

# Repairing mappings under policy views

Angela Bonifati

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

Ugo Comignani

Lyon 1 University & Liris CNRS  
Lyon, France  
ugo.comignani@univ-lyon1.fr

Efthymia Tsamoura

Alan Turing Institute &  
University of Oxford  
Oxford, UK  
efthymia.tsamoura@cs.ox.ac.uk

## ABSTRACT

The problem of data exchange involves a source schema, a target schema and a set of mappings from transforming the data between the two schemas. We study the problem of data exchange in the presence of privacy restrictions on the source. 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. We propose a protocol that provides formal privacy guarantees and is *data-independent*, i.e., if certain criteria are met, then the protocol guarantees that the mappings leak no sensitive information independently of the data that lies in the source. We also propose an algorithm for *repairing* an input mapping w.r.t. a set of policy views, in cases where the input mapping leaks sensitive information. The empirical evaluation of our work shows that the proposed algorithm is quite efficient, repairing sets of 300 s-t tgds in an average time of 5s on a commodity machine. To the best of our knowledge, our work is the first one that studies the problems of exchanging data and repairing mappings under such privacy restrictions. Furthermore, our work is the first to provide practical algorithms for a logical privacy-preservation paradigm, described as an open research challenge in previous work on this area.

## CCS CONCEPTS

• Information systems → Data exchange.

## KEYWORDS

privacy-preserving data integration, data exchange, mapping repairs

## 1 INTRODUCTION

We consider the problem of exchanging data between a source schema  $S$  and a target schema  $T$  via a set of *source-to-target* (s-t) dependencies  $\Sigma_{st}$  that usually come in the form of *tuple-generating dependencies* (tgds). This triple of a source schema, a target schema and a set of dependencies is called a *mapping*. The s-t dependencies specify how and what source data should appear in the target and are expressed as sentences in first-order logic [10].

Our work considers a privacy-aware variant of the data exchange problem, in which the source comes with a set of constraints, representing the data that is *safe* to expose to the target over *all* instances of the source. We also assume that all users, both the malicious and the non-malicious ones, might know the source and the target schema, the data in the target as well as the s-t tgds. Under these assumptions, our work will address the following issues: how could we represent privacy restrictions on the sources and what would it mean for a data exchange setting to be safe under the proposed

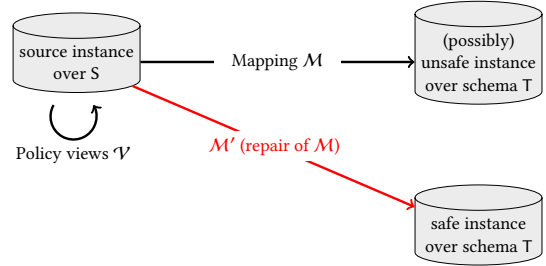


Figure 1: A data exchange setting with mappings and policy views.

privacy restrictions?; 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 leak?; finally, in case of privacy violations, how could we *repair* the s-t tgds?

Regarding the first issue, we assume that the restrictions on the sources are expressed as a set of views, called *policy views*. Inspired by prior work on privacy-preservation [3, 13], 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, the proposed privacy-preservation protocol is *data-independent* allowing us to provide strong privacy-preservation guarantees over all instances of the sources. The above addresses the second aforementioned issue, as well. Regarding the third issue, our work proposes a repairing algorithm for the proposed privacy-preservation protocol. The feature of the proposed repairing algorithm is that it can employ techniques for learning the user preferences during the repairing process. The empirical evaluation of our work over an existing benchmark shows that the proposed algorithm is quite efficient. Indeed it can repair a set of 300 s-t tgds in less than 5s on a commodity machine.

Our secure data exchange setting is exemplified in Figure 1 and in the following running example inspired by a real world scenario from an hospital in the UK<sup>1</sup>.

EXAMPLE 1. Consider the source schema  $S$  consisting of the following relations:  $P$ ,  $H_N$ ,  $H_S$ ,  $O$  and  $S$ . Relation  $P$  stores for each person registered with the NHS, 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 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

<sup>1</sup><https://www.nhs.uk/>

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  $V_1$ – $V_4$ . The policy views define the information that is safe to make available to public. View  $V_1$  projects the ethnicity groups and the hospital admittance reasons for patients in the north of UK;  $V_2$  projects the counties and the hospital admittance reasons for patients in the north of UK;  $V_3$  projects the treatments and the progress of patients of oncology departments;  $V_4$  projects the ethnicity groups of the school students. The policy views are safe w.r.t. the NSS privacy preservation protocol. Indeed, the NSS privacy preservation protocol considers as unsafe any non-evident piece of information that can potentially de-anonymize an individual. For example, views  $V_1$  and  $V_2$  do not leak any sensitive information, since the results concern patients from a very large geographical area and, thus, the probability of de-anonymizing a patient is significantly small. For similar reasons, views  $V_3$  and  $V_4$  are considered to be safe: the probability of de-anonymizing patients of the oncology department from  $V_3$  is zero, since there is no way to link a patient to his treatment or his progress, while  $V_4$  projects information which is already evident to public.

$$P(i, n, e, c) \wedge H_N(i, d) \leftrightarrow V_1(e, d) \quad (1)$$

$$P(i, n, e, c) \wedge H_S(i, d) \leftrightarrow V_2(c, d) \quad (2)$$

$$O(i, t, p) \leftrightarrow V_3(t, p) \quad (3)$$

$$S(i, n, e, c) \leftrightarrow V_4(e) \quad (4)$$

Finally, consider the following set of  $s$ - $t$  dependencies  $\Sigma_{st}$ . The dependencies  $\mu_e$  and  $\mu_c$  project similar information with the views  $V_1$  and  $V_2$ , respectively. They focus, however, on patients in the north of UK. Finally, the dependency  $\mu_s$  projects the ethnicity groups of students who have been in some oncology department.

$$P(i, n, e, c) \wedge H_N(i, d) \rightarrow \text{EthDis}(e, d) \quad (\mu_e)$$

$$P(i, n, e, c) \wedge H_N(i, d) \rightarrow \text{CountyDis}(c, d) \quad (\mu_c)$$

$$S(i, n, e, c) \wedge O(i, t, p) \rightarrow \text{SO}(e) \quad (\mu_s)$$

The questions addressed in our paper are the following ones: Are the  $s$ - $t$  dependencies safe w.r.t. the policy views? Are there any formal guarantees for privacy preservation in the context of policy views? If the  $s$ - $t$  dependencies are not safe w.r.t. the policy views, how could we repair them and provide formal privacy preservation guarantees?

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 logical foundations of information disclosure in ontology-based data integration have been laid in [3] in the presence of boolean policies. Instead, we focus on non-boolean policies. Taking a step forward, we also propose an algorithm for repairing a set of unsafe  $s$ - $t$  tgds w.r.t. our privacy preservation protocol. To the best of our knowledge, our work is the first to provide practical algorithms for a logical privacy-preservation paradigm, described as an open research challenge in [3, 13]. We leave out probabilistic approaches and anonymization techniques [12, 16], which involve modifications of the underlying data instances and are orthogonal to our approach. A careful treatment of related work is deferred to Section 2.

The source code and the experimental scenarios are publicly available at <https://github.com/ucomignani/MapRepair.git>.

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 paper.

## 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 [13]. 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 [3], 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 practical algorithm for checking the safety of schema mappings and for repairing the mappings in order to resume safety 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 [12], 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 [12] 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.

**Controlled Query Evaluation** Controlled Query Evaluation is a confidentiality enforcement framework introduced in [15] and refined in [7],[5] and [6], 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 [11]. 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.

## Repairing mappings under policy views

**Data privacy** Previous work has addressed access control to protect database instances at different levels of granularity [14], in order to combine encrypted query processing and authorization rules. Our work 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 [8] and k-anonymity [16].

### 3 PRELIMINARIES

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  $R \in S$ , where  $*$  is called the *critical constant* and  $\vec{*}$  is an  $n$ -ary vector. A *substitution*  $\sigma$  is a mapping from variables into constants or labelled nulls.

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  $\delta$  as the *body*, denoted as  $\text{body}(\delta)$ , and to the right-hand side as the *head*, denoted as  $\text{head}(\delta)$ . An instance  $I$  satisfies a dependency  $\delta$ , written  $I \models \delta$  if each homomorphism from  $\text{body}(\delta)$  into  $I$  can be extended to a homomorphism  $h'$  from  $\text{head}(\delta)$  into  $I$ . An instance  $I$  satisfies a set of dependencies  $\Sigma$ , written as  $I \models \Sigma$ , if  $I \models \delta$  holds, for each  $\delta \in \Sigma$ . The *solutions* of an instance  $I$  w.r.t.  $\Sigma$  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.  $\Sigma$ .

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  $\sigma$  is an *answer* to a CQ  $Q$  on an instance  $I$  if the domain of  $\sigma$  is the free variables of  $Q$ , and if  $\sigma$  can be extended to a homomorphism from  $\bigwedge_i A_i$  into  $I$ . We denote by  $Q(I)$ , the answers to  $Q$  on  $I$ .

Given an instance  $I$  and a set of dependencies  $\Sigma$ , the chase iteratively computes a *universal* solution of  $I$  w.r.t.  $\Sigma$  [4, 10]. Starting from  $I_0 = I$ , at each iteration  $i$ , it computes a new instance  $I_i$  by applying a tgd or an egd chase step:

**tgd chase step.** Consider an instance  $I_i$ , a tgd  $\delta$  of the form  $\forall \vec{x} \lambda(\vec{x}) \rightarrow \exists \vec{y} \rho(\vec{x}, \vec{y})$  and a homomorphism  $h$  from  $\lambda(\vec{x})$  into  $I_i$ , such that there does not exist an extension of  $h$  to a homomorphism from  $\rho(\vec{x}, \vec{y})$  into  $I_i$ . Such homomorphisms are called *active triggers*. Applying the tgd chase step for  $\delta$  and  $h$  to  $I_i$  results in a new

instance  $I_{i+1} = I_i \cup h'(\rho(\vec{x}, \vec{y}))$ , where  $h'$  is a substitution such that  $h'(x_j) = h(x_j)$  for each variable  $x_j \in \vec{x}$ , and  $h'(y_j)$ , for each  $y_j \in \vec{y}$ , is a fresh labeled null not occurring in  $I_i$ .

**egd chase step.** Consider an instance  $I_i$ , an egd  $\delta$  of the form  $\forall \vec{x} \lambda(\vec{x}) \rightarrow x_i = x_j$  and a homomorphism  $h$  from  $\lambda(\vec{x})$  into  $I_i$ , such that  $h(x_i) \neq h(x_j)$ . Applying the egd chase step for  $\delta$  and  $h$  to  $I_i$  fails if  $\{h(x_i), h(x_j)\} \subseteq \text{Const}$ , and otherwise it results in a new instance  $I_{i+1} = \nu(I_i)$ , where  $\nu = \{h(x_j) \mapsto h(x_i)\}$  if  $h(x_i) \in \text{Const}$ , and  $\nu = \{h(x_i) \mapsto h(x_j)\}$  if  $h(x_i) \notin \text{Const}$ . We denote by  $\text{chase}(I, \Sigma)$ , the chase of  $I$  w.r.t.  $\Sigma$ .

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  $(S, T, \Sigma)$ , where, generally,  $\Sigma = \Sigma_s \cup \Sigma_{st} \cup \Sigma_t$ .  $\Sigma_s$ ,  $\Sigma_{st}$  and  $\Sigma_t$  denote the source, s-t and target dependencies over  $S$  and  $T$ , respectively. We usually refer to the dependencies in  $\Sigma_{st}$  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  $\mu$ . We denote by  $\text{exported}(\mu)$ , the set of exported variables of  $\mu$ . The inverse of set of s-t dependencies  $\Sigma_{st}$ , denoted as  $\Sigma_{st}^{-1}$  is the set consisting, for each mapping  $\mu$  in  $\Sigma_{st}$  of the form  $\lambda(\vec{x}) \rightarrow \rho(\vec{x}, \vec{y})$ , a mapping  $\mu^{-1}$  of the form  $\rho(\vec{x}, \vec{y}) \rightarrow \lambda(\vec{x})$ . In this paper, we will focus on the scenario, where  $\Sigma_s$  and  $\Sigma_t$  are empty, so  $\Sigma$  will be equal to  $\Sigma_{st}$ . Furthermore, in this paper we focus on *GLAV* mappings, i.e., s-t dependencies corresponding to a set of views.

The *certain answers* of a CQ  $Q$  over  $T$  w.r.t.  $I$  and  $\mathcal{M}$ , denoted as  $\text{certain}(Q, I, \mathcal{M})$ , are the intersection of all answers to  $Q$  over all solutions of  $I$  w.r.t.  $\Sigma$ . Given a finite, null-free instance  $I$  of the source schema, the objective of *data exchange* is to compute a universal solution of  $I$  w.r.t. the dependencies  $\Sigma$  from  $\mathcal{M}$ .

## 4 PRIVACY PRESERVATION

In this section, we introduce our notion of privacy preservation. Let  $\mathcal{V}$  be a set of *policy views* over  $S$ . The policy views represent the information that is safe to expose for instances  $I$  of  $S$ . We denote by  $\mathcal{M}_{\mathcal{V}} = (S, V, \mathcal{V})$  the mapping from  $S$  to  $V$ , where  $V$  denotes the schema of the views occurring in  $\mathcal{V}$ . Our goal is to verify whether a user-defined mapping  $\mathcal{M} = (S, T, \Sigma)$  is safe w.r.t. a view mapping  $\mathcal{M}_{\mathcal{V}}$ . 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  $S$ . Below, 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 upon the protocol introduced in [3]. Below, we formalize the notion of privacy preservation from [3] and we extend it for non-boolean conjunctive queries. First we recapitulate the notion of indistinguishability of two source instances.

**Definition 1.** Two instances  $I$  and  $I'$  of a source schema  $S$  are indistinguishable with respect to a mapping  $\mathcal{M} = (S, T, \Sigma)$ , denoted as  $I \equiv_{\mathcal{M}} I'$ , if  $\text{certain}(Q, I, \mathcal{M}) = \text{certain}(Q, I', \mathcal{M})$  for each CQ  $Q$  over  $T$ .  $\square$

Informally, Definition 1 tells us that two source instances are indistinguishable from each other if the target instances have the same certain answers.

**Definition 2.** A mapping  $\mathcal{M} = (S, T, \Sigma)$  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  $p(I') = \emptyset$ .  $\square$

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 [3]. 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*  $\text{visChase}_{\Sigma}(\Sigma)$  of  $\Sigma$  under the critical instance  $\text{Crt}_{\Sigma}$  of  $S$ . The visible chase computes a *universal source instance*— that is an instance, such that the visible part of any instance of  $S$  (i.e., the subinstance that becomes available through the mappings) can be mapped into it. The only constant occurring in the instance computed by  $\text{visChase}_{\Sigma}(\Sigma)$  is the critical constant  $*$  and it represents any other constant that can occur in the source instance.

For the purpose of repairing the mappings efficiently, 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 [3]. However, instead of keeping these facts in a single set, we keep them in separate bags. Before presenting Algorithm 1, we will introduce some new notions.

**Definition 3.** Consider an instance  $I$ . Consider also a s-t tgd  $\delta$  and a homomorphism  $h$  from  $\text{body}(\delta)$  into  $I$ , such that  $h(x) \in \text{Nulls}$ , for some  $x \in \text{exported}(\delta)$ . Then, we say that the egd

$$\text{body}(\delta) \rightarrow \bigwedge_{\forall x \in \text{exported}(\delta): h(x) \in \text{Nulls}} x = * \quad (5)$$

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

**Definition 4.** Consider an instance  $I$ , whose facts are organized into the bags  $\beta_1, \dots, \beta_m$ . Consider also a derived egd  $\delta$  of the form (5) and an active trigger  $h$  for  $\delta$  in  $I$ . A bag  $\beta_i$  is *relevant* for  $\delta$  and  $h$  in  $I$ , where  $1 \leq i \leq m$ , if some fact  $F \in h(\text{body}(\delta))$  occurs in  $\beta_i$  and if some  $h(x)$  is a labeled null occurring in  $\beta_i$ , where  $x \in \text{exported}(\delta)$ .

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

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  $\text{Crt}_{\Sigma}$  using the s-t tgds, line 1. It then chases  $I_0$  with the inverse s-t tgds  $\Sigma^{-1}$ , 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  $\delta$  in  $I$ , Algorithm 1 adds a fresh bag with facts  $h'(\text{head}(\delta))$ , if  $\delta \in \Sigma \cup \Sigma^{-1}$ , line 9; otherwise, if

$\delta \in \Sigma_{\approx}$ , then it adds the derived bag for  $\delta$  and  $h$  in  $I$ , see Definition 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 [3]. Let  $B = \text{visChase}_{\Sigma}(\Sigma)$ . We will denote by  $I_{\Sigma}(\Sigma)$ , the instance  $\bigcup_{\beta \in B} \beta$ .

---

#### Algorithm 1 $\text{visChase}_{\Sigma}(\Sigma)$

---

```

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

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

12: procedure  $\text{bagChaseEGDs}(\Sigma_{\approx}, B)$ 
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 (5) do
17:       for each active trigger  $h : \text{body}(\delta) \rightarrow I_{i-1}$  do
18:         if  $h(x) \neq *$ , for some  $x \in \text{exported}(\delta)$  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 2.** We demonstrate the visible chase algorithm over the policy views and the s-t dependencies from Example (1).

We first present the computation of  $I_{\Sigma}(\mathcal{V}) = \bigcup_{\beta \in \text{visChase}_{\Sigma}(\mathcal{V})} \beta$ . The critical instance  $\text{Crt}_{\Sigma}$  of  $S$  consists of the facts shown in Eq. (6).

$$\begin{array}{lll} P(*, *, *, *) & H_N(*, *) & H_S(*, *) \\ O(*, *, *) & S(*, *, *, *) & \end{array} \quad (6)$$

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

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

where the constants prefixed by  $n$  are labeled nulls created while chasing  $\text{Crt}_{\Sigma}$  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 3. Thus,  $I_{\Sigma}(\mathcal{V}) = I_1$ .

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

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

Since there exists a homomorphism from the body of  $\mu_e$  into  $I_1'$  mapping the exported variable  $e$  into the labeled null  $n_e$ , and since there

## Repairing mappings under policy views

exists another homomorphism from the body of  $\mu_c$  into  $I'_1$  mapping the exported variable  $c$  into the labeled null  $n_c$ ,  $\Sigma_{\approx}$  will comprise the egds  $\epsilon_1$  and  $\epsilon_2$  shown below

$$P(i, n, e, c) \wedge H_N(i, d) \rightarrow e \approx * \quad (\epsilon_1)$$

$$P(i, n, e, c) \wedge H_N(i, d) \rightarrow c \approx * \quad (\epsilon_2)$$

The last step of the visible chase involves chasing  $I'_1$  using  $\Sigma_{\approx}$ . WLOG, assume that the chase considers first  $\epsilon_1$  and then  $\epsilon_2$ . During the first step of the chase, there exists a homomorphism from  $\text{body}(\epsilon_1)$  into  $I'_1$ . Hence,  $n_e = *$ . During the second step of the chase, there exists a homomorphism from  $\text{body}(\epsilon_2)$  into  $I'_1$  and, hence,  $n_c = *$ . The instance computed at the end of the second round of the chase will consist of the facts

$$\begin{array}{lll} P(n_i, n_n, *, *) & H_N(n_i, *) & H_N(n'_i, *) \\ S(n''_i, n''_n, *, n'_c) & O(n''_i, n''_t, n''_p) & \end{array} \quad (7)$$

Since there exists no active trigger for  $\epsilon_1$  or  $\epsilon_2$  in the instance of Eq. (7), the chase will terminate.

The facts in  $I_S(\Sigma_{st})$  will be organized into the following bags  $\beta_1$ – $\beta_5$  (one bag per line)

$$\begin{array}{l} \text{SO}(e) \xrightarrow{\langle \mu_s^{-1}, h_1 \rangle} S(n''_i, n''_n, *, n'_c), O(n''_i, n''_t, n''_p) \\ \text{CountyDis}(c, d) \xrightarrow{\langle \mu_c^{-1}, h_2 \rangle} P(n'_i, n'_n, n_e, *), H_N(n'_i, *) \\ \text{EthDis}(e, d) \xrightarrow{\langle \mu_e^{-1}, h_3 \rangle} P(n_i, n_n, *, n_c), H_N(n_i, *) \\ P(n'_i, n'_n, n_e, *), H_N(n'_i, *) \xrightarrow{\langle \epsilon_1, h_4 \rangle} P(n'_i, n'_n, *, *), H_N(n'_i, *) \\ P(n_i, n_n, *, n_c), H_N(n_i, *) \xrightarrow{\langle \epsilon_2, h_5 \rangle} P(n_i, n_n, *, *), H_N(n_i, *) \end{array}$$

$$\begin{array}{l} h_1 = \{i \mapsto n'_i, n \mapsto n'_n, e \mapsto n_e, c \mapsto *, d \mapsto *\} \\ h_2 = \{c \mapsto *, d \mapsto *\} \\ h_3 = \{e \mapsto *, d \mapsto *\} \\ h_4 = \{i \mapsto n'_i, n \mapsto n'_n, e \mapsto n_e, c \mapsto *, d \mapsto *\} \\ h_5 = \{i \mapsto n_i, n \mapsto n_n, e \mapsto *, c \mapsto n_c, d \mapsto *\} \end{array}$$

The contents of the bags correspond to the right-hand side of the arrows. However, for presentation purposes, we also show the related dependency  $\delta$  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).

## 4.2 Preserving the privacy of policy views

We consider a mapping consisting of CQ views  $\mathcal{M} = (S, T, \Sigma)$  to be safe w.r.t. a view mapping consisting of CQ views  $\mathcal{M}_V = (S, V, \mathcal{V})$ , if  $\mathcal{M}$  does not disclose the information that is also not disclosed by  $\mathcal{M}_V$ . Definition 5 and Theorem 1 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.  $\mathcal{M}_V$ .

**Definition 5.** A mapping  $\mathcal{M}_2 = (S, T_2, \Sigma_2)$  preserves the privacy of a mapping  $\mathcal{M}_1 = (S, T_1, \Sigma_1)$  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  $S$ .  $\square$

**Theorem 1.** A mapping  $\mathcal{M}_2 = (S, T_2, \Sigma_2)$  preserves the privacy of a mapping  $\mathcal{M}_1 = (S, T_1, \Sigma_1)$  on all instances of  $S$ , if and only if there exists a homomorphism  $h$  from  $I_S(\Sigma_2)$  into  $I_S(\Sigma_1)$ , such that  $h(*) = *$ .  $\square$

**PROOF.** (Sketch) First we show that the following holds

**Lemma 1.** A mapping  $\mathcal{M} = (S, T, \Sigma)$  does not disclose a constants-free CQ  $p$  over  $S$  on any instance of  $S$ , iff  $\vec{*} \notin p(J)$ , where  $J = I_S(\Sigma_{st})$ .

**PROOF.** By adapting the proof technique of Theorem 16 from [3], we can show that  $J = I_S(\Sigma_{st})$  is a *universal* source instance  $I_S(\Sigma)$  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  $I_S(\Sigma)$  mapping each schema constant into the critical constant  $*$ . Due to the existence of a homomorphism  $h$  from  $I'$  into  $I_S(\Sigma)$ , 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 2, it follows that  $\mathcal{M} = (S, T, \Sigma)$  does not disclose a constants-free CQ  $p$  over  $S$  on any instance of  $S$ .  $\square$

Lemma 1 states that, in order to check if a constants-free CQ is safe according to Definition 2, we need to check if the critical tuple is among the answers to  $p$  over the instance computed by  $\text{visChases}(\Sigma)$ . Next, we show the following lemma.

**Lemma 2.** 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 2.** (2) $\Rightarrow$ (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) $\Rightarrow$ (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.  $\square$

Lemma 2 can be restated as follows

**Lemma 3.** *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  $(S, T, \Sigma)$ , where  $T$  is a target schema and  $\Sigma$  is a set of s-t dependencies, we know from Proposition 1 that if  $\mathcal{M}$  discloses  $p$  on some instance of  $S$ , then there exists a homomorphism of  $p$  into  $\text{visChase}_S(\Sigma)$  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  $J_1 = I_S(\Sigma_1)$  and  $J_2 = I_S(\Sigma_2)$ . 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 3, it follows that there exists a homomorphism  $\phi : J_2 \rightarrow J_1$ , 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<sub>1</sub>). From assumption (A<sub>1</sub>) 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  $\phi : I_S(\Sigma_2) \rightarrow I_S(\Sigma_1)$ , such that  $\phi(*) = *$ .  $\square$

Theorem 1 states that in order to verify whether  $\mathcal{M}_2$  is safe w.r.t.  $\mathcal{M}_1$ , we need to compute  $I_S(\Sigma_1)$  and  $I_S(\Sigma_2)$  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  $I_S(\Sigma_1)$  is *safe* w.r.t.  $I_S(\Sigma_2)$ , or simply *safe*, and we say that it is *unsafe* otherwise.

**EXAMPLE 3.** *Continuing with Example 1, we can see that the s-t tgds are not safe w.r.t. the policy views according to Theorem 1, since there does not exist a homomorphism from the instance  $I_S(\Sigma_{st})$  into the instance  $I_S(\mathcal{V})$ . This means that there exists information which is disclosed by  $\Sigma_{st}$  in some instance that satisfies  $\Sigma_{st}$ , but it is not disclosed by  $\mathcal{V}$ . Indeed, from  $S(n'_1, n'_n, *, n'_c)$  and  $O(n'_1, n'_t, n'_p)$ , 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  $\mu_s$ . 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  $\mathcal{V}_3$  and  $\mathcal{V}_4$ .*

---

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


---

```

1:  $\Sigma_1 := \text{frepair}(\Sigma, \mathcal{V}, \text{prf})$ 
2:  $\Sigma_2 := \text{srepair}(\Sigma_1, \mathcal{V}, \text{prf}, n)$ 
3: return  $\Sigma_2$ 

```

---

*Furthermore, by looking at the facts  $P(n_i, n_n, *, *)$  and  $H_N(n_i, *)$ , 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  $\mu_e$  and  $\mu_c$ . Note that the policy views  $\mathcal{V}_1$  and  $\mathcal{V}_2$  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  $\mathcal{V}$ .

Algorithm 2 summarizes the steps of the proposed algorithm. The inputs to it are, apart from  $\Sigma$  and  $\mathcal{V}$ , a positive integer  $n$  which will be used during the second step of the repairing process and a preference mechanism  $\text{prf}$  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  $I_S(\Sigma)$  is safe according to Theorem 1, 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  $\Sigma$  into a *partially-safe* set of s-t dependencies  $\Sigma_1$ , while the second step rewrites the output of the first one into a new set of s-t dependencies  $\Sigma_2$ , such that  $I_S(\Sigma_2)$  is safe. As we will explain later on, partial-safety ensures that the intermediate instance  $I_1$  produced by  $\text{visChase}_S(\Sigma_1)$  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  $I_S(\Sigma)$  into  $I_S(\mathcal{V})$ , a first test towards checking for such a homomorphism is to look if the mappings in  $\Sigma$  would lead to such a homomorphism or not. For instance, by looking at  $\mu_s$  in Example 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  $I_S(\mathcal{V})$ .

**Definition 6.** A mapping  $\mathcal{M} = (S, T, \Sigma)$  is *partially-safe* w.r.t.  $\mathcal{M}_V = (S, V, \mathcal{V})$  on all instances of  $S$ , if there exists a homomorphism from  $\text{chase}(\Sigma^{-1}, \text{Crt}_T) \setminus \text{Crt}_T$  into  $I_S(\mathcal{V})$ .  $\square$

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

**Proposition 1.** A mapping  $\mathcal{M} = (S, T, \Sigma)$  is partially-safe w.r.t.  $\mathcal{M}_V = (S, V, \mathcal{V})$  on all instances of  $S$ , if for each  $\mu \in \Sigma$ , there exists a

## Repairing mappings under policy views

homomorphism from  $\text{body}(\mu)$  into  $I_S(\mathcal{V})$  mapping each  $x \in \text{exported}(\mu)$  into the critical constant  $*$ .  $\square$

Note that according to Proposition 1, in our running example  $\Sigma_{st}$  would be partially-safe, if  $\mu_s \notin \Sigma_{st}$ , then since there exist homomorphisms from the bodies of  $\mu_s$  and  $\mu_c$  into  $I_S(\mathcal{V})$ , mapping their exported variables into  $*$ . It is also easy to show the following

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

Proposition 1 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  $\mu_r$  satisfies the criterion in Proposition 1.

We make use of the result of Proposition 1 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  $\mu$  w.r.t. the variable repetitions in the bodies of the rules and the exported variables. For performance reasons, we do not examine rewritings that introduce atoms in the bodies of the rules. However, this does not compromise the *completeness* of Algorithm 2 as we show at end of this section. Below, we present the steps of Algorithm 3.

For each s-t tgd  $\mu$  and for each atom  $B \in \text{body}(\mu)$ , Algorithm 3 constructs a fresh atom  $C$  and adds  $C$  to a set  $C$ . The set of atoms  $C$  provides us with the means to identify all repairs of  $\mu$  that involve breaking joins and hiding exported variables. In particular, each homomorphism  $\xi$  from  $C$  into  $I_S(\mathcal{V})$  corresponds to one repair of  $\mu$ . In lines 12–25, Algorithm 3 modifies each atom  $B \in \text{body}(\mu)$  by taking into account prior body atom modifications. The prior modifications are accumulated in the relation  $\rho$  and the mapping  $\psi$ . The relation  $\rho$  keeps for each variable  $x$  from  $\text{body}(\mu)$ , the fresh variables that were used to replace  $x$  during prior steps of the repairing process, while  $\psi$  is a substitution from the partially repaired body into  $I_S(\mathcal{V})$ . In particular, at the end of the  $i$ -th iteration of the loop in line 12,  $\psi$  holds the substitution from the first repaired  $i$  atoms from  $\text{body}(\mu)$  into  $I_S(\mathcal{V})$ . 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  $\mu$ , w.r.t. a homomorphism  $\xi$ , lines 9–27. Let  $C = \nu(B)$  be the fresh body atom that was constructed out of  $B$  in line 5. For each atom  $B \in \text{body}(\mu)$  and for each  $p \in \text{pos}(B)$ , if the variable  $y$  in position  $p$  of  $C$  is not mapped to the critical constant  $*$  via  $\xi$  and  $B|_p$  is a 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  $\xi$ , 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  $\psi$ , if it is not already

---

### Algorithm 3 $\text{repair}(\Sigma, \mathcal{V}, \text{prf})$

---

```

1: for each  $\mu \in \Sigma$  do
2:    $v := \emptyset, C := \emptyset$ 
3:   for each  $B \in \text{body}(\mu)$ , where  $B = R(\bar{x})$  do
4:     create a vector of fresh variables  $\bar{y}$ 
5:     create the atom  $C = R(\bar{y})$ 
6:     add  $(B, C)$  to  $v$ 
7:     add  $C$  to  $C$ 
8:    $\mathcal{R}_\mu := \emptyset$ 
9:   for each homomorphism  $\xi : C \rightarrow I_S(\mathcal{V})$  do
10:     $\rho := \emptyset, \psi := \emptyset$ 
11:     $\mu_r := \mu$ 
12:    for each  $B \in \text{body}(\mu_r)$  do
13:       $C = \nu(B)$ 
14:      for each  $p \in \text{pos}(B)$  do
15:         $x = B|_p, y = C|_p$ 
16:        if  $x \in \text{exported}(\mu)$  and  $* \neq \xi(y)$  or  $x \in \text{dom}(\psi)$  and  $\psi(x) \neq \xi(y)$  then
17:          if  $\exists x' \text{ s.t. } (x, x') \in \rho$  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  $x \notin \text{dom}(\psi)$  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  $\Sigma$ 

```

---

there, lines 24–25. Finally, the algorithm chooses the best repair according to the preference function, lines 28–31.

EXAMPLE 4. We demonstrate an example of Algorithm 3.

Since Algorithm 3 focuses on  $I_S(\mathcal{V})$  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

$$I_S(\mathcal{V}) = \{R_1(*, n_1, n_2), S_1(n_1, n_2, n_2), S_1(n_1, n_3, *), S_1(n_1, *, *)\}$$

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

$$R_1(x, y, z) \wedge S_1(y, z, z) \rightarrow T_1(x, z) \quad (\mu_1)$$

Note that  $\mathcal{M}$  is not partially-safe. Algorithm 3 computes two repairs for  $\mu_1$  by applying the steps described below. First, it computes the atoms  $R_1(x_1, x_2, x_3)$   $S_1(x_4, x_5, x_6)$  and adds them to  $C$ , lines 3–7. Then, it identifies the following three homomorphisms from  $C$  into  $I_S(\mathcal{V})$ :

$$\xi_1 = \{x_1 \mapsto *, x_2 \mapsto n_1, x_3 \mapsto n_2, x_4 \mapsto n_1, x_5 \mapsto n_2, x_6 \mapsto n_2\}$$

$$\xi_2 = \{x_1 \mapsto *, x_2 \mapsto n_1, x_3 \mapsto n_2, x_4 \mapsto n_1, x_5 \mapsto n_3, x_6 \mapsto *\}$$

$$\xi_3 = \{x_1 \mapsto *, x_2 \mapsto n_1, x_3 \mapsto n_2, x_4 \mapsto n_1, x_5 \mapsto *, x_6 \mapsto *\}$$

From  $\xi_1$ , we can see that the joins in the body of  $\mu_1$  are safe; however, it is unsafe to export  $z$ . From  $\xi_2$ , we can see that it 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  $\xi_1$  and  $\xi_2$ , line 9. When  $B = R_1(x, y, z)$  and  $p < 3$ , Algorithm 3 computes  $\psi$  to  $\{x \mapsto *, y \mapsto n_1\}$ , since there is no violation according to line 16. When  $B = R_1(x, y, z)$  and  $p = 3$ , however, a violation is detected. This is due to the facts that  $z$  is an exported variable and  $\xi(x_3) = n_2$ . Algorithm 3 tackles this violation by creating a fresh variable  $z_1$ . Then, it adds the relation  $(z, z_1)$  to  $\rho$ , replaces  $z$  in  $B|_3$  by  $z_1$  and adds the mapping  $\{z_1 \mapsto n_2\}$  to  $\psi$ , lines 19–23. Algorithm 3 then

considers  $S_1(y, z, z)$ . 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  $\xi(x_3) = n_2$ . Since  $(z, z_1) \in \rho$  and  $\psi(z_1) = \xi_1(x_5)$ , Algorithm 3 replaces  $z$  in the second position of  $S_1(y, z, z)$  by  $z_1$ , line 19. By applying a similar reasoning, we can see that the variable  $z$  sitting in  $S_1(y, z, z)|_3$  is also replaced by  $z_1$ . Hence, the first repair of  $\mu$  is

$$R_1(x, y, z_1) \wedge S_1(y, z_1, z_1) \rightarrow T_1(x) \quad (r_1)$$

Algorithm 3, then proceeds by repairing  $\mu_1$  based on  $\xi_2$ . When  $B = R_1(x, y, z)$ , Algorithm 3 proceeds as described above and computes  $\psi$  to  $\{x \mapsto *, y \mapsto n_1, z_1 \mapsto n_2\}$ . When  $B = S_1(y, z, z)$  and  $p = 1$ , then no violation is encountered since  $\psi(y) = \xi_1(x_4)$ , while when  $B = S_1(y, z, z)$  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  $\{z_2 \mapsto n_3\}$  to  $\psi$ . When  $B = S_1(y, z, z)$  and  $p = 3$ , then no violation is met, since  $z \in \text{exported}(\mu)$  and  $\xi_2(x_6) = *$ . Hence, the second repair of  $\mu_1$  is

$$R_1(x, y, z_1) \wedge S_1(y, z_2, z) \rightarrow T_1(x, z) \quad (r_2)$$

Finally, we can see that the repair for  $\mu_1$  w.r.t.  $\xi_3$  is

$$R_1(x, y, z_1) \wedge S_1(y, z, z) \rightarrow T_1(x, z) \quad (r_3)$$

**Proposition 2.** For any  $\mathcal{M} = (S, T, \Sigma)$ , any  $\mathcal{M}_V = (S, V, \mathcal{V})$  and any preference function  $\text{prf}$ , Algorithm `frepair` returns a mapping  $\mathcal{M}' = (S, T, \Sigma')$  that is partially-safe w.r.t.  $\mathcal{M}_V$  on all instances of  $S$ .  $\square$

**PROOF.** (Sketch) From Proposition 1, a mapping  $\mathcal{M} = (S, T, \Sigma)$  is partially-safe w.r.t.  $\mathcal{M}_V = (S, V, \mathcal{V})$  on all instances of  $S$ , if for each  $\mu \in \Sigma$ , there exists a homomorphism from  $\text{body}(\mu)$  into  $I_S(\mathcal{V})$  mapping each  $x \in \text{exported}(\mu)$  into the critical constant  $*$ . Since for each  $\mu \in \Sigma$  `frepair` computes a set of repaired tgds  $\mathcal{R}_\mu$ , it follows that Proposition 2 holds, if such a homomorphism exists, for each repaired tgd in  $\mathcal{R}_\mu$ . The proof proceeds as follows. Let  $\mu_r^i$  and  $\psi^i$  denote the repaired s-t tgd and the homomorphism  $\psi$  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  $\text{body}(\mu_r^i)$ . Since each  $C \in C$  is an atom of distinct fresh variables, since  $\xi$  is a homomorphism from  $C$  to  $I_S(\mathcal{V})$  and since  $\psi(B^i) = \mu_r^i|_i$ , it follows that in order to prove Proposition 1, we have to show that the following claim holds, for each  $i \geq 0$ :

- $\phi$ .  $\psi^i$  is a homomorphism from the first  $i$  atoms in the body of  $\mu_r^i$  into  $I_S(\mathcal{V})$  mapping each exported variable occurring in  $B^0, \dots, B^i$  into the critical constant  $*$ .

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

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

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

## 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, where  $\Sigma_{st}$  comprises only  $\mu_e$  and  $\mu_c$ . Both  $\mu_e$  and  $\mu_c$  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  $\Sigma_{st}$  from above. Since  $\Sigma_{st}$  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  $\Sigma_{st}$ . 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  $\epsilon_1$ , which in turn has been created due to the fact that  $e$  is an exported variable in  $\mu_e$ . By hiding the exported variable  $e$  from  $\mu_e$ , we actually prevent the creation of  $\epsilon_1$  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.** 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  $I_S(\mathcal{V}) = \{R_1(n_1, n_1, *), R_1(*, *, n_2), S_1(*)\}$ , where  $n_1$  and  $n_2$  are labelled nulls. Consider also the mapping  $\mathcal{M}$  consisting of the following s-t dependencies:

$$R_1(x, x, y) \wedge S_1(y) \rightarrow T_1(y) \quad (\mu_2)$$

$$R_1(x, x, y) \rightarrow T_2(x) \quad (\mu_3)$$

It is easy to see that  $\mathcal{M}$  is partially-safe, but unsafe in overall. Indeed,  $I_S(\Sigma)$  will consist of the following bags (for presentation purposes, we adopt the notation from Example 2):

$$T_1(*) \xrightarrow{\langle \mu_2^{-1}, \theta_1 \rangle} R_1(n_3, n_3, *), S_1(*)$$

$$T_2(*) \xrightarrow{\langle \mu_3^{-1}, \theta_2 \rangle} R_1(*, *, n_4)$$

$$R_1(n_3, n_3, *), S_1(*) \xrightarrow{\langle \epsilon_3, \theta_3 \rangle} R_1(*, *, *), S_1(*)$$

where  $\epsilon_3 := R_1(x, x, y) \rightarrow x = *$ ,  $\theta_1 = \{y \mapsto *\}$ ,  $\theta_2 = \{x \mapsto *\}$  and  $\theta_3 = \{x \mapsto n_3, y \mapsto *\}$ . Note that  $\epsilon_3$  has been created out of  $\mu_3$ , since there exists a homomorphism from  $\text{body}(\mu_3)$  into  $R_1(n_3, n_3, *)$  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  $\mu_3$ . By doing this, we block the creation of  $\epsilon$ , and hence the unsafe unification.

A second approach is to keep  $x$  as an exported variable in  $\mu_3$ , but modify the body of  $\mu_2$  by breaking the join between the first and the second position of  $R_1$

$$R_1(x, z, y) \wedge S_1(y) \rightarrow T_1(y) \quad (\mu'_2)$$

By doing this, we prevent the creation of  $\epsilon$ , since the instance computed at line 2 of Algorithm 1 would consist of the facts  $R_1(n_3, n_5, *)$ ,



## Repairing mappings under policy views

$R_1(*, *, n_4)$ ,  $S_1(*)$  and, hence, there would be no homomorphism from  $\text{body}(\mu_3)$  into it. Note that the modification of  $\mu_2$  to  $\mu'_2$  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  $\beta$ , denoted as  $\text{depth}(\beta)$ , coincides with the highest derivation depth of the facts in  $\beta$ . The *support* of a bag  $\beta$ , denoted as  $\beta^<$ , is inductively defined as follows: if  $\text{depth}(\beta) = 1$ , then  $\beta^< = \beta$ ; otherwise, if  $\text{depth}(\beta) > 1$ , then  $\cup_{\beta' < \beta} \beta'^<$ . Consider an active trigger  $h$  for  $\delta$  in  $I$  leading to the creation of a bag  $\beta$ . We use the following notation:  $\text{dependency}(\beta) = \delta$ ,  $\text{trigger}(\beta) = h$  and  $\text{premise}(\beta) = h(\text{body}(\delta))$ . Two bags  $\beta_1$  and  $\beta_2$  are candidates for  $\text{modifyBody}$  if  $\beta_1 < \beta_2$ ,  $\text{depth}(\beta_1) = 1$ ,  $\text{depth}(\beta_2) = 2$  and there exists at least one repeated variable in the body of  $\text{tgd}(\beta_1)$ .

Algorithm 4 presents an iterative process for repairing a partially-safe  $\Sigma$ , 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  $\Sigma_i$ , where  $\Sigma_0 = \Sigma$ , and incrementally computes the visible chase of the new 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  $\Sigma_0$  to  $\Sigma$ , lines 1. Then, at each iteration  $i$ , it first identifies the lowest depth unsafe bag, line 7, and attempts to repair the dependencies from  $\Sigma_i$  that lead to its creation, lines 7–22. If  $i < n$ , it proposes two different repairs for  $\Sigma_i$ , one based on hiding exported variables through  $\text{hideExported}$  (Algorithm 5), and the second based on eliminating joins through  $\text{modifyBody}$  (Algorithm 6), lines 10–19. Algorithm 4 applies the  $\text{modifyBody}$  if there exist two bags in the support of  $\beta$  that are candidates for  $\text{modifyBody}$ . Informally, Algorithm 4 tries to apply  $\text{modifyBody}$  as early as possible (condition  $\text{depth}(\beta_1) = 1$ ,  $\text{depth}(\beta_2) = 2$ ) and when there are one or more repeated variables in the body of  $\text{tgd}(\beta_1)$  (recall Example 5). Otherwise, if  $i = n$ , it either applies the function  $\text{hideExported}$ , or it eliminates the s-t tgds that are responsible for unsafe unifications.

**EXAMPLE 6.** We demonstrate Algorithm 4 over a simplified version of the running example, where  $\Sigma'_{st} = \{\mu_e, \mu_c\}$ . It is seen that  $\text{visChase}_S(\Sigma'_{st})$  will consist of the bags  $\{\beta_2, \beta_3, \beta_4, \beta_5\}$ . We assume that  $n = \infty$ . During the first iteration of Algorithm 4, the lowest depth bag for which there exists a homomorphism violation is  $\beta_4$ . Since  $i < n$ , the algorithm tries to repair  $\Sigma'_{st}$  by calling  $\text{hideExported}$  and  $\text{modifyBody}$  with arguments (apart from  $\mathcal{V}$  and  $\text{prf}$ )  $\beta_4$  and  $\beta_2, \beta_4$ , respectively.

Algorithm 5 first computes  $v = \{n'_1 \mapsto x_1, n'_n \mapsto x_2, n_e \mapsto x_3\}$ , lines 3–4, and then computes all homomorphisms from

$$v(J) = \{P(x_1, x_2, x_3, *), H_N(x_1, *)\}$$

into the instance  $I_S(\mathcal{V})$ , line 6. We can see that there exists only one such homomorphism  $\xi = \{x_1 \mapsto n'_1, x_2 \mapsto n'_n, x_3 \mapsto n_e\}$ . We have  $\text{tgd}(\beta_4) = \mu_e$ . The first two iterations of the loop in lines 8–12 have no effect, since despite that  $\xi(x_1) = n'_1$  and  $\xi(x_2) = n'_n$ , the variables  $i$  and  $n$  from  $\mu_e$  that are mapped to  $n'_1$  and  $n'_n$  via  $\text{trigger}(\beta_4) = h_4$  are not exported ones. During the last iteration, since  $\xi(x_3) = n_e$ , since

---

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

---

```

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

```

---



---

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

---

```

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

```

---



---

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

---

```

1:  $\mathcal{R} := \emptyset$ 
2: if  $\exists$  one or more repeated variables in  $\text{body}(\mu_1)$  then
3:   for each  $\xi : \text{body}(\mu_2) \rightarrow \text{body}(\mu_1)$  mapping some  $x_1 \in \text{exported}(\mu_1)$ 
   into some  $x_2 \notin \text{exported}(\mu_2)$  do
4:     Let  $B \subseteq \text{body}(\mu_1)$ , s.t.  $\xi(\text{body}(\mu_2)) = B$ 
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  $\text{prf}$ 
12: return  $\mu_r$ 

```

---

$h_4(e) = n_e$  and since  $e$  is an exported variable, Algorithm 5 removes variable  $e$  from the exported variables of  $\mu_e$  and returns  $\mu'_e$

$$P(i, n, e, c) \wedge H_N(i, d) \leftrightarrow \text{EthDis}(d) \quad (\mu'_e)$$

Algorithm 4 then calls  $\text{modifyBody}$ . The function does not return any repair, since there does not exist any variable repetition in the body of  $\mu_e$ . Hence, Algorithm 4 computes  $\Sigma_1 = \{\mu'_e, \mu_c\}$  and proceeds in the next iteration. The instance  $I_S(\Sigma_1)$  will consist of the following bags

$\beta'_2$  and  $\beta'_3$  with their corresponding homomorphisms shown below:

$$\text{CountyDis}(c, d) \xrightarrow{\langle \mu_c^{-1}, h'_2 \rangle} P(n'_i, n'_n, n_e, *), H_N(n'_i, *)$$

$$\text{EthDis}(d) \xrightarrow{\langle \mu_e^{-1}, h'_3 \rangle} P(n_i, n_n, n'_e, n_c), H_N(n_i, *)$$

$$h'_2 = \{c \mapsto *, d \mapsto *\}$$

$$h'_3 = \{d \mapsto *\}$$

Algorithm 4 terminates, since all bags are safe.

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

**PROOF.** (Sketch) First note that since `srepair` takes as input a partially-safe mapping  $M = (S, T, \Sigma)$ , it follows from Definition 6 that there exists a homomorphism from  $\text{chase}(\Sigma^{-1}, \text{Crt}_T) \setminus \text{Crt}_T$  into  $I_S(\mathcal{V})$ . Furthermore, from Proposition 1, we know that for each  $\mu \in \Sigma$ , there exists a homomorphism from  $\text{body}(\mu)$  into  $I_S(\mathcal{V})$  mapping each  $x \in \text{exported}(\mu)$  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  $\beta$  in  $B_n$ , if there does not exist a bag  $\beta' \in B_n$ , such that  $\beta < \beta'$ , it follows that  $M'$  preserves the privacy of  $M_V$  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  $\beta_n^0, \dots, \beta_n^M$  be the unsafe bags in  $B_n$ . Assume also that for each  $1 \leq l \leq M$ ,  $\beta_n^l$  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  $\mu^l = \text{tgd}(\varepsilon^l)$ , for each  $0 \leq l \leq M$  and let  $\mu_r^l$  be the repaired s-t tgd. Finally, let  $\beta_{n+1}^0, \dots, \beta_{n+1}^N$  be the bags in  $B_{n+1}$ , line 23 of Algorithm 4. Based on the above, in order to show that Theorem 2 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  $(\Sigma \setminus \bigcup_{l=0}^M \mu^l) \cup \bigcup_{l=0}^M \mu_r^l$  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  $\mu^l$ , which leads to an unsafe unification, line 11 of Algorithm 5, is turned into a non-exported variable.  $\square$

By combining Proposition 2 and Theorem 2 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 3.** For each mapping  $M = (S, T, \Sigma)$ , each  $M_V = (S, V, \mathcal{V})$  and each preference function  $\text{prf}$  that always prefers the repairs computed by `hideExported` from the repairs computed by `modifyBody`,

	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.**

Algorithm 2 returns a non-empty mapping that is safe w.r.t.  $M_V$ , if such a mapping exists.  $\square$

**PROOF.** (Sketch) From Algorithm 3, we can see that `repair` 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 `repair`. 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`, show that Proposition 3 holds.  $\square$

## 6 EXPERIMENTS

We investigate the efficiency of our repairing algorithm with the use of hard-coded preference function and with a preference function based on a learning approach. The source code and the experimental scenarios are publicly available at <https://github.com/ucomignani/MapRepair.git>.

We evaluated our algorithm using a set of 3,600 scenarios with each scenario consisting of a set of policy views and a set of s-t tgds. The source schemas and the policy views have been synthetically generated using iBench, the state-of-the-art data integration benchmark [1]. We considered relations of up to five attributes and we created GAV mappings using the iBench configuration recommended by [1]. We generated policy views by applying the iBench operators copy, merging, deletion of attributes and self-join ten times each. 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  $n_{vars}$ .

We implemented our algorithm in Java and we used the Weka library [9] that provides an off-the-shelf implementation of the k-NN algorithm<sup>2</sup>. 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 five runs preceded by one discarded cold run.

### 6.1 Running time of repair

First, we study the impact of the number of s-t tgds and of the number of 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, while, in case of ties, it chooses the repair with the maximum number of joins. We range the number of s-t tgds from 100 to 300 by steps of 50 and the number of body atoms from three to five. The results are shown in Figure (2a). Figure (2a)

<sup>2</sup><https://github.com/ucomignani/MapRepair/blob/master/learning.md>

## Repairing mappings under policy views

shows that the performance of our algorithm is pretty high; the median repairing time is less than 1.5s, while 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.

Figure (2b) shows the time breakdown for repair. The first column 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 32 times greater than time to compute the visible chase and 40 times 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 reduced to five and nine, respectively. Overall, the absolute values of the rewriting times are kept low for these scenarios and gracefully scale while increasing the number of s-t tgds and the number of atoms in their bodies.

### 6.2 Time breakdown between frepair and srepair

Figure (2c) 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 overhead that is incurred during the incremental computation of the visual chase after repairing a s-t tgd (line 23 of Algorithm 4). Figure (2d) shows the correlation between the number of active triggers detected while incrementally computing the visual chase and the running time of srepair for scenarios with 100 s-t tgds using the ANOVA method ( $p$ -value  $< 2.2e^{-16}$ ). Figure (2d) 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 validating its efficiency.

**6.2.1 Evaluating learning accuracy and efficiency.** 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 (i.e., the first repair if  $\Delta_{FV} < 0$ , else the other repair) and in case of ties, it chooses the repair with the maximum number of joins (i.e., the first repair if  $\Delta_J < 0$ , else the second repair).
- $P_{avg}$ , which computes the average value  $\Delta = \frac{\Delta_{FV} + \Delta_J}{2}$  and chooses the first repair, if  $\Delta < 0$ ; otherwise, it chooses the second repair.

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 scenarios from Section 6.1 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) shows the confusion matrix associated to learning  $P_{max}$ . The confusion matrix outlines the choices undertaken during the iterations of the k-NN algorithm. In our case, Table (2a) shows that  $\mu_1$  has been selected 230 times, while  $\mu_2$  has been chosen 395,680 times. We can thus see that  $\mu_2$  is chosen in the vast majority of the cases. Notice that  $\mu_2$  is also the default value in cases where the preference function weights equally  $\mu_1$  and  $\mu_2$ .

Apart from the confusion matrix, we also measured the accuracy of learning the preference function, by weighing the closeness of the learned mapping to the golden standard mapping.

We used the Matthews Correlation Coefficient metric (MCC) [2] 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  $\mu_1$  and  $\mu_2$ ). This measure is calculated using the following:

- $N_{1,1}$  the number of predictions of  $\mu_1$  when  $\mu_1$  is expected
- $N_{2,2}$  the number of predictions of  $\mu_2$  when  $\mu_2$  is expected
- $N_{1,2}$  the number of predictions of  $\mu_1$  when  $\mu_2$  is expected
- $N_{2,1}$  the number of predictions of  $\mu_2$  when  $\mu_1$  is expected

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

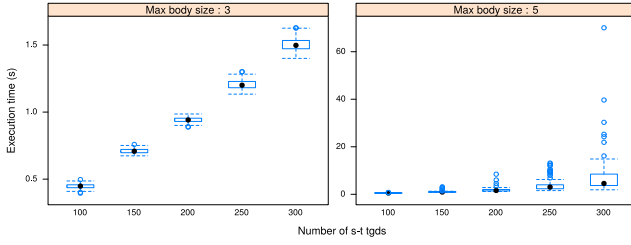
The results of  $MCC$  range from  $-1$  for the cases where the model perfectly predicts the inverse of the expected values, to  $1$  for the cases where the model predicts the expected values. The value  $MCC = 0$  means that there is no correlation between the predicted value and the expected one.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

**Learning  $P_{avg}$ .** Table (2b) 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 ( $< 0.02\%$ ). This error is still acceptable for the learning, since only  $< 0.02\%$  of the predictions are erroneous. This is corroborated by a  $MCC$  value equal to  $0.93$ , thus leading to a still acceptable fit of our preference function also in the case of  $P_{avg}$ .

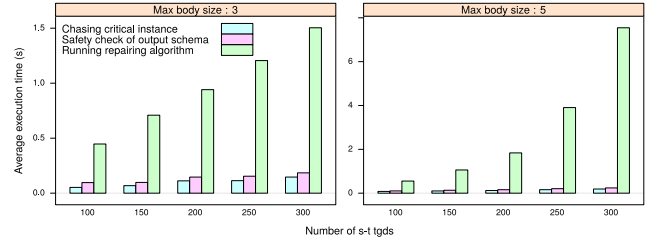
**6.2.2 Running time of repair with a learned preference function.** In the last experiment, we want to measure the impact of learning on the performance of our algorithm. To this end, we compare the running time of repair when adopting a hard-coded preference function (as in the results reported in Figure 2) and when adopting a learned preference function. Figure 3 shows the running times for the same scenarios used in Figure 2. 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.

## 7 CONCLUSION

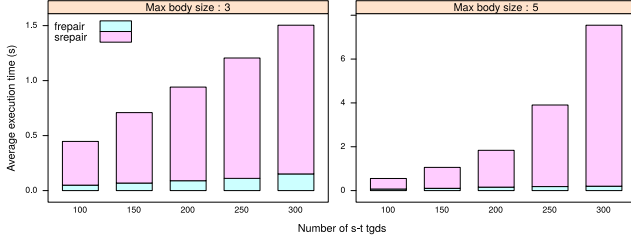
We have addressed the problem of safety checking w.r.t. a set of policy views in a data exchange scenario. We have also proposed efficient repairing algorithms that sanitize the mappings w.r.t. the



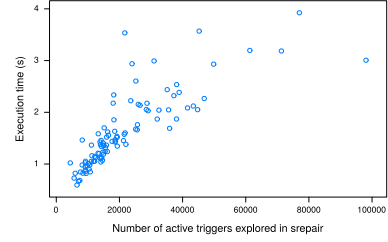
(a) Repairing times.



(b) Time comparisons.



(c) Time breakdown between frepair and repair.



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

Figure 2: Summary of the performance-related experimental results.

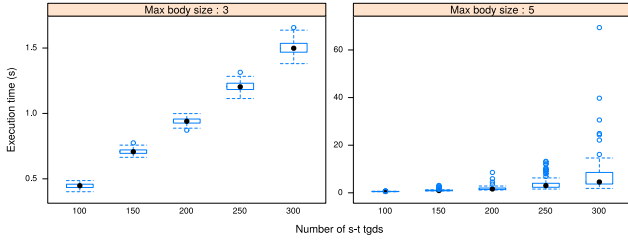


Figure 3: Repairing times with ML.

prediction	golden standard	
	$\mu_1$	$\mu_2$
$\mu_1$	230	0
$\mu_2$	0	395680

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

prediction	golden standard	
	$\mu_1$	$\mu_2$
$\mu_1$	290	1
$\mu_2$	42	395577

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

Table 2: Confusion matrix for the golden standards.

policy views, in cases where the former leak sensitive information. Our approach is inherently data-independent and leads to obtaining rewritings of the mappings guaranteeing privacy preservation at a schema level. As such, our approach is orthogonal to several data-dependent privacy-preservation methods, that can be used on the companion source and target instances to further corroborate the privacy guarantees. We envision several extensions of our work, such as the study of general GLAV mappings and the interplay between data-independent and data-dependent privacy methods.

## REFERENCES

- [1] Patricia C Arocena, Boris Glavic, Radu Ciucanu, and Renée J Miller. 2015. The iBench integration metadata generator. In *VLDB*.
- [2] Pierre Baldi, Soren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. 2000. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics* 16, 5 (2000).
- [3] M. Benedikt, B. Cuenca Grau, and E. Kostylev. 2017. Source Information Disclosure in Ontology-Based Data Integration.. In *AAAI*.
- [4] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *PODS*.
- [5] Joachim Biskup and Piero Bonatti. 2004. Controlled query evaluation for enforcing confidentiality in complete information systems. *International Journal of Information Security* 3, 1 (2004).
- [6] Joachim Biskup and Torben Weibert. 2008. Keeping secrets in incomplete databases. *International Journal of Information Security* 7, 3 (2008), 199–217.
- [7] Piero A. Bonatti, Sarit Kraus, and VS Subrahmanian. 1995. Foundations of secure deductive databases. *IEEE Transactions on Knowledge and Data Engineering* 7, 3 (1995), 406–422.
- [8] 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.
- [9] 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).
- [10] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005).
- [11] Bernardo Cuenca Grau, Evgeny Kharlamov, Egor V. Kostylev, and Dmitriy Zheleznyakov. 2015. Controlled Query Evaluation for Datalog and OWL 2 Profile Ontologies. In *IJCAL*.
- [12] Gerome Miklau and Dan Suciu. 2007. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.* 73, 3 (2007).
- [13] Alan Nash and Alin Deutsch. 2007. Privacy in GLAV Information Integration. In *ICDT*.
- [14] Muhammad I. Sarfraz, Mohamed Nabeel, Jianneng Cao, and Elisa Bertino. 2015. DBMask: Fine-Grained Access Control on Encrypted Relational Databases. In *Proceedings of CODASPY, San Antonio, TX, USA, March 2-4, 2015*. 1–11.
- [15] George L Sicherman, Wiebren De Jonge, and Reind P Van de Riet. 1983. Answering queries without revealing secrets. *TODS* 8, 1 (1983), 41–59.
- [16] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 5 (2002), 557–570.