent sections and contrarily to previous work, our proposed privacy-preservation protocol is *data-independent* allowing us to provide strong privacy-preservation guarantees over all instances of the sources. As such, our work leads to the first practical framework establishing privacy-conscious data exchange. The above addresses the second aforementioned issue in that it enables a schema-level enforcement of the privacy-preserving protocol with strong guarantees. Regarding the third issue, we propose a repairing algorithm for the proposed privacy-preservation protocol in case of detected unsafety. Since multiple repairs are possible, such an algorithm might leverage techniques for learning the user preferences during the repairing process, which is also a desirable feature in privacy enforcement over sensitive data. In order to further illustrate the relevance of our problem, we illustrate a running example inspired by a real-life data exchange process between two different hospitals in the UK[1].

1.1 Illustrative example

Consider the source schema S of NHS consisting of the following relations: P, H_N, H_S, O and S as illustrated in Figure 2 (a). Relation P stores for each person registered with the hospital, his insurance number, his name, his ethnicity group and his county. Relations H_N and H_S store for each patient who has been admitted to some hospital in the north or the south of UK, his

[1] https://www.nhs.uk/

$$\begin{aligned} \text{Source schema } \mathsf{S} &= \{\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}); \mathsf{H}_{\mathsf{N}}(\mathsf{i},\mathsf{d}); \mathsf{H}_{\mathsf{S}}(\mathsf{i},\mathsf{d}); \\ &\qquad \mathsf{O}(\mathsf{i},\mathsf{t},\mathsf{p}); \mathsf{S}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c})\} \\ \text{Target schema } \mathsf{T} &= \{\mathsf{EthDis}(\mathsf{e},\mathsf{d}); \mathsf{CountyDis}(\mathsf{c},\mathsf{d}); \mathsf{SO}(\mathsf{e})\} \\ \text{View schemas } \mathsf{V} &= \{\mathsf{V}_1(\mathsf{e},\mathsf{d}); \mathsf{V}_2(\mathsf{c},\mathsf{d}); \mathsf{V}_3(\mathsf{t},\mathsf{p}); \mathsf{V}_4(\mathsf{e})\} \end{aligned}$$

(a) Schemas S, T and V

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \land \mathsf{H}_{\mathsf{N}}(\mathsf{i},\mathsf{d}) \rightarrow \mathsf{V}_1(\mathsf{e},\mathsf{d}) \tag{1}$$

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \land \mathsf{H}_{\mathsf{S}}(\mathsf{i},\mathsf{d}) \rightarrow \mathsf{V}_2(\mathsf{c},\mathsf{d}) \tag{2}$$

$$\mathsf{O}(\mathsf{i},\mathsf{t},\mathsf{p}) \rightarrow \mathsf{V}_3(\mathsf{t},\mathsf{p}) \tag{3}$$

$$\mathsf{S}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \rightarrow \mathsf{V}_4(\mathsf{e}) \tag{4}$$

(b) Policy views $\mathcal{V}$

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \land \mathsf{H}_{\mathsf{N}}(\mathsf{i},\mathsf{d}) \rightarrow \mathsf{EthDis}(\mathsf{e},\mathsf{d}) \tag{5}$$

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \land \mathsf{H}_{\mathsf{N}}(\mathsf{i},\mathsf{d}) \rightarrow \mathsf{CountyDis}(\mathsf{c},\mathsf{d}) \tag{6}$$

$$\mathsf{S}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \land \mathsf{O}(\mathsf{i},\mathsf{t},\mathsf{p}) \rightarrow \mathsf{SO}(\mathsf{e}) \tag{7}$$

(c) Mapping from S to T.

Figure 2: Schema and tgds in our illustrative example

insurance number and the reason for being admitted to the hospital. Relation O stores information related to patients in oncology departments and, in particular, their insurance numbers, their treatment and their progress. Finally, relation S stores for each student in UK, his insurance number, his name, his ethnicity group and his county.

Consider also the set $\mathcal{V}$ comprising the policy views (1)–(4). The policy views define the information that is safe to make available to the public. View (1) projects the ethnicity groups and the hospital admittance reasons for patients in the north of UK; view (2) projects the counties and the hospital admittance reasons for patients in the south of UK; view (3) projects the treatments and the progress of patients of oncology departments; view (4) projects the ethnicity groups of the school students. The policy views are compliant with the NSS privacy preservation protocol that is adopted at the hospital. Precisely, the NSS privacy preservation protocol considers as unsafe any non-evident piece of information that can potentially de-anonymize an individual. For example, views (1) and (2) do not leak any sensitive information concerning the precise address of patients. Indeed, they include patients from a very large geographical area thus implying that the probability of de-anonymizing a patient is significantly small. Similarly, views (3) and (4) are considered to be safe: the probability of de-anonymizing patients of the oncology department from view (3) is zero, since there is no way to link a patient to his treatment or his progress, while view (4) projects information which is already evident to public.

Finally, consider the following set of source-to-target dependencies Σ_{st}. Dependencies (5) and (6) project similar information with the views (1) and (2), respectively. However, contrarily to the views, they solely focus on patients in the north of UK. Finally, dependency (7) projects the ethnicity groups of students who have been in some oncology department, whereas view V_4 aims at concealing the information about the department in which a student has been admitted.

The above example shows that the policy views defined within one hospital might be in stark contrast with the mappings used to export patient's information to another hospital. This motivates the aforementioned questions (1), (2) and (3) about establishing formal guarantees for privacy preservation as well as enabling repairing of the mappings in order to make them safe. To the best of our knowledge, our work is the first to provide practical algorithms for a logical privacy-preservation paradigm effective in a real system [10], described as an open research challenge in [5, 21]. Our technique is inherently data-independent thus bringing the advantage that both the safety test and the repairing operations are executed on the metadata provided through the mappings and not on the underlying data instances.

The paper is organized as follows. Section 2 discusses the related work. Section 3 presents the basic concepts and notions. Section 4 lays our privacy preservation protocol. Section 5 presents our repairing algorithms and their properties. mechanism. Section 6 outlines the experimental results, while Section 7 concludes our work. The code base along with the experimental data are publicly available at [12].

2 RELATED WORK

Privacy in data integration. Safety of secret queries formulated against a global schema and adhering to the certain answers semantics has been tackled in previous theoretical work [21]. They define the optimal attack that characterizes a set of queries that an attacker can issue to which no further queries can be added to infer more information. They then define the privacy guarantees against the optimal attack by considering the static and the dynamic case, the latter corresponding to modifications of the schemas or the GLAV mappings. The same definition of secret queries and privacy setting is adopted in [5], which instead focuses on boolean conjunctive queries as policy views and on the notion of safety with respect to a given mapping. An ontology-based integration scenario is assumed in which the target instance is produced via a set of mappings starting from an underlying data source. Whereas they study the complexity of the view compliance problem in both data-dependent and data-independent setting, we focus on the latter and extend it to non-boolean conjunctive queries as policy views. We further consider multiple policy views altogether in the design of a practical algorithm for checking the safety of schema mappings and for repairing the mappings in case of violations.

Privacy in data publishing. Data publishing accounts for the settings in which a view exports or publishes the information of an underlying data source. Privacy and information disclosure in data publishing linger over the problem of avoiding the disclosure of the content of the view under a confidential query. A probabilistic formal analysis of the query-view security model has been presented in [20], where they offer a complete treatment of the multi-party collusion and the use of external adversarial knowledge. Access control policies using cryptography are used in [20] to enforce the authorization to an XML document. Our work differs from theirs on both the considered setting, as well as the adopted techniques and the adopted privacy protocol. Striking the balance between utility and privacy in a logic-based framework has been the object of investigation in recent studies focusing on data publishing for Linked Data [13, 14, 19]. The problem there is remarkably different from ours since they focus on publishing a single RDF dataset by applying privacy and utility queries in SPARQL, checking for their compatibility, and for update operations realizing the privacy and utility constraints.

Controlled Query Evaluation. Controlled Query Evaluation is a confidentiality enforcement framework introduced in [23] and refined in [9],[7] and [8], in which a policy declaratively specifies sensitive information and confidentiality is enforced by a censor. Provided a query as input, a censor verifies whether the query leads to a violation of the policy and in case of a violation it returns a distorted answer. It has been recently adopted in ontologies expressed with Datalog-like rules and in lightweight Description Logics [18]. They assume that the policies are only known to database administrators and not to ordinary users and that the data has protected access through a query interface. Our assumptions and setting are quite different, since our multiple policy views are accessible to every user and our goal is to render the s-t mappings safe with respect to a set of policies via repairing and rewriting.

Data privacy. Previous work has addressed access control to protect database instances at different levels of granularity [22], in order to combine encrypted query processing and authorization rules. Our work being logic-based and declarative does not deal with these authorization methods, as well as does not consider any concrete privacy or anonymization algorithms operating on data instances, such as differential privacy [15] and k-anonymity [24]. Further exploring the connection between concrete privacy enforcement and logic-based privacy formalisms is the subject of future investigation.

Data exchange. The vast literature on data exchange [17] has inspired our work. In the considered scenarios, the source and target schema are considered along with s-t mappings and target dependencies, the latter being both egds and tgds. Similarly, past work on degugging schema mappings [11] has focused on all possible routes generated by the exchange process when incomplete or undefined values in one or more variables are exported from the source instance. By opposite, we focus in this paper on the case in which s-t mappings are coupled with source dependencies under the form of policy views, the latter being typical in privacy scenarios and unexplored in the classical data exchange setting.

3 PRELIMINARIES

Relational symbols and critical instances. Let Const, Nulls, and Vars be mutually disjoint, infinite sets of *constant values*, *labeled nulls*, and *variables*, respectively. A *schema* is a set of *relation names* (or just *relations*), each associated with a nonnegative integer called *arity*. A *relational atom* has the form $R(\vec{t})$ where R is an n-ary relation and $\vec{t}$ is an n-tuple of *terms*, where a term is either a constant, a labelled null, or a variable. An *equality atom* has the form $t_1 = t_2$ where t_1 and t_2 are terms. An atom is called ground or fact, when it does not contain any variables. A position in an n-ary atom A is an integer $1 \leq i \leq n$. We denote by $A|_i$, the i-th term of A. An instance I is a set of relational facts. An atom (resp. an instance) is *null-free* if it does not contain labelled nulls. The *critical instance* of a schema S, denoted as Crt_S, is the instance containing a fact of the form $R(\vec{*})$, for each n-ary relation $R \in \mathsf{S}$, where $*$ is called the *critical constant* and $\vec{*}$ is an n-ary vector. A *substitution* σ is a mapping from variables into constants or labelled nulls.

Dependencies and queries A *dependency* describes the semantic relationship between relations. A *Tuple Generating Dependency* (tgd) is a formula of the form $\forall\vec{x}\ \lambda(\vec{x}) \rightarrow \exists\vec{y}\ \rho(\vec{x},\vec{y})$, where $\lambda(\vec{x})$ and $\rho(\vec{x},\vec{y})$ are conjunctions of relational, null-free atoms. An *Equality Generating Dependency* (egd) is a formula of the form $\forall\vec{x}\ \lambda(\vec{x}) \rightarrow x_i = x_j$, where $\lambda(\vec{x})$ is a conjunction of relational, null-free atoms. We usually omit the quantification for brevity. We refer to the left-hand side of a tgd or an egd δ as the *body*, denoted as $\mathsf{body}(\delta)$, and to the right-hand side as the *head*, denoted as $\mathsf{head}(\delta)$. An instance I satisfies a dependency δ, written $I \models \delta$ if each homomorphism from $\mathsf{body}(\delta)$ into I can be extended to a homomorphism h' from $\mathsf{head}(\delta)$ into I. An instance I satisfies a set of dependencies Σ, written as $I \models \Sigma$, if $I \models \delta$ holds, for each $\delta \in \Sigma$. The *solutions* of an instance I w.r.t. Σ is the set of all instances J such that $J \supseteq I$ and $J \models \Sigma$. A solution is called *universal* if it can be homomorphically embedded to each solution of I w.r.t. Σ.

A *conjunctive query* (CQ) is a formula of the form $\exists\vec{y}\ \bigwedge_i A_i$, where A_i are relational, null-free atoms. A CQ is boolean if it does not contain any free variables. A substitution σ is an *answer* to a CQ Q on an instance I if the domain of σ is the free variables of Q, and if σ can be extended to a homomorphism from $\bigwedge_i A_i$ into I. We denote by $Q(I)$, the answers to Q on I.

Let S be a source schema and let T be a target schema. A *mapping* $\mathcal{M}$ from S to T is defined as a triple $(\mathsf{S},\mathsf{T},\Sigma)$, where $\Sigma = \Sigma_{st}$;, i.e. the set of the s-t dependencies over S and T. We usually refer to the dependencies in Σ_{st} as *mappings*. A variable x of a mapping $\mu \in \Sigma_{st}$ is called exported if it occurs both in the body and the head of μ. We denote by $\mathsf{exported}(\mu)$, the set of exported variables of μ. The inverse of set of s-t dependencies Σ_{st}, denoted as Σ_{st}^{-1} is the set consisting, for each mapping μ in Σ_{st} of the form $\lambda(\vec{x}) \rightarrow \rho(\vec{x},\vec{y})$, a mapping μ^{-1} of the form $\rho(\vec{x},\vec{y}) \rightarrow \lambda(\vec{x})$. We focus on *GAV* mappings in this paper, thus assuming that $\vec{y}$ is empty. Moreover, we consider the setting in which Σ only consists of Σ_{st} thus not including Σ_t. This implies that target egds and target tgds are excluded, since, despite their usage in data exchange, their role is less understood in the privacy-preserving variant considered in this paper.

4 PRIVACY PRESERVATION

In this section, we introduce our notion of privacy preservation. Let $\mathcal{V}$ be a set of *policy views* over S, representing the information that is safe to expose for instances I of S. Our goal is to verify whether a user-defined mapping $\mathcal{M} = (\mathsf{S},\mathsf{T},\Sigma)$ is safe w.r.t. the views in a set $\mathcal{V}$. Below, we will introduce a notion for assessing the safety of a GAV mapping $\mathcal{M}_2$ with respect to a GAV mapping $\mathcal{M}_1$, when both make use of the same source schema S. Moreover, let $\Sigma_i = \Sigma_{st_i}$ be the dependencies associated with $\mathcal{M}_i$.

4.1 A formal privacy-preservation protocol

Our notion of privacy preservation builds on the logical foundations introduced in [5] for ontology-based data integration for boolean queries. However, we extend the notion of privacy preservation from [5] to a relational data exchange setting in the presence of non-boolean conjunctive queries. First, we define the notion of disclosure of a CQ by a mapping as follows:

Definition 4.1. A mapping $\mathcal{M} = (\mathsf{S},\mathsf{T},\Sigma)$ *does not disclose a CQ* p *over* S *on any instance of* S, if for each instance I of S there exists an instance I' such that $I \equiv_{\mathcal{M}} I'$ and $p(I') = \emptyset$.

The problem of checking whether a mapping $\mathcal{M}$ over S does not disclose a boolean and constants-free CQ p on any instance of S is decidable for GAV mappings consisting of CQ views [5]. In particular, $\mathcal{M}$ does not disclose p on any instance of S if and only if there does not exist a homomorphism from p into the *unique* instance computed by the *visible chase* $\mathsf{visChase}_\mathsf{S}(\Sigma)$ of Σ under the critical instance Crt_S of S. The visible chase computes a *universal source instance* defined as follows:

Definition 4.2. 7 Given a mapping $\mathcal{M} = (\mathsf{S},\mathsf{T},\Sigma)$, an instance I is a *universal source instance* over S if for any instance J over the source schema S, there exists an homomorphism h from J into I

such that for any constant c from J that is made visible through $\mathcal{M}$, $h(c) = *$.

The only constant occurring in the instance computed by $\mathsf{visChase_S}(\Sigma)$ is the critical constant $*$ and it represents any other constant that can occur in the source instance.

We introduce our own variant of the visible chase, which organizes the facts derived during chasing into subinstances called *bags*. Algorithm 1 describes the steps of the proposed variant. Please note that Algorithm 1 derives the same set of facts with the algorithm from [5]. However, instead of keeping these facts in a single set, we keep them in separate bags. Before presenting Algorithm 1, we will introduce a couple of useful new notions. The first notion serves the need of defining derived egds that allow to unify as many labeled nulls as possible with the critical constant in the target instance. The second notion allows to define relevant bags for which this unification must hold. Both notions are exploited by the visible chase (Algorithm 1) whose last step triggers the obtained egds.

Definition 4.3. Consider an instance I. Consider also a s-t tgd δ and a homomorphism h from $\mathsf{body}(\delta)$ into I, such that $h(x) \in \mathsf{Nulls}$, for some $x \in \mathsf{exported}(\delta)$. Then, we say that the egd

$$\mathsf{body}(\delta) \rightarrow \bigwedge_{\forall x \in \mathsf{exported}(\delta): h(x) \in \mathsf{Nulls}} x \approx * \tag{8}$$

is *derived* from δ in I. For an egd ϵ that is derived from a s-t tgd δ in I, $\mathsf{tgd}(\epsilon)$ denotes δ. For a set of s-t tgds Σ and an instance I, $\Sigma_\approx$ is the set comprising for each $\delta \in \Sigma$, the egd that is derived from δ in I.

Definition 4.4. Consider an instance I, whose facts are organized into the bags $\beta_1, \ldots, \beta_m$. Consider also a derived egd δ of the form (8) and an active trigger h for δ in I. A bag β_i is *relevant* for δ and h in I, where $1 \le i \le m$, if some fact $F \in h(\mathsf{body}(\delta))$ occurs in β_i and if some $h(x)$ is a labeled null occurring in β_i, where $x \in \mathsf{exported}(\delta)$.

Let $\beta_{j_1}, \ldots, \beta_{j_k} \subseteq \beta_1, \ldots, \beta_m$ be the set of bags that are relevant for δ and h in I. Let $\nu = \{h(x_j) \mapsto h(x_i)\}$ if $h(x_i) = *$, and $\nu = \{h(x_i) \mapsto h(x_j)\}$ if $h(x_i) \notin \mathsf{Const}$, where x_i, x_j are variables from $\mathsf{exported}(\delta)$. Then, the *derived* bag β for δ and h in I consists of the facts in $\bigcup_{l=1}^{k} \nu(\beta_{j_l})$. The bags $\beta_{j_1}, \ldots, \beta_{j_k}$ are called the *predecessors* of β. We use $\beta_{j_l} \prec \beta$ to denote that β_{j_l} is a predecessor of β, for $1 \le l \le k$.

We are now ready to proceed with the description of Algorithm 1. Given a s-t mapping, Algorithm 1 computes a universal source instance whose facts are organized into bags. Algorithm 1 first computes the instance I_0 by chasing $\mathsf{Crt_S}$ using the s-t tgds, line 1. It then chases I_0 with the inverse s-t tgds Σ^{-1}, line 2. and proceeds by chasing I_1 with the set of all derived egds $\Sigma_\approx$, for each $\delta \in \Sigma$ in I_1, line 4. Algorithm 1 computes a fresh bag at each chase step. In particular, for each active trigger h for δ in I, Algorithm 1 adds a fresh bag with facts $h'(\mathsf{head}(\delta))$, if $\delta \in \Sigma \cup \Sigma^{-1}$, line 9; otherwise, if $\delta \in \Sigma_\approx$, then it adds the derived bag for δ and h in I, see Definition 4.4, line 20.

Note that, $\Sigma_\approx$ aims at "disambiguating" as many labeled nulls occurring in I_1 as possible, by unifying them with the critical constant $*$. Since $*$ represents the information that is "visible" to a third-party, chasing with $\Sigma_\approx$ computes the *maximal information* from the source instance a third-party has access to. Note that Algorithm 1 always terminates [5]. Let $B = \mathsf{visChase_S}(\Sigma)$. We will denote by $I_\mathsf{S}(\Sigma)$, the instance $\bigcup_{\beta \in B} \beta$.

Algorithm 1 $\mathsf{visChase_S}(\Sigma)$

```
1:  $B_0 := \mathsf{bagChaseTGDs}(\Sigma, \mathsf{Crt_S})$
2:  $B_1 := \mathsf{bagChaseTGDs}(\Sigma^{-1}, \bigcup_{\beta \in B_0} \beta \setminus \mathsf{Crt_S})$
3:  Let $\Sigma_\approx$ be the set of all derived egds $\Sigma_\approx$, for each $\delta \in \Sigma$ in $I_1$
4:  return $\mathsf{bagChaseEGDs}(\Sigma_\approx, B_0 \cup B_1)$

5:  procedure $\mathsf{bagChaseTGDs}(\Sigma, I)$
6:      $B := \emptyset$
7:      for each $\delta \in \Sigma$ do
8:          for each active trigger $h : \mathsf{body}(\delta) \rightarrow I$ do
9:              create a fresh bag $\beta$ with facts $h'(\mathsf{head}(\delta))$
10:             add $\beta$ to $B$
11:     return $B$

12: procedure $\mathsf{bagChaseEGDs}(\Sigma_\approx, B)$
13:     $i := 0$; $I_i := \bigcup_{\beta \in B} \beta$
14:     do
15:         $i := i + 1$
16:         for each ($\delta \in \Sigma_\approx$ of the form (8) do
17:             for each active trigger $h : \mathsf{body}(\delta) \rightarrow I_{i-1}$ do
18:                 if $h(x) \neq *$, for some $x \in \mathsf{exported}(\delta)$ then
19:                     Let $\beta$ be the derived bag for $\delta$ and $h$ in $I_{i-1}$
20:                     add $\beta$ to $B$
21:                     $I_i := I_i \cup \beta$
22:     while $I_{i-1} \neq I_i$
23:     return $B$
```

Example 4.5. We demonstrate the visible chase algorithm over the policy views and the s-t dependencies from Example 1.1. We show how the algorithm runs first on the policy views $\mathcal{V}$ and then show the computation on Σ_{st}.

We first present the computation of $I_\mathsf{S}(\mathcal{V}) = \bigcup_{\beta \in \mathsf{visChase_S}(\mathcal{V})} \beta$.

The critical instance $\mathsf{Crt_S}$ of S consists of the facts shown in the following Eq. (9)

$$\mathsf{P}(*, *, *, *) \quad \mathsf{H_N}(*, *) \quad \mathsf{H_S}(*, *) \quad \mathsf{O}(*, *, *) \quad \mathsf{S}(*, *, *, *) \tag{9}$$

where $*$ is the critical constant.

The instance I_1 computed by chasing the output of line 1 using $\mathcal{V}^{-1}$ will consist of the facts

$$\begin{array}{llll} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}) & \mathsf{H_N}(\mathsf{n_i}, *) & \mathsf{O}(\mathsf{n_i''}, *, *) & (I_1) \\ \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *) & \mathsf{H_S}(\mathsf{n_i'}, *) & \mathsf{S}(\mathsf{n_i'''}, \mathsf{n_n'''}, *, \mathsf{n_c'''}) & \end{array}$$

where the constants prefixed by n are labeled nulls created while chasing $\mathsf{Crt_S}$ with the inverse mappings. Since there exists no homomorphism from the body of any s-t tgd into I_1 mapping an exported variable into a labeled null, $\Sigma_\approx$ will be empty, see Definition 4.3. Thus, $I_\mathsf{S}(\mathcal{V}) = I_1$.

We next present the computation of $I_\mathsf{S}(\Sigma_{st}) = \bigcup_{\beta \in \mathsf{visChase_S}(\Sigma_{st})} \beta$. The instance I_1' computed by chasing the output of line 1 by Σ_{st}^{-1} will consist of the facts

$$\begin{array}{llll} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}) & \mathsf{H_N}(\mathsf{n_i}, *) & \mathsf{S}(\mathsf{n_i''}, \mathsf{n_n''}, *, \mathsf{n_c'}) & (I_1') \\ \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *) & \mathsf{H_N}(\mathsf{n_i'}, *) & \mathsf{O}(\mathsf{n_i''}, \mathsf{n_t''}, \mathsf{n_p''}) & \end{array}$$

Since there exists a homomorphism from the body of μ_e into I_1' mapping the exported variable e into the labeled null $\mathsf{n_e}$, and since there exists another homomorphism from the body of μ_c into I_1' mapping the exported variable c into the labeled null $\mathsf{n_c}$, $\Sigma_\approx$ will comprise the egds ϵ_1 and ϵ_2 shown below

$$\mathsf{P}(\mathsf{i}, \mathsf{n}, \mathsf{e}, \mathsf{c}) \wedge \mathsf{H_N}(\mathsf{i}, \mathsf{d}) \rightarrow \mathsf{e} \approx * \tag{ϵ_1}$$

$$\mathsf{P}(\mathsf{i}, \mathsf{n}, \mathsf{e}, \mathsf{c}) \wedge \mathsf{H_N}(\mathsf{i}, \mathsf{d}) \rightarrow \mathsf{c} \approx * \tag{ϵ_2}$$

The last step of the visible chase involves chasing I_1' using $\Sigma_{\approx}$. W.l.o.g, assume that the chase considers first ϵ_1 and then ϵ_2. During the first step of the chase, there exists a homomorphism from $\mathsf{body}(\epsilon_1)$ into I_1'. Hence, $\mathsf{n_e} = *$. During the second step of the chase, there exists a homomorphism from $\mathsf{body}(\epsilon_2)$ into I_1' and, hence, $\mathsf{n_c} = *$. The instance computed at the end of the second round of the chase will consist of the facts

$$\begin{array}{lll} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, *) & \mathsf{H_N}(\mathsf{n_i}, *) & \mathsf{H_N}(\mathsf{n_i'}, *) \\ \mathsf{S}(\mathsf{n_i''}, \mathsf{n_n''}, *, \mathsf{n_c'}) & \mathsf{O}(\mathsf{n_i''}, \mathsf{n_t''}, \mathsf{n_p''}) & \end{array} \tag{10}$$

Since there exists no active trigger for ϵ_1 or ϵ_2 in the above instance (Eq (10)), the chase will terminate.

The facts in $I_\mathsf{S}(\Sigma_{st})$ will be organized into the following bags β_1–β_5 (one bag per line)

$$\begin{aligned}
\mathsf{SO}(\mathsf{e}) &\xrightarrow{\langle \mu_s^{-1}, h_1 \rangle} \mathsf{S}(\mathsf{n_i''}, \mathsf{n_n''}, *, \mathsf{n_c'}), \mathsf{O}(\mathsf{n_i''}, \mathsf{n_t''}, \mathsf{n_p''}) \\
\mathsf{CountyDis}(\mathsf{c}, \mathsf{d}) &\xrightarrow{\langle \mu_c^{-1}, h_2 \rangle} \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *), \mathsf{H_N}(\mathsf{n_i'}, *) \\
\mathsf{EthDis}(\mathsf{e}, \mathsf{d}) &\xrightarrow{\langle \mu_e^{-1}, h_3 \rangle} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}), \mathsf{H_N}(\mathsf{n_i}, *) \\
\mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *), \mathsf{H_N}(\mathsf{n_i'}, *) &\xrightarrow{\langle \epsilon_1, h_4 \rangle} \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, *, *), \mathsf{H_N}(\mathsf{n_i'}, *) \\
\mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}), \mathsf{H_N}(\mathsf{n_i}, *) &\xrightarrow{\langle \epsilon_2, h_5 \rangle} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, *), \mathsf{H_N}(\mathsf{n_i}, *)
\end{aligned}$$

$$\begin{aligned}
h_1 &= \{\mathsf{i} \mapsto \mathsf{n_i'}, \mathsf{n} \mapsto \mathsf{n_n'}, \mathsf{e} \mapsto \mathsf{n_e}, \mathsf{c} \mapsto *, \mathsf{d} \mapsto *\} \\
h_2 &= \{\mathsf{c} \mapsto *, \mathsf{d} \mapsto *\} \\
h_3 &= \{\mathsf{e} \mapsto *, \mathsf{d} \mapsto *\} \\
h_4 &= \{\mathsf{i} \mapsto \mathsf{n_i'}, \mathsf{n} \mapsto \mathsf{n_n'}, \mathsf{e} \mapsto \mathsf{n_e}, \mathsf{c} \mapsto *, \mathsf{d} \mapsto *\} \\
h_5 &= \{\mathsf{i} \mapsto \mathsf{n_i}, \mathsf{n} \mapsto \mathsf{n_n}, \mathsf{e} \mapsto *, \mathsf{c} \mapsto \mathsf{n_c}, \mathsf{d} \mapsto *\}
\end{aligned}$$

The contents of the bags correspond to the right-hand side of the arrows. However, for presentation purposes, we also show the related dependency δ and the homomorphism h that lead to the derivation of each bag (shown at the top of each arrow), as well as, the facts in $h(\mathsf{body}(\delta))$ (left-hand side of each arrow). The obtained bags will be part of the universal source instance $I_\mathsf{S}(\Sigma_{st})$. Such an instance will be used in Section 5 in order to apply the notion of safety in the repairing of the underlying mappings Σ_{st}.

4.2 Preserving the privacy of policy views

We consider a mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ to be safe w.r.t. a view mapping $\mathcal{M}_\mathsf{V} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ (with $\mathcal{V}$ being the set of policy views and V being the schema of the views as shown in Figure 1), if $\mathcal{M}$ does not disclose the information that is also not disclosed by $\mathcal{M}_\mathsf{V}$. Definition 4.6 and Theorem 4.7 presented below formalize our notion of privacy preservation and show that there exists a simple process for verifying whether $\mathcal{M}$ is safe w.r.t. $\mathcal{M}_\mathsf{V}$.

Definition 4.6. A mapping $\mathcal{M}_2 = (\mathsf{S}, \mathsf{T}_2, \Sigma_2)$ *preserves the privacy of a mapping* $\mathcal{M}_1 = (\mathsf{S}, \mathsf{T}_1, \Sigma_1)$ *on all instances of* S, if for each constants-free CQ p over S, if p is not disclosed by $\mathcal{M}_1$ on any instance of S, then p is not disclosed by $\mathcal{M}_2$ on any instance of S.

Theorem 4.7. *A mapping* $\mathcal{M}_2 = (\mathsf{S}, \mathsf{T}_2, \Sigma_2)$ preserves the privacy of a mapping $\mathcal{M}_1 = (\mathsf{S}, \mathsf{T}_1, \Sigma_1)$ on all instances of S, *if and only if there exists a homomorphism h from* $I_\mathsf{S}(\Sigma_2)$ *into* $I_\mathsf{S}(\Sigma_1)$, *such that* $h(*) = *$.

Proof. (Sketch) First we show that the following holds:

Lemma 4.8. *A mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ does not disclose a constants-free CQ p over S on any instance of S, *iff* $\vec{*} \notin p(J)$, *where* $J = I_\mathsf{S}(\Sigma_{st})$.

Proof. By adapting the proof technique of Theorem 16 from [5], we can show that $J = I_\mathsf{S}(\Sigma_{st})$ is a *universal* source instance $I_\mathsf{S}(\Sigma)$ satisfying the following property: for each pair of source instances I and I', such that I' is indistinguishable from I w.r.t. the mapping $\mathcal{M}$, there exists a homomorphism h from I' into $I_\mathsf{S}(\Sigma)$ mapping each schema constant into the critical constant $*$. Due to the existence of a homomorphism h from I' into $I_\mathsf{S}(\Sigma)$, for each pair of indistinguishable source instances I and I', we can see that if $\vec{*} \notin p(J)$ for a constants-free CQ p, then $p(I') = \emptyset$. Due to the above and due to Definition 4.1, it follows that $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ does not disclose a constants-free CQ p over S on any instance of S. □

Lemma 4.8 states that, in order to check if a constants-free CQ is safe according to Definition 4.1, we need to check if the critical tuple is among the answers to p over the instance computed by $\mathsf{visChase}_\mathsf{S}(\Sigma)$. Next, we show the following lemma.

Lemma 4.9. *Given two instances* I_1 *and* I_2, *the following are equivalent*

(1) for each CQ p, *if* $\vec{u} \in p(I_1)$, *then* $\vec{u} \in p(I_2)$, *where* $\vec{u}$ *is a vector of constants*

(2) there exists a homomorphism from I_1 *to* I_2 *preserving the constants of* I_1

Proof of Lemma 4.9. (2)⇒(1). Suppose that there exists a homomorphism h from I_1 to I_2 preserving the constants of I_1. Suppose also that $\vec{u} \in p(I_1)$, with p being a CQ. This means that there exists a homomorphism h_1 from p into I_1 mapping each free variable x_i of p into u_i, for each $1 \leq i \leq n$, where n is the number of free variables of p. Since the composition of two homomorphisms is a homomorphism and since h preserves the constants of I_1 due to the base assumptions, this means that $h \circ h_1$ is a homomorphism from p into I_2 mapping each free variable x_i of p into t_i, for each $1 \leq i \leq n$. This completes this part of the proof.

(1)⇒(2). Let p_1 be a CQ formed by creating a non-ground atom $R(y_1, \dots, y_n)$ for each ground atom $R(u_1, \dots, u_n) \in I_1$, by taking the conjunction of these non-ground atoms and by converting into an existentially quantified variable every variable created out of some labelled null. Let $\vec{x}$ denote the free variables of p_1 and let $n = |\vec{x}|$. From the above, it follows that there exists a homomorphism h_1 from p_1 into I_1 mapping each $x_i \in \vec{x}$ into some constant occurring in I_1. Let $\vec{u} \in p_1(I_1)$. From (1), it follows that $\vec{u} \in p_1(I_2)$ and, hence, there exists a homomorphism h_2 from p_1 into I_2 mapping each $x_i \in \vec{x}$ into u_i, for each $1 \leq i \leq n$. Since h_1 ranges over all constants of I_1 and since $h_1(x_i) = h_2(x_i)$ holds for each $1 \leq i \leq n$, it follows that there exists a homomorphism from I_1 to I_2 preserving the constants of I_1. This completes the second part of the proof. □

Lemma 4.9 can be restated as follows:

Lemma 4.10. *Given two instances* I_1 *and* I_2, *the following are equivalent*

(1) for each CQ p, *if* $\vec{t} \notin p(I_2)$, *then* $\vec{t} \notin p(I_1)$

(2) there exists a homomorphism from I_1 *to* I_2

We are now ready to return to the main part of the proof. Given a CQ p over a source schema S, and a mapping $\mathcal{M}$ defined as the triple $(\mathsf{S}, \mathsf{T}, \Sigma)$, where T is a target schema and Σ is a set of s-t dependencies, we know from Proposition 4.8 that if $\mathcal{M}$ discloses p on some instance of S, then there exists a homomorphism of p into $\mathsf{visChase}_\mathsf{S}(\Sigma)$ mapping the free variables of p into the critical constant $*$.

From the above, we know that $\mathcal{M}_2$ does not preserve the privacy of $\mathcal{M}_1$ if there exists a CQ p over S, such that $\vec{*} \notin J_1$ and $\vec{*} \in J_2$, where $J_1 = I_{\mathsf{S}}(\Sigma_1)$ and $J_2 = I_{\mathsf{S}}(\Sigma_2)$. We will now prove that $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ iff there exists a homomorphism from J_2 into J_1 that preserves the critical constant $*$. This will be referred to as conjecture (C).

($\Rightarrow$) If $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$, then for each CQ p, if $\vec{*} \notin p(J_1)$, then $\vec{*} \notin p(J_2)$. From the above and from Lemma 4.10, it follows that there exists a homomorphism $\phi : J_2 \longrightarrow J_1$, such that $\phi(*) = *$.

($\Leftarrow$) The proof proceeds by contradiction. Assume that there exists a homomorphism h from J_2 into J_1 preserving $*$, but $\mathcal{M}_2$ does not preserve the privacy of $\mathcal{M}_1$. We will refer to this assumption as assumption (A_1). From assumption (A_1) and the discussion above it follows that there exists a CQ p over S such that $\vec{*} \notin p(J_1)$ and $\vec{*} \in p(J_2)$. Let h_2 be the homomorphism from p into J_2 mapping its free variables into $*$. Since the composition of two homomorphisms is a homomorphism, this means that $h \circ h_2$ is a homomorphism from p into J_1 mapping its free variables into $*$, i.e., $\vec{*} \in p(J_1)$. This contradicts our original assumption and hence concludes the proof of conjecture (C). Conjecture (C) witnesses the decidability of the instance-independent privacy preservation problem: in order to verify whether $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ we only need to check if there exists a homomorphism $\phi : I_{\mathsf{S}}(\Sigma_2) \longrightarrow I_{\mathsf{S}}(\Sigma_1)$, such that $\phi(*) = *$. □

Theorem 4.7 states that in order to verify whether $\mathcal{M}_2$ is safe w.r.t. $\mathcal{M}_1$, we need to compute $I_{\mathsf{S}}(\Sigma_1)$ and $I_{\mathsf{S}}(\Sigma_2)$ and check if there exists a homomorphism from the second instance into the first one that maps $*$ into itself. If there exists such a homomorphism, we say that $I_{\mathsf{S}}(\Sigma_1)$ is *safe* w.r.t. $I_{\mathsf{S}}(\Sigma_2)$, or simply safe, and we say that it is unsafe otherwise.

Example 4.11. Continuing with Example 1.1, we can see that the s-t tgds are not safe w.r.t. the policy views according to Theorem 4.7, since there does not exist a homomorphism from the instance $I_{\mathsf{S}}(\Sigma_{st})$ into the instance $I_{\mathsf{S}}(\mathcal{V})$. This means that there exists information which is disclosed by Σ_{st} in some instance that satisfies Σ_{st}, but it is not disclosed by $\mathcal{V}$. Indeed, from $\mathsf{S}(\mathsf{n}_\mathsf{i}'', \mathsf{n}_\mathsf{n}'', *, \mathsf{n}_\mathsf{c}')$ and $\mathsf{O}(\mathsf{n}_\mathsf{i}'', \mathsf{n}_\mathsf{t}'', \mathsf{n}_\mathsf{p}'')$, we can see that we can potentially leak the identity of a student who has been to an oncology department. This can happen if there exists only one student in the school coming from a specific ethnicity group and this ethnicity group is returned by μ_s. Please note that the policy views are safe w.r.t. this leak. Indeed, it is impossible to derive this information through reasoning over the returned tuples under the input instance and the views V_3 and V_4.

Furthermore, by looking at the facts $\mathsf{P}(\mathsf{n}_\mathsf{i}, \mathsf{n}_\mathsf{n}, *, *)$ and $\mathsf{H}_\mathsf{N}(\mathsf{n}_\mathsf{i}, *)$, we can see that we can potentially leak the identity and the disease of a patient who has been admitted to some hospital in the north of UK. This can happen if there exists only one patient who relates to the county and the ethnicity group returned by μ_e and μ_c. Note that the policy views V_1 and V_2 do not leak this information, since it is impossible to obtain the county and the ethnicity group of an NHS patient at the same time.

5 REPAIRING UNSAFE MAPPINGS

In Section 4, we presented our privacy preservation protocol and a technique for verifying whether a mapping is safe w.r.t. another one, over all source instances. This section presents an algorithm for repairing an unsafe mapping $\mathcal{M}$ w.r.t. a set of policy views $\mathcal{V}$. This is a fundamental operation needed to amend mappings whenever the policy views are modified and become unsafe (e.g. in the presence of data protection regulations).

Algorithm 2 $\mathsf{repair}(\Sigma, \mathcal{V}, \mathsf{prf}, n)$

```
$\Sigma_1 := \mathsf{frepair}(\Sigma, \mathcal{V}, \mathsf{prf})$
$\Sigma_2 := \mathsf{srepair}(\Sigma_1, \mathcal{V}, \mathsf{prf}, n)$
**return** $\Sigma_2$
```

Algorithm 2 summarizes the steps of the proposed algorithm. The inputs to it are, apart from Σ and $\mathcal{V}$, a positive integer n which will be used during the second step of the repairing process and a preference mechanism prf for ranking the possible repairs. In the simplest scenario, the preference mechanism implements a fixed function for ranking the different repairs. However, it can also employ supervised learning techniques in order to progressively learn the user preferences by looking at his prior decisions.

Since a mapping $\mathcal{M}$ is safe w.r.t. $\mathcal{V}$ if the instance $I_{\mathsf{S}}(\Sigma)$ is safe according to Theorem 4.7, Algorithm 2 rewrites the tgds in $\mathcal{M}$, such that the derived visible chase instances are safe. The rewriting takes place in two steps. The first step rewrites Σ into a *partially-safe* set of s-t dependencies Σ_1, while the second step rewrites the output of the first one into a new set of s-t dependencies Σ_2, such that $I_{\mathsf{S}}(\Sigma_2)$ is safe. As we will explain later on, partial-safety ensures that the intermediate instance I_1 produced by $\mathsf{visChase}_{\mathsf{S}}(\Sigma_1)$ at line 2 of Algorithm 1 is safe, but it does not provide strong privacy guarantees. The benefit of this two-step approach is that it allows repairing one or a small set of dependencies at a time.

5.1 Computing partially-safe mappings

Since the problem of safety is reduced to the problem of checking for a homomorphism from $I_{\mathsf{S}}(\Sigma)$ into $I_{\mathsf{S}}(\mathcal{V})$, a first test towards checking for such a homomorphism is to look if the mappings in Σ would lead to such a homomorphism or not. For instance, by looking at μ_s in Example 1.1 it is easy to see that it leaks sensitive information, since it involves a join between students and oncology departments, which does not occur in $I_{\mathsf{S}}(\mathcal{V})$.

Definition 5.1. A mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ is *partially-safe w.r.t.* $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ *on all instances of* S, if there exists a homomorphism from $\mathsf{chase}(\Sigma^{-1}, \mathsf{Crt}_{\mathsf{T}}) \setminus \mathsf{Crt}_{\mathsf{T}}$ into $I_{\mathsf{S}}(\mathcal{V})$.

From Algorithm 1, it follows that Σ is partially-safe iff the intermediate instance I_1 computed by $\mathsf{visChase}_{\mathsf{S}}(\Sigma)$ is safe.

Proposition 5.2. *A mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ *is* partially-safe w.r.t. $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ on all instances of S, *if for each* $\mu \in \Sigma$, *there exists a homomorphism from* $\mathsf{body}(\mu)$ *into* $I_{\mathsf{S}}(\mathcal{V})$ *mapping each* $x \in \mathsf{exported}(\mu)$ *into the critical constant* $*$.

Note that according to Proposition 5.2, in our running example Σ_{st} would be partially-safe, if $\mu_s \notin \Sigma_{st}$, then since there exist homomorphisms from the bodies of μ_s and μ_c into $I_{\mathsf{S}}(\mathcal{V})$, mapping their exported variables into $*$. It is also easy to show the following

Remark 1. *A mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ *is* safe w.r.t. $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ on all instances of S, *only if it is partially-safe w.r.t.* $\mathcal{M}_{\mathsf{V}}$ *on all instances of* S. □

Proposition 5.2 presents a quite convenient, yet somewhat expected, finding: in order to obtain a partially-safe mapping, it suffices to repair each s-t dependency *independently of the others*. Furthermore, the repair of each $\mu \in \Sigma$ involves breaking joins and hiding exported variables, such that the repaired dependency μ_r satisfies the criterion in Proposition 5.2.

We make use of the result of Proposition 5.2 in Algorithm 3. Algorithm 3 obtains, for each $\mu \in \Sigma$, a set of rewritings $\mathcal{R}_\mu$, out of which we will choose the best rewriting according to prf. The set $\mathcal{R}_\mu$ consists of *all* rewritings that differ from μ w.r.t. the variable repetitions in the bodies of the rules and the exported variables. Below, we present the steps of Algorithm 3.

For each s-t tgd μ and for each atom $B \in \mathsf{body}(\mu)$, Algorithm 3 constructs a fresh atom C and adds C to a set $\mathcal{C}$. The set of atoms $\mathcal{C}$ provides us with the means to identify all repairs of μ that involve breaking joins and hiding exported variables. In particular, each homomorphism ξ from $\mathcal{C}$ into $I_\mathsf{S}(\mathcal{V})$ corresponds to one repair of μ. In lines 12–25, Algorithm 3 modifies each atom $B \in \mathsf{body}(\mu)$ by taking into account prior body atom modifications. The prior modifications are accumulated in the relation ρ and the mapping ψ. The relation ρ keeps for each variable x from $\mathsf{body}(\mu)$, the fresh variables that were used to replace x during prior steps of the repairing process, while ψ is a substitution from the partially repaired body into $I_\mathsf{S}(\mathcal{V})$. In particular, at the end of the i-th iteration of the loop in line 12, ψ holds the substitution from the first repaired i atoms from $\mathsf{body}(\mu)$ into $I_\mathsf{S}(\mathcal{V})$. We adopt this approach instead of replacing variable x in position p always by a fresh variable, in order to minimize the number of the joins we break.

Below, we describe how Algorithm 3 modifies each body atom of μ, w.r.t. a homomorphism ξ, lines 9–27. Let $C = \nu(B)$ be the fresh body atom that was constructed out of B in line 5. For each atom $B \in \mathsf{body}(\mu)$ and for each $p \in \mathsf{pos}(B)$, if the variable y in position p of C is not mapped to the critical constant $*$ via ξ and $B|_p$ is an exported variable, this means that *the variable sitting in position p of B should not be exported* (see first condition in line 16). Similarly, if the variable sitting in position p of B is mapped to a different constant than the one that y maps via ξ, then this means that *the variable sitting in position p of B introduces an unsafe join* (see second condition in line 16). In the presence of these violations, we must replace variable x in position p of B, either by a variable that was used in a prior step of the repairing process, line 17–18), or by a fresh variable, lines 19–23. Otherwise, if there is no violation so far, then we add the mapping $\{x \mapsto \xi(y)\}$ to ψ, if it is not already there, lines 24–25. Finally, the algorithm chooses the best repair according to the preference function, lines 28–31.

Proposition 5.3. *For any $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, any $\mathcal{M}_\mathsf{V} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ and any preference function* prf*, Algorithm* frepair *returns a mapping $\mathcal{M}' = (\mathsf{S}, \mathsf{T}, \Sigma')$ that is partially-safe w.r.t. $\mathcal{M}_\mathsf{V}$ on all instances of* S.

Proof. (Sketch) From Proposition 5.2, a mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ is partially-safe w.r.t. $\mathcal{M}_\mathsf{V} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ on all instances of S, if for each $\mu \in \Sigma$, there exists a homomorphism from $\mathsf{body}(\mu)$ into $I_\mathsf{S}(\mathcal{V})$ mapping each $x \in \mathsf{exported}(\mu)$ into the critical constant $*$. Since for each $\mu \in \Sigma$ frepair computes a set of repaired tgds $\mathcal{R}_\mu$, it follows that Proposition 5.3 holds, if such a homomorphism exists, for each repaired tgd in $\mathcal{R}_\mu$. The proof proceeds as follows. Let μ_r^i and ψ^i denote the repaired s-t tgd and the homomorphism ψ computed at the end of each iteration i of the steps in lines 12–25 of Algorithm 3. Let also B^i denote the i-th atom in $\mathsf{body}(\mu_r)$. Since each $C \in \mathcal{C}$ is an atom of distinct fresh variables, since ξ is a homomorphism from $\mathcal{C}$ to $I_\mathsf{S}(\mathcal{V})$ and since $\psi(B^i) = \mu_r|_i$, it follows that in order to prove Proposition 5.2, we have to show that the following claim holds, for each $i \geq 0$:

Algorithm 3 frepair($\Sigma, \mathcal{V}$, prf)

```
1: for each $\mu \in \Sigma$ do
2:     $\nu := \emptyset, \mathcal{C} := \emptyset$
3:     for each $B \in \mathsf{body}(\mu)$, where $B = R(\vec{x})$ do
4:         create a vector of fresh variables $\vec{y}$
5:         create the atom $C = R(\vec{y})$
6:         add $(B, C)$ to $\nu$
7:         add $C$ to $\mathcal{C}$
8:     $\mathcal{R}_\mu := \emptyset$
9:     for each homomorphism $\xi : \mathcal{C} \rightarrow I_\mathsf{S}(\mathcal{V})$ do
10:        $\rho := \emptyset, \psi := \emptyset$
11:        $\mu_r := \mu$
12:        for each $B \in \mathsf{body}(\mu_r)$ do
13:            $C = \nu(B)$
14:            for each $p \in \mathsf{pos}(B)$ do
15:                $x = B|_p, y = C|_p$
16:                if $x \in \mathsf{exported}(\mu)$ and $* \neq \xi(y)$ or $x \in \mathsf{dom}(\psi)$ and $\psi(x) \neq \xi(y)$ then
17:                    if $\exists x'$ s.t. $(x, x') \in \rho$ and $\psi(x') = \xi(y)$ then
18:                        $B|_p = x'$
19:                    else
20:                        create a fresh variable $x'$
21:                        add $(x, x')$ to $\rho$
22:                        add $\{x' \mapsto \xi(y)\}$ to $\psi$
23:                        $B|_p = x'$
24:                else if $x \notin \mathsf{dom}(\psi)$ then
25:                    add $\{x \mapsto \xi(y)\}$ to $\psi$
26:        if $\mu_r \neq \mu$ then
27:            add $\mu_r$ to $\mathcal{R}_\mu$
28:    if $\mathcal{R}_\mu \neq \emptyset$ then f
29:        choose the best repair $\mu_r$ of $\mu$ from $\mathcal{R}_\mu$ based on prf
30:        remove $\mu$ from $\Sigma$
31:        add $\mu_r$ to $\Sigma$
32: return $\Sigma$
```

- ϕ. ψ^i is a homomorphism from the first i atoms in the body of μ_r into $I_\mathsf{S}(\mathcal{V})$ mapping each exported variable occurring in $B^0, \ldots, B^i$ into the critical constant $*$.

For $i = 0$, ϕ trivially holds. For $i + 1$ and assuming that ϕ holds for i let $C^{i+1} = \nu(B^{i+1})$, line 13. The proof of claim ϕ depends upon the proof of the following claim, for each iteration $p \geq 0$ of the steps in lines 14–25:

- θ. $\psi^{i+1}(B^{i+1}|_p) = \xi(y)$, where $y = C^{i+1}|_p$.

The claim θ trivially holds for $p = 0$, while for $p > 0$, it directly follows from the steps in lines 16–25. Since ϕ holds for i, since the steps in lines 16–25 do not modify the variable mappings in ψ^i and due to θ, it follows that ϕ holds for $i + 1$, concluding the proof of Proposition 5.3. □

Example 5.4. We demonstrate an example of Algorithm 3.

Since Algorithm 3 focuses on $I_\mathsf{S}(\mathcal{V})$ overlooking the actual views in $\mathcal{V}$, we will not explicitly define $\mathcal{V}$. Instead, we will only assume that the visible chase computes the instance

$$I_\mathsf{S}(\mathcal{V}) = \{\mathsf{R}_1(*, \mathsf{n}_1, \mathsf{n}_2), \mathsf{S}_1(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_2), \mathsf{S}_1(\mathsf{n}_1, \mathsf{n}_3, *), \mathsf{S}_1(\mathsf{n}_1, *, *)\}$$

where n_1–n_3 are labeled nulls. Consider also the mapping $\mathcal{M}$ consisting of the following s-t dependency

$$\mathsf{R}_1(x, y, z) \wedge \mathsf{S}_1(y, z, z) \rightarrow \mathsf{T}_1(x, z) \qquad (\mu_1)$$

Note that $\mathcal{M}$ is not partially-safe. Algorithm 3 computes two repairs for μ_1 by applying the steps described below. First, it computes the atoms $\mathsf{R}_1(x_1, x_2, x_3)$ $\mathsf{S}_1(x_4, x_5, x_6)$ and adds them to C, lines 3–7. Then, it identifies the following three homomorphisms from C into $I_\mathsf{S}(\mathcal{V})$:

$$\xi_1 = \{x_1 \mapsto *, x_2 \mapsto \mathsf{n}_1, x_3 \mapsto \mathsf{n}_2, x_4 \mapsto \mathsf{n}_1, x_5 \mapsto \mathsf{n}_2, x_6 \mapsto \mathsf{n}_2\}$$

$$\xi_2 = \{x_1 \mapsto *, x_2 \mapsto \mathsf{n}_1, x_3 \mapsto \mathsf{n}_2, x_4 \mapsto \mathsf{n}_1, x_5 \mapsto \mathsf{n}_3, x_6 \mapsto *\}$$

$$\xi_3 = \{x_1 \mapsto *, x_2 \mapsto \mathsf{n}_1, x_3 \mapsto \mathsf{n}_2, x_4 \mapsto \mathsf{n}_1, x_5 \mapsto *, x_6 \mapsto *\}$$

From ξ_1, we can see that the joins in the body of μ_1 are safe; however, it is unsafe to export z. From ξ_2, we can see that is safe to reveal the third position of S_1; however, it is unsafe to join the second and the third position of S_1. Algorithm 3 then iterates over ξ_1 and ξ_2, line 9. When $B = \mathsf{R}_1(x, y, z)$ and $p < 3$, Algorithm 3 computes ψ to $\{x \mapsto *, y \mapsto \mathsf{n}_1\}$, since there is no violation according to line 16. When $B = \mathsf{R}_1(x, y, z)$ and $p = 3$, however, a violation is detected. This is due to the fact that z is an exported variable and $\xi(x_3) = \mathsf{n}_2$. Algorithm 3 tackles this violation by creating a fresh variable z_1. Then, it adds the relation (z, z_1) to ρ, replaces z in $B|_3$ by z_1 and adds the mapping $\{z_1 \mapsto \mathsf{n}_2\}$ to ψ, lines 19–23. Algorithm 3 then considers $\mathsf{S}_1(y, z, z)$. When $p = 1$, no violation is encountered, since $\psi(y) = \xi_1(x_4)$. However, when $p = 2$, a homomorphism violation is encountered, since z is an exported variable and since $\xi(x_3) = \mathsf{n}_2$. Since $(z, z_1) \in \rho$ and $\psi(z_1) = \xi_1(x_5)$, Algorithm 3 replaces z in the second position of $\mathsf{S}_1(y, z, z)$ by z_1, line 19. By applying a similar reasoning, we can see that the variable z siting in $\mathsf{S}_1(y, z, z)|_3$ is also replaced by z_1. Hence, the first repair of μ is

$$\mathsf{R}_1(x, y, z_1) \land \mathsf{S}_1(y, z_1, z_1) \rightarrow \mathsf{T}_1(x) \qquad (r_1)$$

Algorithm 3, then proceeds by repairing μ_1 based on ξ_2. When $B = \mathsf{R}_1(x, y, z)$, Algorithm 3 proceeds as described above and computes ψ to $\{x \mapsto *, y \mapsto \mathsf{n}_1, z_1 \mapsto \mathsf{n}_2\}$. When $B = \mathsf{S}_1(y, z, z)$ and $p = 1$, then no violation is encountered since $\psi(y) = \xi_1(x_4)$, while when $B = \mathsf{S}_1(y, z, z)$ and $p = 2$, there is a violation. Since the condition in line 18 is not met, Algorithm 3 creates a fresh variable z_2 and adds the mapping $\{z_2 \mapsto \mathsf{n}_3\}$ to ψ. When $B = \mathsf{S}_1(y, z, z)$ and $p = 3$, then no violation is met, since $z \in \mathsf{exported}(\mu)$ and $\xi_2(x_6) = *$. Hence, the second repair of μ_1 is

$$\mathsf{R}_1(x, y, z_1) \land \mathsf{S}_1(y, z_2, z) \rightarrow \mathsf{T}_1(x, z) \qquad (r_2)$$

Finally, we can see that the repair for μ_1 w.r.t. ξ_3 is

$$\mathsf{R}_1(x, y, z_1) \land \mathsf{S}_1(y, z, z) \rightarrow \mathsf{T}_1(x, z) \qquad (r_3)$$

5.2 Computing safe mappings

Unifications of one or more labeled nulls occurring in I_1 with the critical constant $*$, might lead to unsafe instances. Consider, for instance, a simplified variant of Example 1.1, where Σ_{st} comprises only μ_e and μ_c. Both μ_e and μ_c are partially-safe, as we have explained above. However, the unification of the labeled nulls n_n and n_c produces an unsafe instance. Algorithm 4 aims at repairing the output of the previous step, such that no unsafe unification of a labeled null with $*$ takes place.

Consider again the simplified variant of Σ_{st} from above. Since Σ_{st} is partially-safe, it suffices to look for homomorphism violations in I_i, for $i \geq 1$. A first observation is that the homomorphism violations are "sitting" within the bags. This is due to the fact that each bag stores *all* the facts associated with the bodies of one or more s-t tgds from Σ_{st}. A second observation is that one way for preventing unsafe unifications is to hide exported variables. For example, let us focus on the unsafe unification of n_e with $*$. This unification takes place due to ϵ_1, which in turn has been created due to the fact that e is an exported variable in μ_e. By hiding the exported variable e from μ_e, we actually prevent the creation of ϵ_1 and hence, we block the unsafe unification of e with $*$. Hiding exported variables is one way for preventing unsafe unifications with the critical constant. Another way for preventing unsafe unifications is to break joins in the bodies of the rules.

Example 5.5. This example demonstrates a second approach for preventing unsafe labeled null unifications.

Consider a set of policy views $\mathcal{V}$ leading to the following instance $I_\mathsf{S}(\mathcal{V}) = \{\mathsf{R}_1(\mathsf{n}_1, \mathsf{n}_1, *), \mathsf{R}_1(*, *, \mathsf{n}_2), \mathsf{S}_1(*)\}$, where n_1 and n_2 are labelled nulls. Consider also the mapping $\mathcal{M}$ consisting of the following s-t dependencies:

$$\mathsf{R}_1(x, x, y) \land \mathsf{S}_1(y) \rightarrow \mathsf{T}_1(y) \qquad (\mu_2)$$

$$\mathsf{R}_1(x, x, y) \rightarrow \mathsf{T}_2(x) \qquad (\mu_3)$$

It is easy to see that $\mathcal{M}$ is partially-safe, but unsafe in overall. Indeed, $I_\mathsf{S}(\Sigma)$ will consist of the following bags (for presentation purposes, we adopt the notation from Example 4.5):

$$\mathsf{T}_1(*) \xrightarrow{\langle \mu_2^{-1}, \theta_1 \rangle} \mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_3, *), \mathsf{S}_1(*)$$

$$\mathsf{T}_2(*) \xrightarrow{\langle \mu_3^{-1}, \theta_2 \rangle} \mathsf{R}_1(*, *, \mathsf{n}_4)$$

$$\mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_3, *), \mathsf{S}_1(*) \xrightarrow{\langle \epsilon_3, \theta_3 \rangle} \mathsf{R}_1(*, *, *), \mathsf{S}_1(*)$$

where $\epsilon_3 := \mathsf{R}_1(x, x, y) \rightarrow x = *$, $\theta_1 = \{y \mapsto *\}$, $\theta_2 = \{x \mapsto *\}$ and $\theta_3 = \{x \mapsto \mathsf{n}_3, y \mapsto *\}$. Note that ϵ_3 has been created out of μ_3, since there exists a homomorphism from $\mathsf{body}(\mu_3)$ into $\mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_3, *)$ mapping the exported variable x into n_3.

One approach for preventing the unsafe unification of n_3 with $*$ is to hide the exported variable x from μ_3. By doing this, we block the creation of ε, and hence the unsafe unification.

A second approach is to keep x as an exported variable in μ_3, but modify the body of μ_2 by breaking the join between the first and the second position of R_1

$$\mathsf{R}_1(x, z, y) \land \mathsf{S}_1(y) \rightarrow \mathsf{T}_1(y) \qquad (\mu_2')$$

By doing this, we prevent the creation of ε, since the instance computed at line 2 of Algorithm 1 would consist of the facts $\mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_5, *)$, $\mathsf{R}_1(*, *, \mathsf{n}_4)$, $\mathsf{S}_1(*)$ and, hence, there would be no homomorphism from $\mathsf{body}(\mu_3)$ into it. Note that the modification of μ_2 to μ_2' is safe. Intuitively, this holds, since we break joins, and thus, we export less information.

Before presenting Algorithm 4, we will introduce some new notation. The *depth* of each bag β, denoted as $\mathsf{depth}(\beta)$, coincides with the highest derivation depth of the facts in β. The *support* of a bag β, denoted as $\beta^{\prec}$, is inductively defined as follows: if $\mathsf{depth}(\beta) = 1$, then $\beta^{\prec} = \beta$; otherwise, if $\mathsf{depth}(\beta) > 1$, then $\cup_{\beta' \prec \beta} \beta'^{\prec}$. Consider an active trigger h for δ in I leading to the creation of a bag β. We use the following notation: $\mathsf{dependency}(\beta) = \delta$, $\mathsf{trigger}(\beta) = h$ and $\mathsf{premise}(\beta) = h(\mathsf{body}(\delta))$. Two bags β_1 and β_2 are candidates for modifyBody if $\beta_1 \prec \beta_2$, $\mathsf{depth}(\beta_1) = 1$, $\mathsf{depth}(\beta_2) = 2$ and there exists at least one repeated variable in the body of $\mathsf{tgd}(\beta_1)$.

Algorithm 4 presents an iterative process for repairing a partially-safe Σ, by employing the three ideas we described above: checking for homomorphism violations within each bag and preventing unsafe unifications either by hiding exported variable, or by modifying the bodies of the s-t tgds. In brief, at each iteration $i \geq 0$, the algorithm repairs one or more dependencies from Σ_i, where $\Sigma_0 = \Sigma$, and incrementally computes the visible chase of the new

Algorithm 4 srepair$(\Sigma, \mathcal{V}, \mathsf{prf}, n)$

```
1:  $\Sigma_0 := \Sigma$
2:  $B_0 :=$ visChases$_{\mathsf{S}}(\Sigma)$
3:  $i := 0$
4:  do
5:      $\Sigma_{i+1} := \Sigma_i$
6:      $cont :=$ false
7:      if $\exists$ unsafe $\beta \in B_i$, s.t. depth$(\beta) \leq$ depth$(\beta')$, $\forall$ unsafe bag $\beta' \in B_i$ then
8:          $cont :=$ true
9:          if $i < n$ then
10:             $r_1 := \emptyset$; $r_2 :=$ hideExported$(\beta, \mathcal{V}, \mathsf{prf})$
11:             if $\exists \beta_1, \beta_2 \in \beta^{\prec}$, s.t. $\beta_1, \beta_2$ are candidates for modifyBody then
12:                 $r_1 :=$ modifyBody(tgd$(\beta_1)$, tgd$(\beta_2)$, prf)
13:             if $r_1 \neq \emptyset$ and it is preferred over $r_2$ w.r.t. prf then
14:                 remove tgd$(\beta_1)$ from $\Sigma_{i+1}$
15:                 add $r_1$ to $\Sigma_{i+1}$
16:             else
17:                 remove tgd$(\beta)$ from $\Sigma_{i+1}$
18:                 add $r_2$ to $\Sigma_{i+1}$
19:         else
20:             if $\nexists \beta'$, s.t., $\beta \prec \beta' \in B_i$ then
21:                 add hideExported$(\beta, \mathcal{V}, \mathsf{prf})$ to $\Sigma_{i+1}$
22:             else remove tgd$(\beta)$ from $\Sigma_{i+1}$
23:     compute $J_{i+1}$ from $\Sigma_i$, $\Sigma_{i+1}$ and $B_i$
24:     $i = i + 1$
25: while $cont$ and $i \leq n$
26: return $\Sigma_n$
```

set of dependencies, lines 4–25. Algorithm 4 terminates either when the dependencies are safe, or when the maximum number of iterations n is reached, line 25, in which case it repairs all unsafe dependencies by hiding their exported variables. The algorithm starts by initializing Σ_0 to Σ, lines 1. Then, at each iteration i, it first identifies the lowest depth unsafe bag, line 7, and attempts to repair the dependencies from Σ_i that lead to its creation, lines 7–22. If $i < n$, it proposes two different repairs for Σ_i, one based on hiding exported variables through hideExported (Algorithm 5), and the second based on eliminating joins through modifyBody (Algorithm 6), lines 10–19. Algorithm 4 applies the modifyBody if there exist two bags in the support of β that are candidates for modifyBody. Informally, Algorithm 4 tries to apply modifyBody as early as possible (condition depth$(\beta_1) = 1$, depth$(\beta_2) = 2$) and when there are one or more repeated variables in the body of tgd(β_1) (recall Example 5.5). Otherwise, if $i = n$, it either applies the function hideExported, or it eliminates the s-t tgds that are responsible for unsafe unifications.

Note that when we reach the maximum number of iterations we do not apply modifyBody. This is due to the fact that modifyBody might lead to unsafe unification of labeled nulls to $*$ that were not taking place before the modifying the s-t tgd through modifyBody. In contrast, hideExported is a safe modification, since it does not lead to new unsafe unifications.

Theorem 5.6. *For any partially-safe* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, *any* $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$*, any preference function* prf *and* $n \geq 0$*, Algorithm* srepair *returns a mapping* $\mathcal{M}' = (\mathsf{S}, \mathsf{T}, \Sigma')$ *that* preserves the privacy of $\mathcal{M}_{\mathsf{V}}$ on all instances of S.

Proof. (Sketch) Since srepair takes as input a partially-safe mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, it follows from Definition 5.1 that there exists a homomorphism from chase$(\Sigma^{-1}, \mathsf{Crt}_{\mathsf{T}}) \setminus \mathsf{Crt}_{\mathsf{T}}$ into $I_{\mathsf{S}}(\mathcal{V})$. Furthermore, from Proposition 5.2, we know that for each $\mu \in \Sigma$, there exists a homomorphism from body(μ) into $I_{\mathsf{S}}(\mathcal{V})$ mapping each $x \in$ exported(μ) into the critical constant $*$. Due to the above, since the steps in lines 16–20 of Algorithm 1 do not introduce new labeled nulls and since srepair applies the procedure hideExported to each unsafe bag β in B_n, if there does not exist a bag $\beta' \in B_n$, such that $\beta \prec \beta'$, it follows that $\mathcal{M}'$ preserves the privacy of $\mathcal{M}_{\mathsf{V}}$ on all instances of S, if hideExported prevents dangerous unifications of labeled nulls with the critical constant in line 4 of Algorithm 1. In particular, assume that we are in the n-th iteration of the steps in lines 4–25 of Algorithm 4. Let $\beta_n^0, \ldots, \beta_n^M$ be the unsafe bags in B_n. Assume also that for each $1 \leq l \leq M$, β_n^l, was derived due to some active trigger h^l, for some derived egd $\varepsilon^l \in \Sigma_{\approx}$ in I_j, where $j \geq 0$, line 17 of Algorithm 1. Let $\mu^l = $ tgd(ε^l), for each $0 \leq l \leq M$ and let μ_r^l be the repaired s-t tgd. Finally, let $\beta_{n+1}^0, \ldots, \beta_{n+1}^N$ be the bags in B_{n+1}, line 23 of Algorithm 4. Based on the above, in order to show that Theorem 5.6 holds, we need to show that (i) the number of bags in B_{n+1} is $\leq$ the number of bags in B_n and that (ii) the s-t tgds in $\left(\Sigma \setminus \bigcup_{l=0}^{M} \mu^l\right) \cup \bigcup_{l=0}^{M} \mu_r^l$ are safe. In order to show (i) and (ii), we consider the steps in Algorithm 5: for each $1 \leq l \leq M$, each exported variable y occurring in μ^l, which leads to an unsafe

Algorithm 5 hideExported$(\beta, \mathcal{V}, \mathsf{prf})$

```
1:  $J :=$ premise$(\beta)$
2:  $\nu := \emptyset$
3:  for each $n \in$ Nulls occurring into $J$ do
4:      add $\{n \mapsto x\}$ to $\nu$, where $x$ is a fresh variable
5:  $\mathcal{R} := \emptyset$
6:  for each $\xi : \nu(J) \rightarrow I_{\mathsf{S}}(\mathcal{V})$ do
7:      $\mu :=$ tgd$(\beta)$
8:      for each $x \in$ dom$(\xi)$ do
9:          if $\xi(x) \neq *$ then
10:             for each $y \in$ exported$(\mu)$ do
11:                 if $\tau(y) = \nu^{-1}(x)$, where $\tau =$ trigger$(\beta)$ then
12:                     remove $y$ from exported$(\mu)$
13:     if $\mu \neq$ tgd$(\beta)$ then
14:         add $\mu$ to $\mathcal{R}$
15: choose the best repair $\mu_r$ of $\mu$ from $\mathcal{R}$ based on prf
16: return $\mu_r$
```

Algorithm 6 modifyBody$(\mu_1, \mu_2, \mathsf{prf})$

```
1:  $\mathcal{R} := \emptyset$
2:  if $\exists$ one or more repeated variables in body$(\mu_1)$ then
3:      for each $\xi$ : body$(\mu_2) \rightarrow$ body$(\mu_1)$ mapping some $x_1 \in$ exported$(\mu_1)$
            into some $x_2 \notin$ exported$(\mu_2)$ do
4:          Let $B \subseteq$ body$(\mu_1)$, s.t. $\xi($body$(\mu_2)) = B$
5:          Let $V$ be the set of repeated variables from $B$
6:          Let $P$ be the set of positions from $B$, where all variables from $V$ occur
7:          for each non-empty $S \subset P$ do
8:              $\mu := \mu_1$
9:              replace the variables in positions $S$ of $\mu$ by fresh variables
10:             add $\mu$ to $\mathcal{R}$
11: choose the best repair $\mu_r$ of $\mu$ from $\mathcal{R}$ based on prf
12: return $\mu_r$
```

	min	max	step
# s-t tgds per scenario (n_{dep})	100	300	50
# body atom per s-t tgds (n_{atoms})	1	3 (5)	–
# exported variables per s-t tgds (n_{vars})	5	8	–

Table 1: Properties of the generated iBench scenarios.

unification, line 11 of Algorithm 5, is turned into a non-exported variable. □

By combining Proposition 5.3 and Theorem 5.6 we can prove the correctness of Algorithm 2. Furthermore, if the preference function always prefers the repairs computed by hideExported from the repairs computed by modifyBody, we can show the following:

Proposition 5.7. *For each mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, *each* $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ *and each preference function* prf *that always prefers the repairs computed by* hideExported *from the repairs computed by* modifyBody, *Algorithm 2 returns a non-empty mapping that is safe w.r.t.* $\mathcal{M}_{\mathsf{V}}$, *if such a mapping exists.*

Proof. (Sketch) From Algorithm 3, we can see that frepair always computes a non-empty partially-safe mapping, if such a mapping exists. Note that a mapping, where no variable is exported and no repeated variables occur in the body of the s-t tgds is always partially-safe as long as, the predicates in the bodies of the s-t tgds are the same with the ones occurring in the policy views. Please also note that such a mapping is always considered by frepair. The above argument, along with the fact that a partially-safe mapping can be transformed into a safe one by turning exported variables into non-exported ones by means of the function hideExported, shows that Proposition 5.7 holds. □

6 EXPERIMENTS

We gauge the efficiency of our repairing algorithm on two types of preference function: a hardcoded one and a learning-based preference function.

We evaluated our algorithm using a set of 3.6K diverse mapping scenarios each of which consisting of a set of policy views and a set of s-t tgds. The characteristics of the scenarios are summarized in Table 1. In each scenario, we used a different number of s-t tgds n_{dep}, a different number of body atoms n_{atoms} and a different number of exported variables n_{vars}. The source schemas and the policy views have been synthetically generated using iBench, the state-of-the-art data integration benchmark [2]. We considered relations of up to five attributes and we created mappings using the iBench configuration recommended by the authors of [2]. We generated a set of varied policy views by applying the iBench operators copy, merge, deletion of attributes and self-join, each of which has been applied 10 times.

We implemented our algorithm in Java and we used the Weka library [16] that provides an off-the-shelf implementation of the k-NN algorithm for the learning-based preference function. We ran our experiments on a laptop with one 2.6GHz 2-core processor, 16Gb of RAM, running Debian 9.

In the remainder, all data points have been computed as an average on a total of 5 runs preceded by one discarded cold run.

	golden standard	
prediction	μ_1	μ_2
μ_1	230	0
μ_2	0	395680

(a) P_{max} **confusion matrix.**

	golden standard	
prediction	μ_1	μ_2
μ_1	290	1
μ_2	42	395577

(b) P_{avg} **confusion matrix.**

Table 2: Confusion matrix for the golden standards.

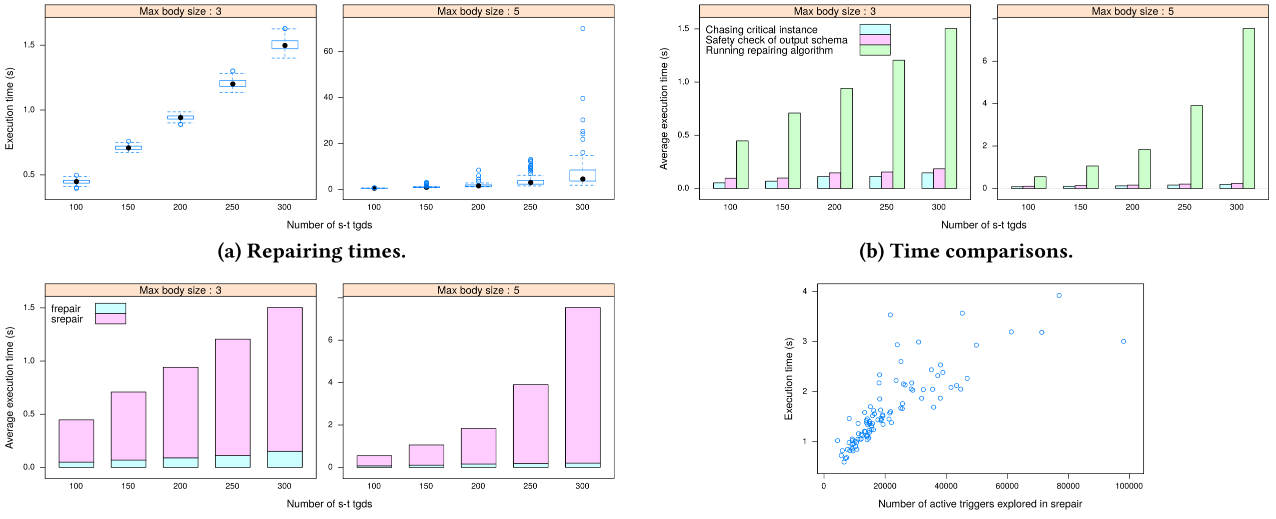
Running time of repair**.** First, we study the impact of the number of s-t tgds and body atoms on the running time of repair. We adopt a fixed preference function that chooses the repair with the maximum number of exported variables. In case of ties, the repair with the maximum number of joins is preferred. We vary the number of s-t tgds from 100 to 300 by steps of 50 and the number of body atoms from 3 to 5, respectively. The obtained results are shown in Figure (3a), illustrating the fact that the median repairing time is less than 1.5s in most cases. For the most complex scenario containing up to five body atoms per s-t tgd, the median running time is less than 8s with 71s being the maximum. These results clearly show the high performance of our repair method along with its scalability. The reader should keep in mind that the repairing process is triggered prior to exchanging the data between source and target and might be rerun each time a mapping (set of s-t tgds) is modified or each time a policy view is modified, thus bringing the overhead to be quite reasonable in both cases.

Figure (3b) shows the time breakdown for repair. The first bar shows the average running time to run the visible chase over the input s-t mappings, the second one shows the average running time for checking the safety of the computed bags and the third one shows the average running time for repairing the s-t tgds. The results show that the repairing time is $32x$ greater than the time to compute the visible chase and $40x$ greater than the time to check the safety of the chase bags for scenarios with 300 s-t tgds. In the simplest scenarios, these numbers are much lower (reduced to $5x$ and $9x$, respectively). Overall, the absolute values of the rewriting times are kept low (of the order of few seconds) for all these scenarios and gracefully scale while increasing the number of s-t tgds and the number of atoms in their bodies.

Time breakdown between frepair **and** srepair**.** Figure (3c) shows the average running time for frepair and srepair for the considered scenarios. We can see that srepair is the most time-consuming step of our algorithm. We can also see that the running time of srepair increases more in comparison to the running time of frepair when increasing the number of the s-t tgds and the number of atoms in their bodies. This is due to the incurred overhead during the incremental computation of the visible chase after repairing a s-t tgd (line 23 of Algorithm 4). Figure (3d) shows the correlation between the number of active triggers detected while incrementally computing the visible chase and the running time of srepair for scenarios with 100 s-t tgds using the ANOVA method (*p-value* $< 2.2e^{-16}$). Figure (3d) shows that the most complex scenarios lead to the detection of more than $45,000$ active triggers. Despite the high number of the detected active triggers, the running time of srepair is kept low thus confirming its efficiency.

Leveraging learning-based preferences. We adopted the following steps in order to evaluate the performance of our learning approach. First, we defined the following two golden standard preference functions that we will try to learn:

- P_{max}, which chooses the repair with the maximum number of exported variables and in case of ties, it chooses the repair with the maximum number of joins.
- P_{avg}, which computes the average number of exported variables and joins for each repair, and choose the one with the maximum average value.



(a) Repairing times.

(b) Time comparisons.

(c) Time breakdown between frepair **and** srepair**.**

(d) Running time of srepair **over 100 s-t tgds.**

Figure 3: Summary of the performance-related experimental results.

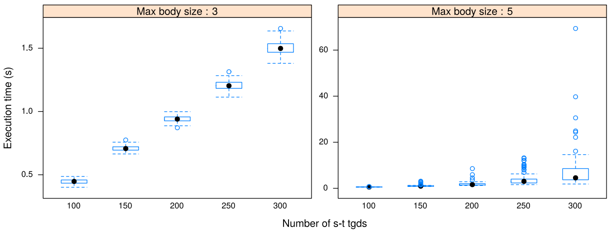


Figure 4: Repairing times with ML classifiers.

For both preference functions, we created a training set of 10, 000 measurements for the k-NN classifier by running the repairing algorithm on fresh scenarios of 50 s-t tgds and five body atoms per s-t tgd. For each input vector $\langle \delta_{FV}, \delta_J \rangle$ whose repair we wanted to predict, we computed the Euclidean distance between $\langle \delta_{FV}, \delta_J \rangle$ and the vectors of the training set. We also set the value of parameter k to 1. This parameter controls the number of neighbors used to predict the output. Higher values of this parameter led to comparable predictions and are omitted for space reasons. Finally, we used the trained k-NN classifier as a preference function in srepair, rerun the above scenarios and compared the returned repairs with the ones returned when applying the golden standards P_{max} and P_{avg} as preference functions.

Learning P_{max}. Table (2a) (left) reports the confusion matrix associated to learning P_{max}, including the choices made by the k-NN classifier during its iterations.

Let us call μ_1 and μ_2 two possible repairs of an s-t tgd as evaluated by the k-NN classifier. We can observe that the prediction of μ_1 was correct (and equal to the golden standard in the training set) in 230 cases, while the prediction of μ_2 was correct in 395,680 cases.

This confirms the fact that μ_2 is the best repair across the iterations of the k-NN algorithm and is also chosen in case μ_1 and μ_2 are equally weighed by the preference function.

Furthermore, we also report the accuracy of learning the preference function, obtained by measuring the closeness of the learned mapping to the golden standard mapping.

We used the Matthews Correlation Coefficient metric (MCC) [3] to compare the repairs returned by the trained k-NN classifier and the ones returned when applied P_{max}. This is a classical measure that allows to evaluate the quality of ML classifiers when ranking is computed between two possible values (in our case, the choice between μ_1 and μ_2). This measure has been computed using the following formula:

$$MCC = \frac{N_{1,1} \times N_{2,2} - N_{1,2} \times N_{2,1}}{\sqrt{(N_{1,1} + N_{1,2})(N_{1,1} + N_{2,1})(N_{2,2} + N_{1,2})(N_{2,2} + N_{2,1})}}$$

where $N_{i,j}$ is the number of predictions of μ_i when μ_j is expected. The results of MCC range from -1 for the cases where the model perfectly predicts the inverse of the expected values, to 1 for the cases where the model predicts the expected values. The value $MCC = 0$ means that there is no correlation between the predicted value and the expected one. By applying MCC to the learning of P_{max}, we observed that the data are clearly discriminated, thus leading to high-quality of our prediction in this case ($MCC = 1$).

Learning P_{avg}. Table (2b) (right) shows the confusion matrix associated to learning P_{avg}. We can see that the predictions are less accurate in this case. The data is not as clearly discriminated as before, leading to a fairly negligible error rate ($< 0.02\%$). However, the latter is still acceptable for learning, since only $< 0.02\%$ of the predictions are erroneous. This is corroborated by an MCC value equal to 0.93, thus leading to a fairly acceptable quality of the prediction in this case too.

Running time of repair **with ML classifiers.** In the last experiment, we want to measure the impact of learning on the performance of our algorithm. To this end, we compare the running time of repair when adopting a hard-coded preference function (as in the results reported in Figure 3) and when adopting a learned preference function. Figure 4 shows the running times for the same scenarios used in Figure 3. We can easily observe that the runtimes are rather similar with and without learning and the difference amounts to a few milliseconds. This further corroborates the utility of learning the preference function and shows that the learning is robust and does not deteriorate the performances of our algorithm.

Qualitative study. In order to illustrate the utility of our approach, in this experiment we study possible rewritings of a mapping defined over the NHS schema.The NHS schema focuses on storing information concerning patients admitted in hospitals. Here, we consider the dependencies involving general information on patients. This includes administrative information

Relation	#atts
birth	34
patient	17
mothers_social_data	8
pis_e_prescribing	27
death	50

(a) Source schema characteristics

Relation	#atts
birth_export	34
patient_export	17
pis_e_prescribing_export	27
death_export	50

(b) Target schema characteristics

Relation	#atts
link_death_drugs	3
death_causes	20
prescribed_drugs_evolution	9
mothers_social	8
fathers_social	6

(c) Policy views schema characteristics

(1) $\mathsf{birth}(\ldots) \rightarrow \mathsf{birth\_export}(\ldots)$

(2) $\mathsf{patient}(\ldots) \rightarrow \mathsf{patient\_export}(\ldots)$

(3) $\mathsf{pis\_e\_prescribing}(\ldots) \rightarrow \mathsf{pis\_e\_prescribing\_export}(\ldots)$

(4) $\mathsf{death}(\ldots) \rightarrow \mathsf{death\_export}(\ldots)$

(d) Mapping over NHS

$\mathsf{death}(\ldots) \wedge \mathsf{pis\_e\_prescribing}(\ldots) \rightarrow \mathsf{link\_death\_drugs\_data}(\ldots)$

$\mathsf{death}(\ldots) \rightarrow \mathsf{death\_causes}(\ldots)$

$\mathsf{pis\_e\_prescribing}(\ldots) \rightarrow \mathsf{prescribed\_drugs\_evolution}(\ldots)$

$\mathsf{birth}(\ldots) \wedge \mathsf{patient}(\ldots) \rightarrow \mathsf{mother\_social}(\ldots)$

$\mathsf{birth}(\ldots) \rightarrow \mathsf{fathers\_social}(\ldots)$

(e) Policy views over NHS

Table 3: Properties of the NHS dataset.

Rewrited	#possible	#frontier variables in repairs	
tgd	repairs	min	max
(1)	3	25	29
(2)	2	13	14
(3)	2	18	18
(4)	2	25	25

Table 4: Properties of the repairing process.

(relation patient in the source schema), social and medical information on the patient himself (relations birth and death), social data on patients' mothers (relation mother_social_data) and information on drugs prescriptions (relation pis_e_prescribing).

The characteristics of the source schema are summarized in Table 3a. The mapping to rewrite and the characteristics of its target schema are summarized in Tables 3d and 3b, respectively. The set of policy views and the characteristics of their target schema are reported in Tables 3e and 3c, respectively.

The link_death_drugs_data view allows to link the prescribed drugs with patient pathology, but no personal information is exported to prevent the identification of the patient. The death_causes view gives access to the causes of death of the admitted patients. The prescribed_drugs_evolution view gives access to drug prescriptions information without any identifying information. The mother_social and fathers_social views give access to patients' mothers and fathers social information.

In Table 4, we show the number of possible repairs for each tgd in Table 3d. It can be seen that the tgd (1) has three possible repairs, exporting from 25 to 29 variables, respectively. Analogously, the tgd (2) leads to two possible repairs, allowing to export from 13 to 14 variables each. Both tgds (3) and (4) lead to two possible repairs, with a constant number of exported variables for each tgd. These rewritings are distinguished by the exported variables, and one can decide to choose which repairing fits best her needs either visually or by leveraging the user preference function, as shown in our previous experiment.

7 CONCLUSION

We have studied the problem of data exchange in the presence of privacy restrictions expressed as policy views on the source schema. We have proposed a repairing process for the mappings that are unsafe under the source policy views. Our approach is inherently data-independent and leads to repairing the mappings guaranteeing privacy preservation at a schema level. As such, our approach is orthogonal to several data-dependent privacy-preservation methods (such as differential privacy methods), that can be used on the source and target instances to further corroborate the privacy guarantees. The study of such fruitful combinations of methods is devoted to future work.

We also envision several other extensions of our work, such as the study of more expressive GLAV mappings and the interplay between data-independent and data-dependent privacy methods as well as the usage of other learning methods.

ACKNOWLEDGEMENTS

Research by the first author is funded by ANR (under Grant No. 18-CE23-0002 QualiHealth)

REFERENCES

[1] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. 2014. *Foundations of Data Exchange.* Cambridge University Press.

[2] Patricia C Arocena, Boris Glavic, Radu Ciucanu, and Renée J Miller. 2015. The iBench integration metadata generator. In *Proceedings of VLDB.*

[3] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. 2000. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics* 16, 5 (2000).

[4] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). 2011. *Schema Matching and Mapping.* Springer.

[5] M. Benedikt, B. Cuenca Grau, and E. Kostylev. 2017. Source Information Disclosure in Ontology-Based Data Integration.. In *AAAI.*

[6] Philip A. Bernstein. 2005. The many roles of meta data in data integration. In *In ACM SIGMOD.*

[7] Joachim Biskup and Piero Bonatti. 2004. Controlled query evaluation for enforcing confidentiality in complete information systems. *International Journal of Information Security* 3, 1 (2004).

[8] Joachim Biskup and Torben Weibert. 2008. Keeping secrets in incomplete databases. *International Journal of Information Security* 7, 3 (2008), 199–217.

[9] Piero A. Bonatti, Sarit Kraus, and VS Subrahmanian. 1995. Foundations of secure deductive databases. *IEEE TKDE* 7, 3 (1995), 406–422.

[10] Angela Bonifati, Ugo Comignani, and Efthymia Tsamoura. 2019. MapRepair: Mapping and Repairing under Policy Views (demo). In *ACM SIGMOD.* 1873–1876.

[11] Laura Chiticariu and Wang Chiew Tan. 2006. Debugging Schema Mappings with Routes. In *Proceedings of VLDB.* 79–90.

[12] Ugo Comignani. 2020. MapRepair - open source code. https://github.com/ucomignani/MapRepair.git.

[13] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. 2018. Query-Based Linked Data Anonymization. In *ISWC 2018.* 530–546.

[14] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. 2019. RDF Graph Anonymization Robust to Data Linkage. In *WISE 2019.* 491–506.

[15] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.

[16] Frank Eibe, MA Hall, and IH Witten. 2016. The WEKA Workbench. Online Appendix for" Data Mining: Practical Machine Learning Tools and Techniques. *Morgan Kaufmann* (2016).

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005).

[18] Bernardo Cuenca Grau, Evgeny Kharlamov, Egor V. Kostylev, and Dmitriy Zheleznyakov. 2015. Controlled Query Evaluation for Datalog and OWL 2 Profile Ontologies. In *IJCAI.*

[19] Bernardo Cuenca Grau and Egor V. Kostylev. 2016. Logical Foundations of Privacy-Preserving Publishing of Linked Data. In *AAAI.* 943–949.

[20] Gerome Miklau and Dan Suciu. 2007. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.* 73, 3 (2007).

[21] Alan Nash and Alin Deutsch. 2007. Privacy in GLAV Information Integration. In *ICDT.*

[22] Muhammad I. Sarfraz, Mohamed Nabeel, Jianneng Cao, and Elisa Bertino. 2015. DBMask: Fine-Grained Access Control on Encrypted Relational Databases. In *In CODASPY.* 1–11.

[23] George L Sicherman, Wiebren De Jonge, and Reind P Van de Riet. 1983. Answering queries without revealing secrets. *ACM TODS* 8, 1 (1983), 41–59.

[24] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 5 (2002), 557–570.

CONTENTS

ABSTRACT 1
1. INTRODUCTION 1
1.1. ILLUSTRATIVE EXAMPLE 1
2. RELATED WORK 2
3. PRELIMINARIES 3
4. PRIVACY PRESERVATION 3
4.1. A FORMAL PRIVACY-PRESERVATION PROTOCOL 3
4.2. PRESERVING THE PRIVACY OF POLICY VIEWS 5
5. REPAIRING UNSAFE MAPPINGS 6
5.1. COMPUTING PARTIALLY-SAFE MAPPINGS 6
5.2. COMPUTING SAFE MAPPINGS 8
6. EXPERIMENTS 10
7. CONCLUSION 12
ACKNOWLEDGEMENTS 12
REFERENCES 12

Exchanging Data under Policy Views

Angela Bonifati
Lyon 1 University & Liris CNRS
Lyon, France
angela.bonifati@univ-lyon1.fr

Ugo Comignani
Grenoble INP Ensimag & LIG CNRS
Grenoble, France
ugo.comignani@grenoble-inp.fr

Efthymia Tsamoura
Samsung AI Research
Cambridge, UK
efi.tsamoura@samsung.com

ABSTRACT

Exchanging data between data sources is a fundamental problem in many data science and data integration tasks. In this paper, we focus on the data exchange problem in the presence of privacy constraints on the source data, which has been disregarded in the literature to date. By leveraging a logical privacy-preservation paradigm, the privacy restrictions are expressed as a set of *policy views* representing the information that is safe to expose over *all* instances of the source in order to exchange them with the target. We introduce a protocol that provides formal privacy guarantees and is *data-independent*, i.e., under certain criteria, it guarantees that the mappings leak no sensitive information independently of the instances lying in the source. Moreover, we design an algorithm for *repairing* an input mapping w.r.t. a set of policy views, in cases where the input mapping leaks sensitive information. We show that the repairing can build upon hard-coded and learning-based user preference functions and we show the trade-offs. Our empirical evaluation shows that repairing mappings is quite efficient, leading to repairing sets of 300 s-t tgds in an average time of 5s on a commodity machine. It also shows that the repairing based on learning is robust and has comparable runtimes with the hard-coded one.

KEYWORDS

privacy-preserving data integration, data exchange, mapping repairs

1 INTRODUCTION

Data exchange is a key process in data science and data integration pipelines, leading to translating data compliant with a source schema S and lying in a source database to a target database with a non-overlapping target schema T [1, 4, 17]. Data exchange is also part of metadata management operations [6], since the schema mappings between source and target also known as *source-to-target* (s-t) dependencies Σ_{st} (s-t tgds) are declarative expressions manipulating schema elements, i.e. metadata rather than data.

Despite a wealth of research on the topic, the privacy-aware variant of the data exchange problem has received little attention to date. However, recent data protection regulations such as EU GDPR or CCPA in the US bring the attention to the problem of protecting personal data when transferring data across countries and institutions, thus motivating our work. In a privacy-aware data exchange scenario (as exemplified in Figure 1), the source schema comes with a set of constraints called policy views $\mathcal{V}$ representing the data that is *safe* to expose to the target over *all instances* of the source. The policy views can be considered as user views on the data of the source and can encode possible formulations of the different purposes the data will undergo during the exchange process as in many data protection regulations.

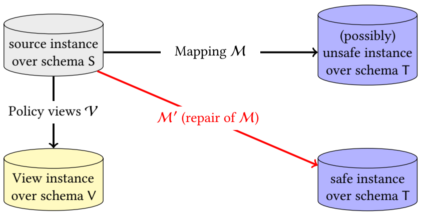

source instance over schema S
Mapping $\mathcal{M}$
(possibly) unsafe instance over schema T
Policy views $\mathcal{V}$
$\mathcal{M}'$ (repair of $\mathcal{M}$)
View instance over schema V
safe instance over schema T

Figure 1: A privacy-aware data exchange setting with mappings and policy views.

This process entails the repairing of the original mapping $\mathcal{M}$ into a mapping $\mathcal{M}'$ in order to make the exported target instance safe. However, in order to realize such a data exchange scenario, one needs to address the following issues: (1) given a set of privacy restrictions on the source schema, what would it mean for a data exchange setting to be safe under the proposed privacy restrictions?; (2) assuming that the privacy-preservation protocol is fixed, how could we assess the safety of a data exchange setting w.r.t. the privacy restrictions and provide *strong guarantees* of no privacy leakage?; finally, in case of privacy violations, (3) how could we *repair* the s-t tgds (and transform the mapping $\mathcal{M}$ into a repaired mapping $\mathcal{M}'$)?

To address the first issue, we build upon prior work on the logical foundations of privacy-preserving data integration [5, 21], and we tailor them to a data exchange setting. Hence, we define a set of s-t tgds to be safe w.r.t. the policy views if *every positive information* that is kept secret by the policy views is also kept secret by the s-t tgds. As we will see in subsequent sections and contrarily to previous work, our proposed privacy-preservation protocol is *data-independent* allowing us to provide strong privacy-preservation guarantees over all instances of the sources. As such, our work leads to the first practical framework establishing privacy-conscious data exchange. The above addresses the second aforementioned issue in that it enables a schema-level enforcement of the privacy-preserving protocol with strong guarantees. Regarding the third issue, we propose a repairing algorithm for the proposed privacy-preservation protocol in case of detected unsafety. Since multiple repairs are possible, such an algorithm might leverage techniques for learning the user preferences during the repairing process, which is also a desirable feature in privacy enforcement over sensitive data. In order to further illustrate the relevance of our problem, we illustrate a running example inspired by a real-life data exchange process between two different hospitals in the UK[1].

1.1 Illustrative example

Consider the source schema S of NHS consisting of the following relations: P, H_N, H_S, O and S as illustrated in Figure 2 (a). Relation P stores for each person registered with the hospital, his insurance number, his name, his ethnicity group and his county. Relations H_N and H_S store for each patient who has been admitted to some hospital in the north or the south of UK, his

[1] https://www.nhs.uk/

$$\begin{aligned}
&\text{Source schema } \mathsf{S} = \{\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}); \mathsf{H}_\mathsf{N}(\mathsf{i},\mathsf{d}); \mathsf{H}_\mathsf{S}(\mathsf{i},\mathsf{d});\\
&\qquad\qquad\qquad\qquad \mathsf{O}(\mathsf{i},\mathsf{t},\mathsf{p}); \mathsf{S}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c})\}\\
&\text{Target schema } \mathsf{T} = \{\mathsf{EthDis}(\mathsf{e},\mathsf{d}); \mathsf{CountyDis}(\mathsf{c},\mathsf{d}); \mathsf{SO}(\mathsf{e})\}\\
&\text{View schemas } \mathsf{V} = \{\mathsf{V}_1(\mathsf{e},\mathsf{d}); \mathsf{V}_2(\mathsf{c},\mathsf{d}); \mathsf{V}_3(\mathsf{t},\mathsf{p}); \mathsf{V}_4(\mathsf{e})\}
\end{aligned}$$

(a) Schemas S, T **and** V

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \wedge \mathsf{H}_\mathsf{N}(\mathsf{i},\mathsf{d}) \to \mathsf{V}_1(\mathsf{e},\mathsf{d}) \tag{1}$$

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \wedge \mathsf{H}_\mathsf{S}(\mathsf{i},\mathsf{d}) \to \mathsf{V}_2(\mathsf{c},\mathsf{d}) \tag{2}$$

$$\mathsf{O}(\mathsf{i},\mathsf{t},\mathsf{p}) \to \mathsf{V}_3(\mathsf{t},\mathsf{p}) \tag{3}$$

$$\mathsf{S}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \to \mathsf{V}_4(\mathsf{e}) \tag{4}$$

(b) Policy views $\mathcal{V}$

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \wedge \mathsf{H}_\mathsf{N}(\mathsf{i},\mathsf{d}) \to \mathsf{EthDis}(\mathsf{e},\mathsf{d}) \tag{5}$$

$$\mathsf{P}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \wedge \mathsf{H}_\mathsf{N}(\mathsf{i},\mathsf{d}) \to \mathsf{CountyDis}(\mathsf{c},\mathsf{d}) \tag{6}$$

$$\mathsf{S}(\mathsf{i},\mathsf{n},\mathsf{e},\mathsf{c}) \wedge \mathsf{O}(\mathsf{i},\mathsf{t},\mathsf{p}) \to \mathsf{SO}(\mathsf{e}) \tag{7}$$

(c) Mapping from S **to** T.

Figure 2: Schema and tgds in our illustrative example

insurance number and the reason for being admitted to the hospital. Relation O stores information related to patients in oncology departments and, in particular, their insurance numbers, their treatment and their progress. Finally, relation S stores for each student in UK, his insurance number, his name, his ethnicity group and his county.

Consider also the set $\mathcal{V}$ comprising the policy views (1)–(4). The policy views define the information that is safe to make available to the public. View (1) projects the ethnicity groups and the hospital admittance reasons for patients in the north of UK; view (2) projects the counties and the hospital admittance reasons for patients in the south of UK; view (3) projects the treatments and the progress of patients of oncology departments; view (4) projects the ethnicity groups of the school students. The policy views are compliant with the NSS privacy preservation protocol that is adopted at the hospital. Precisely, the NSS privacy preservation protocol considers as unsafe any non-evident piece of information that can potentially de-anonymize an individual. For example, views (1) and (2) do not leak any sensitive information concerning the precise address of patients. Indeed, they include patients from a very large geographical area thus implying that the probability of de-anonymizing a patient is significantly small. Similarly, views (3) and (4) are considered to be safe: the probability of de-anonymizing patients of the oncology department from view (3) is zero, since there is no way to link a patient to his treatment or his progress, while view (4) projects information which is already evident to public.

Finally, consider the following set of source-to-target dependencies Σ_{st}. Dependencies (5) and (6) project similar information with the views (1) and (2), respectively. However, contrarily to the views, they solely focus on patients in the north of UK. Finally, dependency (7) projects the ethnicity groups of students who have been in some oncology department, whereas view V_4 aims at concealing the information about the department in which a student has been admitted.

The above example shows that the policy views defined within one hospital might be in stark contrast with the mappings used to export patient's information to another hospital. This motivates the aforementioned questions (1), (2) and (3) about establishing formal guarantees for privacy preservation as well as enabling repairing of the mappings in order to make them safe. To the best of our knowledge, our work is the first to provide practical algorithms for a logical privacy-preservation paradigm effective in a real system [10], described as an open research challenge in [5, 21]. Our technique is inherently data-independent thus bringing the advantage that both the safety test and the repairing operations are executed on the metadata provided through the mappings and not on the underlying data instances.

The paper is organized as follows. Section 2 discusses the related work. Section 3 presents the basic concepts and notions. Section 4 lays our privacy preservation protocol. Section 5 presents our repairing algorithms and their properties. mechanism. Section 6 outlines the experimental results, while Section 7 concludes our work. The code base along with the experimental data are publicly available at [12].

2 RELATED WORK

Privacy in data integration. Safety of secret queries formulated against a global schema and adhering to the certain answers semantics has been tackled in previous theoretical work [21]. They define the optimal attack that characterizes a set of queries that an attacker can issue to which no further queries can be added to infer more information. They then define the privacy guarantees against the optimal attack by considering the static and the dynamic case, the latter corresponding to modifications of the schemas or the GLAV mappings. The same definition of secret queries and privacy setting is adopted in [5], which instead focuses on boolean conjunctive queries as policy views and on the notion of safety with respect to a given mapping. An ontology-based integration scenario is assumed in which the target instance is produced via a set of mappings starting from an underlying data source. Whereas they study the complexity of the view compliance problem in both data-dependent and data-independent setting, we focus on the latter and extend it to non-boolean conjunctive queries as policy views. We further consider multiple policy views altogether in the design of a practical algorithm for checking the safety of schema mappings and for repairing the mappings in case of violations.

Privacy in data publishing. Data publishing accounts for the settings in which a view exports or publishes the information of an underlying data source. Privacy and information disclosure in data publishing linger over the problem of avoiding the disclosure of the content of the view under a confidential query. A probabilistic formal analysis of the query-view security model has been presented in [20], where they offer a complete treatment of the multi-party collusion and the use of external adversarial knowledge. Access control policies using cryptography are used in [20] to enforce the authorization to an XML document. Our work differs from theirs on both the considered setting, as well as the adopted techniques and the adopted privacy protocol. Striking the balance between utility and privacy in a logic-based framework has been the object of investigation in recent studies focusing on data publishing for Linked Data [13, 14, 19]. The problem there is remarkably different from ours since they focus on publishing a single RDF dataset by applying privacy and utility queries in SPARQL, checking for their compatibility, and for update operations realizing the privacy and utility constraints.

Controlled Query Evaluation. Controlled Query Evaluation is a confidentiality enforcement framework introduced in [23] and refined in [9],[7] and [8], in which a policy declaratively specifies sensitive information and confidentiality is enforced by a censor. Provided a query as input, a censor verifies whether the query leads to a violation of the policy and in case of a violation it returns a distorted answer. It has been recently adopted in ontologies expressed with Datalog-like rules and in lightweight Description Logics [18]. They assume that the policies are only known to database administrators and not to ordinary users and that the data has protected access through a query interface. Our assumptions and setting are quite different, since our multiple policy views are accessible to every user and our goal is to render the s-t mappings safe with respect to a set of policies via repairing and rewriting.

Data privacy. Previous work has addressed access control to protect database instances at different levels of granularity [22], in order to combine encrypted query processing and authorization rules. Our work being logic-based and declarative does not deal with these authorization methods, as well as does not consider any concrete privacy or anonymization algorithms operating on data instances, such as differential privacy [15] and k-anonymity [24]. Further exploring the connection between concrete privacy enforcement and logic-based privacy formalisms is the subject of future investigation.

Data exchange. The vast literature on data exchange [17] has inspired our work. In the considered scenarios, the source and target schema are considered along with s-t mappings and target dependencies, the latter being both egds and tgds. Similarly, past work on degugging schema mappings [11] has focused on all possible routes generated by the exchange process when incomplete or undefined values in one or more variables are exported from the source instance. By opposite, we focus in this paper on the case in which s-t mappings are coupled with source dependencies under the form of policy views, the latter being typical in privacy scenarios and unexplored in the classical data exchange setting.

3 PRELIMINARIES

Relational symbols and critical instances. Let Const, Nulls, and Vars be mutually disjoint, infinite sets of *constant values*, *labeled nulls*, and *variables*, respectively. A *schema* is a set of *relation names* (or just *relations*), each associated with a nonnegative integer called *arity*. A *relational atom* has the form $R(\vec{t})$ where R is an n-ary relation and $\vec{t}$ is an n-tuple of *terms*, where a term is either a constant, a labelled null, or a variable. An *equality atom* has the form $t_1 = t_2$ where t_1 and t_2 are terms. An atom is called ground or fact, when it does not contain any variables. A position in an n-ary atom A is an integer $1 \leq i \leq n$. We denote by $A|_i$, the i-th term of A. An instance I is a set of relational facts. An atom (resp. an instance) is *null-free* if it does not contain labelled nulls. The *critical instance* of a schema S, denoted as Crt_S, is the instance containing a fact of the form $R(\vec{*})$, for each n-ary relation $R \in \mathsf{S}$, where $*$ is called the *critical constant* and $\vec{*}$ is an n-ary vector. A *substitution* σ is a mapping from variables into constants or labelled nulls.

Dependencies and queries A *dependency* describes the semantic relationship between relations. A *Tuple Generating Dependency* (tgd) is a formula of the form $\forall\vec{x}\ \lambda(\vec{x}) \rightarrow \exists\vec{y}\ \rho(\vec{x},\vec{y})$, where $\lambda(\vec{x})$ and $\rho(\vec{x},\vec{y})$ are conjunctions of relational, null-free atoms. An *Equality Generating Dependency* (egd) is a formula of the form $\forall\vec{x}\ \lambda(\vec{x}) \rightarrow x_i = x_j$, where $\lambda(\vec{x})$ is a conjunction of relational, null-free atoms. We usually omit the quantification for brevity. We refer to the left-hand side of a tgd or an egd δ as the *body*, denoted as $\mathsf{body}(\delta)$, and to the right-hand side as the *head*, denoted as $\mathsf{head}(\delta)$. An instance I satisfies a dependency δ, written $I \models \delta$ if each homomorphism from $\mathsf{body}(\delta)$ into I can be extended to a homomorphism h' from $\mathsf{head}(\delta)$ into I. An instance I satisfies a set of dependencies Σ, written as $I \models \Sigma$, if $I \models \delta$ holds, for each $\delta \in \Sigma$. The *solutions* of an instance I w.r.t. Σ is the set of all instances J such that $J \supseteq I$ and $J \models \Sigma$. A solution is called *universal* if it can be homomorphically embedded to each solution of I w.r.t. Σ.

A *conjunctive query* (CQ) is a formula of the form $\exists\vec{y}\ \bigwedge_i A_i$, where A_i are relational, null-free atoms. A CQ is boolean if it does not contain any free variables. A substitution σ is an *answer* to a CQ Q on an instance I if the domain of σ is the free variables of Q, and if σ can be extended to a homomorphism from $\bigwedge_i A_i$ into I. We denote by $Q(I)$, the answers to Q on I.

Let S be a source schema and let T be a target schema. A *mapping* $\mathcal{M}$ from S to T is defined as a triple $(\mathsf{S},\mathsf{T},\Sigma)$, where $\Sigma = \Sigma_{st}$;, i.e. the set of the s-t dependencies over S and T. We usually refer to the dependencies in Σ_{st} as *mappings*. A variable x of a mapping $\mu \in \Sigma_{st}$ is called exported if it occurs both in the body and the head of μ. We denote by $\mathsf{exported}(\mu)$, the set of exported variables of μ. The inverse of set of s-t dependencies Σ_{st}, denoted as Σ_{st}^{-1} is the set consisting, for each mapping μ in Σ_{st} of the form $\lambda(\vec{x}) \rightarrow \rho(\vec{x},\vec{y})$, a mapping μ^{-1} of the form $\rho(\vec{x},\vec{y}) \rightarrow \lambda(\vec{x})$. We focus on *GAV* mappings in this paper, thus assuming that $\vec{y}$ is empty. Moreover, we consider the setting in which Σ only consists of Σ_{st} thus not including Σ_t. This implies that target egds and target tgds are excluded, since, despite their usage in data exchange, their role is less understood in the privacy-preserving variant considered in this paper.

4 PRIVACY PRESERVATION

In this section, we introduce our notion of privacy preservation. Let $\mathcal{V}$ be a set of *policy views* over S, representing the information that is safe to expose for instances I of S. Our goal is to verify whether a user-defined mapping $\mathcal{M} = (\mathsf{S},\mathsf{T},\Sigma)$ is safe w.r.t. the views in a set $\mathcal{V}$. Below, we will introduce a notion for assessing the safety of a GAV mapping $\mathcal{M}_2$ with respect to a GAV mapping $\mathcal{M}_1$, when both make use of the same source schema S. Moreover, let $\Sigma_i = \Sigma_{st_i}$ be the dependencies associated with $\mathcal{M}_i$.

4.1 A formal privacy-preservation protocol

Our notion of privacy preservation builds on the logical foundations introduced in [5] for ontology-based data integration for boolean queries. However, we extend the notion of privacy preservation from [5] to a relational data exchange setting in the presence of non-boolean conjunctive queries. First, we define the notion of disclosure of a CQ by a mapping as follows:

Definition 4.1. A mapping $\mathcal{M} = (\mathsf{S},\mathsf{T},\Sigma)$ *does not disclose a CQ* p *over* S *on any instance of* S, if for each instance I of S there exists an instance I' such that $I \equiv_{\mathcal{M}} I'$ and $p(I') = \emptyset$.

The problem of checking whether a mapping $\mathcal{M}$ over S does not disclose a boolean and constants-free CQ p on any instance of S is decidable for GAV mappings consisting of CQ views [5]. In particular, $\mathcal{M}$ does not disclose p on any instance of S if and only if there does not exist a homomorphism from p into the *unique* instance computed by the *visible chase* $\mathsf{visChase}_\mathsf{S}(\Sigma)$ of Σ under the critical instance Crt_S of S. The visible chase computes a *universal source instance* defined as follows:

Definition 4.2. 7 Given a mapping $\mathcal{M} = (\mathsf{S},\mathsf{T},\Sigma)$, an instance I is a *universal source instance* over S if for any instance J over the source schema S, there exists an homomorphism h from J into I

such that for any constant c from J that is made visible through $\mathcal{M}$, $h(c) = *$.

The only constant occurring in the instance computed by $\mathsf{visChase}_\mathsf{S}(\Sigma)$ is the critical constant $*$ and it represents any other constant that can occur in the source instance.

We introduce our own variant of the visible chase, which organizes the facts derived during chasing into subinstances called *bags*. Algorithm 1 describes the steps of the proposed variant. Please note that Algorithm 1 derives the same set of facts with the algorithm from [5]. However, instead of keeping these facts in a single set, we keep them in separate bags. Before presenting Algorithm 1, we will introduce a couple of useful new notions. The first notion serves the need of defining derived egds that allow to unify as many labeled nulls as possible with the critical constant in the target instance. The second notion allows to define relevant bags for which this unification must hold. Both notions are exploited by the visible chase (Algorithm 1) whose last step triggers the obtained egds.

Definition 4.3. Consider an instance I. Consider also a s-t tgd δ and a homomorphism h from $\mathsf{body}(\delta)$ into I, such that $h(x) \in \mathsf{Nulls}$, for some $x \in \mathsf{exported}(\delta)$. Then, we say that the egd

$$\mathsf{body}(\delta) \rightarrow \bigwedge_{\forall x \in \mathsf{exported}(\delta) : h(x) \in \mathsf{Nulls}} x \approx * \tag{8}$$

is *derived* from δ in I. For an egd ϵ that is derived from a s-t tgd δ in I, $\mathsf{tgd}(\epsilon)$ denotes δ. For a set of s-t tgds Σ and an instance I, $\Sigma_\approx$ is the set comprising for each $\delta \in \Sigma$, the egd that is derived from δ in I.

Definition 4.4. Consider an instance I, whose facts are organized into the bags $\beta_1, \dots, \beta_m$. Consider also a derived egd δ of the form (8) and an active trigger h for δ in I. A bag β_i is *relevant* for δ and h in I, where $1 \leq i \leq m$, if some fact $F \in h(\mathsf{body}(\delta))$ occurs in β_i and if some $h(x)$ is a labeled null occurring in β_i, where $x \in \mathsf{exported}(\delta)$.

Let $\beta_{j_1}, \dots, \beta_{j_k} \subseteq \beta_1, \dots, \beta_m$ be the set of bags that are relevant for δ and h in I. Let $v = \{h(x_j) \mapsto h(x_i)\}$ if $h(x_i) = *$, and $v = \{h(x_i) \mapsto h(x_j)\}$ if $h(x_i) \notin \mathsf{Const}$, where x_i, x_j are variables from $\mathsf{exported}(\delta)$. Then, the *derived* bag β for δ and h in I consists of the facts in $\bigcup_{l=1}^{k} v(\beta_{j_l})$. The bags $\beta_{j_1}, \dots, \beta_{j_k}$ are called the *predecessors* of β. We use $\beta_{j_l} \prec \beta$ to denote that β_{j_l} is a predecessor of β, for $1 \leq l \leq k$.

We are now ready to proceed with the description of Algorithm 1. Given a s-t mapping, Algorithm 1 computes a universal source instance whose facts are organized into bags. Algorithm 1 first computes the instance I_0 by chasing Crt_S using the s-t tgds, line 1. It then chases I_0 with the inverse s-t tgds Σ^{-1}, line 2. and proceeds by chasing I_1 with the set of all derived egds $\Sigma_\approx$, for each $\delta \in \Sigma$ in I_1, line 4. Algorithm 1 computes a fresh bag at each chase step. In particular, for each active trigger h for δ in I, Algorithm 1 adds a fresh bag with facts $h'(\mathsf{head}(\delta))$, if $\delta \in \Sigma \cup \Sigma^{-1}$, line 9; otherwise, if $\delta \in \Sigma_\approx$, then it adds the derived bag for δ and h in I, see Definition 4.4, line 20.

Note that, $\Sigma_\approx$ aims at "disambiguating" as many labeled nulls occurring in I_1 as possible, by unifying them with the critical constant $*$. Since $*$ represents the information that is "visible" to a third-party, chasing with $\Sigma_\approx$ computes the *maximal information* from the source instance a third-party has access to. Note that Algorithm 1 always terminates [5]. Let $B = \mathsf{visChase}_\mathsf{S}(\Sigma)$. We will denote by $I_\mathsf{S}(\Sigma)$, the instance $\bigcup_{\beta \in B} \beta$.

Algorithm 1 $\mathsf{visChase}_\mathsf{S}(\Sigma)$

```
1:  $B_0 := \mathsf{bagChaseTGDs}(\Sigma, \mathsf{Crt}_\mathsf{S})$
2:  $B_1 := \mathsf{bagChaseTGDs}(\Sigma^{-1}, \bigcup_{\beta \in B_0} \beta \setminus \mathsf{Crt}_\mathsf{S})$
3:  Let $\Sigma_\approx$ be the set of all derived egds $\Sigma_\approx$, for each $\delta \in \Sigma$ in $I_1$
4:  **return** $\mathsf{bagChaseEGDs}(\Sigma_\approx, B_0 \cup B_1)$

5:  **procedure** $\mathsf{bagChaseTGDs}(\Sigma, I)$
6:      $B := \emptyset$
7:      **for each** $\delta \in \Sigma$ **do**
8:          **for each** active trigger $h : \mathsf{body}(\delta) \rightarrow I$ **do**
9:              **create** a fresh bag $\beta$ with facts $h'(\mathsf{head}(\delta))$
10:             **add** $\beta$ to $B$
11:     **return** $B$

12: **procedure** $\mathsf{bagChaseEGDs}(\Sigma_\approx, B)$
13:     $i := 0$; $I_i := \bigcup_{\beta \in B} \beta$
14:     **do**
15:         $i := i + 1$
16:         **for each** ($\delta \in \Sigma_\approx$ of the form (8) **do**
17:             **for each** active trigger $h : \mathsf{body}(\delta) \rightarrow I_{i-1}$ **do**
18:                 **if** $h(x) \neq *$, for some $x \in \mathsf{exported}(\delta)$ **then**
19:                     Let $\beta$ be the derived bag for $\delta$ and $h$ in $I_{i-1}$
20:                     **add** $\beta$ to $B$
21:                     $I_i := I_i \cup \beta$
22:     **while** $I_{i-1} \neq I_i$
23:     **return** $B$
```

Example 4.5. We demonstrate the visible chase algorithm over the policy views and the s-t dependencies from Example 1.1. We show how the algorithm runs first on the policy views $\mathcal{V}$ and then show the computation on Σ_{st}.

We first present the computation of $I_\mathsf{S}(\mathcal{V}) = \bigcup_{\beta \in \mathsf{visChase}_\mathsf{S}(\mathcal{V})} \beta$.

The critical instance Crt_S of S consists of the facts shown in the following Eq. (9)

$$\mathsf{P}(*, *, *, *) \quad \mathsf{H_N}(*, *) \quad \mathsf{H_S}(*, *) \quad \mathsf{O}(*, *, *) \quad \mathsf{S}(*, *, *, *) \tag{9}$$

where $*$ is the critical constant.

The instance I_1 computed by chasing the output of line 1 using $\mathcal{V}^{-1}$ will consist of the facts

$$\begin{array}{lll} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}) & \mathsf{H_N}(\mathsf{n_i}, *) & \mathsf{O}(\mathsf{n_i''}, *, *) \\ \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *) & \mathsf{H_S}(\mathsf{n_i'}, *) & \mathsf{S}(\mathsf{n_i'''}, \mathsf{n_n'''}, *, \mathsf{n_c'''}) \end{array} \tag{I_1}$$

where the constants prefixed by n are labeled nulls created while chasing Crt_S with the inverse mappings. Since there exists no homomorphism from the body of any s-t tgd into I_1 mapping an exported variable into a labeled null, $\Sigma_\approx$ will be empty, see Definition 4.3. Thus, $I_\mathsf{S}(\mathcal{V}) = I_1$.

We next present the computation of $I_\mathsf{S}(\Sigma_{st}) = \bigcup_{\beta \in \mathsf{visChase}_\mathsf{S}(\Sigma_{st})} \beta$. The instance I_1' computed by chasing the output of line 1 by Σ_{st}^{-1} will consist of the facts

$$\begin{array}{lll} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}) & \mathsf{H_N}(\mathsf{n_i}, *) & \mathsf{S}(\mathsf{n_i''}, \mathsf{n_n''}, *, \mathsf{n_c'}) \\ \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *) & \mathsf{H_N}(\mathsf{n_i'}, *) & \mathsf{O}(\mathsf{n_i''}, \mathsf{n_t''}, \mathsf{n_p''}) \end{array} \tag{I_1'}$$

Since there exists a homomorphism from the body of μ_e into I_1' mapping the exported variable e into the labeled null $\mathsf{n_e}$, and since there exists another homomorphism from the body of μ_c into I_1' mapping the exported variable c into the labeled null $\mathsf{n_c}$, $\Sigma_\approx$ will comprise the egds ϵ_1 and ϵ_2 shown below

$$\mathsf{P}(\mathsf{i}, \mathsf{n}, \mathsf{e}, \mathsf{c}) \land \mathsf{H_N}(\mathsf{i}, \mathsf{d}) \rightarrow \mathsf{e} \approx * \tag{ϵ_1}$$

$$\mathsf{P}(\mathsf{i}, \mathsf{n}, \mathsf{e}, \mathsf{c}) \land \mathsf{H_N}(\mathsf{i}, \mathsf{d}) \rightarrow \mathsf{c} \approx * \tag{ϵ_2}$$

The last step of the visible chase involves chasing I_1' using $\Sigma_{\approx}$. W.l.o.g, assume that the chase considers first ϵ_1 and then ϵ_2. During the first step of the chase, there exists a homomorphism from body(ϵ_1) into I_1'. Hence, $\mathsf{n_e} = *$. During the second step of the chase, there exists a homomorphism from body(ϵ_2) into I_1' and, hence, $\mathsf{n_c} = *$. The instance computed at the end of the second round of the chase will consist of the facts

$$\begin{array}{ccc} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, *) & \mathsf{H_N}(\mathsf{n_i}, *) & \mathsf{H_N}(\mathsf{n_i'}, *) \\ \mathsf{S}(\mathsf{n_i''}, \mathsf{n_n''}, *, \mathsf{n_c'}) & \mathsf{O}(\mathsf{n_i''}, \mathsf{n_t''}, \mathsf{n_p''}) & \end{array} \tag{10}$$

Since there exists no active trigger for ϵ_1 or ϵ_2 in the above instance (Eq (10)), the chase will terminate.

The facts in $I_{\mathsf{S}}(\Sigma_{st})$ will be organized into the following bags β_1–β_5 (one bag per line)

$$\begin{aligned}
\mathsf{SO}(\mathsf{e}) &\xrightarrow{\langle \mu_s^{-1}, h_1 \rangle} \mathsf{S}(\mathsf{n_i''}, \mathsf{n_n''}, *, \mathsf{n_c'}), \mathsf{O}(\mathsf{n_i''}, \mathsf{n_t''}, \mathsf{n_p''}) \\
\mathsf{CountyDis}(\mathsf{c}, \mathsf{d}) &\xrightarrow{\langle \mu_c^{-1}, h_2 \rangle} \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *), \mathsf{H_N}(\mathsf{n_i'}, *) \\
\mathsf{EthDis}(\mathsf{e}, \mathsf{d}) &\xrightarrow{\langle \mu_e^{-1}, h_3 \rangle} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}), \mathsf{H_N}(\mathsf{n_i}, *) \\
\mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, \mathsf{n_e}, *), \mathsf{H_N}(\mathsf{n_i'}, *) &\xrightarrow{\langle \epsilon_1, h_4 \rangle} \mathsf{P}(\mathsf{n_i'}, \mathsf{n_n'}, *, *), \mathsf{H_N}(\mathsf{n_i'}, *) \\
\mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, \mathsf{n_c}), \mathsf{H_N}(\mathsf{n_i}, *) &\xrightarrow{\langle \epsilon_2, h_5 \rangle} \mathsf{P}(\mathsf{n_i}, \mathsf{n_n}, *, *), \mathsf{H_N}(\mathsf{n_i}, *)
\end{aligned}$$

$$\begin{aligned}
h_1 &= \{\mathsf{i} \mapsto \mathsf{n_i'}, \mathsf{n} \mapsto \mathsf{n_n'}, \mathsf{e} \mapsto \mathsf{n_e}, \mathsf{c} \mapsto *, \mathsf{d} \mapsto *\} \\
h_2 &= \{\mathsf{c} \mapsto *, \mathsf{d} \mapsto *\} \\
h_3 &= \{\mathsf{e} \mapsto *, \mathsf{d} \mapsto *\} \\
h_4 &= \{\mathsf{i} \mapsto \mathsf{n_i'}, \mathsf{n} \mapsto \mathsf{n_n'}, \mathsf{e} \mapsto \mathsf{n_e}, \mathsf{c} \mapsto *, \mathsf{d} \mapsto *\} \\
h_5 &= \{\mathsf{i} \mapsto \mathsf{n_i}, \mathsf{n} \mapsto \mathsf{n_n}, \mathsf{e} \mapsto *, \mathsf{c} \mapsto \mathsf{n_c}, \mathsf{d} \mapsto *\}
\end{aligned}$$

The contents of the bags correspond to the right-hand side of the arrows. However, for presentation purposes, we also show the related dependency δ and the homomorphism h that lead to the derivation of each bag (shown at the top of each arrow), as well as, the facts in $h(\text{body}(\delta))$ (left-hand side of each arrow). The obtained bags will be part of the universal source instance $I_{\mathsf{S}}(\Sigma_{st})$. Such an instance will be used in Section 5 in order to apply the notion of safety in the repairing of the underlying mappings Σ_{st}.

4.2 Preserving the privacy of policy views

We consider a mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ to be safe w.r.t. a view mapping $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ (with $\mathcal{V}$ being the set of policy views and V being the schema of the views as shown in Figure 1), if $\mathcal{M}$ does not disclose the information that is also not disclosed by $\mathcal{M}_{\mathsf{V}}$. Definition 4.6 and Theorem 4.7 presented below formalize our notion of privacy preservation and show that there exists a simple process for verifying whether $\mathcal{M}$ is safe w.r.t. $\mathcal{M}_{\mathsf{V}}$.

Definition 4.6. A mapping $\mathcal{M}_2 = (\mathsf{S}, \mathsf{T}_2, \Sigma_2)$ *preserves the privacy of a mapping* $\mathcal{M}_1 = (\mathsf{S}, \mathsf{T}_1, \Sigma_1)$ *on all instances of* S, if for each constants-free CQ p over S, if p is not disclosed by $\mathcal{M}_1$ on any instance of S, then p is not disclosed by $\mathcal{M}_2$ on any instance of S.

Theorem 4.7. *A mapping* $\mathcal{M}_2 = (\mathsf{S}, \mathsf{T}_2, \Sigma_2)$ preserves the privacy of a mapping $\mathcal{M}_1 = (\mathsf{S}, \mathsf{T}_1, \Sigma_1)$ on all instances of S, *if and only if there exists a homomorphism h from* $I_{\mathsf{S}}(\Sigma_2)$ *into* $I_{\mathsf{S}}(\Sigma_1)$, *such that* $h(*) = *$.

Proof. (Sketch) First we show that the following holds:

Lemma 4.8. *A mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ does not disclose a constants-free CQ p over S on any instance of S, *iff* $\vec{*} \notin p(J)$, *where* $J = I_{\mathsf{S}}(\Sigma_{st})$.

Proof. By adapting the proof technique of Theorem 16 from [5], we can show that $J = I_{\mathsf{S}}(\Sigma_{st})$ is a *universal* source instance $I_{\mathsf{S}}(\Sigma)$ satisfying the following property: for each pair of source instances I and I', such that I' is indistinguishable from I w.r.t. the mapping $\mathcal{M}$, there exists a homomorphism h from I' into $I_{\mathsf{S}}(\Sigma)$ mapping each schema constant into the critical constant $*$. Due to the existence of a homomorphism h from I' into $I_{\mathsf{S}}(\Sigma)$, for each pair of indistinguishable source instances I and I', we can see that if $\vec{*} \notin p(J)$ for a constants-free CQ p, then $p(I') = \emptyset$. Due to the above and due to Definition 4.1, it follows that $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ does not disclose a constants-free CQ p over S on any instance of S. □

Lemma 4.8 states that, in order to check if a constants-free CQ is safe according to Definition 4.1, we need to check if the critical tuple is among the answers to p over the instance computed by visChase$_{\mathsf{S}}(\Sigma)$. Next, we show the following lemma.

Lemma 4.9. *Given two instances* I_1 *and* I_2, *the following are equivalent*

1. *for each CQ* p, *if* $\vec{u} \in p(I_1)$, *then* $\vec{u} \in p(I_2)$, *where* $\vec{u}$ *is a vector of constants*
2. *there exists a homomorphism from* I_1 *to* I_2 *preserving the constants of* I_1

Proof of Lemma 4.9. (2)⟹(1). Suppose that there exists a homomorphism h from I_1 to I_2 preserving the constants of I_1. Suppose also that $\vec{u} \in p(I_1)$, with p being a CQ. This means that there exists a homomorphism h_1 from p into I_1 mapping each free variable x_i of p into u_i, for each $1 \leq i \leq n$, where n is the number of free variables of p. Since the composition of two homomorphisms is a homomorphism and since h preserves the constants of I_1 due to the base assumptions, this means that $h \circ h_1$ is a homomorphism from p into I_2 mapping each free variable x_i of p into t_i, for each $1 \leq i \leq n$. This completes this part of the proof.

(1)⟹(2). Let p_1 be a CQ formed by creating a non-ground atom $R(y_1, \ldots, y_n)$ for each ground atom $R(u_1, \ldots, u_n) \in I_1$, by taking the conjunction of these non-ground atoms and by converting into an existentially quantified variable every variable created out of some labelled null. Let $\vec{x}$ denote the free variables of p_1 and let $n = |\vec{x}|$. From the above, it follows that there exists a homomorphism h_1 from p_1 into I_1 mapping each $x_i \in \vec{x}$ into some constant occurring in I_1. Let $\vec{u} \in p_1(I_1)$. From (1), it follows that $\vec{u} \in p_1(I_2)$ and, hence, there exists a homomorphism h_2 from p_1 into I_2 mapping each $x_i \in \vec{x}$ into u_i, for each $1 \leq i \leq n$. Since h_1 ranges over all constants of I_1 and since $h_1(x_i) = h_2(x_i)$ holds for each $1 \leq i \leq n$, it follows that there exists a homomorphism from I_1 to I_2 preserving the constants of I_1. This completes the second part of the proof. □

Lemma 4.9 can be restated as follows:

Lemma 4.10. *Given two instances* I_1 *and* I_2, *the following are equivalent*

1. *for each CQ* p, *if* $\vec{t} \notin p(I_2)$, *then* $\vec{t} \notin p(I_1)$
2. *there exists a homomorphism from* I_1 *to* I_2

We are now ready to return to the main part of the proof. Given a CQ p over a source schema S, and a mapping $\mathcal{M}$ defined as the triple $(\mathsf{S}, \mathsf{T}, \Sigma)$, where T is a target schema and Σ is a set of s-t dependencies, we know from Proposition 4.8 that if $\mathcal{M}$ discloses p on some instance of S, then there exists a homomorphism of p into visChase$_{\mathsf{S}}(\Sigma)$ mapping the free variables of p into the critical constant $*$.

From the above, we know that $\mathcal{M}_2$ does not preserve the privacy of $\mathcal{M}_1$ if there exists a CQ p over S, such that $\vec{*} \notin J_1$ and $\vec{*} \in J_2$, where $J_1 = I_{\mathsf{S}}(\Sigma_1)$ and $J_2 = I_{\mathsf{S}}(\Sigma_2)$. We will now prove that $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ iff there exists a homomorphism from J_2 into J_1 that preserves the critical constant $*$. This will be referred to as conjecture (C).

($\Rightarrow$) If $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$, then for each CQ p, if $\vec{*} \notin p(J_1)$, then $\vec{*} \notin p(J_2)$. From the above and from Lemma 4.10, it follows that there exists a homomorphism $\phi : J_2 \to J_1$, such that $\phi(*) = *$.

($\Leftarrow$) The proof proceeds by contradiction. Assume that there exists a homomorphism h from J_2 into J_1 preserving $*$, but $\mathcal{M}_2$ does not preserve the privacy of $\mathcal{M}_1$. We will refer to this assumption as assumption (A_1). From assumption (A_1) and the discussion above it follows that there exists a CQ p over S such that $\vec{*} \notin p(J_1)$ and $\vec{*} \in p(J_2)$. Let h_2 be the homomorphism from p into J_2 mapping its free variables into $*$. Since the composition of two homomorphisms is a homomorphism, this means that $h \circ h_2$ is a homomorphism from p into J_1 mapping its free variables into $*$, i.e., $\vec{*} \in p(J_1)$. This contradicts our original assumption and hence concludes the proof of conjecture (C). Conjecture (C) witnesses the decidability of the instance-independent privacy preservation problem: in order to verify whether $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ we only need to check if there exists a homomorphism $\phi : I_{\mathsf{S}}(\Sigma_2) \to I_{\mathsf{S}}(\Sigma_1)$, such that $\phi(*) = *$. □

Theorem 4.7 states that in order to verify whether $\mathcal{M}_2$ is safe w.r.t. $\mathcal{M}_1$, we need to compute $I_{\mathsf{S}}(\Sigma_1)$ and $I_{\mathsf{S}}(\Sigma_2)$ and check if there exists a homomorphism from the second instance into the first one that maps $*$ into itself. If there exists such a homomorphism, we say that $I_{\mathsf{S}}(\Sigma_1)$ is *safe* w.r.t. $I_{\mathsf{S}}(\Sigma_2)$, or simply safe, and we say that it is unsafe otherwise.

Example 4.11. Continuing with Example 1.1, we can see that the s-t tgds are not safe w.r.t. the policy views according to Theorem 4.7, since there does not exist a homomorphism from the instance $I_{\mathsf{S}}(\Sigma_{st})$ into the instance $I_{\mathsf{S}}(\mathcal{V})$. This means that there exists information which is disclosed by Σ_{st} in some instance that satisfies Σ_{st}, but it is not disclosed by $\mathcal{V}$. Indeed, from $\mathsf{S}(\mathsf{n}_{\mathsf{i}}'', \mathsf{n}_{\mathsf{n}}'', *, \mathsf{n}_{\mathsf{c}}')$ and $\mathsf{O}(\mathsf{n}_{\mathsf{i}}'', \mathsf{n}_{\mathsf{t}}'', \mathsf{n}_{\mathsf{p}}'')$, we can see that we can potentially leak the identity of a student who has been to an oncology department. This can happen if there exists only one student in the school coming from a specific ethnicity group and this ethnicity group is returned by μ_s. Please note that the policy views are safe w.r.t. this leak. Indeed, it is impossible to derive this information through reasoning over the returned tuples under the input instance and the views V_3 and V_4.

Furthermore, by looking at the facts $\mathsf{P}(\mathsf{n}_{\mathsf{i}}, \mathsf{n}_{\mathsf{n}}, *, *)$ and $\mathsf{H}_{\mathsf{N}}(\mathsf{n}_{\mathsf{i}}, *)$, we can see that we can potentially leak the identity and the disease of a patient who has been admitted to some hospital in the north of UK. This can happen if there exists only one patient who relates to the county and the ethnicity group returned by μ_e and μ_c. Note that the policy views V_1 and V_2 do not leak this information, since it is impossible to obtain the county and the ethnicity group of an NHS patient at the same time.

5 REPAIRING UNSAFE MAPPINGS

In Section 4, we presented our privacy preservation protocol and a technique for verifying whether a mapping is safe w.r.t. another one, over all source instances. This section presents an algorithm for repairing an unsafe mapping $\mathcal{M}$ w.r.t. a set of policy views $\mathcal{V}$. This is a fundamental operation needed to amend mappings whenever the policy views are modified and become unsafe (e.g. in the presence of data protection regulations).

Algorithm 2 $\mathsf{repair}(\Sigma, \mathcal{V}, \mathsf{prf}, n)$

```
$\Sigma_1 := \mathsf{frepair}(\Sigma, \mathcal{V}, \mathsf{prf})$
$\Sigma_2 := \mathsf{srepair}(\Sigma_1, \mathcal{V}, \mathsf{prf}, n)$
**return** $\Sigma_2$
```

Algorithm 2 summarizes the steps of the proposed algorithm. The inputs to it are, apart from Σ and $\mathcal{V}$, a positive integer n which will be used during the second step of the repairing process and a preference mechanism prf for ranking the possible repairs. In the simplest scenario, the preference mechanism implements a fixed function for ranking the different repairs. However, it can also employ supervised learning techniques in order to progressively learn the user preferences by looking at his prior decisions.

Since a mapping $\mathcal{M}$ is safe w.r.t. $\mathcal{V}$ if the instance $I_{\mathsf{S}}(\Sigma)$ is safe according to Theorem 4.7, Algorithm 2 rewrites the tgds in $\mathcal{M}$, such that the derived visible chase instances are safe. The rewriting takes place in two steps. The first step rewrites Σ into a *partially-safe* set of s-t dependencies Σ_1, while the second step rewrites the output of the first one into a new set of s-t dependencies Σ_2, such that $I_{\mathsf{S}}(\Sigma_2)$ is safe. As we will explain later on, partial-safety ensures that the intermediate instance I_1 produced by $\mathsf{visChase}_{\mathsf{S}}(\Sigma_1)$ at line 2 of Algorithm 1 is safe, but it does not provide strong privacy guarantees. The benefit of this two-step approach is that it allows repairing one or a small set of dependencies at a time.

5.1 Computing partially-safe mappings

Since the problem of safety is reduced to the problem of checking for a homomorphism from $I_{\mathsf{S}}(\Sigma)$ into $I_{\mathsf{S}}(\mathcal{V})$, a first test towards checking for such a homomorphism is to look if the mappings in Σ would lead to such a homomorphism or not. For instance, by looking at μ_s in Example 1.1 it is easy to see that it leaks sensitive information, since it involves a join between students and oncology departments, which does not occur in $I_{\mathsf{S}}(\mathcal{V})$.

Definition 5.1. A mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ is *partially-safe w.r.t.* $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ *on all instances of* S, if there exists a homomorphism from $\mathsf{chase}(\Sigma^{-1}, \mathsf{Crt}_{\mathsf{T}}) \setminus \mathsf{Crt}_{\mathsf{T}}$ into $I_{\mathsf{S}}(\mathcal{V})$.

From Algorithm 1, it follows that Σ is partially-safe iff the intermediate instance I_1 computed by $\mathsf{visChase}_{\mathsf{S}}(\Sigma)$ is safe.

Proposition 5.2. *A mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ *is* partially-safe w.r.t. $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ on all instances of S, *if for each* $\mu \in \Sigma$, *there exists a homomorphism from* $\mathsf{body}(\mu)$ *into* $I_{\mathsf{S}}(\mathcal{V})$ *mapping each* $x \in \mathsf{exported}(\mu)$ *into the critical constant* $*$.

Note that according to Proposition 5.2, in our running example Σ_{st} would be partially-safe, if $\mu_s \notin \Sigma_{st}$, then since there exist homomorphisms from the bodies of μ_s and μ_c into $I_{\mathsf{S}}(\mathcal{V})$, mapping their exported variables into $*$. It is also easy to show the following

Remark 1. *A mapping* $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ *is* safe w.r.t. $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ on all instances of S, *only if it is partially-safe w.r.t.* $\mathcal{M}_{\mathsf{V}}$ *on all instances of* S. □

Proposition 5.2 presents a quite convenient, yet somewhat expected, finding: in order to obtain a partially-safe mapping, it suffices to repair each s-t dependency *independently of the others.* Furthermore, the repair of each $\mu \in \Sigma$ involves breaking joins and hiding exported variables, such that the repaired dependency μ_r satisfies the criterion in Proposition 5.2.

We make use of the result of Proposition 5.2 in Algorithm 3. Algorithm 3 obtains, for each $\mu \in \Sigma$, a set of rewritings $\mathcal{R}_\mu$, out of which we will choose the best rewriting according to prf. The set $\mathcal{R}_\mu$ consists of *all* rewritings that differ from μ w.r.t. the variable repetitions in the bodies of the rules and the exported variables. Below, we present the steps of Algorithm 3.

For each s-t tgd μ and for each atom $B \in \mathsf{body}(\mu)$, Algorithm 3 constructs a fresh atom C and adds C to a set $\mathcal{C}$. The set of atoms $\mathcal{C}$ provides us with the means to identify all repairs of μ that involve breaking joins and hiding exported variables. In particular, each homomorphism ξ from $\mathcal{C}$ into $I_\mathsf{S}(\mathcal{V})$ corresponds to one repair of μ. In lines 12–25, Algorithm 3 modifies each atom $B \in \mathsf{body}(\mu)$ by taking into account prior body atom modifications. The prior modifications are accumulated in the relation ρ and the mapping ψ. The relation ρ keeps for each variable x from body(μ), the fresh variables that were used to replace x during prior steps of the repairing process, while ψ is a substitution from the partially repaired body into $I_\mathsf{S}(\mathcal{V})$. In particular, at the end of the i-th iteration of the loop in line 12, ψ holds the substitution from the first repaired i atoms from body(μ) into $I_\mathsf{S}(\mathcal{V})$. We adopt this approach instead of replacing variable x in position p always by a fresh variable, in order to minimize the number of the joins we break.

Below, we describe how Algorithm 3 modifies each body atom of μ, w.r.t. a homomorphism ξ, lines 9–27. Let $C = \nu(B)$ be the fresh body atom that was constructed out of B in line 5. For each atom $B \in \mathsf{body}(\mu)$ and for each $p \in \mathsf{pos}(B)$, if the variable y in position p of C is not mapped to the critical constant $*$ via ξ and $B|_p$ is an exported variable, this means that *the variable sitting in position p of B should not be exported* (see first condition in line 16). Similarly, if the variable sitting in position p of B is mapped to a different constant than the one that y maps via ξ, then this means that *the variable sitting in position p of B introduces an unsafe join* (see second condition in line 16). In the presence of these violations, we must replace variable x in position p of B, either by a variable that was used in a prior step of the repairing process, line 17–18), or by a fresh variable, lines 19–23. Otherwise, if there is no violation so far, then we add the mapping $\{x \mapsto \xi(y)\}$ to ψ, if it is not already there, lines 24–25. Finally, the algorithm chooses the best repair according to the preference function, lines 28–31.

Proposition 5.3. *For any $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, any $\mathcal{M}_\mathsf{V} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ and any preference function* prf, *Algorithm* frepair *returns a mapping $\mathcal{M}' = (\mathsf{S}, \mathsf{T}, \Sigma')$ that is partially-safe w.r.t. $\mathcal{M}_\mathsf{V}$ on all instances of* S.

Proof. (Sketch) From Proposition 5.2, a mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$ is partially-safe w.r.t. $\mathcal{M}_\mathsf{V} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ on all instances of S, if for each $\mu \in \Sigma$, there exists a homomorphism from body(μ) into $I_\mathsf{S}(\mathcal{V})$ mapping each $x \in \mathsf{exported}(\mu)$ into the critical constant $*$. Since for each $\mu \in \Sigma$ frepair computes a set of repaired tgds $\mathcal{R}_\mu$, it follows that Proposition 5.3 holds, if such a homomorphism exists, for each repaired tgd in $\mathcal{R}_\mu$. The proof proceeds as follows. Let μ_r^i and ψ^i denote the repaired s-t tgd and the homomorphism ψ computed at the end of each iteration i of the steps in lines 12–25 of Algorithm 3. Let also B^i denote the i-th atom in body(μ_r). Since each $C \in \mathcal{C}$ is an atom of distinct fresh variables, since ξ is a homomorphism from $\mathcal{C}$ to $I_\mathsf{S}(\mathcal{V})$ and since $\psi(B^i) = \mu_r|_i$, it follows that in order to prove Proposition 5.2, we have to show that the following claim holds, for each $i \geq 0$:

Algorithm 3 frepair($\Sigma, \mathcal{V}$, prf)

```
1:  for each $\mu \in \Sigma$ do
2:      $\nu := \emptyset, \mathcal{C} := \emptyset$
3:      for each $B \in \mathsf{body}(\mu)$, where $B = R(\vec{x})$ do
4:          create a vector of fresh variables $\vec{y}$
5:          create the atom $C = R(\vec{y})$
6:          add $(B, C)$ to $\nu$
7:          add $C$ to $\mathcal{C}$
8:      $\mathcal{R}_\mu := \emptyset$
9:      for each homomorphism $\xi : \mathcal{C} \to I_\mathsf{S}(\mathcal{V})$ do
10:         $\rho := \emptyset, \psi := \emptyset$
11:         $\mu_r := \mu$
12:         for each $B \in \mathsf{body}(\mu_r)$ do
13:             $C = \nu(B)$
14:             for each $p \in \mathsf{pos}(B)$ do
15:                 $x = B|_p, y = C|_p$
16:                 if $x \in \mathsf{exported}(\mu)$ and $* \neq \xi(y)$ or $x \in \mathsf{dom}(\psi)$ and $\psi(x) \neq \xi(y)$ then
17:                     if $\exists x'$ s.t. $(x, x') \in \rho$ and $\psi(x') = \xi(y)$ then
18:                         $B|_p = x'$
19:                     else
20:                         create a fresh variable $x'$
21:                         add $(x, x')$ to $\rho$
22:                         add $\{x' \mapsto \xi(y)\}$ to $\psi$
23:                         $B|_p = x'$
24:                 else if $x \notin \mathsf{dom}(\psi)$ then
25:                     add $\{x \mapsto \xi(y)\}$ to $\psi$
26:         if $\mu_r \neq \mu$ then
27:             add $\mu_r$ to $\mathcal{R}_\mu$
28:     if $\mathcal{R}_\mu \neq \emptyset$ then f
29:         choose the best repair $\mu_r$ of $\mu$ from $\mathcal{R}_\mu$ based on prf
30:         remove $\mu$ from $\Sigma$
31:         add $\mu_r$ to $\Sigma$
32: return $\Sigma$
```

- ϕ. ψ^i is a homomorphism from the first i atoms in the body of μ_r into $I_\mathsf{S}(\mathcal{V})$ mapping each exported variable occurring in $B^0, \ldots, B^i$ into the critical constant $*$.

For $i = 0$, ϕ trivially holds. For $i + 1$ and assuming that ϕ holds for i let $C^{i+1} = \nu(B^{i+1})$, line 13. The proof of claim ϕ depends upon the proof of the following claim, for each iteration $p \geq 0$ of the steps in lines 14–25:

- θ. $\psi^{i+1}(B^{i+1}|_p) = \xi(y)$, where $y = C^{i+1}|_p$.

The claim θ trivially holds for $p = 0$, while for $p > 0$, it directly follows from the steps in lines 16–25. Since ϕ holds for i, since the steps in lines 16–25 do not modify the variable mappings in ψ^i and due to θ, it follows that ϕ holds for $i + 1$, concluding the proof of Proposition 5.3. □

Example 5.4. We demonstrate an example of Algorithm 3.

Since Algorithm 3 focuses on $I_\mathsf{S}(\mathcal{V})$ overlooking the actual views in $\mathcal{V}$, we will not explicitly define $\mathcal{V}$. Instead, we will only assume that the visible chase computes the instance

$$I_\mathsf{S}(\mathcal{V}) = \{\mathsf{R}_1(*, \mathsf{n}_1, \mathsf{n}_2), \mathsf{S}_1(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_2), \mathsf{S}_1(\mathsf{n}_1, \mathsf{n}_3, *), \mathsf{S}_1(\mathsf{n}_1, *, *)\}$$

where n_1–n_3 are labeled nulls. Consider also the mapping $\mathcal{M}$ consisting of the following s-t dependency

$$\mathsf{R}_1(x, y, z) \wedge \mathsf{S}_1(y, z, z) \to \mathsf{T}_1(x, z) \qquad (\mu_1)$$

Note that $\mathcal{M}$ is not partially-safe. Algorithm 3 computes two repairs for μ_1 by applying the steps described below. First, it computes the atoms $\mathsf{R}_1(x_1, x_2, x_3)$ $\mathsf{S}_1(x_4, x_5, x_6)$ and adds them to C, lines 3–7. Then, it identifies the following three homomorphisms from C into $I_\mathsf{S}(\mathcal{V})$:

$$\xi_1 = \{x_1 \mapsto *, x_2 \mapsto \mathsf{n}_1, x_3 \mapsto \mathsf{n}_2, x_4 \mapsto \mathsf{n}_1, x_5 \mapsto \mathsf{n}_2, x_6 \mapsto \mathsf{n}_2\}$$
$$\xi_2 = \{x_1 \mapsto *, x_2 \mapsto \mathsf{n}_1, x_3 \mapsto \mathsf{n}_2, x_4 \mapsto \mathsf{n}_1, x_5 \mapsto \mathsf{n}_3, x_6 \mapsto *\}$$
$$\xi_3 = \{x_1 \mapsto *, x_2 \mapsto \mathsf{n}_1, x_3 \mapsto \mathsf{n}_2, x_4 \mapsto \mathsf{n}_1, x_5 \mapsto *, x_6 \mapsto *\}$$

From ξ_1, we can see that the joins in the body of μ_1 are safe; however, it is unsafe to export z. From ξ_2, we can see that is safe to reveal the third position of S_1; however, it is unsafe to join the second and the third position of S_1. Algorithm 3 then iterates over ξ_1 and ξ_2, line 9. When $B = \mathsf{R}_1(x, y, z)$ and $p < 3$, Algorithm 3 computes ψ to $\{x \mapsto *, y \mapsto \mathsf{n}_1\}$, since there is no violation according to line 16. When $B = \mathsf{R}_1(x, y, z)$ and $p = 3$, however, a violation is detected. This is due to the fact that z is an exported variable and $\xi(x_3) = \mathsf{n}_2$. Algorithm 3 tackles this violation by creating a fresh variable z_1. Then, it adds the relation (z, z_1) to ρ, replaces z in $B|_3$ by z_1 and adds the mapping $\{z_1 \mapsto \mathsf{n}_2\}$ to ψ, lines 19–23. Algorithm 3 then considers $\mathsf{S}_1(y, z, z)$. When $p = 1$, no violation is encountered, since $\psi(y) = \xi_1(x_4)$. However, when $p = 2$, a homomorphism violation is encountered, since z is an exported variable and since $\xi(x_3) = \mathsf{n}_2$. Since $(z, z_1) \in \rho$ and $\psi(z_1) = \xi_1(x_5)$, Algorithm 3 replaces z in the second position of $\mathsf{S}_1(y, z, z)$ by z_1, line 19. By applying a similar reasoning, we can see that the variable z siting in $\mathsf{S}_1(y, z, z)|_3$ is also replaced by z_1. Hence, the first repair of μ is

$$\mathsf{R}_1(x, y, z_1) \wedge \mathsf{S}_1(y, z_1, z_1) \rightarrow \mathsf{T}_1(x) \qquad (r_1)$$

Algorithm 3, then proceeds by repairing μ_1 based on ξ_2. When $B = \mathsf{R}_1(x, y, z)$, Algorithm 3 proceeds as described above and computes ψ to $\{x \mapsto *, y \mapsto \mathsf{n}_1, z_1 \mapsto \mathsf{n}_2\}$. When $B = \mathsf{S}_1(y, z, z)$ and $p = 1$, then no violation is encountered since $\psi(y) = \xi_1(x_4)$, while when $B = \mathsf{S}_1(y, z, z)$ and $p = 2$, there is a violation. Since the condition in line 18 is not met, Algorithm 3 creates a fresh variable z_2 and adds the mapping $\{z_2 \mapsto \mathsf{n}_3\}$ to ψ. When $B = \mathsf{S}_1(y, z, z)$ and $p = 3$, then no violation is met, since $z \in \mathsf{exported}(\mu)$ and $\xi_2(x_6) = *$. Hence, the second repair of μ_1 is

$$\mathsf{R}_1(x, y, z_1) \wedge \mathsf{S}_1(y, z_2, z) \rightarrow \mathsf{T}_1(x, z) \qquad (r_2)$$

Finally, we can see that the repair for μ_1 w.r.t. ξ_3 is

$$\mathsf{R}_1(x, y, z_1) \wedge \mathsf{S}_1(y, z, z) \rightarrow \mathsf{T}_1(x, z) \qquad (r_3)$$

5.2 Computing safe mappings

Unifications of one or more labeled nulls occurring in I_1 with the critical constant $*$, might lead to unsafe instances. Consider, for instance, a simplified variant of Example 1.1, where Σ_{st} comprises only μ_e and μ_c. Both μ_e and μ_c are partially-safe, as we have explained above. However, the unification of the labeled nulls n_n and n_c produces an unsafe instance. Algorithm 4 aims at repairing the output of the previous step, such that no unsafe unification of a labeled null with $*$ takes place.

Consider again the simplified variant of Σ_{st} from above. Since Σ_{st} is partially-safe, it suffices to look for homomorphism violations in I_i, for $i \geq 1$. A first observation is that the homomorphism violations are "sitting" within the bags. This is due to the fact that each bag stores *all* the facts associated with the bodies of one or more s-t tgds from Σ_{st}. A second observation is that one way for preventing unsafe unifications is to hide exported variables. For example, let us focus on the unsafe unification of n_e with $*$. This unification takes place due to ϵ_1, which in turn has been created due to the fact that e is an exported variable in μ_e. By hiding the exported variable e from μ_e, we actually prevent the creation of ϵ_1 and hence, we block the unsafe unification of e with $*$. Hiding exported variables is one way for preventing unsafe unifications with the critical constant. Another way for preventing unsafe unifications is to break joins in the bodies of the rules.

Example 5.5. This example demonstrates a second approach for preventing unsafe labeled null unifications.

Consider a set of policy views $\mathcal{V}$ leading to the following instance $I_\mathsf{S}(\mathcal{V}) = \{\mathsf{R}_1(\mathsf{n}_1, \mathsf{n}_1, *), \mathsf{R}_1(*, *, \mathsf{n}_2), \mathsf{S}_1(*)\}$, where n_1 and n_2 are labelled nulls. Consider also the mapping $\mathcal{M}$ consisting of the following s-t dependencies:

$$\mathsf{R}_1(x, x, y) \wedge \mathsf{S}_1(y) \rightarrow \mathsf{T}_1(y) \qquad (\mu_2)$$
$$\mathsf{R}_1(x, x, y) \rightarrow \mathsf{T}_2(x) \qquad (\mu_3)$$

It is easy to see that $\mathcal{M}$ is partially-safe, but unsafe in overall. Indeed, $I_\mathsf{S}(\Sigma)$ will consist of the following bags (for presentation purposes, we adopt the notation from Example 4.5):

$$\mathsf{T}_1(*) \xrightarrow{\langle \mu_2^{-1}, \theta_1 \rangle} \mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_3, *), \mathsf{S}_1(*)$$
$$\mathsf{T}_2(*) \xrightarrow{\langle \mu_3^{-1}, \theta_2 \rangle} \mathsf{R}_1(*, *, \mathsf{n}_4)$$
$$\mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_3, *), \mathsf{S}_1(*) \xrightarrow{\langle \epsilon_3, \theta_3 \rangle} \mathsf{R}_1(*, *, *), \mathsf{S}_1(*)$$

where $\epsilon_3 := \mathsf{R}_1(x, x, y) \rightarrow x = *$, $\theta_1 = \{y \mapsto *\}$, $\theta_2 = \{x \mapsto *\}$ and $\theta_3 = \{x \mapsto \mathsf{n}_3, y \mapsto *\}$. Note that ϵ_3 has been created out of μ_3, since there exists a homomorphism from $\mathsf{body}(\mu_3)$ into $\mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_3, *)$ mapping the exported variable x into n_3.

One approach for preventing the unsafe unification of n_3 with $*$ is to hide the exported variable x from μ_3. By doing this, we block the creation of ε, and hence the unsafe unification.

A second approach is to keep x as an exported variable in μ_3, but modify the body of μ_2 by breaking the join between the first and the second position of R_1

$$\mathsf{R}_1(x, z, y) \wedge \mathsf{S}_1(y) \rightarrow \mathsf{T}_1(y) \qquad (\mu_2')$$

By doing this, we prevent the creation of ε, since the instance computed at line 2 of Algorithm 1 would consist of the facts $\mathsf{R}_1(\mathsf{n}_3, \mathsf{n}_5, *)$, $\mathsf{R}_1(*, *, \mathsf{n}_4)$, $\mathsf{S}_1(*)$ and, hence, there would be no homomorphism from $\mathsf{body}(\mu_3)$ into it. Note that the modification of μ_2 to μ_2' is safe. Intuitively, this holds, since we break joins, and thus, we export less information.

Before presenting Algorithm 4, we will introduce some new notation. The *depth* of each bag β, denoted as $\mathsf{depth}(\beta)$, coincides with the highest derivation depth of the facts in β. The *support* of a bag β, denoted as $\beta^{\prec}$, is inductively defined as follows: if $\mathsf{depth}(\beta) = 1$, then $\beta^{\prec} = \beta$; otherwise, if $\mathsf{depth}(\beta) > 1$, then $\cup_{\beta' \prec \beta} \beta'^{\prec}$. Consider an active trigger h for δ in I leading to the creation of a bag β. We use the following notation: $\mathsf{dependency}(\beta) = \delta$, $\mathsf{trigger}(\beta) = h$ and $\mathsf{premise}(\beta) = h(\mathsf{body}(\delta))$. Two bags β_1 and β_2 are candidates for modifyBody if $\beta_1 \prec \beta_2$, $\mathsf{depth}(\beta_1) = 1$, $\mathsf{depth}(\beta_2) = 2$ and there exists at least one repeated variable in the body of $\mathsf{tgd}(\beta_1)$.

Algorithm 4 presents an iterative process for repairing a partially-safe Σ, by employing the three ideas we described above: checking for homomorphism violations within each bag and preventing unsafe unifications either by hiding exported variable, or by modifying the bodies of the s-t tgds. In brief, at each iteration $i \geq 0$, the algorithm repairs one or more dependencies from Σ_i, where $\Sigma_0 = \Sigma$, and incrementally computes the visible chase of the new

Algorithm 4 srepair($\Sigma, \mathcal{V}$, prf, n)

```
1: $\Sigma_0 := \Sigma$
2: $B_0 :=$ visChases$_{\mathsf{S}}(\Sigma)$
3: $i := 0$
4: do
5:     $\Sigma_{i+1} := \Sigma_i$
6:     cont := false
7:     if $\exists$ unsafe $\beta \in B_i$, s.t. depth($\beta$) $\leq$ depth($\beta'$), $\forall$ unsafe bag $\beta' \in B_i$ then
8:         cont := true
9:         if $i < n$ then
10:            $r_1 := \emptyset$; $r_2 :=$ hideExported($\beta, \mathcal{V}$, prf)
11:            if $\exists \beta_1, \beta_2 \in \beta^{\prec}$, s.t. $\beta_1, \beta_2$ are candidates for modifyBody then
12:                $r_1 :=$ modifyBody(tgd($\beta_1$), tgd($\beta_2$), prf)
13:            if $r_1 \neq \emptyset$ and it is preferred over $r_2$ w.r.t. prf then
14:                remove tgd($\beta_1$) from $\Sigma_{i+1}$
15:                add $r_1$ to $\Sigma_{i+1}$
16:            else
17:                remove tgd($\beta$) from $\Sigma_{i+1}$
18:                add $r_2$ to $\Sigma_{i+1}$
19:        else
20:            if $\nexists \beta'$, s.t., $\beta \prec \beta' \in B_i$ then
21:                add hideExported($\beta, \mathcal{V}$, prf) to $\Sigma_{i+1}$
22:            else remove tgd($\beta$) from $\Sigma_{i+1}$
23:    compute $J_{i+1}$ from $\Sigma_i$, $\Sigma_{i+1}$ and $B_i$
24:    $i = i + 1$
25: while cont and $i \leq n$
26: return $\Sigma_n$
```

set of dependencies, lines 4–25. Algorithm 4 terminates either when the dependencies are safe, or when the maximum number of iterations n is reached, line 25, in which case it repairs all unsafe dependencies by hiding their exported variables. The algorithm starts by initializing Σ_0 to Σ, lines 1. Then, at each iteration i, it first identifies the lowest depth unsafe bag, line 7, and attempts to repair the dependencies from Σ_i that lead to its creation, lines 7–22. If $i < n$, it proposes two different repairs for Σ_i, one based on hiding exported variables through hideExported (Algorithm 5), and the second based on eliminating joins through modifyBody (Algorithm 6), lines 10–19. Algorithm 4 applies the modifyBody if there exist two bags in the support of β that are candidates for modifyBody. Informally, Algorithm 4 tries to apply modifyBody as early as possible (condition depth(β_1) = 1, depth(β_2) = 2) and when there are one or more repeated variables in the body of tgd(β_1) (recall Example 5.5). Otherwise, if $i = n$, it either applies the function hideExported, or it eliminates the s-t tgds that are responsible for unsafe unifications.

Note that when we reach the maximum number of iterations we do not apply modifyBody. This is due to the fact that modifyBody might lead to unsafe unification of labeled nulls to $*$ that were not taking place before the modifying the s-t tgd through modifyBody. In contrast, hideExported is a safe modification, since it does not lead to new unsafe unifications.

Theorem 5.6. *For any partially-safe $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, any $\mathcal{M}_{\mathsf{V}} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$, any preference function* prf *and $n \geq 0$, Algorithm* srepair *returns a mapping $\mathcal{M}' = (\mathsf{S}, \mathsf{T}, \Sigma')$ that* preserves the privacy of $\mathcal{M}_{\mathsf{V}}$ on all instances of S.

Proof. (Sketch) Since srepair takes as input a partially-safe mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, it follows from Definition 5.1 that there

Algorithm 5 hideExported($\beta, \mathcal{V}$, prf)

```
1: $J :=$ premise($\beta$)
2: $\nu := \emptyset$
3: for each $n \in$ Nulls occurring into $J$ do
4:     add $\{n \mapsto x\}$ to $\nu$, where $x$ is a fresh variable
5: $\mathcal{R} := \emptyset$
6: for each $\xi : \nu(J) \to I_{\mathsf{S}}(\mathcal{V})$ do
7:     $\mu :=$ tgd($\beta$)
8:     for each $x \in$ dom($\xi$) do
9:         if $\xi(x) \neq *$ then
10:            for each $y \in$ exported($\mu$) do
11:                if $\tau(y) = \nu^{-1}(x)$, where $\tau =$ trigger($\beta$) then
12:                    remove $y$ from exported($\mu$)
13:    if $\mu \neq$ tgd($\beta$) then
14:        add $\mu$ to $\mathcal{R}$
15: choose the best repair $\mu_r$ of $\mu$ from $\mathcal{R}$ based on prf
16: return $\mu_r$
```

Algorithm 6 modifyBody(μ_1, μ_2, prf)

```
1: $\mathcal{R} := \emptyset$
2: if $\exists$ one or more repeated variables in body($\mu_1$) then
3:     for each $\xi$ : body($\mu_2$) $\to$ body($\mu_1$) mapping some $x_1 \in$ exported($\mu_1$)
               into some $x_2 \notin$ exported($\mu_2$) do
4:         Let $B \subseteq$ body($\mu_1$), s.t. $\xi$(body($\mu_2$)) = $B$
5:         Let $V$ be the set of repeated variables from $B$
6:         Let $P$ be the set of positions from $B$, where all variables from $V$ occur
7:         for each non-empty $S \subset P$ do
8:             $\mu := \mu_1$
9:             replace the variables in positions $S$ of $\mu$ by fresh variables
10:            add $\mu$ to $\mathcal{R}$
11: choose the best repair $\mu_r$ of $\mu$ from $\mathcal{R}$ based on prf
12: return $\mu_r$
```

exists a homomorphism from chase(Σ^{-1}, Crt$_{\mathsf{T}}$) $\setminus$ Crt$_{\mathsf{T}}$ into $I_{\mathsf{S}}(\mathcal{V})$. Furthermore, from Proposition 5.2, we know that for each $\mu \in \Sigma$, there exists a homomorphism from body(μ) into $I_{\mathsf{S}}(\mathcal{V})$ mapping each $x \in$ exported(μ) into the critical constant $*$. Due to the above, since the steps in lines 16–20 of Algorithm 1 do not introduce new labeled nulls and since srepair applies the procedure hideExported to each unsafe bag β in B_n, if there does not exist a bag $\beta' \in B_n$, such that $\beta \prec \beta'$, it follows that $\mathcal{M}'$ preserves the privacy of $\mathcal{M}_{\mathsf{V}}$ on all instances of S, if hideExported prevents dangerous unifications of labeled nulls with the critical constant in line 4 of Algorithm 1. In particular, assume that we are in the n-th iteration of the steps in lines 4–25 of Algorithm 4. Let $\beta_n^0, \ldots, \beta_n^M$ be the unsafe bags in B_n. Assume also that for each $1 \leq l \leq M$, β_n^l, was derived due to some active trigger h^l, for some derived egd $\varepsilon^l \in \Sigma_{\approx}$ in I_j, where $j \geq 0$, line 17 of Algorithm 1. Let $\mu^l =$ tgd(ε^l), for each $0 \leq l \leq M$ and let μ_r^l be the repaired s-t tgd. Finally, let $\beta_{n+1}^0, \ldots, \beta_{n+1}^N$ be the bags in B_{n+1}, line 23 of Algorithm 4. Based on the above, in order to show that Theorem 5.6 holds, we need to show that (i) the number of bags in B_{n+1} is $\leq$ the number of bags in B_n and that (ii) the s-t tgds in $\left(\Sigma \setminus \bigcup_{l=0}^{M} \mu^l\right) \cup \bigcup_{l=0}^{M} \mu_r^l$ are safe. In order to show (i) and (ii), we consider the steps in Algorithm 5: for each $1 \leq l \leq M$, each exported variable y occurring in μ^l, which leads to an unsafe

	min	max	step
# s-t tgds per scenario (n_{dep})	100	300	50
# body atom per s-t tgds (n_{atoms})	1	3 (5)	–
# exported variables per s-t tgds (n_{vars})	5	8	–

Table 1: Properties of the generated iBench scenarios.

unification, line 11 of Algorithm 5, is turned into a non-exported variable. □

By combining Proposition 5.3 and Theorem 5.6 we can prove the correctness of Algorithm 2. Furthermore, if the preference function always prefers the repairs computed by hideExported from the repairs computed by modifyBody, we can show the following:

Proposition 5.7. *For each mapping $\mathcal{M} = (\mathsf{S}, \mathsf{T}, \Sigma)$, each $\mathcal{M}_\mathsf{V} = (\mathsf{S}, \mathsf{V}, \mathcal{V})$ and each preference function* prf *that always prefers the repairs computed by* hideExported *from the repairs computed by* modifyBody*, Algorithm 2 returns a non-empty mapping that is safe w.r.t. $\mathcal{M}_\mathsf{V}$, if such a mapping exists.*

Proof. (Sketch) From Algorithm 3, we can see that frepair always computes a non-empty partially-safe mapping, if such a mapping exists. Note that a mapping, where no variable is exported and no repeated variables occur in the body of the s-t tgds is always partially-safe as long as, the predicates in the bodies of the s-t tgds are the same with the ones occurring in the policy views. Please also note that such a mapping is always considered by frepair. The above argument, along with the fact that a partially-safe mapping can be transformed into a safe one by turning exported variables into non-exported ones by means of the function hideExported, shows that Proposition 5.7 holds. □

6 EXPERIMENTS

We gauge the efficiency of our repairing algorithm on two types of preference function: a hardcoded one and a learning-based preference function.

We evaluated our algorithm using a set of 3.6K diverse mapping scenarios each of which consisting of a set of policy views and a set of s-t tgds. The characteristics of the scenarios are summarized in Table 1. In each scenario, we used a different number of s-t tgds n_{dep}, a different number of body atoms n_{atoms} and a different number of exported variables n_{vars}. The source schemas and the policy views have been synthetically generated using iBench, the state-of-the-art data integration benchmark [2]. We considered relations of up to five attributes and we created mappings using the iBench configuration recommended by the authors of [2]. We generated a set of varied policy views by applying the iBench operators copy, merge, deletion of attributes and self-join, each of which has been applied 10 times.

We implemented our algorithm in Java and we used the Weka library [16] that provides an off-the-shelf implementation of the k-NN algorithm for the learning-based preference function. We ran our experiments on a laptop with one 2.6GHz 2-core processor, 16Gb of RAM, running Debian 9.

In the remainder, all data points have been computed as an average on a total of 5 runs preceded by one discarded cold run.

Running time of repair**.** First, we study the impact of the number of s-t tgds and body atoms on the running time of repair. We adopt a fixed preference function that chooses the repair with the maximum number of exported variables. In case of ties, the repair with the maximum number of joins is preferred. We vary the number of s-t tgds from 100 to 300 by steps of 50 and the number of body atoms from 3 to 5, respectively. The obtained results are shown in Figure (3a), illustrating the fact that the median repairing time is less than 1.5s in most cases. For the most complex scenario containing up to five body atoms per s-t tgd, the median running time is less than 8s with 71s being the maximum. These results clearly show the high performance of our repair method along with its scalability. The reader should keep in mind that the repairing process is triggered prior to exchanging the data between source and target and might be rerun each time a mapping (set of s-t tgds) is modified or each time a policy view is modified, thus bringing the overhead to be quite reasonable in both cases.

Figure (3b) shows the time breakdown for repair. The first bar shows the average running time to run the visible chase over the input s-t mappings, the second one shows the average running time for checking the safety of the computed bags and the third one shows the average running time for repairing the s-t tgds. The results show that the repairing time is $32x$ greater than the time to compute the visible chase and $40x$ greater than the time to check the safety of the chase bags for scenarios with 300 s-t tgds. In the simplest scenarios, these numbers are much lower (reduced to $5x$ and $9x$, respectively). Overall, the absolute values of the rewriting times are kept low (of the order of few seconds) for all these scenarios and gracefully scale while increasing the number of s-t tgds and the number of atoms in their bodies.

Time breakdown between frepair **and** srepair**.** Figure (3c) shows the average running time for frepair and srepair for the considered scenarios. We can see that srepair is the most time-consuming step of our algorithm. We can also see that the running time of srepair increases more in comparison to the running time of frepair when increasing the number of the s-t tgds and the number of atoms in their bodies. This is due to the incurred overhead during the incremental computation of the visible chase after repairing a s-t tgd (line 23 of Algorithm 4). Figure (3d) shows the correlation between the number of active triggers detected while incrementally computing the visible chase and the running time of srepair for scenarios with 100 s-t tgds using the ANOVA method (*p-value* $< 2.2e^{-16}$). Figure (3d) shows that the most complex scenarios lead to the detection of more than 45,000 active triggers. Despite the high number of the detected active triggers, the running time of srepair is kept low thus confirming its efficiency.

Leveraging learning-based preferences. We adopted the following steps in order to evaluate the performance of our learning approach. First, we defined the following two golden standard preference functions that we will try to learn:

- P_{max}, which chooses the repair with the maximum number of exported variables and in case of ties, it chooses the repair with the maximum number of joins.
- P_{avg}, which computes the average number of exported variables and joins for each repair, and choose the one with the maximum average value.

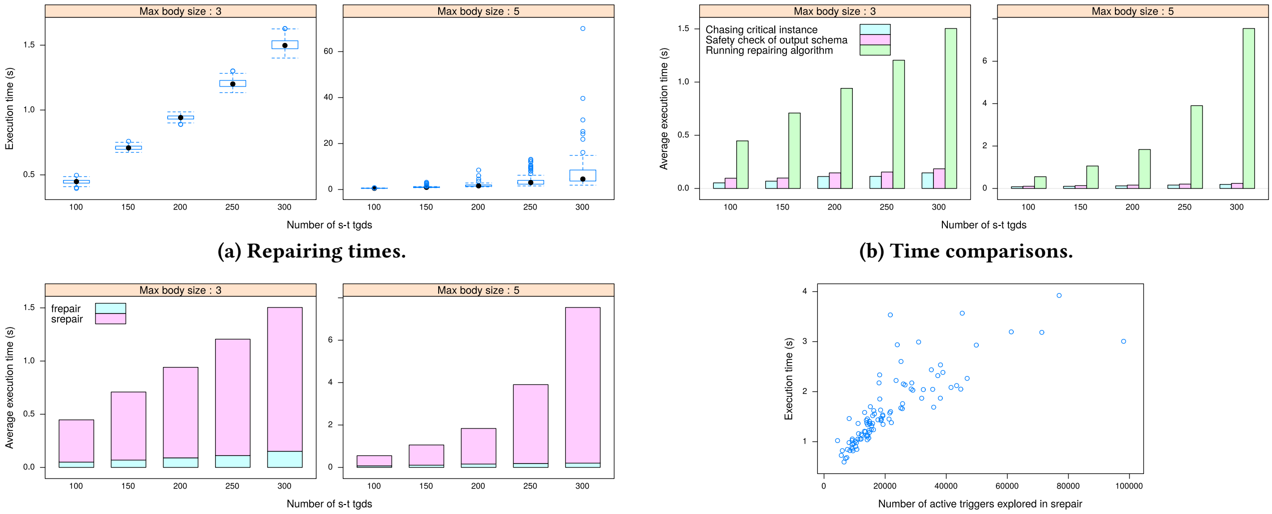
	golden standard	
prediction	μ_1	μ_2
μ_1	230	0
μ_2	0	395680

(a) P_{max} confusion matrix.

	golden standard	
prediction	μ_1	μ_2
μ_1	290	1
μ_2	42	395577

(b) P_{avg} confusion matrix.

Table 2: Confusion matrix for the golden standards.



(a) Repairing times.

(b) Time comparisons.

(c) Time breakdown between frepair **and** srepair**.**

(d) Running time of srepair **over 100 s-t tgds.**

Figure 3: Summary of the performance-related experimental results.

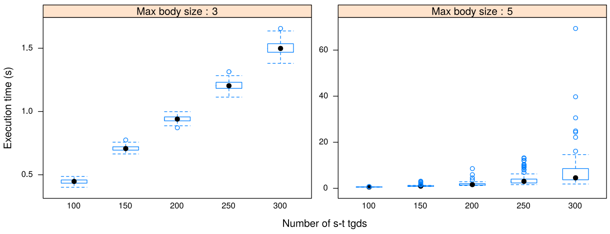


Figure 4: Repairing times with ML classifiers.

For both preference functions, we created a training set of 10, 000 measurements for the k-NN classifier by running the repairing algorithm on fresh scenarios of 50 s-t tgds and five body atoms per s-t tgd. For each input vector $\langle \delta_{FV}, \delta_J \rangle$ whose repair we wanted to predict, we computed the Euclidean distance between $\langle \delta_{FV}, \delta_J \rangle$ and the vectors of the training set. We also set the value of parameter k to 1. This parameter controls the number of neighbors used to predict the output. Higher values of this parameter led to comparable predictions and are omitted for space reasons. Finally, we used the trained k-NN classifier as a preference function in srepair, rerun the above scenarios and compared the returned repairs with the ones returned when applying the golden standards P_{max} and P_{avg} as preference functions.

Learning P_{max}. Table (2a) (left) reports the confusion matrix associated to learning P_{max}, including the choices made by the k-NN classifier during its iterations.

Let us call μ_1 and μ_2 two possible repairs of an s-t tgd as evaluated by the k-NN classifier. We can observe that the prediction of μ_1 was correct (and equal to the golden standard in the training set) in 230 cases, while the prediction of μ_2 was correct in 395,680 cases.

This confirms the fact that μ_2 is the best repair across the iterations of the k-NN algorithm and is also chosen in case μ_1 and μ_2 are equally weighed by the preference function.

Furthermore, we also report the accuracy of learning the preference function, obtained by measuring the closeness of the learned mapping to the golden standard mapping.

We used the Matthews Correlation Coefficient metric (MCC) [3] to compare the repairs returned by the trained k-NN classifier and the ones returned when applied P_{max}. This is a classical measure that allows to evaluate the quality of ML classifiers when ranking is computed between two possible values (in our case, the choice between μ_1 and μ_2). This measure has been computed using the following formula:

$$MCC = \frac{N_{1,1} \times N_{2,2} - N_{1,2} \times N_{2,1}}{\sqrt{(N_{1,1} + N_{1,2})(N_{1,1} + N_{2,1})(N_{2,2} + N_{1,2})(N_{2,2} + N_{2,1})}}$$

where $N_{i,j}$ is the number of predictions of μ_i when μ_j is expected. The results of MCC range from -1 for the cases where the model perfectly predicts the inverse of the expected values, to 1 for the cases where the model predicts the expected values. The value $MCC = 0$ means that there is no correlation between the predicted value and the expected one. By applying MCC to the learning of P_{max}, we observed that the data are clearly discriminated, thus leading to high-quality of our prediction in this case ($MCC = 1$).

Learning P_{avg}. Table (2b) (right) shows the confusion matrix associated to learning P_{avg}. We can see that the predictions are less accurate in this case. The data is not as clearly discriminated as before, leading to a fairly negligible error rate ($< 0.02\%$). However, the latter is still acceptable for learning, since only $< 0.02\%$ of the predictions are erroneous. This is corroborated by an MCC value equal to 0.93, thus leading to a fairly acceptable quality of the prediction in this case too.

Running time of repair **with ML classifiers.** In the last experiment, we want to measure the impact of learning on the performance of our algorithm. To this end, we compare the running time of repair when adopting a hard-coded preference function (as in the results reported in Figure 3) and when adopting a learned preference function. Figure 4 shows the running times for the same scenarios used in Figure 3. We can easily observe that the runtimes are rather similar with and without learning and the difference amounts to a few milliseconds. This further corroborates the utility of learning the preference function and shows that the learning is robust and does not deteriorate the performances of our algorithm.

Qualitative study. In order to illustrate the utility of our approach, in this experiment we study possible rewritings of a mapping defined over the NHS schema.The NHS schema focuses on storing information concerning patients admitted in hospitals. Here, we consider the dependencies involving general information on patients. This includes administrative information

Relation	#atts
birth	34
patient	17
mothers_social_data	8
pis_e_prescribing	27
death	50

(a) Source schema characteristics

Relation	#atts
birth_export	34
patient_export	17
pis_e_prescribing_export	27
death_export	50

(b) Target schema characteristics

Relation	#atts
link_death_drugs	3
death_causes	20
prescribed_drugs_evolution	9
mothers_social	8
fathers_social	6

(c) Policy views schema characteristics

(1) $\mathsf{birth}(\ldots) \rightarrow \mathsf{birth\_export}(\ldots)$

(2) $\mathsf{patient}(\ldots) \rightarrow \mathsf{patient\_export}(\ldots)$

(3) $\mathsf{pis\_e\_prescribing}(\ldots) \rightarrow \mathsf{pis\_e\_prescribing\_export}(\ldots)$

(4) $\mathsf{death}(\ldots) \rightarrow \mathsf{death\_export}(\ldots)$

(d) Mapping over NHS

$\mathsf{death}(\ldots) \land \mathsf{pis\_e\_prescribing}(\ldots) \rightarrow \mathsf{link\_death\_drugs\_data}(\ldots)$

$\mathsf{death}(\ldots) \rightarrow \mathsf{death\_causes}(\ldots)$

$\mathsf{pis\_e\_prescribing}(\ldots) \rightarrow \mathsf{prescribed\_drugs\_evolution}(\ldots)$

$\mathsf{birth}(\ldots) \land \mathsf{patient}(\ldots) \rightarrow \mathsf{mother\_social}(\ldots)$

$\mathsf{birth}(\ldots) \rightarrow \mathsf{fathers\_social}(\ldots)$

(e) Policy views over NHS

Table 3: Properties of the NHS dataset.

Rewrited	#possible	#frontier variables in repairs	
tgd	repairs	min	max
(1)	3	25	29
(2)	2	13	14
(3)	2	18	18
(4)	2	25	25

Table 4: Properties of the repairing process.

(relation patient in the source schema), social and medical information on the patient himself (relations birth and death), social data on patients' mothers (relation mother_social_data) and information on drugs prescriptions (relation pis_e_prescribing).

The characteristics of the source schema are summarized in Table 3a. The mapping to rewrite and the characteristics of its target schema are summarized in Tables 3d and 3b, respectively. The set of policy views and the characteristics of their target schema are reported in Tables 3e and 3c, respectively.

The link_death_drugs_data view allows to link the prescribed drugs with patient pathology, but no personal information is exported to prevent the identification of the patient. The death_causes view gives access to the causes of death of the admitted patients. The prescribed_drugs_evolution view gives access to drug prescriptions information without any identifying information. The mother_social and fathers_social views give access to patients' mothers and fathers social information.

In Table 4, we show the number of possible repairs for each tgd in Table 3d. It can be seen that the tgd (1) has three possible repairs, exporting from 25 to 29 variables, respectively. Analogously, the tgd (2) leads to two possible repairs, allowing to export from 13 to 14 variables each. Both tgds (3) and (4) lead to two possible repairs, with a constant number of exported variables for each tgd. These rewritings are distinguished by the exported variables, and one can decide to choose which repairing fits best her needs either visually or by leveraging the user preference function, as shown in our previous experiment.

7 CONCLUSION

We have studied the problem of data exchange in the presence of privacy restrictions expressed as policy views on the source schema. We have proposed a repairing process for the mappings that are unsafe under the source policy views. Our approach is inherently data-independent and leads to repairing the mappings guaranteeing privacy preservation at a schema level. As such, our approach is orthogonal to several data-dependent privacy-preservation methods (such as differential privacy methods), that can be used on the source and target instances to further corroborate the privacy guarantees. The study of such fruitful combinations of methods is devoted to future work.

We also envision several other extensions of our work, such as the study of more expressive GLAV mappings and the interplay between data-independent and data-dependent privacy methods as well as the usage of other learning methods.

ACKNOWLEDGEMENTS

Research by the first author is funded by ANR (under Grant No. 18-CE23-0002 QualiHealth)

REFERENCES

[1] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. 2014. *Foundations of Data Exchange.* Cambridge University Press.

[2] Patricia C Arocena, Boris Glavic, Radu Ciucanu, and Renée J Miller. 2015. The iBench integration metadata generator. In *Proceedings of VLDB.*

[3] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. 2000. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics* 16, 5 (2000).

[4] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). 2011. *Schema Matching and Mapping.* Springer.

[5] M. Benedikt, B. Cuenca Grau, and E. Kostylev. 2017. Source Information Disclosure in Ontology-Based Data Integration.. In *AAAI.*

[6] Philip A. Bernstein. 2005. The many roles of meta data in data integration. In *In ACM SIGMOD.*

[7] Joachim Biskup and Piero Bonatti. 2004. Controlled query evaluation for enforcing confidentiality in complete information systems. *International Journal of Information Security* 3, 1 (2004).

[8] Joachim Biskup and Torben Weibert. 2008. Keeping secrets in incomplete databases. *International Journal of Information Security* 7, 3 (2008), 199–217.

[9] Piero A. Bonatti, Sarit Kraus, and VS Subrahmanian. 1995. Foundations of secure deductive databases. *IEEE TKDE* 7, 3 (1995), 406–422.

[10] Angela Bonifati, Ugo Comignani, and Efthymia Tsamoura. 2019. MapRepair: Mapping and Repairing under Policy Views (demo). In *ACM SIGMOD.* 1873–1876.

[11] Laura Chiticariu and Wang Chiew Tan. 2006. Debugging Schema Mappings with Routes. In *Proceedings of VLDB.* 79–90.

[12] Ugo Comignani. 2020. MapRepair - open source code. https://github.com/ucomignani/MapRepair.git.

[13] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. 2018. Query-Based Linked Data Anonymization. In *ISWC 2018.* 530–546.

[14] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. 2019. RDF Graph Anonymization Robust to Data Linkage. In *WISE 2019.* 491–506.

[15] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.

[16] Frank Eibe, MA Hall, and IH Witten. 2016. The WEKA Workbench. Online Appendix for" Data Mining: Practical Machine Learning Tools and Techniques. *Morgan Kaufmann* (2016).

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005).

[18] Bernardo Cuenca Grau, Evgeny Kharlamov, Egor V. Kostylev, and Dmitriy Zheleznyakov. 2015. Controlled Query Evaluation for Datalog and OWL 2 Profile Ontologies. In *IJCAI.*

[19] Bernardo Cuenca Grau and Egor V. Kostylev. 2016. Logical Foundations of Privacy-Preserving Publishing of Linked Data. In *AAAI.* 943–949.

[20] Gerome Miklau and Dan Suciu. 2007. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.* 73, 3 (2007).

[21] Alan Nash and Alin Deutsch. 2007. Privacy in GLAV Information Integration. In *ICDT.*

[22] Muhammad I. Sarfraz, Mohamed Nabeel, Jianneng Cao, and Elisa Bertino. 2015. DBMask: Fine-Grained Access Control on Encrypted Relational Databases. In *In CODASPY.* 1–11.

[23] George L Sicherman, Wiebren De Jonge, and Reind P Van de Riet. 1983. Answering queries without revealing secrets. *ACM TODS* 8, 1 (1983), 41–59.

[24] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 5 (2002), 557–570.