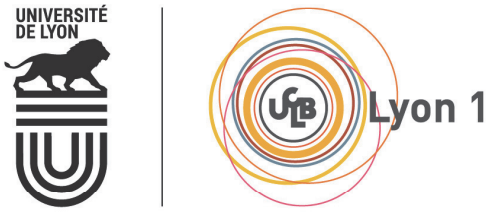
# Interactive mapping specification and repairing in the presence of policy views

Ugo Comignani



N° d'ordre NNT : 2019LYSE1127

**THESE de DOCTORAT DE L'UNIVERSITE DE LYON**
opérée au sein de
**l'Université Claude Bernard Lyon 1**

**Ecole Doctorale 512**
**Ecole Doctorale en Informatique et Mathématiques de Lyon**

**Spécialité de doctorat :**
**Informatique**

Soutenue publiquement le 19/09/2019, par :
**Ugo Comignani**

# Interactive mapping specification and repairing in the presence of policy views

Devant le jury composé de :

| | | |
|---|---|---|
| PICHLER, Reinhard | Professeur, Technische Universität Wien | Rapporteur |
| SENELLART, Pierre | Professeur, École normale supérieure | Rapporteur |
| BERTI-ÉQUILLE, Laure | Directrice de recherche, Aix-Marseille Université | Examinatrice |
| MUGNIER, Marie-Laure | Professeure, Université de Montpellier | Examinatrice |
| KHEDDOUCI, Hamamache | Professeur, Université, Lyon 1 | Examinateur |
| BONIFATI, Angela | Professeure, Université Lyon 1 | Directrice de thèse |
| COQUERY, Emmanuel | Maître de conférence, Université Lyon 1 | Co-directeur de thèse |
| THION, Romuald | Maître de conférence, Université Lyon 1 | Co-directeur de thèse |

# Abstract

Data exchange between sources over heterogeneous schemas is an ever-growing field of study with the increased availability of data, oftentimes available in open access, and the pooling of such data for data mining or learning purposes. However, the description of the data exchange process from a source to a target instance defined over a different schema is a cumbersome task, even for users acquainted with data exchange.

In this thesis, we address the problem of allowing a non-expert user to specify a source-to-target mapping, and the problem of ensuring that the specified mapping does not leak information forbidden by the security policies defined over the source. To do so, we first provide an interactive process in which users provide small examples of their data, and answer simple boolean questions in order to specify their intended mapping. Then, we provide another process to rewrite this mapping in order to ensure its safety with respect to the source policy views.

As such, the first main contribution of this thesis is to provide a formal definition of the problem of interactive mapping specification, as well as a formal resolution process for which desirable properties are proved. Then, based on this formal resolution process, practical algorithms are provided. The approach behind these algorithms aims at reducing the number of boolean questions users have to answers by making use of quasi-lattice structures to order the set of possible mappings to explore, allowing an efficient pruning of the space of explored mappings. In order to improve this pruning, an extension of this approach to the use of integrity constraints is also provided. The second main contribution is a repairing process allowing to ensure that a mapping is "safe" with respect to a set of policy views defined on its source schema, i.e., that it does not leak sensitive information. A privacy-preservation protocol is provided to visualize the information leaks of a mapping, as well as a process to rewrite an input mapping into a safe one with respect to a set of policy views. As in the first contribution, this process comes with proofs of desirable properties. In order to reduce the number of interactions needed with the user, the interactive part of the repairing process is also enriched with the

possibility of learning which rewriting is preferred by users, in order to obtain a completely automatic process.

Last but not least, we present extensive experiments over the open source prototypes built from two contributions of this thesis.

# Résumé

La migration de données entre des sources aux schémas hétérogènes est un domaine en pleine croissance avec l'augmentation de la quantité de données en accès libre, et le regroupement des données à des fins d'apprentissage automatisé et de fouilles. Cependant, la description du processus de transformation des données d'une instance source vers une instance définie sur un schéma différent est un processus complexe même pour un utilisateur expert dans ce domaine.

Cette thèse aborde le problème de la définition de mapping par un utilisateur non expert dans le domaine de la migration de données, ainsi que la vérification du respect par ce mapping des contraintes d'accès ayant été définies sur les données sources. Pour cela, dans un premier temps nous proposons un système dans lequel l'utilisateur fournit un ensemble de petits exemples de ses données, et est amené à répondre à des questions booléennes simples afin de générer un mapping correspondant à ses besoins. Dans un second temps, nous proposons un système permettant de réécrire le mapping produit de manière à assurer qu'il respecte un ensemble de vues de contrôle d'accès définis sur le schéma source du mapping.

Plus précisément, le premier grand axe de cette thèse est la formalisation du problème de la définition interactive de mappings, ainsi que la description d'un cadre formel pour la résolution de celui-ci. Cette approche formelle pour la résolution du problème de définition interactive de mappings est accompagnée de preuves de bonnes propriétés. À la suite de cela, basés sur le cadre formel défini précédemment, nous proposons des algorithmes permettant de résoudre efficacement ce problème en pratique. Ces algorithmes visent à réduire le nombre de questions auxquelles l'utilisateur doit répondre afin d'obtenir un mapping correspondant à ces besoins. Pour cela, les mappings possibles sont ordonnés dans des structures de treillis imbriqués, afin de permettre un élagage efficace de l'espace des mappings à explorer. Nous proposons également une extension de cette approche à l'utilisation de contraintes d'intégrité afin d'améliorer l'efficacité de l'élagage.

Le second axe majeur vise à proposer un processus de réécriture de mapping qui, étant donné un ensemble de vues de contrôle d'accès de référence, permet

d'assurer que le mapping réécrit ne laisse l'accès à aucune information n'étant pas accessible via les vues de contrôle d'accès. Pour cela, nous définissons un protocole de contrôle d'accès permettant de visualiser les informations accessibles ou non à travers un ensemble de vues de contrôle d'accès. Ensuite, nous décrivons un ensemble d'algorithmes permettant la réécriture d'un mapping en un mapping sûr vis-à-vis d'un ensemble de vues de contrôle d'accès. Comme précédemment, cette approche est complétée de preuves de bonnes propriétés. Afin de réduire le nombre d'interactions nécessaires avec l'utilisateur lors de la réécriture d'un mapping, une approche permettant l'apprentissage des préférences de l'utilisateur est proposée, cela afin de permettre le choix entre un processus interactif ou automatique.

L'ensemble des algorithmes décrit dans cette thèse ont fait l'objet d'un prototypage et les expériences réalisées sur ceux-ci sont présentées dans cette thèse.

# Acknowledgements

I want to express my deepest thank to my thesis advisor, Professor Angela Bonifati. Her patience, support and thoughtful guidance have been invaluable all along my Ph.D study. I couldn't have expected a better mentor to grow as a research scientist.

I thank my co-advisors, Doctor Emmanuel Coquery and Doctor Romuald Thion, for their support and advice during these three years.

I also thank Doctor Efthymia Tsamoura for our collaboration, for all her advice and to have allowed me to come to Oxford for an enlightening research stay.

I would also thank Professor Reinhard Pichler and Professor Pierre Senellart for accepting to be my thesis referees, and Professor Laure Berti-Équille, Professor Marie-Laure Mugnier and Professor Hamamache Kheddouci for accepting to be my thesis examiners.

I thank my parents and friends for their constant support during these three years.

Finally, I want to thank my spouse Charlotte Turpin for her love, her indefectible support and the countless sacrifices she has made to help me get to this point.

# Contents

# Introduction

The problem of data exchange is the problem of moving data from an instance under a source schema into another instance defined under a different target schema. This problem is an ever-growing field of study (Mottin *et al.* [MLVP17], Kolaitis *et al.* [Kol18], Paton [Pat19]) particularly in recent years with the increasing availability of large amount of data defined over heterogeneous schemas, and the development of large scale analytics dedicated to explore such high amount of information. Data exchange finds applications in various domains such as business applications, for example the aggregation of multiple customer databases after companies merges and acquisitions, or in the field of healthcare data with, for example, the regrouping of the isolated information from multiple hospitals into large databases shared between health professionals.

The modalities of exchange of the data are usually described through a so-called schema mapping from the source schema to the target schema, taking the form of first-order logical formulas. However, while the specification of such a mapping is a cumbersome task for data curation specialists, it becomes unfeasible for non-expert users, who are unacquainted with the semantics and languages of the involved transformations.

In this thesis, we address the problem of mapping specification by non-expert users. To this extent, we propose the framework illustrated in Figure 1 which focuses on the resolution of two problems induced by the *specification of schema mappings by non-expert users*:

(i) the *specification of a mapping* by using a few simple exemplar tuples to infer the underlying set of tuple-generating dependencies; and iterating the inference process via simple user interactions under the form of boolean queries on the validity of the initial exemplar tuples;

(ii) the *rewriting of specified mappings with respect to sets of reference policy views* in order to prevent any undesired information leakage. In other words, we want to rewrite mappings such that they only expose the information that is safe to expose over *all* instances of the source schema.

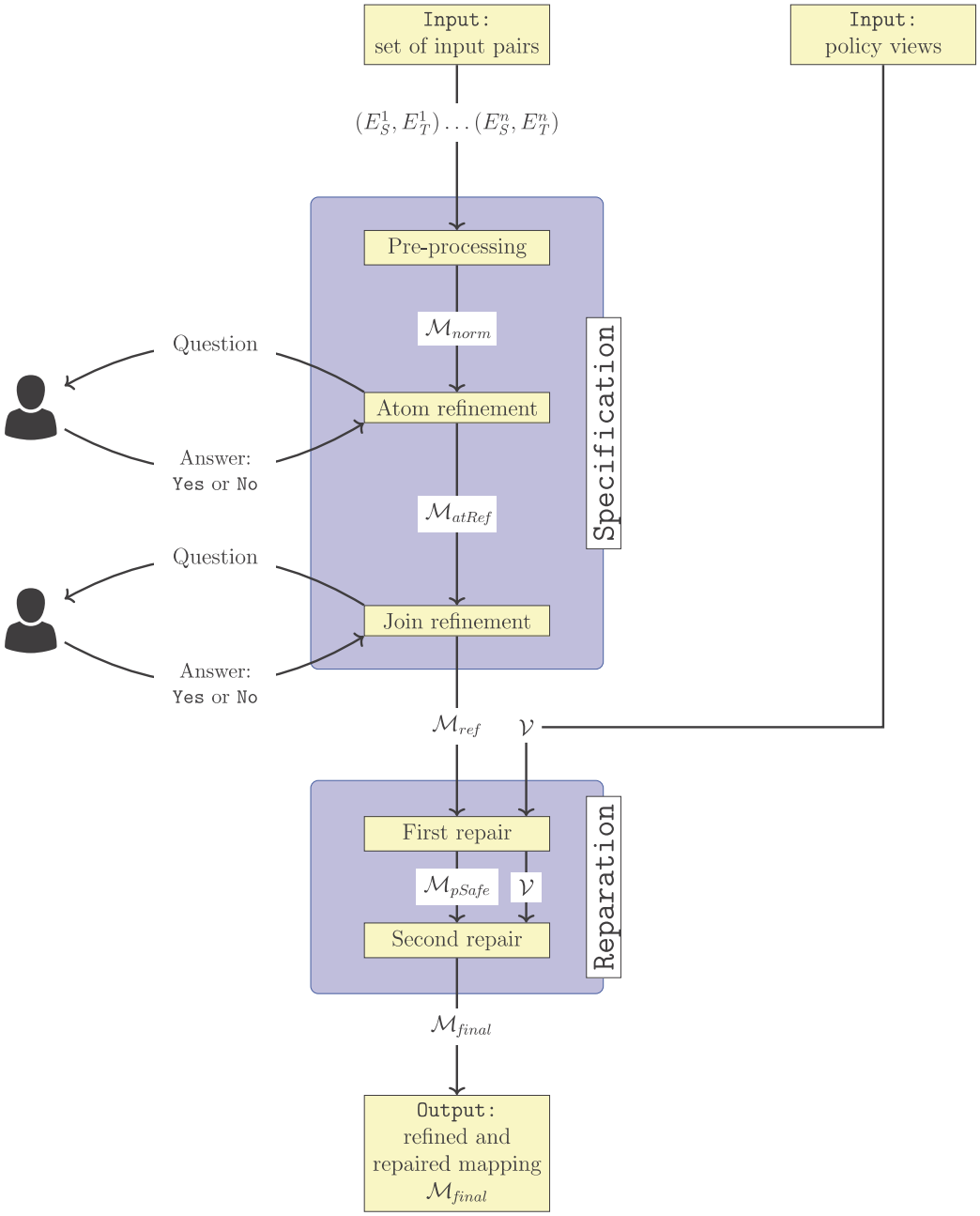



Figure 1: Interactive mapping specification and repairing process.

In order to solve the first problem, we present a quasi-lattice based exploration of the space of all possible mappings that satisfy arbitrary user exemplar tuples. As illustrated in Figure 1, the specification part of our approach uses three steps:

- the first pre-processing step extracts and normalizes $\mathcal{M}_{norm}$, which is the most specific mapping valid with respect to the input exemplar tuples;
- the second step explores the possibly valid atom conjunctions in the tuple-generating dependencies in $\mathcal{M}_{norm}$ in order to produce a partially refined mapping $\mathcal{M}_{atRef}$;
- the third step explores the possibly valid joins between variable occurrences in the tuple-generating dependencies in $\mathcal{M}_{atRef}$ in order to obtain the final refined mapping $\mathcal{M}_{ref}$.

Along the exploration, we challenge users to retain the mappings that fit their requirements best and to dynamically prune the exploration space, thus reducing the number of interactions needed with users. We prove that after the refinement process, the obtained mappings are correct. We also formally define the notion of *fully informative exemplar tuples sets*, which are sets of exemplar tuples exemplifying every dependency expected in the output mapping. Then we prove the completeness of the refinement process if such *fully informative exemplar tuples sets* are provided as input of our framework. Additionally, we provide the worst-case complexity of our framework in terms of the maximal number of interactions with the users. We present an extensive experimental analysis devoted to measuring the feasibility of our interactive mapping specification strategies and the inherent quality of the obtained mappings.

In order to solve the second problem, 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* the input mapping $\mathcal{M}_{ref}$ with respect to a set of policy views $\mathcal{V}$, in cases where the input mapping leaks sensitive information. Finally, we present an extensive experimental analysis devoted to measuring the efficiency of our repairing process in various settings, as well as an evaluation of a learning approach used to simulate the user's preferences during the interactive part of our repairing process.

## Outline

The outline of this thesis is as follows:

**Chapter 1** presents the *interactive mapping specification problem* from a formal point of view. First, we propose a framework to solve this problem as well as the definition of a data example allowing to obtain an output mapping logically equivalent to an expected mapping. Then, we provide proofs of good properties of our framework, including its *completeness* and *correctness*.

**Chapter 2** presents a practical framework to solve the *interactive mapping specification problem* formalized in the preceding chapter. At first, we provide a detailed description of the steps of the practical framework presented in this chapter, as well as proofs of good properties of the mapping output. Next, we show how the introduction of integrity constraints in the IMS problem allows to solve this problem more efficiently, i.e., to reduce the number of candidates mappings to explore.

**Chapter 3** presents a privacy-aware variant of the data exchange problem. In this setting, 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. Under these assumptions, we provide a definition of the safety of mapping under our privacy restrictions and a way to assess a mapping safety with respect to the privacy restrictions defined over its source schema. Then, in case of privacy violations, we provide repairing methods allowing to rewrite an input mapping in a mapping which is safe with respect to the privacy restrictions.

**Chapter 4** presents the experimental study conducted to assess the efficiency of the approaches developed in the previous chapters. These experimentations use the two implemented prototypes `MapSpec` and `MapRepair`.

Finally, we conclude our work by presenting possible directions of future research.

## Publications

The work carried out in this thesis has been published in the following venues:

- **Chapter 1** (*The Interactive Mapping Specification problem*) is based on our article published in the ACM Transactions on Database Systems journal [BCCT19].
- **Chapter 2** (*A practical framework for Interactive Mapping Specification*) is based on our full paper in the proceedings of the ACM SIGMOD'17 Conference [BCCT17] and its journal version published in the ACM Transactions on Database Systems journal [BCCT19].

– **Chapter 3** (*Mapping under policy views*) is based on our demo paper presented in the proceedings of the ACM SIGMOD'19 Conference [BCT19a], and an article currently under submission and available on the arXiv repository [BCT19b].

– **Chapter 4** (*Experimental assessment*) is based on the prototypes and experiments presented in the previously enumerated papers [BCCT17, BCCT19, BCT19a, BCT19b].

# Chapter 1

# The Interactive Mapping Specification problem

In this chapter, we describe the *interactive mapping specification problem* from a formal viewpoint. First, we propose a framework to solve this problem as well as the definition of a class of *exemplar tuples* allowing to obtain an output mapping logically equivalent to an expected mapping. Then, we provide proofs of good properties of our framework, including its *completeness* and *correctness*.

**Chapter organization** In Section 1.1, we introduce some basic notions on data exchange. In Section 1.2, we define the *Interactive Mapping Specification* problem (IMS), and we propose a formal model targeting its resolution. In Section 1.3, we provide proofs of the good properties of our formal model. In Section 1.4, we discuss related work on mapping specification.

## 1.1 Basic Notions

In this section, we give some basic notions from the database literature (Abiteboul *et al.* [AHV95]) and more specifically from the data exchange literature (Fagin *et al.* [FKMP05], Kolaitis *et al.* [Kol05], Arenas *et al.* [ABLM14]), as well as notations that will be used in the following sections.

**Tuple-generating dependencies** To formally define a *tuple-generating dependency*, we first recall some basic definitions in the relational model (Codd [Cod70]). In this model, a schema $\mathbf{S}$ is a nonempty finite set of relation symbols $\{R_1, \ldots, R_n\}$ with each $R_i$ having an arity $n_i \geq 0$. For the sake of readability, to each $n$-ary relation symbol $R$, we associate a set of attributes

$\{Attr_1, \ldots, Attr_n\}$ of cardinality $n$. We use the notation $R(Attr_1, \ldots, Attr_n)$ to denote a relation symbol and its corresponding set of attributes. We also recall that a relational atom $R(\mathrm{t}_1, \ldots, \mathrm{t}_n)$ is an atomic formula where $R$ is an n-ary relation symbol and $\{\mathrm{t}_1; \ldots; \mathrm{t}_n\}$ are terms (i.e., constants, variables or labelled nulls). In the following, a *relational atom* will be simply called an *atom* whenever it is clear from the context.

A *tuple-generating dependency* (tgd for short) is an embedded dependency (Fagin [Fag80]) in which the right and left-hand sides are conjunctions of relational atoms. This is formally defined as follows:

Definition 1.1 (Tuple-generating dependency).
*Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be two schemas.*
*Then a* tuple-generating dependency *from* $\mathbf{S}$ *to* $\mathbf{T}$ *is a first-order logical formula of the form:*

$$\forall \overline{x}, \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$$

*such that :*

- $\overline{x}$ *and* $\overline{y}$ *are vectors of variables;*
- $\phi$ *is a conjunction of relational atoms over relation symbols in* $\mathbf{S}$*;*
- $\psi$ *is a conjunction of relational atoms over relation symbols in* $\mathbf{T}$*.*

In this thesis, we focus on a particular class of *tuple-generating dependencies*, the *source-to-target tuple-generating dependencies* (s-t tgds for short), in which schemas $\mathbf{S}$ and $\mathbf{T}$ are disjoint schemas.

Given a tgd, we define the notions of connected atoms and connected components as follows:

Definition 1.2 (Connected atoms).
*Let* $\sigma : \forall \overline{x}, \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$ *be a tgd.*
*Two atoms* $a_1, a_2 \in \psi(\overline{x}, \overline{y})$ *are* connected *if they have at least one variable from* $\overline{y}$ *in common.*

Definition 1.3 (Connected components).
*Let* $\sigma : \forall \overline{x}, \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$ *be a tgd.*
*Let* $\mathsf{atoms}(\psi(\overline{x}, \overline{y}))$ *be the set of atoms in the conjunction* $\psi(\overline{x}, \overline{y})$*.*
*Then a* connected component *of* $\sigma$ *is a set of atoms* $\mathcal{E} \subseteq \mathsf{atoms}(\psi(\overline{x}, \overline{y}))$ *such that:*

- *the connections between atoms in* $\mathcal{E}$ *form a path between every pair of atoms in* $\mathcal{E}$*;*

– *there is no atom* $a \in \psi(\overline{x}, \overline{y})$ *such that* $a \notin \mathcal{E}$ *and* $a$ *is connected to an atom in* $\mathcal{E}$.

The two previous notions are illustrated in the following example:

*Example* 1.1. *Given a tgd* $\sigma : S(x, y) \rightarrow \exists z, T(x, z) \wedge U(z, y) \wedge V(x)$, *the two connected components of its right-hand side are:* $\{T(x, z); U(z, y)\}$ *as these two atoms are connected by the existential variable* $z$, *and the second connected component* $\{V(x)\}$ *containing only* $V(x)$ *due to the fact that this atom is not connected with another atom.*

**Schema mappings** A *(schema) mapping* (Popa *et al.* [PVH+02], Barcelo [Bar09], Fagin *et al.* [FKMP05], Bonifati *et al.* [BMPV11]) between two databases schemas is a specification of relations between these two schemas. The specification of such mappings is at the core of the resolution of data exchange and data integration problems. While the relation described by a schema mapping can be defined with diverse logical formalisms, here we focus on mappings containing only *source-to-target tgds*.

Thus, the *schema mappings* used in this thesis are formally described as follows:

Definition 1.4 (Schema mapping).
*Let* $\mathbf{S}$ *be a source schema.*
*Let* $\mathbf{T}$ *be a target schema.*
*Let* $\Sigma$ *be a set of* s-t tgds *from* $\mathbf{S}$ *to* $\mathbf{T}$.
*Then a* schema mapping from $\mathbf{S}$ to $\mathbf{T}$ *is a triple* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$.

It should be noted that this triple can be extended to include sets of source constraints ($\Sigma_{\mathbf{S}}$) and target constraints ($\Sigma_{\mathbf{T}}$). However, the interactive specification of such constraints falls beyond the scope of this thesis.

In the Interactive Mapping Specification problem, we consider the class of *GLAV schema mappings*, i.e., *Global-Local-As-View schema mappings* (first introduced in Friedman *et al.* [FLM+99]), which is the class of mappings composed by tgds without limitation in their number of atoms neither in their left-hand side $\phi$ nor in their right-hand side $\psi$.

However, it should be noted that two other classes of schema mappings, the GAV schema mappings (Chawathe *et al.* [CGMH+94], Lenzerini *et al.* [Len02], Garcia-Molina *et al.* [GMUW08]) and LAV schema mappings[1] (Levy *et al.* [LRO96], Lenzerini *et al.* [Len02], Garcia-Molina *et al.* [GMUW08]), have been described:

[1]These two classes are compared in details in Levy [Lev00] and Lenzerini *et al.* [Len02].

Definition 1.5 (Global-As-View (GAV) schema mapping).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a schema mapping.*
$\mathcal{M}$ *is a* Global-As-View schema mapping *if for all s-t tgd* $\sigma \in \Sigma$, $\sigma$ *has the form:*

$$\sigma : \forall \overline{x}, \phi(\overline{x}) \rightarrow U(\overline{x})$$

*where* $U$ *is a relation symbol in* $\mathbf{T}$.

Definition 1.6 (Local-As-View (LAV) schema mapping).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a schema mapping.*
$\mathcal{M}$ *is a* Local-As-View schema mapping *if for all s-t tgd* $\sigma \in \Sigma$, $\sigma$ *has the form:*

$$\sigma : \forall \overline{x}, U(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$$

*where* $U$ *is a relation symbol in* $\mathbf{T}$.

The tgds contained in the *GAV schema mappings* and *LAV schema mappings* are called *GAV tuple-generating dependencies* and *LAV tuple-generating dependencies*, respectively.

**Solutions and mapping equivalence** A solution to a mapping for a given source instance (i.e., an instance that is not allowed to contain labelled nulls) is a target instance (i.e., an instance allowed to contain labelled nulls) which is formally defined as follows:

Definition 1.7 (Solution).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a schema mapping.*
*Let* $I$ *be a source instance over the source schema* $\mathbf{S}$.
*Let* $J$ *be a target instance over the target schema* $\mathbf{T}$.
*Then* $J$ *is a* solution *for* $I$ *under* $\mathcal{M}$ *if, and only if,* $(I, J)$ *satisfies the s-t tgds in* $\Sigma$ *(i.e., if* $(I, J)$ *is a model of* $\Sigma$*, denoted by the notation* $(I, J) \models \Sigma$*).*

We also define the notions of *logical entailment* and *logical equivalence* (Chang *et al.* [CK90]) between mappings as follows:

Definition 1.8 (Logical entailment between two mappings).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *and* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *be two schema mappings.*
*Then* $\mathcal{M}$ logically entails $\mathcal{M}'$ *if, and only if, for every pair of instances* $(I, J)$*:*

$$(I, J) \models \Sigma \text{ implies that } (I, J) \models \Sigma'$$

We denote that a mapping $\mathcal{M}$ *logically entails* a mapping $\mathcal{M}'$ by the notation $\mathcal{M} \models \mathcal{M}'$. When we compare two mappings, we say that $\mathcal{M}$ is *more general* than $\mathcal{M}'$ if $\mathcal{M}$ *logically entails* $\mathcal{M}'$. Informally, this expresses the fact

that on a given instance the tgds in $\mathcal{M}$ can be applied more often than the ones in $\mathcal{M}'$, but never less often.

From the definition of logical entailment of mappings, we derive the definitions of *logical equivalence* between two mappings:

Definition 1.9 (Logical equivalence between two mappings).
*Let $\mathcal{M}$ and $\mathcal{M}'$ be two schema mappings.*
*Then $\mathcal{M}$ and $\mathcal{M}'$ are* logically equivalent *if, and only if:*

$$\mathcal{M} \models \mathcal{M}' \text{ and } \mathcal{M}' \models \mathcal{M}$$

In the following, the notation $\mathcal{M} \equiv \mathcal{M}'$ denotes the *logical equivalence* between two mappings $\mathcal{M}$ and $\mathcal{M}'$.

**Universal solutions and chase procedure** In a data exchange problem, if the source instance $I$ has a solution under the mapping $\mathcal{M}$, then there might exists an infinity of solutions to $I$ under $\mathcal{M}$. This is illustrated in the following example:

*Example* 1.2. *Given a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that:*

$$\Sigma = \{S(x, y) \rightarrow \exists z, T(x, z) \wedge T(z, y)\}$$

*Given a source instance $I = \{S(\mathtt{a}, \mathtt{b})\}$ over $\mathbf{S}$. Then we can construct an infinity of solutions $J_i$, including:*

$J_1 = \{T(\mathtt{a}, \mathtt{b}); T(\mathtt{b}, \mathtt{b})\}$ $\quad$ $J_2 = \{T(\mathtt{a}, \mathtt{a}); T(\mathtt{a}, \mathtt{b})\}$
$J_3 = \{T(\mathtt{a}, \mathtt{n}); T(\mathtt{n}, \mathtt{b})\}$ $\quad$ $J_4 = \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{n}_1, \mathtt{b}); T(\mathtt{n}_2, \mathtt{n}_2)\}$
$J_5 = \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{n}_1, \mathtt{b}); T(\mathtt{a}, \mathtt{n}_2); T(\mathtt{n}_2, \mathtt{b})\}$ $\quad$ ...

It is also worth noting that in our configuration where we consider mappings consisting of source-to-target tgds only, the source instance always have a solution (Fagin *et al.* [FKMP05]).

In this thesis, we rely on the notion of *universal solution* as described by Fagin *et al.* [FKMP05]. Such *universal solutions* are solutions that describe all other solutions for a source instance under a given mapping. To give a formal definition of a *universal solution*, we first need to define the notion of *homomorphism between two instances* as follows:

Definition 1.10 (Homomorphism between instances).
*Let $I$ and $I'$ be two instances.*
*Let $\mathcal{C}$ and $\mathcal{L}$ be two disjoint countably infinite sets of constants and labelled*

*nulls, respectively.*
*Let* $\{\mathtt{e}_1; \ldots; \mathtt{e}_n\}$ *be the set of elements of* $\mathcal{C} \cup \mathcal{L}$ *occurring in* $I$*.*
*Let* $\{\mathtt{e}'_1; \ldots; \mathtt{e}'_n\}$ *be the set of elements of* $\mathcal{C} \cup \mathcal{L}$ *occurring in* $I'$*.*

*A homomorphism from* $I$ *to* $I'$ *is a function* $h$ *from* $\{\mathtt{e}_1, \ldots, \mathtt{e}_n\}$ *to* $\{\mathtt{e}'_1; \ldots; \mathtt{e}'_n\}$ *such that for any tuple* $R(\mathtt{e}_1, \ldots, \mathtt{e}_n)$ *in* $I$*, the tuple* $R(h(\mathtt{e}_1), \ldots, h(\mathtt{e}_n))$ *is in* $I'$*.*

Such a homomorphism is illustrated in the following example:

*Example* 1.3. *Borrowing instances from Example 1.2, as follows:*

$$J_3 = \{T(\mathtt{a}, \mathtt{n}); T(\mathtt{n}, \mathtt{b})\} \qquad J_4 = \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{n}_1, \mathtt{b}); T(\mathtt{n}_2, \mathtt{n}_2)\}$$

*we can exhibit a homomorphism* $h : J_3 \to J_4$ *such that:*

$$h(\mathtt{a}) = \mathtt{a} \qquad h(\mathtt{b}) = \mathtt{b} \qquad h(\mathtt{n}) = \mathtt{n}_1$$

*We can verify that for the tuple* $T(\mathtt{a}, \mathtt{n})$ *in* $J_3$*, the tuple* $T(h(\mathtt{a}), h(\mathtt{n})) = T(\mathtt{a}, \mathtt{n}_1)$ *belongs to* $J_4$*. Analogously, for the tuple* $T(\mathtt{n}, \mathtt{b})$ *in* $J_3$*, the tuple* $T(h(\mathtt{n}), h(\mathtt{b})) = T(\mathtt{n}_1, \mathtt{b})$ *belongs to* $J_4$*.*

Using Definition 1.10, we describe a *universal solution* for an instance under a mapping $\mathcal{M}$ as a solution having a homomorphism into any other solutions. More formally:

Definition 1.11 (Universal solution).
*Let* $I$ *and* $J$ *be two instances.*
*Let* $\mathcal{M}$ *be a schema mapping.*
*Then* $J$ *is a* universal solution *for* $I$ *under* $\mathcal{M}$ *if:*

- $J$ *is a solution for* $I$ *under* $\mathcal{M}$
- *for every instance* $J'$ *which is a solution for* $I$ *under* $\mathcal{M}$*, there exists a homomorphism* $h : J \to J'$

In order to produce such a *universal solution* as result of a data exchange problem, we use the *chase procedure* (Onet [One13]). Precisely, we focus on the *oblivious chase procedure* (Fagin *et al.* [FKMP05], Maier *et al.* [MMS79]). As we focus only on tgds in this chapter, this procedure is defined as follows:

Definition 1.12 (Chase procedure over tgds).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Let* $I$ *be a source instance over* $\mathbf{S}$*.*
*Then, to produce an output instance* $J$ *over* $\mathbf{T}$*, the chase procedure is applied by repeatedly applying the following operation until no new tuples are*

*produced (modulo a renaming of the fresh constants): for each tgd* $\phi(\overline{x}, \overline{y}) \rightarrow \exists \overline{z}, \psi(\overline{x}, \overline{z}) \in \Sigma$*, if there exists a substitution* $\mu$ *of* $\overline{x} \cup \overline{y}$ *such that all atoms in* $\phi(\overline{x}, \overline{y})$ *can be mapped to tuples in* $I$*, then* $\mu$ *is extended to a substitution* $\mu'$ *by picking a new fresh constant for each variable in* $\overline{z}$ *and finally all atoms of* $\psi(\overline{x}, \overline{z})$ *instantiated to tuples with* $\mu'$ *are added to* $J$*.*

In the following, we denote by $\text{CHASE}(\Sigma, I)$ the result of applying the *chase procedure* over a set of tgds $\Sigma$ and on an instance $I$. In our setting with mappings containing only s-t tgds, the chase procedure has been shown to always terminate (the conditions of the termination have been studied in details in the works of Fagin *et al.* [FKMP05], Fagin [Fag83] and Deutsch *et al.* [DT03, DNR08]).

In this chapter, we will also make use of the chase procedure to test the logical implication of mappings. To this extent, we use the left-hand side and right-hand side of tgds as instances, and use the following property:

THEOREM 1.1 (Logical implication test with the chase procedure, Maier *et al.* [MMS79]).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *and* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *be two schema mappings.*
*Then* $\mathcal{M} \models \mathcal{M}'$ *if, and only if:*

$$\forall (\sigma' : \phi_{\sigma'} \rightarrow \psi_{\sigma'}) \in \Sigma', \psi_{\sigma'} \subseteq \text{CHASE}(\Sigma, \phi_{\sigma'})$$

This theorem will find its use in the proof we provide about the desirable properties of our approach.

**Normalization of mappings** In order to define a *normal form* of a mapping, we borrow two notions from Gottlob *et al.* [GPS11]: the *split-reduction* and the *σ-redundancy suppression.*

The *split-reduction* focuses on breaking an initial tgd into a logically equivalent set of tgds such that there is no overlapping over existentially quantified variables. We formally define the *split-reduction* as follows:

DEFINITION 1.13 (split-reduction).
*Let* $\sigma : \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$ *be a tgd.*
*Then* $\sigma$ *is* split-reduced *if there is no pair of tgds:*

$$\sigma_1 : \phi_1(\overline{x}) \rightarrow \exists \overline{y}_1, \psi_1(\overline{x}, \overline{y}_1) \text{ and } \sigma_2 : \phi_2(\overline{x}) \rightarrow \exists \overline{y}_2, \psi(\overline{x}, \overline{y}_2)$$

*such that* $\sigma_1 \not\equiv \sigma_2$ *and* $\{\sigma\} \equiv \{\sigma_1; \sigma_2\}$*.*

*A mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *is split-reduced if, for all tgd* $\sigma \in \Sigma$*,* $\sigma$ *is split-reduced.*

According to Gottlob *et al.* [GPS11], given a mapping $\mathcal{M}$, it is always possible to find a split-reduced mapping $\mathcal{M}'$ that is equivalent to $\mathcal{M}$.

The *split-reduction* is illustrated in the following example:

*Example* 1.4. *We consider a tgd:*

$$\sigma : S(x, y) \rightarrow \exists z, T(x, z) \wedge U(z, y) \wedge V(x)$$

*The* split-reduction *of* $\sigma$ *will lead to separate its connected components into the tgds:*

$$\sigma_1 : S(x, y) \rightarrow \exists z, T(x, z) \wedge U(z, y) \qquad \textit{and} \qquad \sigma_2 : S(x, y) \rightarrow V(x)$$

*It should be noticed that we have the logical equivalence* $\{\sigma\} \equiv \{\sigma_1; \sigma_2\}$*, and that the joins between occurrences of the existential variable* $z$ *have not been broken by the split-reduction.*

The second normalization rule, the *$\sigma$-redundancy suppression*, works at eliminating the *$\sigma$-redundancy* in the mappings. The notion of *$\sigma$-redundancy* is defined as follows:

Definition 1.14 ($\sigma$-redundancy).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a schema mapping and* $\sigma \in \Sigma$ *a tgd.*
*Then* $\mathcal{M}$ *is* $\sigma$-redundant*, with respect to logical equivalence, if and only if* $\Sigma \setminus \{\sigma\} \equiv \Sigma$.

In order to check if a tgd $\sigma \in \Sigma$ is *$\sigma$-redundant* with other tgds in $\Sigma$, we can rely on the *chase procedure* as a proof procedure for the implication problem by checking whether $\Sigma \setminus \{\sigma\} \models \sigma$ (Theorem 1.1 on page 25).

In the following, a mapping will be said in *normal form* if it is both *split-reduced* and does not contain any *$\sigma$-redundant* tgds.

We do not borrow the other normalization rules from Gottlob *et al.* [GPS11], the *core computation*, as it would not lead to improvement of our approach and comes with a non-negligible cost. Indeed, Gottlob *et al.* [GPS11] show in their paper that if we let $\alpha$ denote the maximum arity of the relation symbols in the source and target schemas, and if we let $b$ denote the maximum number of atoms in the tgds to normalize, then the cost of an application of a *core rule* is $O(\alpha b^b)$. For the sake of comparison, the cost of an application of the *split-reduction rule* is $O(\alpha b^2)$.

**Canonical mappings** In order to specify a mapping from a set of pairs of instances, our approach relies on producing the most specific mapping from these pairs, and then refine this over-constrained mapping into a less constrained one that better corresponds to users' needs. Tho this extent, we borrow the notion

of *canonical mapping* from Alexe *et al.* [AtCKT11a], such a mapping being the most specific mapping for a set of pairs of instances.

To formally define such a *canonical mapping*, we need to be able to switch between the tuples used in the instances and the atoms of the canonical mapping's tgds. To this extent, we first define a bijection $\theta$ from constants to variables:

Definition 1.15 (Bijection $\theta$).
*Let $\mathcal{C}$ and $\mathcal{V}$ be two disjoint countably infinite sets of constants and variables, respectively.*
*Then we define a bijection $\theta : \mathcal{C} \to \mathcal{V}$ from constants to variables.*

Then, we extend this bijection to the following bijection $\bar{\theta}$ from tuples to atoms:

Definition 1.16 (bijection $\bar{\theta}$).
*Let $R(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ over $R$ be a tuple.*
*Then:*

$$\bar{\theta}(R(\mathsf{c}_1, \ldots, \mathsf{c}_n)) = R(\theta(\mathsf{c}_1), \ldots, \theta(\mathsf{c}_n))$$

This bijection naturally extends to a bijection from a set of tuples to a conjunction of atoms.

From the bijection $\bar{\theta}$, we define the notion of *canonical tgd* for a pair of instances as follows:

Definition 1.17 (canonical tgd).
*Let $(I, J)$ be a pair of instances.*
*Then a* canonical tgd *for $(I, J)$ is a tgd $\phi \to \psi$ such that $\phi = \bar{\theta}(I)$ and $\psi = \bar{\theta}(J)$.*

A *canonical tgd* for a pair of instances is illustrated in the following example:

*Example* 1.5. *Given a source instance $I = \{R(\mathsf{a}, \mathsf{b}); S(\mathsf{b}, \mathsf{c})\}$ and a target instance $J = \{T(\mathsf{a}, \mathsf{d}); U(\mathsf{d}, \mathsf{e})\}$, the following tgd is a* canonical tgd *for the pair $(I, J)$:*

$$R(a, b) \wedge S(b, c) \to \exists d, e, T(a, d) \wedge U(d, e)$$

Finally, we define a *canonical mapping* for a set of pairs of instances:

Definition 1.18 (canonical mapping).
*Let $\mathcal{S} = \{(I_1, J_1); \ldots; (I_n, J_n)\}$ be a set of pairs of instances such that each instance $I_i$ is defined over $\mathbf{S}$ and each instance $J_i$ is defined over $\mathbf{T}$.*
*Then a* canonical mapping $\mathcal{M}_{can} = (\mathbf{S}, \mathbf{T}, \Sigma_{can})$ *for $\mathcal{S}$ is a mapping such that:*

$$\Sigma_{can} = \{\bar{\theta}(I) \to \bar{\theta}(J) \mid (I, J) \in \mathcal{S}\}$$

For the sake of illustration, we give the following example of a canonical mapping:

*Example* 1.6. *Given a set of pairs of instances $\mathcal{S}$ containing the two following pairs of instances:*

$$(I_1, J_1) = (\{R(\mathtt{a}, \mathtt{b}); S(\mathtt{b}, \mathtt{c})\}, \{T(\mathtt{a}, \mathtt{d}); U(\mathtt{d}, \mathtt{e})\})$$
$$(I_2, J_2) = (\{V(\mathtt{a}, \mathtt{b})\}, \{T(\mathtt{a}, \mathtt{c})\})$$

*then the mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that:*

$$\Sigma = \{R(a, b) \wedge S(b, c) \rightarrow \exists d, e, T(a, d) \wedge U(d, e), \quad V(a, b) \rightarrow \exists c, T(a, c)\}$$

*is a* canonical mapping *for $\mathcal{S}$.*

In the next section, we provide a formal definition of the *Interactive Mapping Specification problem* as well as a model for its resolution.

## 1.2 Formal definitions

In this section we give a formal definition of the *Interactive Mapping Specification problem* (IMS), and we provide a model for its resolution. In order to be able to formally define the *IMS problem*, we will first introduce the notion of *exemplar tuples*, which are pairs of instances used in the input of the *IMS problem*. We also introduce a particular class of exemplar tuples sets allowing the retrieval of a mapping logically equivalent to the mapping expected by the users. Then, we provide the formal definition of the *IMS problem* and a characterization of the set of candidate tgds which is explored during the resolution of the *IMS problem*.

### 1.2.1 Exemplar tuples

In order to specify a mapping with our approach, users are intended to provide examples of the behavior that they expect from this mapping. These provided examples take the form of a *set of exemplar tuples*. An *exemplar tuple for a mapping $\mathcal{M}$* is a pair $(E_S, E_T)$ formed by a source and a target instance. In order to produce an exemplar tuple for the mapping $\mathcal{M}$ they expect, it is not mandatory for the users to provide a target instance $E_T$ which is a universal solution for $E_S$ under $\mathcal{M}$. Instead, given a source instance $E_S$, users can populate $E_T$ with tuples from a subset of the universal solution to $E_S$ under their expected mapping $\mathcal{M}$. This is captured by the following formal definition of an exemplar tuple:

DEFINITION 1.19 (Exemplar tuple).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Then an* exemplar tuple *for* $\mathcal{M}$ *is a pair of instances* $(E_S, E_T)$ *such that:*

$$E_T \neq \emptyset \text{ and } \mu(E_T) \subseteq \text{CHASE}(\Sigma, E_S)$$

*with* $\mu$ *being an homomorphism from the labelled nulls in* $E_T$ *to the labelled nulls in* $\text{CHASE}(\Sigma, E_S)$.

Henceforth, an *exemplar tuple for a mapping* $\mathcal{M}$ will be simply called an *exemplar tuple* whenever the mapping $\mathcal{M}$ is clear from the context. We also introduce the notation $\mathcal{E}$ which will be used to denote a *set of exemplar tuples.* The notion of *exemplar tuple* for a mapping is illustrated in the following example:

*Example* 1.7. *Given a mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *such that:*

$$\begin{aligned}\Sigma = \{&S(x, y) \rightarrow \exists z,\, T(x, z);\\ &S'(x, y) \rightarrow \exists z,\, T(x, z) \wedge T'(z, y)\}\end{aligned}$$

*and given the following source instance over* $\mathbf{S}$*:*

$$E_S = \{S(\mathtt{a}, \mathtt{b}); S(\mathtt{c}, \mathtt{b}); S'(\mathtt{d}, \mathtt{e})\}$$

*Then the following pairs of instances are some of the possible exemplar tuples for* $\mathcal{M}$*:*

$$(E_S, \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{c}, \mathtt{n}_2); T(\mathtt{d}, \mathtt{n}_3); T'(\mathtt{n}_3, \mathtt{e})\}) \tag{1.1}$$
$$(E_S, \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{c}, \mathtt{n}_2)\}) \tag{1.2}$$
$$(\{S(\mathtt{a}, \mathtt{b}); S(\mathtt{c}, \mathtt{b})\}, \{T(\mathtt{a}, \mathtt{n}_1)\}) \tag{1.3}$$

*It should be noted that the exemplar tuples (1.2) and (1.3) only exemplify the tgd of* $\Sigma$*:*

$$S(x, y) \rightarrow \exists z,\, T(x, z)$$

*By opposite, the pair of instances:*

$$({E_S}', {E_T}') = (\{S(\mathtt{a}, \mathtt{b}); S(\mathtt{c}, \mathtt{b})\}, \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{c}, \mathtt{n}_2); T'(\mathtt{n}_3, \mathtt{e})\}) \tag{1.4}$$

*is not an exemplar tuple for* $\mathcal{M}$*. Indeed, the result of chasing* ${E_S}'$ *under* $\mathcal{M}$ *is an instance:*

$$\text{CHASE}(\Sigma, {E_S}') = \{T(\mathtt{a}, \mathtt{n}_1); T(\mathtt{c}, \mathtt{n}_2)\}$$

*for which we see that* ${E_T}' \not\subseteq \text{CHASE}(\Sigma, {E_S}')$ *due to tuple* $T'(\mathtt{n}_3, \mathtt{e})$ *that cannot be deduced from* ${E_S}'$*.*

However, despite providing good properties to approximate the mapping expected by the users, we will show in the Section 1.3.1 that the notion of *exemplar tuple* is not sufficient to ensure that the mapping we output is logically equivalent to the mapping expected by the users. To ensure this property, the provided set of exemplar tuples needs to allow the retrieval of all tgds of the expected mapping. Consequently, the exemplar tuples set provided as input by the users' needs to exemplify every tgd in the expected mapping.

This is formally captured by the following definition of a *fully informative* exemplar tuples set:

Definition 1.20 (Fully informative exemplar tuples set).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping in normal form.*
*Then a* fully informative exemplar tuples set *for* $\mathcal{M}$ *is a set of exemplar tuples* $\mathcal{E}$ *such that each tgd in* $\Sigma$ *is exemplified at least once, i.e.:*

$$\forall \sigma \in \Sigma, \exists (E_S, E_T) \in \mathcal{E}, \exists E'_S \subseteq E_S, (\textsc{chase}(\sigma, E'_S) \neq \emptyset) \\ \wedge (\mu(\textsc{chase}(\sigma, E'_S)) \subseteq E_T)$$

*with* $\mu$ *being an homomorphism from the labelled nulls in* $\textsc{chase}(\sigma, E'_S)$ *to the labelled nulls in* $E_T$*.*

It should be noted that there is no need for the tgds of the expected mapping to be exemplified in separate exemplar tuples. This is illustrated in the following example:

*Example* 1.8. *In this example, we reuse from Example 1.7 the mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *such that:*

$$\Sigma = \{S(x, y) \rightarrow \exists z,\, T(x, z); \\ S'(x, y) \rightarrow \exists z,\, T(x, z) \wedge T'(z, y)\}$$

*and the exemplar tuple (1.3):*

$$(E_S, E_T) = (\{S(\mathtt{a}, \mathtt{b}); S(\mathtt{c}, \mathtt{b})\}, \{T(\mathtt{a}, \mathtt{n}_1)\})$$

*As stated in Example 1.7,* $(E_S, E_T)$ *is an exemplar tuple for* $\Sigma$*. However, the tgd:*

$$S'(x, y) \rightarrow \exists z,\, T(x, z) \wedge T'(z, y)$$

*is not exemplified by this exemplar tuple. Thus, the singleton containing this exemplar tuple is not a fully informative exemplar tuple set as defined in Definition 1.20.*

*A* fully informative exemplar tuple set *for* $\Sigma$ *can be obtained by either adding a new exemplar tuple as in the following set:*

$$\begin{array}{l}\{\ (\{S(\mathtt{a},\mathtt{b});S(\mathtt{c},\mathtt{b})\},\{T(\mathtt{a},\mathtt{n}_1)\});\\ \quad(\{S'(\mathtt{a},\mathtt{b})\},\{T(\mathtt{a},\mathtt{n}_1);T'(\mathtt{n}_1,\mathtt{b})\})\ \}\end{array} \tag{1.5}$$

*or by adding new tuples in the initial exemplar tuple:*

$$(\{S(\mathtt{a},\mathtt{b});S(\mathtt{c},\mathtt{b});S'(\mathtt{d},\mathtt{e})\};\{T(\mathtt{a},\mathtt{n}_1);T(\mathtt{d},\mathtt{n}_2);T'(\mathtt{n}_2,\mathtt{e})\}) \tag{1.6}$$

*At the opposite, the singleton containing the exemplar tuple:*

$$(\{S(\mathtt{a},\mathtt{b});S(\mathtt{c},\mathtt{b});S'(\mathtt{d},\mathtt{e})\},\{T(\mathtt{a},\mathtt{n}_1);T(\mathtt{d},\mathtt{n}_2)) \tag{1.7}$$

*is not a fully informative exemplar tuples set as it lacks a tuple* $T'(\mathtt{n}_1,\mathtt{b})$ *in order to exemplify the connected component in the tgd:*

$$S'(x,y)\rightarrow\exists z,\,T(x,z)\wedge T'(z,y)$$

In the definition of a *fully informative exemplar tuples set* (Definition 1.20 on page 30), the set ${E_S}'$ captures the fact that in an exemplar tuple $(E_S,E_T)$ in a *fully informative exemplar tuples set* for a mapping $\mathcal{M}=(\mathbf{S},\mathbf{T},\Sigma)$, $E_T$ does not need to be equal to the chased instance $\textsc{chase}(\Sigma,E_S)$. We illustrate this in the following example:

*Example* 1.9. *Given the mapping* $\mathcal{M}=(\mathbf{S},\mathbf{T},\Sigma)$ *such that* $\Sigma$ *is a singleton containing the tgd* $\sigma:S(x)\rightarrow T(x)$. *Given the pair of instances:*

$$(E_S,E_T)=(\{S(\mathtt{a});S(\mathtt{b})\},\{T(\mathtt{a})\})$$

*which is an exemplar tuple for* $\Sigma$.

*The result of chasing* $E_S$ *under* $\mathcal{M}$ *leads to the following instance:*

$$\textsc{chase}(\Sigma,E_S)=\{T(\mathtt{a});T(\mathtt{b})\}$$

*Thus, we have* $\textsc{chase}(\Sigma,E_S)\not\subseteq E_T$.

*However, we can exhibit the subset* ${E_S}'=\{S(\mathtt{a})\}$ *such that:*

$$\textsc{chase}(\sigma,{E_S}')=\{T(\mathtt{a})\}\subseteq E_S\ \textit{and}\ \textsc{chase}(\sigma,{E_S}')\neq\emptyset$$

*Hence, according to the definition of a* fully informative exemplar tuples set *(Definition 1.20 on page 30), the exemplar tuple* $(E_S,E_T)$ *exemplifies the tgd* $\sigma$, *despite the fact that* $\textsc{chase}(\Sigma,E_S)\not\subseteq E_T$.

Moreover, the definition of a *fully informative exemplar tuples set* ensures that every connected component in the tgds of the exemplified mapping is illustrated at least once, as shown by the following example:

*Example* 1.10. *Given a mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *such that* $\Sigma$ *is a singleton containing the tgd* $\sigma : S(x, y) \rightarrow \exists z, T(x, z) \wedge T'(z, y)$.
*Given two exemplar tuples for* $\mathcal{M}$*:*

$$(E_{S1}, E_{T1}) = (\{S(\mathtt{a}, \mathtt{b})\}, \{T(\mathtt{a}, \mathtt{n}); T'(\mathtt{n}, \mathtt{b})\})$$
$$(E_{S2}, E_{T2}) = (\{S(\mathtt{a}, \mathtt{b})\}, \{T(\mathtt{a}, \mathtt{n})\})$$

*The result of chasing* $E_{S1}$ *under* $\mathcal{M}$ *leads to the following instance:*

$$\textsc{chase}(\Sigma, E_{S1}) = \{T(\mathtt{a}, \mathtt{n}); T'(\mathtt{n}, \mathtt{b})\}$$

*Thus, we have* $\textsc{chase}(\sigma, E_{S1}) \neq \emptyset$ *and* $\textsc{chase}(\sigma, E_{S1}) \subseteq E_{T1}$*. As* $\sigma$ *is the only tgd in* $\Sigma$*, this means that an* exemplar-tuple set *containing* $(E_{S1}, E_{T1})$ *is* fully informative *for* $\mathcal{M}$*.*

*Analogously, the result of chasing* $E_{S2}$ *under* $\mathcal{M}$ *leads to the following instance:*

$$\textsc{chase}(\sigma, E_{S2}) = \{T(\mathtt{a}, \mathtt{n}); T'(\mathtt{n}, \mathtt{b})\}$$

*However, in this case we have* $\textsc{chase}(\sigma, E_{S2}) \not\subseteq E_{T2}$*. This is due to the fact that the exemplar tuple* $(E_{S2}, E_{T2})$ *does not exemplify the entire connected component in* $\sigma$ *but only one of its atoms (i.e.* $T(x, z)$*). Consequently, an* exemplar-tuple set *containing only* $(E_{S2}, E_{T2})$ *is not* fully informative *for* $\mathcal{M}$*.*

These two notions of *exemplar tuples* and *fully informative exemplar tuples set* constitute the input of the *Interactive Mapping Specification* problem that we will describe in detail in next section.

### 1.2.2 Interactive Mapping Specification

The *Interactive Mapping Specification* problem (IMS) is formally defined as follows:

Definition 1.21 (IMS).
*Let* $\mathcal{M}_{exp}$ *be a mapping expected by the users.*
*Let* $\mathcal{E}$ *be a set of exemplar tuples for* $\mathcal{M}_{exp}$*.*
*Then, the* Interactive Mapping Specification *problem is to discover, by means of boolean interactions (i.e., questions about the validity of exemplar tuples), a mapping* $\mathcal{M}'$ *such that:*

- $\forall (E_S, E_T) \in \mathcal{E}, (E_S, E_T) \models \mathcal{M}'$

– $\mathcal{M}_{exp} \models \mathcal{M}'$

The choice of using exemplar tuples both as input of the IMS and basis of the asked questions comes from the intuition that non-expert users will not be able to express a mapping using a logical language. At the opposite, they can easily rely on their domain knowledge to answer about the validity of an exemplar tuple $(E_S, E_T)$, i.e., to tell whether the information contained in the source instance $E_S$ is sufficient to infer the tuples in the target instance $E_T$. Thus, the question asked to the users about the validity of an exemplar tuple $(E_S, E_T)$ will take the following form:

*"Are the tuples $E_S$ enough to produce $E_T$?"*

Albeit important in practice, handling errors that can be done by users when they answer the questions does not fall under the scope of this thesis. Consequently, we will consider that our questions are answered by an oracle using the following procedure:

Definition 1.22 (*oracle* answering procedure).
*Let $\mathcal{M}_{exp} = \langle S, T, \Sigma_{exp} \rangle$ be the mapping expected by the* oracle.
*Let $E_S$ and $E_T$ be two instances over $\mathbf{S}$ and $\mathbf{T}$, respectively.*
*Then, the* oracle *answers* `true` *to the question:*

*"Are the tuples $E_S$ enough to produce $E_T$?"*

*if:*

$$E_T \subseteq \text{CHASE}(\Sigma_{exp}, E_S)$$

From the definition of the IMS problem, we derive the definition of the set of the candidate tgds which is explored by our framework in order to solve the IMS problem. Intuitively, this set is composed by the tgds that do not lead to break the conditions over the mapping $\mathcal{M}'$ in the definition of the IMS problem. This set is formally defined as follows:

Definition 1.23 (Explored set of candidate tgds).
*Let $\mathcal{E}$ be a set of exemplar tuples.*
*Let $\mathcal{M}_{can} = (\mathbf{S}, \mathbf{T}, \Sigma_{can})$ be the canonical mapping computed from $\mathcal{E}$.*
*Then the* set of candidate tgds *is defined as follows:*

$$\Sigma_{candidates} = \bigcup_{(\phi \rightarrow \psi) \in \Sigma_{can}} \{\phi' \rightarrow \psi' \mid \phi' \neq \emptyset \wedge \psi' \neq \emptyset \wedge (\exists \mu \text{ a morphism such that } \mu(\phi') \subseteq \phi \wedge \mu(\psi') \subseteq \psi)\}$$

From this set, we also define the set $\mathcal{Q}$ of questions that can be asked by our framework during the resolution of the IMS problem:

Definition 1.24 (Set of asked questions).
*Let $\mathcal{E}$ be a set of exemplar tuples.*
*Let $\Sigma_{candidates}$ be the set of tgds explored by our framework for $\mathcal{E}$.*
*Let $\bar{\theta}^{-1}$ be the inverse of the bijection $\bar{\theta}$ from tuples to atoms (Definition 1.16 on page 27).*
*Then the set $\mathcal{Q}$ of questions that can be asked by our framework is the set:*

$$\mathcal{Q} = \{\ \text{“Are the tuples } \bar{\theta}^{-1}(\phi) \text{ enough to produce } \bar{\theta}^{-1}(\psi)\text{?”} \mid (\phi \rightarrow \psi) \in \Sigma_{candidates}\}$$

This set will be explored in order to obtain the desired output mapping. This exploration is done by producing a new mapping given a question, the oracle's answer to this question and the previously inferred mapping. This is expressed by the following transition rule:

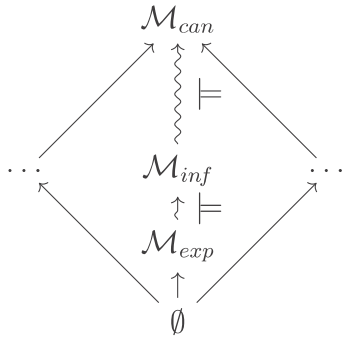
Definition 1.25 (Transition rule).
*Let $\mathcal{M}$ be a mapping.*
*Let $q$ be a question about the validity of the tgd $\phi \rightarrow \psi$ from the set of candidates explored by our framework.*
*Let $answer(q, u)$ be the function asking a question $q$ to a user $u$ and returning her/his answer.*
*Then we have:*

$$\mathcal{M} \xrightarrow{q} \mathcal{M}' \text{ such that: } \texttt{if } answer(q, Oracle) \texttt{ then } \mathcal{M}' = \mathcal{M} \cup (\phi \rightarrow \psi) \texttt{ else } \mathcal{M}' = \mathcal{M}$$

Remark 1.1.
*The process is non-deterministic as, for a given mapping $\mathcal{M}$, any unexplored tgd can be picked up from the the set of candidates to generate a question.*

In order to obtain a solution to an IMS problem, our framework begins with the canonical mapping computed from the set of exemplar tuples provided by the users. Then, this mapping is rewritten iteratively by applying the transition rule from Definition 1.25 over the questions from the set $\mathcal{Q}$ from Definition 1.24. More formally, we define this succession of the application of the transition rule as follows:

Definition 1.26 (Exploration).
*Let $\mathcal{E}$ be a set of exemplar tuples.*
*Let $\mathcal{M}_{can}$ be the canonical mapping computed from $\mathcal{E}$.*




Given a mapping $\mathcal{M}_{exp}$ expected by a user :

$\mathcal{M}_{inf} \models \mathcal{M}_{can}$ is proved in Theorem 1.2

$\mathcal{M}_{exp} \models \mathcal{M}_{inf}$ is proved in Theorem 1.3

Confluence to a mapping $\mathcal{M}_{final}$ is proved in Theorem 1.4

Convergence is proved in Theorem 1.5

If a *fully informative* exemplar tuples set is provided :

$\mathcal{M}_{final} \equiv \mathcal{M}_{exp}$ is proved in Theorem 1.6

Figure 1.1: Summary of the main theorems about our framework (Section 1.3).

*Let $\mathcal{Q}$ be the set of questions that can be asked over $\mathcal{M}_{can}$.*
*Then,* an exploration of the set $\mathcal{Q}$ *is a series of applications of the transition rule:*

$$\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{inf} \text{ such that } \{q_1; \ldots; q_n\} \in \mathcal{Q}$$

The mapping $\mathcal{M}_{inf}$ obtained at the end of an exploration is called an *inferred mapping.*

In the next section, we will prove good properties that are provided by the *inferred mapping.*

## 1.3 Guarantees of the process

In this section, we will prove the correctness of the exploration of a set of questions. We also prove the convergence and confluence properties of an exploration of a set of questions. Finally, we will prove that the completeness of our approach is obtained if the users provide a set of exemplar tuples which is fully informative. These properties are summarized in Figure 1.1.

### 1.3.1 Correctness

In this section, we show that, given a mapping $\mathcal{M}_{exp}$ expected by the users and a canonical mapping $\mathcal{M}_{can}$ derived from an input set $\mathcal{E}$ for $\mathcal{M}_{can}$, the

exploration of the set of questions $\mathcal{Q}$ always leads to an inferred mapping $\mathcal{M}_{inf}$ such that $\mathcal{M}_{inf} \models \mathcal{M}_{can}$ and $\mathcal{M}_{exp} \models \mathcal{M}_{inf}$.

First, we show that the inferred mapping $\mathcal{M}_{inf}$ always imply the canonical mapping $\mathcal{M}_{can}$:

THEOREM 1.2 (Inferred mapping imply the canonical mapping).
*Let* $\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{inf}$ *be an exploration.*
*Then:*

$$\mathcal{M}_{inf} \models \mathcal{M}_{can}$$

*Proof.* This theorem follows from the definition of the transition rule (Definition 1.25 on page 34). By induction over the steps of the exploration we have:

- for $\mathcal{M}_{can} \xrightarrow{q_1} \mathcal{M}_1$, either :
  - if $answer(q_1, Oracle) = \texttt{false}$ we have $\mathcal{M}_1 = \mathcal{M}_{can}$
  - if $answer(q_1, Oracle) = \texttt{true}$, given $\sigma_{q_1}$ the tgd associated with $q_1$, we have $\mathcal{M}_1 = \mathcal{M}_{can} \cup \sigma_{q_1}$.

  Thus, in both cases we have $\mathcal{M}_1 \supseteq \mathcal{M}_{can}$, and consequently $\mathcal{M}_1 \models \mathcal{M}_{can}$.
- for each step of the exploration $\mathcal{M}_i \xrightarrow{q_{i+1}} \mathcal{M}_{i+1}$, analogously to the previous induction step, we find that $\mathcal{M}_{i+1} \supseteq \mathcal{M}_i$, and consequently $\mathcal{M}_{i+1} \models \mathcal{M}_i$.

Thus, by transitivity, we have $\mathcal{M}_{inf} \models \mathcal{M}_{can}$. □

Intuitively, Theorem 1.2 shows that our framework will never output mappings that are less general than the canonical mapping.

In addition, we also show that our framework only produces mappings that are implied by the mapping expected by the users :

THEOREM 1.3 (Expected mapping is a model of the inferred mapping).
*Let* $\mathcal{M}_{exp} = (\mathbf{S}, \mathbf{T}, \Sigma_{exp})$ *be the mapping expected by Oracle.*
*Let* $\mathcal{E}$ *be a set of exemplar tuples for* $\mathcal{M}_{exp}$.
*Let* $\mathcal{M}_{can} = (\mathbf{S}, \mathbf{T}, \Sigma_{can})$ *be the canonical mapping computed from* $\mathcal{E}$.
*Let* $\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{inf}$ *be an exploration.*
*Then:*

$$\mathcal{M}_{exp} \models \mathcal{M}_{inf}$$

*Proof.* The proof is done by induction over the length of an exploration:

– At first, we show that $\mathcal{M}_{exp} \models \mathcal{M}_{can}$. Using the definition of the canonical mappings (Definition 1.18 on page 27), we define the set of tgds $\Sigma_{can}$ of $\mathcal{M}_{can}$ as follows:

$$\Sigma_{can} = \{\bar{\theta}(E_S) \rightarrow \bar{\theta}(E_T) \mid (E_S, E_T) \in \mathcal{E}\}$$

Moreover, according to the definition of the exemplar tuples (Definition 1.19 on page 29), we have:

$$\forall (E_S, E_T) \in \mathcal{E}, E_T \subseteq \textsc{chase}(\Sigma_{exp}, E_S)$$

As $\bar{\theta}$ is an isomorphism, we can do the following substitution:

$$\forall (\bar{\theta}(E_S) \rightarrow \bar{\theta}(E_T)) \in \Sigma_{can}, \bar{\theta}(E_T) \subseteq \textsc{chase}(\Sigma_{exp}, \bar{\theta}(E_S))$$

which means, by the property of the mapping implication from Theorem 1.1 on page 25, that $\mathcal{M}_{exp} \models \mathcal{M}_{can}$.

– We now show that, given an exploration: $\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{inf}$ such that $\mathcal{M}_{exp} \models \mathcal{M}_{inf}$, then any application of the transition rule over $\mathcal{M}_{inf}$ leads to a mapping $\mathcal{M}'_{inf}$ such that $\mathcal{M}_{exp} \models \mathcal{M}'_{inf}$.

Given $\mathcal{M}_{inf}$, the application of a new rewriting rule leads to the following exploration:

$$\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{inf} \xrightarrow{q_{n+1}} \mathcal{M}'_{inf}$$

with $q_{n+1}$ being a question over a tgd $\phi \rightarrow \psi$. By induction over oracle's answer to question $q_{n+1}$, we have :

  – if $answer(q_{n+1}, Oracle) =$ `false` then, using the definition of the transition rule (Definition 1.25 on page 34), we have $\mathcal{M}'_{inf} = \mathcal{M}_{inf}$. Thus, by the induction hypothesis we have : $\mathcal{M}_{exp} \models \mathcal{M}'_{inf}$.
  – if $answer(q_{n+1}, Oracle) =$ `true` then, using the definition of the transition rule, we have $\mathcal{M}'_{inf} = \mathcal{M}_{inf} \cup \{\phi \rightarrow \psi\}$. Also, by definition of the oracle's answering procedure (Definition 1.22 on page 33), if the oracle answer `true` to $q_{n+1}$ then $\psi \subseteq \textsc{chase}(\phi, \mathcal{M}_{exp})$, i.e., $\mathcal{M}_{exp} \models \{\phi \rightarrow \psi\}$. Since we have $\mathcal{M}_{exp} \models \mathcal{M}_{inf}$ by induction hypothesis, we obtain $\mathcal{M}_{exp} \models (\mathcal{M}_{inf} \cup \{\phi \rightarrow \psi\})$ and thus $\mathcal{M}_{exp} \models \mathcal{M}'_{inf}$.

From above, we obtain that $\mathcal{M}_{exp} \models \mathcal{M}_{inf}$ for any mapping $\mathcal{M}_{inf}$ inferred by our framework.

□

This result shows that for an expected mapping $\mathcal{M}_{exp}$, for any instance $I$ and for any mapping $\mathcal{M}_{inf}$ produced by our framework, we have

$$\textsc{chase}(\mathcal{M}_{inf}, I) \subseteq \textsc{chase}(\mathcal{M}_{exp}, I)$$

In other words, the mapping produced by our framework will never produce tuples that would not be produced by the expected mapping.

### 1.3.2 Convergence to a unique mapping

In this section, we show that there exists a mapping to which our framework converges, regardless of the order used to ask the questions, and we build a definition of a *complete exploration* from this.

We first show the confluence of our framework :

Theorem 1.4 (Confluence).
*Let $\mathcal{M}$ be a mapping.*
*Let $\mathcal{M} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_n$ and $\mathcal{M} \xrightarrow{q'_1} \ldots \xrightarrow{q'_m} \mathcal{M}_m$ be two explorations from $\mathcal{M}$.*
*Then there exists a mapping $\mathcal{M}'$ such that we can find two explorations:*

$$\mathcal{M}_n \xrightarrow{q_{n+1}} \ldots \xrightarrow{q_{n+k}} \mathcal{M}' \text{ and } \mathcal{M}_m \xrightarrow{q'_{m+1}} \ldots \xrightarrow{q'_{m+k'}} \mathcal{M}'$$

*Proof.* When a question is asked, the tgd corresponding to this question will be added or not to the inferred mapping, depending on the oracle answer. This process is completely independent from the previously asked questions and does not modify the set of questions that are asked. Therefore, the order in which questions are asked does not influence the result. Thus, it is easy to construct the two sets of questions $\{q_{n+1}; \ldots; q_{n+k}\}$ and $\{q'_{m+1}; \ldots; q'_{m+k}\}$ as follows:

$$\{q_{n+1}; \ldots; q_{n+k}\} = \{q'_1; \ldots; q'_m\} \setminus \{q_1; \ldots; q_n\}$$
$$\{q'_{m+1}; \ldots; q'_{m+k}\} = \{q_1; \ldots; q_n\} \setminus \{q'_1; \ldots; q'_m\}$$

□

We will now show that our framework always converges to a unique mapping :

Theorem 1.5 (Convergence to a unique mapping).
*Let $\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_k} \mathcal{M}_k \xrightarrow{q_{k+1}} \ldots$ be an infinite exploration.*
*Then:*

$$\exists k \in \mathbb{N} \text{ such that } \forall k' \geq k, \mathcal{M}_k \equiv \mathcal{M}_{k'}$$

*Proof.* This follows from the definition of the set $\mathcal{Q}$ of all questions that can be asked (Definition 1.24 on page 34), and from our theorem of confluence (Theorem 1.4). If the whole set of questions is explored, then asking one of these questions one more time, or asking a question isomorphic to a question of set $\mathcal{Q}$, will only lead to an equivalent mapping. □

This last theorem allows us to define the notion of a *complete exploration* for our framework :

Definition 1.27 (Complete exploration).
*Let $\mathcal{M}_{can}$ be a canonical mapping.*
*Let $\mathcal{Q}$ be the set of questions that can be asked over $\mathcal{M}_{can}$.*
*Then, a* complete exploration of the set $\mathcal{Q}$ *is an exploration :*

$$\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{final} \text{ where } \{q_1; \ldots; q_n\} \in \mathcal{Q}$$

*such that:*

$$\forall q \in \mathcal{Q}, \mathcal{M}_{final} \xrightarrow{q} \mathcal{M}_{final}$$

It has been seen that many different explorations can be done to solve the IMS problem, and consequently many mappings can serve as output of our framework. However, intuitively the mapping $\mathcal{M}_{final}$ obtained after a complete exploration is the mapping for which asking a new question to the users does not help to learn more information about the expected mapping, and thus this mapping seems to be the most relevant mapping to return to the users. Consequently, this mapping will be the one that will be computed by the algorithms we propose in the next chapter to solve the IMS problem.

### 1.3.3 Completeness in the presence of fully informative exemplar tuples sets

In this section, we show that, if the users provide a fully informative exemplar tuples set for their expected mapping, then a complete exploration of the set of questions will always lead to a mapping logically-equivalent to the expected mapping. To prove this, we first show that there always exists an *ideal exemplar tuples set* that allows to retrieve a mapping logically equivalent to the expected mapping. Then, we show that a *fully informative exemplar tuples set* always leads to ask the questions asked for the *ideal exemplar tuples set*. Finally, we use these results to prove the completeness of our approach in the presence of *fully informative exemplar tuples sets*.

We first define an *ideal set of exemplar tuples* as follows :

Definition 1.28 (Ideal exemplar tuples set).
*Let $\mathcal{M}$ be a mapping.*
*Let $\mathcal{E}$ be a set of exemplar tuples for $\mathcal{M}$.*
*Then $\mathcal{E}$ is an* ideal exemplar tuples set *if the canonical mapping $\mathcal{M}_{can}$ extracted from $\mathcal{E}$ is such that $\mathcal{M}_{can} \equiv \mathcal{M}$.*

Such an *ideal exemplar tuples set* can be produced for any GLAV mapping, as shown in the following lemma :

Lemma 1.1 (Mapping with ideal exemplar tuples set).
*For all GLAV mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, there exists an ideal exemplar tuples set $\mathcal{E}_{ideal}$.*

*Proof.* From $\mathcal{M}$ we can construct a set:

$$\mathcal{E} = \{(\bar{\theta}^{-1}(\phi), \bar{\theta}^{-1}(\psi))|(\phi \rightarrow \psi) \in \Sigma\}$$

As each exemplar tuple $(E_S, E_T) \in \mathcal{E}$ comes directly from a tgd $\sigma \in \Sigma$, it follows that $E_T \subseteq \text{CHASE}(\sigma, E_S)$. It follows that $\mathcal{E}$ is an exemplar tuples set for $\mathcal{M}$.

Also, as the canonical mapping $\mathcal{M}_{can} = (\mathbf{S}, \mathbf{T}, \Sigma_{can})$ is obtained by applying the morphism $\bar{\theta}$ from tuples to atoms to each exemplar tuple in $\mathcal{E}$. By definition of the exemplar tuples (Definition 1.19 on page 29), we obtain:

$$\begin{aligned}\Sigma_{can} &= \{(\bar{\theta}(\bar{\theta}^{-1}(\phi)) \rightarrow \bar{\theta}(\bar{\theta}^{-1}(\psi)))|(\phi \rightarrow \psi) \in \Sigma\} \\ &= \{(\phi \rightarrow \psi)|(\phi \rightarrow \psi) \in \Sigma\} \\ &= \Sigma\end{aligned}$$

So $\mathcal{M}_{can} \equiv \mathcal{M}$, and thus the set $\mathcal{E}$ is an ideal exemplar tuples set for $\mathcal{M}$. □

From the definition of an *ideal exemplar tuples set* (Definition 1.28) we derive the notion of *ideal questions set*:

Definition 1.29 (Ideal questions set).
*Let $\mathcal{E}_{ideal}$ be an* ideal exemplar tuples set.
*Then an* ideal questions set *for $\mathcal{E}_{ideal}$ is a set of questions :*

$$\mathcal{Q}_{ideal} = \{\text{ “Are the tuples } E_S \text{ enough to produce } E_T\text{?” } | \ (E_S, E_T) \in \mathcal{E}_{ideal}\}$$

Thus, every possible expected mapping for the IMS problem can be represented with the use of an *ideal set of exemplar tuples*. Now, we show that the questions asked about an *ideal set of exemplar tuples* for an expected mapping always leads to retrieve this expected mapping at the end of a complete exploration :

LEMMA 1.2 (An ideal exemplar tuples set leading to the expected mapping).
*Let* $\mathcal{M}_{exp} = (\mathbf{S}, \mathbf{T}, \Sigma_{exp})$ *be the mapping expected by Oracle.*
*Let* $\mathcal{E}$ *be a set of exemplar tuples for* $\mathcal{M}_{exp}$.
*Let* $\mathcal{M}_{can} = (\mathbf{S}, \mathbf{T}, \Sigma_{can})$ *be the canonical mapping computed from* $\mathcal{E}$.
*Let* $\mathcal{E}_{ideal}$ *be the ideal exemplar tuples set for* $\mathcal{M}_{exp}$.
*Let* $\mathcal{Q}_{ideal}$ *be an ideal questions set for* $\mathcal{E}_{ideal}$.
*Then for any complete exploration* $\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_2} \ldots \xrightarrow{q_n} \mathcal{M}_{final}$ *over the set of questions* $\mathcal{Q}_{ideal}$ *:*

$$\mathcal{M}_{final} \equiv \mathcal{M}_{exp}$$

*Proof.* By construction of $\mathcal{E}_{ideal}$ (cf. proof of Lemma 1.1), the *Oracle* will always answer `true` to each question in $\mathcal{Q}$. We also know by the construction of $\mathcal{E}_{ideal}$ that each tgd in $\mathcal{M}_{exp}$ corresponds to one, and only one, pair $(E_S, E_T) \in \mathcal{E}_{ideal}$.

Thus, each application of the transition rule $\mathcal{M}_i \xrightarrow{q} \mathcal{M}_{i+1}$ over a question $q \in \mathcal{Q}$ will add a tgd from $\mathcal{M}_{exp}$ to $\mathcal{M}_i$. At the end of the exploration over $\mathcal{Q}$, we obtain the mapping $\mathcal{M}_{final} = (\mathbf{S}, \mathbf{T}, \Sigma_{final})$ such that $\Sigma_{final} = \Sigma_{can} \cup \Sigma_{exp}$. Thus, $\Sigma_{final} \supseteq \Sigma_{exp}$ and consequently $\mathcal{M}_{final} \models \mathcal{M}_{exp}$.

From Theorem 1.3, we also have that $\mathcal{M}_{exp} \models \mathcal{M}_{final}$, so $\mathcal{M}_{final} \equiv \mathcal{M}_{exp}$. □

Now, in the following lemma we show that a *fully informative exemplar tuples set* always leads to the questions asked for the *ideal exemplar tuples set*:

LEMMA 1.3 (A fully informative exemplar tuples set leading to the ideal exemplar tuples set).
*Let* $\mathcal{M}_{exp}$ *be the mapping expected by Oracle (supposed normalized).*
*Let* $\mathcal{E}_{FI}$ *be a* fully informative exemplar tuples set *for* $\mathcal{M}_{exp}$.
*Let* $\mathcal{M}_{can}$ *be a canonical mapping for* $\mathcal{E}_{FI}$.
*Let* $\mathcal{Q}$ *be the set of questions asked from* $\mathcal{M}_{can}$.
*Let* $\mathcal{E}_{ideal}$ *be the* ideal exemplar tuples set *for* $\mathcal{M}_{exp}$.
*Let* $\mathcal{Q}_{ideal}$ *be an ideal questions set for* $\mathcal{E}_{ideal}$.
*Then we have:*

$$\mathcal{Q}_{ideal} \subseteq \mathcal{Q}$$

*i.e., our framework leads to explore the ideal exemplar tuples set* $\mathcal{E}_{ideal}$.

*Proof.* For each tgd $\sigma = (\phi \rightarrow \psi) \in \mathcal{M}_{exp}$ there exists an exemplar tuple $(E_S, E_T)$ such that:

$$\exists {E_S}' \subseteq E_S \text{ s.t. } (\textsc{chase}(\sigma, {E_S}') \neq \emptyset) \wedge (\textsc{chase}(\sigma, {E_S}') \subseteq E_T)$$

By construction of $\mathcal{M}_{can}$, for each tgd $\sigma = (\phi \rightarrow \psi) \in \mathcal{M}_{exp}$ there exists a tgd $(\phi' \rightarrow \psi') \in \mathcal{M}_{can}$ such that:

$$\exists \phi'' \subseteq \phi' \text{ s.t. } (\textsc{chase}(\sigma, \bar{\theta}^{-1}(\phi'')) \neq \emptyset) \wedge (\textsc{chase}(\sigma, \bar{\theta}^{-1}(\phi'')) \subseteq \bar{\theta}^{-1}(\psi'))$$

Thus, there exists a substitution $\mu$ such that all atoms in $\phi$ can be mapped to atoms in $\phi''$ and an extension $\mu'$ of $\mu$ mapping all atoms of $\psi$ to atoms in $\psi'' = \bar{\theta}(\textsc{chase}(\sigma, \bar{\theta}^{-1}(\phi'')))$. This leads to:

$$\mu(\phi) \subseteq \phi'' \subseteq \phi' \text{ and } \mu'(\psi) \subseteq \psi'' \subseteq \psi'$$

By construction of $\mathcal{E}_{ideal}$, for each tgd $(\phi \rightarrow \psi) \in \mathcal{M}_{exp}$ there exists $(E_S, E_T) \in \mathcal{E}_{ideal}$ such that $\phi = \bar{\theta}(E_S)$ and $\psi = \bar{\theta}(E_T)$. Thus, we can find a tgd $(\phi' \rightarrow \psi') \in \mathcal{M}_{can}$ such that there is a morphism $\mu$ and its extension $\mu'$ such that $\mu(\bar{\theta}(E_S)) \subseteq \phi'$ and $\mu'(\bar{\theta}(E_T)) \subseteq \psi'$.

From this, from the definition of the explored set of candidate tgds (Definition 1.23 on page 33) and from the definition of the set of asked questions (Definition 1.24 on page 34), it follows that for all exemplar tuples $(E_S, E_T) \in \mathcal{E}_{ideal}$ the question about the validity of tgd $\bar{\theta}(E_S) \rightarrow \bar{\theta}(E_T)$ is in the set $\mathcal{Q}$. Equivalently, this is expressed by $\mathcal{Q}_{ideal} \subseteq \mathcal{Q}$. □

From these lemmas we can show that, if a fully informative exemplar tuples set is provided, our framework will output a mapping logically equivalent to the expected one:

Theorem 1.6 (logical equivalence between output mapping and expected mapping).
*Let $\mathcal{M}_{exp}$ be the mapping expected by Oracle.*
*Let $\mathcal{E}_{FI}$ be a fully informative exemplar tuples set for $\mathcal{M}_{exp}$.*
*Let $\mathcal{M}_{can}$ be the canonical mapping computed from $\mathcal{E}_{FI}$.*
*Let $\mathcal{M}_{can} \xrightarrow{q_1} \ldots \xrightarrow{q_n} \mathcal{M}_{final}$ be a complete exploration performed by our framework.*
*Then:*

$$\mathcal{M}_{final} \equiv \mathcal{M}_{exp}$$

*Proof.* In Lemma 1.2 we show that if our framework asks all the questions of the ideal exemplar tuples set for the expected mapping $\mathcal{M}_{exp}$, then the output mapping $\mathcal{M}$ will be such that $\mathcal{M} \equiv \mathcal{M}_{exp}$. In Lemma 1.3 we show that, given a fully informative exemplar tuples set for $\mathcal{M}_{exp}$, then our framework will ask all questions of the ideal exemplar tuples set for $\mathcal{M}_{exp}$. This proves our theorem. □

### 1.3.4 Cardinality of the set of candidates $\mathcal{M}_{candidates}$

In this section, we give the cardinality of the set $\mathcal{M}_{candidates}$ described in Definition 1.23. This cardinality gives an upper bound of the number of questions that can be asked by our framework without introducing optimisation during exploration of the search space. Thus, it corresponds to a case where every possible conjunction (both in left and right-hand sides) and every possible join (with or without creation of existential variables) are explored.

As $\mathcal{M}_{candidates}$ considers every possible join refinement, including the ones producing new existential variables, then for a given variable the whole lattice of partitions of the variable occurrences is explored. We recall that the number of partitions of a set with $n$ elements is given by the Bell number:

$$\mathcal{B}_n = \sum_{k=0}^{n} \left\{ \begin{array}{c} n \\ k \end{array} \right\}$$

with $\left\{ \begin{array}{c} n \\ k \end{array} \right\}$ being the Stirling number of the second kind, i.e. the number of ways to partition a set of n elements into k blocks (Knuth [Knu97]).

Thus, given a variable $v$ with $n$ occurrences in a tgd, we use the Bell number $\mathcal{B}_n$ to compute the number of candidate join partitions explored for $v$. This exploration should be done for every possible conjunction in the tgds of the refined mapping.

Thus, to compute the cardinality of $\mathcal{M}_{candidates}$, given a canonical mapping $\mathcal{M}_{can}$, given $var(\sigma)$ the set of variables in a tgd $\sigma$, and given $n_{v,\sigma}$ the number of occurrences of a variable $v$ in a tgd $\sigma$, we obtain the following formula of the upper bound the number of questions in the worst-case scenario:

$$|\mathcal{M}_{candidates}| = \sum_{\forall(\phi\rightarrow\psi)\in\mathcal{M}_{can}} \left( \sum_{\substack{\forall\phi',\phi'\neq\emptyset \\ \wedge\exists\mu,\mu(\phi')\subseteq\phi}} \left( \sum_{\substack{\forall\psi',\psi'\neq\emptyset \\ \wedge\exists\mu',\mu'(\psi')\subseteq\psi}} \left( \sum_{\forall v\in var(\phi'\rightarrow\psi')} \mathcal{B}_{n_{v,\phi'\rightarrow\psi'}} \right) \right) \right)$$

Despite the high complexity of this worst-case scenario, the experiments in Chapter 4 will show that the number of asked questions stays reasonable for real-world scenarios, as detailed in Chapter 2. It is worth noting that the worst-case complexity is provided for each step of our practical framework in the respective sections in Chapter 2.

## 1.4 Related Work

In this section, we examine the literature related to mapping specification in data exchange.At first, we examine the pioneer data exchange works in the design of mappings. Finally, we provide some connections of our work with mapping learning.

### 1.4.1 Design and refinement of mappings.

The proposed approaches to define mappings are numerous. A frequent but cumbersome way to solve such a problem is to use procedural approaches. To do so, a developer will have the choice to either write dedicated pieces of code to transform her/his data, or rely on commercial systems such as `Altova MapForce` [Alt], `Talend Data Integration` [Tal] and `Pentaho` [Hit]. However, even if such commercial systems rely on graphical user interface to simplify the most trivial mapping definitions, most complex tasks will lead to express the relation between data with the use of programming languages that are hard to use for non expert users.

In the following, we examine in more details the research works that have proposed more user-friendly solutions to solve data exchange problems.

**Graphical design and refinement of mapping: Clio and Muse.** One of the pioneering work to address the problem of helping the specification of mappings in an interactive way is `Clio` (Fagin *et al.* [FHH+09], Miller *et al.* [MHH+01], Popa *et al.* [PVH+02]. In these works, users where intended to specify the schema correspondences with the use of arrows in a graphical user interface. As this formalism exhibits limited expressiveness, multiple mapping can be valid with respect to a unique schema correspondence. To address this problem, `Clio` provides a ranked set of alternative mappings. Meanwhile, the choice between these alternatives must be made by users knowledgeable in the syntax and semantics of the output mappings.

Extending the functionalities of `Clio`, the framework of Yan et al. [YMHF01] is particularly close to our approach as it is one of the first works to consider the usage of data examples in order to help the understanding and refinement of mappings. Precisely, Yan et al. [YMHF01] leverage on the schema correspondences defined in `Clio` to propose alternative data associations among relevant source instances. At each iteration, the mapping designer is intended to choose one of the proposed data associations, leading to construct the output mapping in an incremental fashion.

Thus, unlike `Clio`, our approach does not assume the existence of an initial mapping or knowledge on the syntax and semantics of the languages used to

describe a mapping, as we only rely on simple boolean questions about data examples to specify the user's intended mapping.

A more recent work from Alexe *et al.* [ACMT08] proposes `Muse`, a mapping design tool built on the top of `Clio`. `Muse` allows to use data examples to differentiate between the alternative mappings like the ones provided by `Clio` with the use of an iterative process. When this can seem similar to the work of Yan et al. [YMHF01], the methods proposed by Alexe *et al.* [ACMT08] in `Muse` are more sophisticated than the ones in Alexe *et al.* [YMHF01]. A major difference is that `Muse` is able to infer desired *grouping semantics* instead of using default grouping functions. For instance, `Muse` can infer if the user desires to group flights by airline name and type of plane or only by airline name. Muse also poses a number of yes/no questions to the designer to clarify the grouping semantics. However, the number of questions is driven by the schema elements along with schema constraints that are used to reduce the number of questions. In our approach, we do not assume prior knowledge of the schema constraints. Recently, the work from Atzeni *et al.* [ABPT19] has proposed another approach to help users to define their mapping by building on the creation of a repository of meta-mappings. These meta-mappings are created by generalizing existing mappings. Consequently, to be efficient this approach assumes the availability of a pool of well-defined mappings. Then, given a source and target schemas, the repository of meta-mappings is explored to return the most relevant meta-mapping for these schemas.

**Debugging systems: `TRAMP` and `Vagabond`.** The dichotomy between the target instance expected by a user, and the solution that is output through the mapping has also been addressed from the angle of the debugging of mappings by Chiticariu *et al.* [CT06] with their debugger `Routes` on top of `Clio`. With `Routes`, the user is intended to provide a mapping to debug a source instance. From this, the user can build test cases for the mapping by probing values in the target instance, and then visualise the provenance trace of these values to interpret how and why these values are computed. This approach closely resembles testing as done for software development. By opposite, our method requires as input source and target exemplar tuples, without any prior mapping connecting them.

The suite of tools `TRAMP` (Glavic *et al.* [GAMH10]) and `Vagabond` (Glavic *et al.* [GDM+11]), which is built on top of `TRAMP`, focuses on the understandability of user errors in mappings by using provenance. However, both `TRAMP` and `Vagabond` expect existing mappings as input and the explanations provided by these systems are to be interpreted by users who are familiar with the mapping language and its underlying semantics.

It should be noted that a recent study from Singh *et al.* [SME+17] addresses the problem of finding the best Entity Matching (EM) rules under the form a GBF (General Boolean Formula), thus including disjunctions, conjunctions and negation starting from negative and positive examples. They assume as input two relations of the same arity whose alignment is already known and output the best GBF that takes into account the actual values of the instances and the similarities between them. Their problem is orthogonal to ours, since we work on a different fragment of logic and on data exchange rather than on entity matching.

### 1.4.2 Theoretical limitations in the use of data examples to characterise a mapping

The use of data examples as a tool to describe mappings has begun in ten Cate *et al.* [TCKT10] and Alexe *et al.* [ACKT11]. In these works, the authors have investigated the possibility of uniquely characterizing a schema mapping by means of a set of data examples.

Hence, such characterization using a finite set of universal data examples was shown to be possible only in the case of LAV dependencies and for fragments of GAV dependencies [ACKT11, TCKT10]. As a negative result, Alexe *et al.* [ACKT11] have shown that even simple s-t tgds, such as a copy $E(x, y) \rightarrow F(x, y)$, cannot be characterized by a finite set of universal data examples under the class of GLAV mappings.

Given this impossibility of a unique characterization of GLAV mappings, for a given schema ten Cate *et al.* [TCKT10] and Alexe *et al.* [ACKT11] made the choice of being less specific by characterizing the set of valid "non-equivalent" mappings with respect to the class of GLAV. They also rely on the notion of "most general mapping" for which it was shown that, given a schema mapping problem, such a most general mapping always exists in the class of GLAV mappings if there exists at least one valid mapping for the considered problem (Alexe *et al.* [AtCKT11a]).

### 1.4.3 Learning mappings

In `EIRENE` [AtCKT11b], the authors show how the user can generate a mapping that fits universal data examples given as input. Whereas `EIRENE` expects a set of *universal* data examples, we lift the universality assumption arguing that universal data examples are hard to be produced by a non-expert user. Moreover, as we will show in our experiments in Chapter 4, universal target instances tend to be *significantly larger* than our exemplar tuples. One previous

work targeting non-expert users is MWeaver [QCJ12], where the user is asked to toss tuples in the target instance by fetching constants within the available complete source instance. However, this work has different assumptions with respect to ours: it aims at searching a source sample among all possible samples satisfying the provided target tuples, focusing on GAV mappings only. Our system inspects a few input tuples, on which interactive refinement is enabled, and expressive GLAV mappings can be inferred via simple user feedback.

ten Cate et al. [CDK13] show how computational learning (i.e., the *exact learning model* introduced by D. Angluin [Ang87] and the Probably Approximately Correct model introduced by L. Valiant [Val84, Val13]) can be used to infer mappings from data examples. Their analysis is restricted to GAV schema mappings. Recently, ten Cate *et al.* [tCKQT18] have employed active learning to learn GAV mappings and proved its utility in practice, by proposing the first practical tool for learning schema mappings. Previous work from Gottlob *et al.* [GS10] and ten Cate *et al.* [CKQT17] has focused on a theoretical framework for understanding the relationship between a source instance and a target instance and to provide through a series of repairs the mapping that describes this relationship. They focus on the complexity of the problem for various logical languages and also on the optimality notion. Their setting is quite different from ours as we do not assume complete instances as input and also we do not aim at refining an approximate mapping given as input via reparation operations.

Other approaches handle the specification of mappings as an optimization problem as in the repair frameworks from Gottlob *et al.* [GS10] and ten Cate *et al.* [CKQT17], as a rule selection problem as in Kolaitis *et al.* [KPQ19], or as a learning problem as in the learning and fitting frameworks such as `Beaver` [JBCJ18] or the frameworks proposed in ten Cate *et al.* [CDK13, tCKQT18], Jin *et al.* [JBC$^+$18] and Alexe *et al.* [AtCKT11a, AtCKT11b]. Another similar approach is proposed in the work of Mandreoli *et al.* [Man17] in which the mapping is learned through user queries.

As such, our IMS framework can be considered as a hybrid between the learning-based frameworks and the repair-based frameworks. On one hand, the IMS framework hardcodes mapping optimization and normalization rules such as those adopted in the pre-processing step and consequent atom and join refinement operations that lead to compact and human-understandable mappings. These operations can be considered as optimization choices with a pre-defined and implicit cost model (sigma-redundancy, split-reduction, atom refinement and join refinement). In a sense, a lattice or semi-lattice is an instantiation of the black-box used in the learning framework, in which the user is allowed to navigate the space of possibilities by focusing on two operations,

that correspond to atom and join modifications.

### 1.4.4 Learning queries

Besides mapping specification and learning, researchers have investigated the problem of inferring relational queries (Abouzied *et al.* [AHS12, AAP+13], Bonifati *et al.* [BCS16], Mottin *et al.* [MLVP14]). The works from Abouzied *et al.* [AHS12, AAP+13] focus on learning quantified Boolean queries by leveraging schema information under the form of primary-foreign key relationships between attributes. Their goal is to disambiguate a natural language specification of the query, whereas we use raw tuples to guess the unknown mapping that the user has in mind. In Bonifati *et al.* [BCS16], the problem of inferring join predicates in relational queries is addressed. Consistent equi-join predicates are inferred by questioning the user on a unique denormalized relation. We differ from their work as follows: we focus on mapping specification and consider the broad class of *GLAV mappings* whereas they focus on query specification for a limited fragment of (equi-join) queries. Finally, Mottin *et al.* [MLVP14] presents the exemplar query evaluation paradigm, which rely on exemplar queries to identify a user sample of the desired result of the query and a similarity function to identify database structures that are similar to the user sample. For the latter, the input database is assumed to be known, which is not an assumption in our framework. Since exemplar queries are answered upon an input database, they are considered as unambiguous, whereas this is not necessarily the case in our framework, whose goal is to refine and disambiguate exemplar tuples to derive the unknown mapping that the user has in mind.

## 1.5 Conclusion

In this section, we have provided basic notions in data exchange as well as fundamental definition of our problem. Among these definitions, in order to ensure good properties of our framework output, we have provided the definitions of particular types of data-examples : the *exemplar tuples* and their restriction the *fully informative exemplar tuples set*.

Then, we have proved that if such exemplar tuples are provided as input of our framework, then our framework will converge to a unique mapping in the universe of the possible inferred mappings. We have also shown that this mapping entails the canonical mapping and is entailed by the expected mapping.

Additionally, we have proved that if a *fully informative exemplar tuples set* is provided, then the output mapping will be logically equivalent to the expected mapping.

# Chapter 2

# A practical framework for Interactive Mapping Specification

In this chapter, we present a practical framework to solve the *interactive mapping specification problem*. This framework is built upon the theoretical framework from the previous chapter. At first, we provide a detailed description of the steps of the practical framework presented in this chapter, as well as proofs of good properties of the mapping output. Next, we show how the introduction of integrity constraints in the IMS problem allows to solve this problem more efficiently, i.e., to reduce the number of candidate mappings to explore.

**Chapter organization** In Section 2.1, we introduce some complementary notions used in this chapter. In Section 2.2, we provide an overview of our practical approach and we introduce the running example that will be used along this chapter. In Section 2.3, and Section 2.4 we present the two main steps of our approach. In Section 2.5, we provide proofs of good properties of the mapping output by our practical approach. In Section 2.6, we show how integrity constraints can be used with our approach in order to reduce the number of interactions needed during the specification of a mapping.

## 2.1 Basic Notions

In this section, we give some basic notions from the data exchange literature as well as notation that will be used in the following sections. We only provide notions that have not been presented in Section 1.1 and Section 1.2 of the previous chapter.

**$\psi$-equivalence and $\phi$-equivalence** In order to avoid redundant questions during the resolution of the IMS problem, we need to identify redundancies in the right-hand sides of the tgds of the canonical mapping obtained from the exemplar tuples in input of our framework. To this extent, we define the notion of *$\psi$-equivalence* as follows:

Definition 2.1 ($\psi$-equivalence).
*Let $\sigma_1 : \phi_1(\overline{x}_1) \rightarrow \exists \overline{y}_1, \psi_1(\overline{x}_1, \overline{y}_1)$ and $\sigma_2 : \phi_2(\overline{x}_2) \rightarrow \exists \overline{y}_2, \psi_2(\overline{x}_2, \overline{y}_2)$ be two tgds. If there exists an isomorphism:*

$$\mu : \psi_2(\overline{x}_2, \overline{y}_2) \xrightarrow{\sim} \psi_1(\overline{x}_1, \overline{y}_1)$$

*such that:*

- $\psi_1 \equiv \mu(\psi_2)$
- *$\mu$ matches existential variables in $\overline{y}_2$ only with existential variables in $\overline{y}_1$*
- *$\mu$ matches universal variables in $\overline{x}_2$ only with universal variables in $\overline{x}_1$*

*then $\sigma_1$ and $\sigma_2$ are $\psi$-equivalent.*

We denote the *$\psi$-equivalence* between two tgds $\sigma_1$ and $\sigma_2$ by the following notation: $\sigma_1 \equiv_\psi \sigma_2$.

The notion of $\psi$-equivalence is illustrated in the following example:

*Example* 2.1. *Given two tgds:*

$$S(x, y) \rightarrow \exists z, T(x, z) \wedge V(z, y) \text{ and } U(u, v) \rightarrow \exists w, T(u, w) \wedge V(w, v)$$

*There exists an isomorphism:*

$$\mu : T(x, z) \wedge V(z, y) \xrightarrow{\sim} T(u, w) \wedge V(w, v)$$

*between their right-hand side atom conjunctions, matching the universal variables $x$ and $y$ to the universal variables $u$ and $v$, and the existential variable $z$ with to the existential variable $w$. Thus, these two tgds are $\psi$-equivalent.*

Analogously, we define the *$\phi$-equivalent* between two tgds as follows:

Definition 2.2 ($\phi$-equivalence).
*Let $\sigma_1 : \phi_1(\overline{x}_1) \rightarrow \exists \overline{y}_1, \psi_1(\overline{x}_1, \overline{y}_1)$ and $\sigma_2 : \phi_2(\overline{x}_2) \rightarrow \exists \overline{y}_2, \psi_2(\overline{x}_2, \overline{y}_2)$ be two tgds. If there exists an isomorphism:*

$$\mu : \phi_2(\overline{x}_2) \xrightarrow{\sim} \phi_1(\overline{x}_1)$$

*then $\sigma_1$ and $\sigma_2$ are $\phi$-equivalent.*

We denote the *$\phi$-equivalence* between two tgds $\sigma_1$ and $\sigma_2$ by the following notation: $\sigma_1 \equiv_\phi \sigma_2$.

**Quasi-lattices and partitions** We define a quasi-lattice as a restriction of a complete lattice to a subset of its nodes, included between an upper and a lower bound sets of nodes. This is formally defined as follows:

Definition 2.3 (Quasi-lattice).
*Let $\mathcal{L} = (L, \leq)$ be a lattice.*
*Then a quasi-lattice for $\mathcal{L}$ is a 4-tuples $\mathcal{L}' = (L', \leq, \mathcal{E}_{low}, \mathcal{E}_{up})$ such that:*

- $L' \subseteq L$
- $\forall e \in L', \exists e_{low} \in \mathcal{E}_{low}, \exists e_{up} \in \mathcal{E}_{up},$ *such that* $e_{low} \leq e \leq e_{up}$
- $\forall e_{low} \in \mathcal{E}_{low}, \nexists e'_{low} \in \mathcal{E}_{low}$ *such that* $e'_{low} < e_{low}$
- $\forall e_{up} \in \mathcal{E}_{up}, \nexists e'_{up} \in \mathcal{E}_{up}$ *such that* $e_{up} < e'_{up}$

We also recall a few notions on partitions (Grätzer [Grä02]). A partition of a set $\mathcal{W}$ is a set $\mathcal{P}$ of disjoint and non-empty subsets called *blocks*, such that $\bigcup_{b \in \mathcal{P}} b = \mathcal{W}$. The set of all partitions of $\mathcal{W}$ is denoted by $\mathrm{Part}(\mathcal{W})$. The fact that two objects of $\mathcal{W}$ are in the same block of a partition $\mathcal{P}$ is denoted by $a \equiv_{\mathcal{P}} b$. The set of all partitions of $\mathcal{W}$ forms a complete lattice under the partial order:

$$\mathcal{P}_0 \leq \mathcal{P}_1 \Leftrightarrow \forall x, y \in \mathcal{W}, \left(x \equiv_{\mathcal{P}_0} y \Rightarrow x \equiv_{\mathcal{P}_1} y\right)$$

In the next section, we give an overview of the practical framework we propose to solve the IMS problem.

## 2.2 Overview of the process and running example

The process used to solve the IMS problem is illustrated in Figure 2.1 on page 54. Intuitively, the process goes through the following steps:

- A first normalization step which begins with the generation of a canonical mapping from the input exemplar tuples set. Next, this canonical mapping is normalized as described in Section 1.1.
- A second step focuses on the refinement of the atom conjunctions in the tgds of the produced mapping. To do that, the framework attempts to get rid of the extraneous atoms in the tgds of the mapping produced during the initial normalization step.

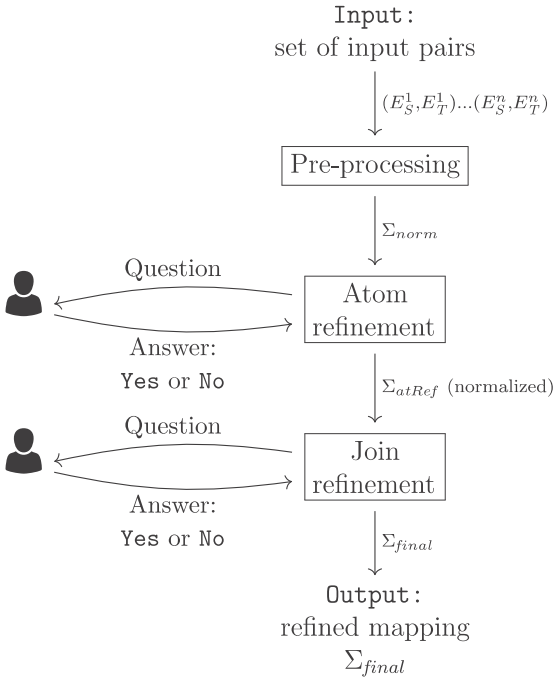



Figure 2.1: Interactive mapping specification process.

– Finally, the framework explores the possible breaking of joins between variable occurrences in the tgds, in order to suppress the irrelevant constraints they convey.

As the mechanisms involved during the pre-processing step have already been detailed in the previous chapter, in the current section we only give a quick illustration of them during the introduction of the running example which is used along this chapter. We also give a quick overview of the atom refinement and join refinement steps that are described in detail in the next two sections.

**Pre-processing step** Our running example is illustrated in Figure 2.2. In this example, for the ease of exposition, we suppose that there are only two exemplar tuples $(E_{S1}, E_{T1})$ and $(E_{S2}, E_{T2})$ in the provided input set. However, in practice, users are likely to provide larger sets of exemplar tuples.

The application of the pre-processing step over the exemplar tuple $(E_{S1}, E_{T1})$ and $(E_{S2}, E_{T2})$ produces a canonical mapping containing the canonical tgds illustrated in Figure 2.2(ii) and 2.2(v) on page 57. Next, the *split-reduction* (Definition 1.13) is applied as illustrated in the following example:

*Example* 2.2. *The split-reduction of the canonical mappings in Figures 2.2(ii) and (v) leads to a set of tgds* $\Sigma_{splitReduced}$. *As the normalization does not affect the left-hand side of the tgds,* $\Sigma_{splitReduced}$ *is populated with tgds whose left-hand sides remain identical with the left-hand sides in the canonical mapping:*

$$\begin{aligned}\phi_1 = \mathit{Flight}(\mathit{idF}_0, \mathit{town}_2, \mathit{town}_1, \mathit{idAir}_0) \wedge \mathit{Flight}(\mathit{idF}_1, \mathit{town}_1, \mathit{town}_2, \mathit{idAir}_1)\\ \wedge\ \mathit{Airl}(\mathit{idAir}_0, \mathit{name}_1, \mathit{town}_1) \wedge \mathit{Airl}(\mathit{idAir}_1, \mathit{name}_2, \mathit{town}_2)\\ \wedge\ \mathit{TA}(\mathit{idAg}, \mathit{name}_3, \mathit{town}_1)\\ \phi_2 = \mathit{Flight}(\mathit{idF}_0, \mathit{town}_2, \mathit{town}_1, \mathit{idAir}_0) \wedge \mathit{Flight}(\mathit{idF}_1, \mathit{town}_1, \mathit{town}_2, \mathit{idAir}_1)\\ \wedge\ \mathit{Airl}(\mathit{idAir}_0, \mathit{name}_1, \mathit{town}_1) \wedge \mathit{Airp}(\mathit{idAp}, \mathit{name}_2, \mathit{town}_2)\\ \wedge\ \mathit{TA}(\mathit{idAg}, \mathit{name}_3, \mathit{town}_1)\end{aligned}$$

*Thus, the split-reduction leads to the following mapping:*

$$\Sigma_{splitReduced} = \{$$
$$\phi_1 \rightarrow \exists idC_2, Co(idC_2, name_3, town_1); \tag{2.1}$$
$$\phi_1 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1); \tag{2.2}$$
$$\phi_1 \rightarrow \exists idC_1, idF_3, Dpt(town_1, idF_3, idC_1) \wedge Arr(town_2, idF_3, idC_1) \wedge Co(idC_1, name_2, town_2); \tag{2.3}$$
$$\phi_2 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1)\} \tag{2.4}$$

Thus, the *split-reduction* allows us to manipulate tgds with small right-hand sides during the refinement of the mapping. Next to the *split-reduction*, the *$\sigma$-redundancy suppression* is applied to the mapping $\Sigma_{splitReduced}$ in order to suppress redundant tgds. We illustrate this step in the following example:

*Example* 2.3. *The $\sigma$-redundancy suppression on $\Sigma_{splitReduced}$ allows to suppress the redundancy induced by tgds (2.2) and (2.3), which are logically equivalent. As the tgd to suppress is chosen arbitrarily, in the following we suppose that the suppressed tgd is tgd (2.3).*

*Moreover, it should be noted that, despite their similar right-hand sides, the $\sigma$-redundancy suppression cannot be applied to tgds (2.2) and (2.4) as their left-hand sides are different.*

*This leads to the normalized mapping $\Sigma_{norm}$:*

$$\Sigma_{norm} = \{$$
$$\phi_1 \rightarrow \exists idC_2, Co(idC_2, name_3, town_1); \tag{2.1}$$
$$\phi_1 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1); \tag{2.2}$$
$$\phi_2 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1)\} \tag{2.4}$$

After this step, the mapping refinement is executed over $\Sigma_{norm}$.

**(i) Source instance $E_{S1}$ :**

*Airl* (*Airline*):

| *IdAirline* | *Name* | *Town* |
|---|---|---|
| airline0 | AAir | L.A. |
| airline1 | MAI | Miami |

*Flight* (*Flight*):

| *IdFlight* | *From* | *To* | *IdAirline* |
|---|---|---|---|
| flight0 | Miami | L.A. | airline0 |
| flight1 | L.A. | Miami | airline1 |

*TA* (*TravelAgency*):

| *IdAgency* | *Name* | *Town* |
|---|---|---|
| ag0 | TC | L.A. |

**(ii) Canonical tgd:**

$$\begin{aligned}&\mathit{Flight}(\mathrm{idF}_0, \mathrm{town}_2, \mathrm{town}_1, \mathrm{idAir}_0)\\ &\quad \wedge \mathit{Flight}(\mathrm{idF}_1, \mathrm{town}_1, \mathrm{town}_2, \mathrm{idAir}_1)\\ &\quad \wedge \mathit{Airl}(\mathrm{idAir}_0, \mathrm{name}_1, \mathrm{town}_1)\\ &\quad \wedge \mathit{Airl}(\mathrm{idAir}_1, \mathrm{name}_2, \mathrm{town}_2)\\ &\quad \wedge \mathit{TA}(\mathrm{idAg}, \mathrm{name}_3, \mathrm{town}_1)\\ &\rightarrow \exists \mathrm{idC}_0, \mathrm{idC}_1, \mathrm{idC}_2, \mathrm{idF}_2, \mathrm{idF}_3,\\ &\qquad \mathit{Co}(\mathrm{idC}_2, \mathrm{name}_3, \mathrm{town}_1)\\ &\qquad \wedge \mathit{Co}(\mathrm{idC}_0, \mathrm{name}_1, \mathrm{town}_1)\\ &\qquad \wedge \mathit{Co}(\mathrm{idC}_1, \mathrm{name}_2, \mathrm{town}_2)\\ &\qquad \wedge \mathit{Dpt}(\mathrm{town}_2, \mathrm{idF}_2, \mathrm{idC}_0)\\ &\qquad \wedge \mathit{Arr}(\mathrm{town}_1, \mathrm{idF}_2, \mathrm{idC}_0)\\ &\qquad \wedge \mathit{Dpt}(\mathrm{town}_1, \mathrm{idF}_3, \mathrm{idC}_1)\\ &\qquad \wedge \mathit{Arr}(\mathrm{town}_2, \mathrm{idF}_3, \mathrm{idC}_1)\end{aligned}$$

**(iii) Target instance $E_{T1}$ :**

*Co* (*Company*):

| *IdCompany* | *Name* | *Town* |
|---|---|---|
| comp0 | AAir | L.A. |
| comp1 | MAI | Miami |
| comp2 | TC | L.A. |

*Dpt* (*Departure*):

| *Town* | *IdFlight* | *IdCompany* |
|---|---|---|
| Miami | flight2 | comp0 |
| L.A. | flight3 | comp1 |

*Arr* (*Arrival*):

| *Town* | *IdFlight* | *IdCompany* |
|---|---|---|
| L.A. | flight2 | comp0 |
| Miami | flight3 | comp1 |

**(iv) Source instance $E_{S2}$ :**

*Airl* (*Airline*):

| *IdAirline* | *Name* | *Town* |
|---|---|---|
| airline0 | AirF | Paris |

*Flight* (*Flight*):

| *IdFlight* | *From* | *To* | *IdAirline* |
|---|---|---|---|
| flight0 | Lyon | Paris | airline0 |
| flight1 | Paris | Lyon | airline1 |

*TA* (*TravelAgency*):

| *IdAgency* | *Name* | *Town* |
|---|---|---|
| ag0 | DT | Paris |

*Airp* (*Airport*):

| *IdAirport* | *Name* | *Town* |
|---|---|---|
| ap0 | SE | Lyon |

**(v) Canonical tgd:**

$$\begin{aligned}&\mathit{Flight}(\mathrm{idF}_0, \mathrm{town}_2, \mathrm{town}_1, \mathrm{idAir}_0)\\ &\quad \wedge \mathit{Flight}(\mathrm{idF}_1, \mathrm{town}_1, \mathrm{town}_2, \mathrm{idAir}_1)\\ &\quad \wedge \mathit{Airl}(\mathrm{idAir}_0, \mathrm{name}_1, \mathrm{town}_1)\\ &\quad \wedge \mathit{TA}(\mathrm{idAg}, \mathrm{name}_3, \mathrm{town}_1)\\ &\quad \wedge \mathit{Airp}(\mathrm{idAp}, \mathrm{name}_2, \mathrm{town}_2)\\ &\rightarrow \exists \mathrm{idC}_0, \mathrm{idF}_2,\\ &\qquad \mathit{Co}(\mathrm{idC}_0, \mathrm{name}_1, \mathrm{town}_1)\\ &\qquad \wedge \mathit{Dpt}(\mathrm{town}_2, \mathrm{idF}_2, \mathrm{idC}_0)\\ &\qquad \wedge \mathit{Arr}(\mathrm{town}_1, \mathrm{idF}_2, \mathrm{idC}_0)\end{aligned}$$

**(vi) Target instance $E_{T2}$ :**

*Co* (*Company*):

| *IdCompany* | *Name* | *Town* |
|---|---|---|
| comp0 | AirF | Paris |

*Dpt* (*Departure*):

| *Town* | *IdFlight* | *IdCompany* |
|---|---|---|
| Lyon | flight2 | comp0 |

*Arr* (*Arrival*):

| *Town* | *IdFlight* | *IdCompany* |
|---|---|---|
| Paris | flight3 | comp0 |

**(vii) Final mapping after refinement:**

$$\begin{aligned}&\Sigma_{final} = \{\\ &\ \mathit{TA}(\mathrm{idAg}, \mathrm{name}_3, \mathrm{town}_1) \rightarrow \exists \mathrm{idC}_2,\ \mathit{Co}(\mathrm{idC}_2, \mathrm{name}_3, \mathrm{town}_1);\\ &\\ &\ \mathit{Flight}(\mathrm{idF}_0, \mathrm{town}_2, \mathrm{town}_1{}', \mathrm{idAir}_0) \wedge \mathit{Airl}(\mathrm{idAir}_0, \mathrm{name}_1, \mathrm{town}_1{}'')\\ &\ \rightarrow \exists \mathrm{idC}_0, \mathrm{idF}_2,\\ &\qquad \mathit{Dpt}(\mathrm{town}_2, \mathrm{idF}_2, \mathrm{idC}_0) \wedge \mathit{Arr}(\mathrm{town}_1{}', \mathrm{idF}_2, \mathrm{idC}_0) \wedge \mathit{Co}(\mathrm{idC}_0, \mathrm{name}_1, \mathrm{town}_1{}'')\\ &\}\end{aligned}$$

Figure 2.2: Running example: exemplar tuples $(E_{S1}, E_{T1})$ and $(E_{S2}, E_{T2})$ (i), (iii), (iv) and (vi), resp.; tgds in the canonical mapping (ii) and (v), and Final mapping (vii).

**Refinement overview** For a given set of exemplar tuples, we have seen in the previous chapter that the number of mappings satisfying it may be large, leading to a big set of questions to ask to the user. Therefore, it is important to provide efficient exploration strategies of the space of mappings in order to reduce the number of questions we need to ask to our users. An important method used here relies on the fact that we can partition the normalized canonical mapping in blocks of $\psi$-equivalent tgds. All tgds in a block of $\psi$-equivalent tgds are handled together to find morphisms between subsets of their left-hand sides. Such morphisms correspond to equivalent tgds that can be extracted from different exemplar tuples, so we need to avoid exploring them more than once in order to reduce the size of the explored space. This is illustrated in the following example:

*Example* 2.4. *During the refinement of our running example, the framework run over the following tgds:*

$$\phi_1 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \quad (2.2)$$

$$\phi_2 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \quad (2.4)$$

*And, during the atom refinement step, the conjunction:*

$$Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1)$$

*is used as a possible left-hand side conjunction refinement for both of the tgds (2.2) and (2.4), leading to only one tgd:*

$$Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \quad (2.5)$$

*Thus, during refinement steps, we want to efficiently spot such logically equivalent tgds in order to ask a question about their validity only once.*

Two successive steps are applied during refinement: the *atom refinement* step and the *join refinement* step. We illustrate such steps in Figure 2.1 on page 54, along with the corresponding user interactions required to obtain the final result, i.e., the refined tgds that meets the user's requirements. The atom refinement step aims at removing unnecessary atoms in the left-hand side of the tgds within the normalized mapping obtained in the pre-processing. The join refinement step applies the removal of unnecessary joins between variable

occurrences in each tgd belonging to the mapping output by the atom refinement step. During both steps, the user is challenged with specific questions devoted to address ambiguities of the provided exemplar tuples and refine the normalized canonical mapping obtained in the pre-processing step. We focus on the atom refinement step in Section 2.3 and we postpone the description of the join refinement step to Section 2.4.

In this chapter, unlike the formal framework presented in Chapter 1 that serves as foundation of our approach, *we focus on universally quantified variables as the targets of the refinement algorithms and assume that the existential variables in the right-hand side of the tgds are unambiguous* (and appear as such in the input exemplar tuples). In other words, value invention (e.g., the production of labelled nulls in `SQL`) in the target exemplar tuples is supposed to be correct and the user is not inquired about them. This also implies that our algorithms *do not create fresh existential variables* in the tgds, as illustrated in the following example:

*Example* 2.5. *Given a tgd:*

$$S(x, y) \wedge U(y, z) \rightarrow T(x, z)$$

*our framework does not consider refinements like:*

$$S(x, y) \rightarrow \exists z, T(x, z) \text{ or } U(y, z) \rightarrow \exists x, T(x, z)$$

*as they lead to transform universally quantified variables into existentially quantified variables.*

This choice comes from the fact that the introduction of such variables would drastically increase the number of mappings to explore, and their coverage would entail non-trivial extensions of our algorithms which are beyond the scope of this work.

## 2.3 Atom refinement step

In this section, we focus on the atom refinement step. As seen in Section 2.2, the normalization produces a *split-reduced* mapping from the canonical mapping in which each tgd has a large left-hand side $\phi$. However, some atoms in $\phi$ may be irrelevant for the users' expected mapping, preventing the triggering of a tgd and consequently preventing the exportation of desirable tuples. To alleviate these ambiguities, we present Algorithm 1 on page 60. This algorithm first groups the refined tgds into a partition of $\psi$-equivalent tgds, and then refines the blocks of this partition.

In the following, we first describe the partition of $\psi$-equivalent tgds which is used as baseline structure for the creation of the quasi-lattices explored by our algorithm. Next, we describe these quasi-lattices and the way they are explored. Finally, we provide formal guarantees for the mapping output by our algorithm as well as an evaluation of the maximum number of asked questions in the worst case.

**Algorithm 1** TgdsAtomRefinement($\Sigma$)

**Input:** A set of tgds $\Sigma$ to be atom refined.
**Output:** A set of tgds $\Sigma'$ where each tgd is atom refined.

1: $\mathcal{P}_\Sigma \leftarrow$ generate partition of $\psi$-equivalent tgds from $\Sigma$
2: $\Sigma' \leftarrow \emptyset$
3: **for all** $b \in \mathcal{P}_\Sigma$ **do**
4: **let** $b$ be $\psi$-equivalent over $\psi_b$
5: $\mathcal{C}_{cand} \leftarrow$ generate set of possible left-hand side candidates from $b$
6: $\mathcal{C}_{valid} \leftarrow$ generate the upper bound of the quasi-lattice over $b$
7: $\mathcal{C}_{invalid} \leftarrow \emptyset$
8: **while** $\mathcal{C}_{cand} \neq \emptyset$ **do**
9: $e \leftarrow$ SelectAtomSet($\mathcal{C}_{cand}, \mathcal{C}_{valid}$)
10: **if** AskAtomSetValidity($e, \psi_b$) **then**
11: add $e$ to $\mathcal{C}_{valid}$
12: remove supersets of $e$ from $\mathcal{C}_{valid}$
13: remove $e$ and its supersets from $\mathcal{C}_{cand}$
14: **else**
15: add $e$ to $\mathcal{C}_{invalid}$
16: remove $e$ and its subsets from $\mathcal{C}_{cand}$
17: **end if**
18: **end while**
19: **for all** $e \in \mathcal{C}_{valid}$ **do**
20: add the tgd $(e \rightarrow \psi)$ to $\Sigma'$
21: **end for**
22: **end for**
23: **return** $\Sigma'$

### 2.3.1 Partition of $\psi$-equivalent tgds

The first step of atom refinement aims at grouping $\psi$-equivalent tgds together. Grouping these tgds together allows to efficiently avoid exploration of redundant questions in the search space. To this end, given $\Sigma_{norm} = \{\sigma_1 \dots \sigma_n\}$ the

set of tgds generated during by the pre-processing step, we create a partition $\mathcal{P}_{norm}$ of $\Sigma_{norm}$ in which each block is constituted by $\psi$-equivalent tgds.

Formally, we define a partition of $\psi$-equivalent tgds as follows:

Definition 2.4 (Partition of $\psi$-equivalent tgds).
*Let $\Sigma = \{\sigma_1 \dots \sigma_n\}$ be a set of tgds.*
*A partition $\mathcal{P}$ of $\Sigma$ is a* partition of $\psi$-equivalent tgds for $\Sigma$ *if:*

$$\forall \sigma_i, \sigma_j \in \Sigma, \sigma_i \equiv_{\mathcal{P}} \sigma_j \Leftrightarrow \sigma_i \equiv_{\psi} \sigma_j$$

Such a partition is illustrated in the following example:

*Example* 2.6. *Continuing our running example, we borrow the set of tgds $\Sigma_{norm}$ from Example 2.3:*

$$\begin{aligned}
\Sigma_{norm} = \{ & \sigma_1 : \phi_1 \rightarrow \exists idC_2, Co(idC_2, name_3, town_1); && (2.1)\\
& \sigma_2 : \phi_1 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\
& \qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1); && (2.2)\\
& \sigma_3 : \phi_2 \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\
& \qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) && (2.4)\\
\}&
\end{aligned}$$

*In these tgds, it can be seen that $\sigma_2$ and $\sigma_3$ are $\psi$-equivalent, contrary to $\sigma_1$. Thus, the partition of $\psi$-equivalent tgds for $\Sigma_{norm}$ is:*

$$\mathcal{P}_{norm} = \{\{\sigma_1\}; \{\sigma_2; \sigma_3\}\}$$

Such a partition serves as baseline structure to create the quasi-lattices that are explored and pruned by our framework, as it will be illustrated in the next section.

### 2.3.2 Quasi-lattice of atom conjunctions

In our framework, given $\mathcal{P}_{norm}$ the partition of $\psi$-equivalent tgds for $\Sigma_{norm}$, we explore each block $\mathcal{B} \in \mathcal{P}_{norm}$ (Algorithm 1 line 3) using a quasi-lattice as baseline structure.

For each block, $\mathcal{B} \in \mathcal{P}_{norm}$ we first define the complete lattice of conjunctions upon which we build our quasi-lattice:

Definition 2.5 (Complete lattice of left-hand side conjunctions).
*Let $\mathcal{B}$ be a block of a partition of $\psi$-equivalent tgds.*

*Let* $\{\phi_1; \ldots; \phi_n\}$ *be the set of the left-hand parts of the tgds in* $\mathcal{B}$.
*Let* $\mathrm{Pow}(\bigcup_{i=1}^{n} \phi_i)$ *be the powerset of the set of all atoms in the left-hand sides of the tgds in* $\mathcal{B}$.
*Let* $\subseteq_h$ *be the inclusion relation with variable renaming through a homomorphism.*
*Then, the complete lattice of left-hand side conjunctions for* $\mathcal{B}$ *is:*

$$\mathcal{L} = (\mathrm{Pow}(\bigcup_{i=1}^{n} \phi_i), \subseteq_h)$$

For all elements $e_x$ and $e_y$ of $\mathrm{Pow}(\bigcup_{i=1}^{n} \phi_i)$ the least-upper bound of the set $\{e_x, e_y\}$ is their union.

Moreover, as atom refinement does not add new constraints in the tgds, it does not create conjunctions which are not subsets of the left-hand side of at least one tgd in $\mathcal{B}$. Hence, we can define the *upper bound* of the quasi-lattice as follows:

Definition 2.6 (Upper bound of a quasi-lattice of conjunctions).
*Let* $\mathcal{B}$ *be a block of a partition of* $\psi$*-equivalent tgds.*
*Let* $\{\phi_1; \ldots; \phi_n\}$ *be the set of the left-hand parts of the tgds in* $\mathcal{B}$.
*Let* $\mathrm{At}(\phi_i)$ *be the set of atoms in the conjunction* $\phi_i$.
*Then the* upper bound *of the explored quasi-lattice is the set:*

$$\{\mathrm{At}(\phi_1); \ldots; \mathrm{At}(\phi_n)\}$$

Thus, during the atom refinement, the explored conjunctions will always be subsets of one of the sets in the upper bound. This restriction takes effect in line 5 and line 6 of Algorithm 1. In particular, at line 6 of Algorithm 1, the whole set of atom conjunctions in the upper bound of the quasi-lattice is placed in the set of valid candidates. This comes from the fact that these atom conjunctions are the most constrained conjunctions that can be explored. Indeed, these conjunctions correspond to the information conveyed by the users' exemplar tuples and, consequently, by the canonical mapping. So, if the users answer 'No' when asked about the validity of these conjunctions, this means that at least one of the pairs of instances provided as input of our process is not a proper exemplar tuples (Definition 1.19 on 29).

We illustrate the upper bound of the quasi-lattices of our running example in the following:

*Example* 2.7. *Considering the tgds in the partition:*

$$\mathcal{P}_{norm} = \{\{\sigma_1\}; \{\sigma_2; \sigma_3\}\}$$

*of Example 2.6 on page 61, the left-hand sides are made of conjunctions $\phi_1$ and $\phi_2$.*

*The elements of the quasi-lattices for $\sigma_2$ and $\sigma_3$ are the subsets of the two following sets of atoms (the sets of atoms in $\phi_1$ for $\sigma_2$, and in $\phi_2$ for $\sigma_3$):*

$$\begin{aligned}\mathrm{At}(\phi_1) = \{&Flight(idF_0, town_2, town_1, idAir_0); Flight(idF_1, town_1, town_2, idAir_1);\\ &Airl(idAir_0, name_1, town_1); Airl(idAir_1, name_2, town_2);\\ &TA(idAg, name_3, town_1)\}\\ \mathrm{At}(\phi_2) = \{&Flight(idF_0, town_2, town_1, idAir_0); Flight(idF_1, town_1, town_2, idAir_1);\\ &Airl(idAir_0, name_1, town_1); Airp(idAp, name_2, town_2);\\ &TA(idAg, name_3, town_1)\}\end{aligned}$$

*It is worth noting that many of the subsets of these two sets are homomorphically equivalent. Such equivalences can be used to leverage common parts of the tgds.*

*For the tgd in the singleton $\{\sigma_1\}$ the* upper bound *of the quasi-lattice is the set of atoms $\mathrm{At}(\phi_1)$, leading to an upper semi-lattice of conjunctions if we consider the lower bound described in the following definition.*

Moreover we recall that, as stated in Section 2.2, our framework does not create new existentially quantified variables in the tgds, we need to prune each set of atoms leading to violate this rule. Recalling that an existential variable in a tgd corresponds to a variable occurring only in the right-hand side (i.e., a variable leading to the creation of new value in the target instance), thus each candidate left-hand side conjunction that does not contain the whole set of right-hand side universal variables will be excluded from the set of candidates. Consequently, we can define the *lower bound* of our quasi-lattices as the set of smallest left-hand side conjunctions containing, at least, all the universal variables of the right-hand side conjunction. More formally:

Definition 2.7 (Lower bound of a quasi-lattice of conjunctions).
*Let $\mathcal{B} = \{\sigma_1; \ldots; \sigma_n\}$ be a block of a partition of $\psi$-equivalent tgds.*
*Let $\mathsf{body}(\sigma)$ be the left-hand part of a tgds $\sigma$.*
*Let $\mathrm{var}(\phi)$ be the set of variables in the conjunction $\phi$.*
*Let $\mathrm{frontier}(\sigma)$ be the set of variables occurring in both sides of the tgd $\sigma$.*
*Let $\subset_h$ be the inclusion relation with variable renaming through a homomorphism.*
*Then the* lower bound *of the explored quasi-lattice is the set:*

$$\begin{aligned}\{\phi \mid &\exists \sigma \in \mathcal{B}, \phi \subseteq \mathsf{body}(\sigma) \wedge \mathrm{frontier}(\sigma) \subseteq \mathrm{var}(\phi)\\ &\wedge (\forall \sigma' \in \mathcal{B}, \nexists \phi' \subseteq \mathsf{body}(\sigma') \textit{ such that } \phi' \subset_h \phi \wedge \mathrm{frontier}(\sigma') \subseteq \mathrm{var}(\phi'))\}\end{aligned}$$

The restriction conveyed by this lower bound takes effect in line 5 of Algorithm 1 where each set which is not a superset of a set in the *lower bound* is pruned.

The *lower bound* of our running example is illustrated in the following example:

*Example* 2.8. *We illustrate the atom refinement on the set of tgds $\Sigma_{norm}$ from Example 2.2 on page 55. As the process stays analogous for each tgd in $\Sigma_{norm}$, we focus on the tgds (2.2) and (2.4) whose right-hand side is:*

$$\begin{aligned}\psi_1 &\equiv_h \psi_2\\ &\equiv_h \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0)\\ &\qquad\qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1)\end{aligned}$$

*The set of universally quantified variables in this conjunction is:*

$$\{town_2, town_1, name_1\}$$

*In order to prevent the creation of new existential variables, a refined tgd needs to contain at least one occurrence of each of these variables in its left-hand side. Consequently, for tgd (2.2) the smallest subsets of the set of atoms* $\mathrm{At}(\phi_1)$ *given in Example 2.7 on page 62 for which this assumption is valid are:*

$$\begin{aligned}&\{Flight(idF_0, town_2, town_1, idAir_0); Airl(idAir_0, name_1, town_1)\},\\ &\{Flight(idF_1, town_1, town_2, idAir_0); Airl(idAir_0, name_1, town_1)\}\\ &\text{and } \{Airl(idAir_0, name_1, town_1); Airl(idAir_1, name_2, town_2)\}\end{aligned}$$

*Analogously, for tgd (2.4), the smallest subsets of the set of atoms* $\mathrm{At}(\phi_2)$ *given in Example 2.7 for which this assumption is valid are:*

$$\begin{aligned}&\{Flight(idF_0, town_2, town_1, idAir_0); Airl(idAir_0, name_1, town_1)\},\\ &\{Flight(idF_1, town_1, town_2, idAir_0); Airl(idAir_0, name_1, town_1)\}\\ &\text{and } \{Airl(idAir_0, name_1, town_1); Airp(idAp, name_2, town_2)\}\end{aligned}$$

*The set containing all of these smallest subsets constitutes the lower bound of our quasi-lattice. It should be noted that this set will contain four elements, as two of the smallest subsets are common between tgd (2.2) and (2.4).*

From these previous definitions, we can now provide an example of the quasi-lattices that are explored during atom refinement of our running example.

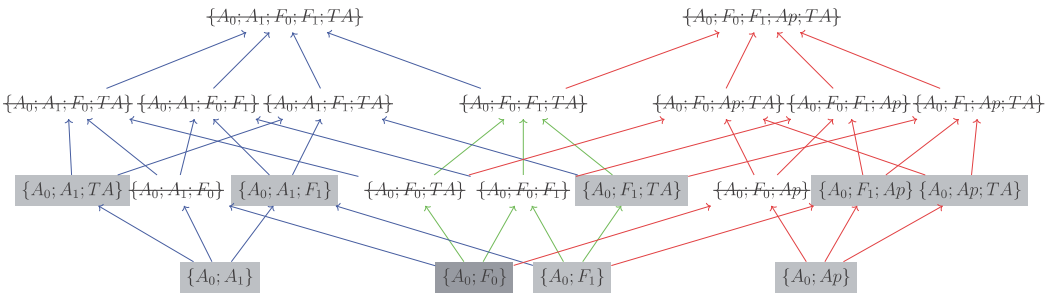



Figure 2.3: Atom sets quasi-lattice on examples 2.9 and 2.10. With atoms: $A_0 = Airl(\text{idAir}_0, \text{name}_1, \text{town}_1)$, $F_0 = Flight(\text{idF}_0, \text{town}_2, \text{town}_1, \text{idAir}_0)$, $A_1 = Airl(\text{idAir}_1, \text{name}_2, \text{town}_2)$, $F_1 = Flight(\text{idF}_1, \text{town}_1, \text{town}_2, \text{idAir}_1)$, $TA = TA(\text{idAg}, \text{name}_3, \text{town}_1)$ and $Ap = Airp(\text{idAp}, \text{name}_2, \text{town}_2)$.

*Example* 2.9. *We recall that we do not allow the creation of new constraints, leading to consider left-hand sides of the tgds (2.2) and (2.4) as the upper bound of our exploration space as shown in Example 2.7 on page 62.*

*We also recall that we do not create existential variables, so each set which is not a superset of a set in the* lower bound *shown in Example 2.8 is pruned.*

*Thus, the search space resulting of this pruning forms the quasi-lattice shown in Figure 2.3 on page 65 with the left side corresponding to sets of atoms specific to tgd (2.2), the right side corresponding to sets of atoms specific to tgd (2.4) and the central part corresponding to common atom sets between (2.2) and (2.4).*

In the next section, we will show how the quasi-lattices can be explored, and especially how the search space can be pruned in order to reduce the number of interactions with the users.

### 2.3.3 Exploring the quasi-lattice

During the exploration of the space of possible candidates for a given quasi-lattice at line 8 of Algorithm 1 on page 60, the user is challenged upon one element of the quasi-lattice at a time. At each iteration, the choice of an element is done using a given exploration strategy, corresponding to the call of SelectAtomSet in line 9 of Algorithm 1. This allows our algorithm to be independent of the chosen strategy, and thus to adapt this strategy to the exemplar tuples or to the kinds of users (e.g., by using a strategy assuming an important number of superfluous tuples in the initial exemplar tuples sets if the system is used by non-expert users).

During the exploration, the user is challenged at line 10 of Algorithm 1 by

being asked a question about the validity of the evaluated element of the quasi-lattice. An important property of the quasi-lattice of atom refinement implies that, once the user validates one of the candidates, then all the supersets of such candidate can be excluded from further exploration, thus effectively pruning the search space. Analogously, once the user invalidates one of the candidates, then all the subsets of such candidate can be excluded from further exploration.

The pruning of the quasi-lattices during their exploration is illustrated in the following example:

*Example* 2.10. *Following previous Example 2.9, we are refining the tgds (2.2) and (2.4) by exploring the quasi-lattice shown in Figure 2.3 on page 65.*

*Assume, for the sake of the example, that we employ a breadth-first bottom-up strategy, starting the exploration of the upper quasi-lattice in Figure 2.3 at its bottom-up level with the set of atoms:*

$$\{Airl(idAir_0, name_1, town_1); Airl(idAir_1, name_2, town_2)\},$$
$$\{Airl(idAir_0, name_1, town_1); Flight(idF_0, town_2, town_1, idAir_0)\},$$
$$\{Airl(idAir_0, name_1, town_1); Flight(idF_1, town_1, town_2, idAir_1)\}$$
$$\text{and } \{Airl(idAir_0, name_1, town_1); Airp(idAp, name_2, town_2)\}$$

*The user is asked about the validity of the set corresponding to the bottom left light gray box of Figure 2.3:*

$$\{Airl(idAir_0, name_1, town_1); Airl(idAir_1, name_2, town_2)\}$$

*with the following question:*

"Are the tuples:
$Airl(\texttt{airline0}, \texttt{AAir}, \texttt{L.A.})$ and $Airl(\texttt{airline1}, \texttt{MAI}, \texttt{Miami})$
enough to produce:
$Dpt(\texttt{Miami}, \texttt{flight2}, \texttt{comp0})$, $Arr(\texttt{L.A.}, \texttt{flight2}, \texttt{comp0})$
and $Co(\texttt{comp0}, \texttt{AAir}, \texttt{L.A.})$?"

*We can observe that a positive answer implies an inconsistency with respect to the application domain, namely that* the second flight company is based in the same town as the departure of the flight*, which is not the case in real-world examples. Hence, the user will be likely to answer 'No' to the above question.*

*Next, now assume that Algorithm 1 proceeds with:*

$$\{Airl(idAir_0, name_1, town_1); Flight(idF_0, town_2, town_1, idAir_0)\}$$

*This atom set is common between the tgds (2.2) and (2.4), consequently we can use tuples from $(E_S^1, E_T^1)$ or $(E_S^2, E_T^2)$ to generate the question. Here we take tuples from $(E_S^1, E_T^1)$, leading to the following question:*

"Are the tuples:
$Flight(\texttt{flight0}, \texttt{Miami}, \texttt{L.A.}, \texttt{airline0})$ and $Airl(\texttt{airline0}, \texttt{AAir}, \texttt{L.A.})$
enough to produce:
$Dpt(\texttt{Miami}, \texttt{flight2}, \texttt{comp0})$, $Arr(\texttt{L.A.}, \texttt{flight2}, \texttt{comp0})$
and $Co(\texttt{comp0}, \texttt{AAir}, \texttt{L.A.})$?"

*Assuming that the user will answer 'Yes' to this question, the supersets of:*

$$\{Airl(idAir_0, name_1, town_1); Flight(idF_0, town_2, town_1, idAir_0)\}$$

*will be pruned (crossed out boxes of Figure 2.3) and the following tgd will be output by the algorithm:*

$$\begin{aligned} &Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ &\qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \end{aligned} \tag{2.6}$$

*We continue the exploration of the current level with sets:*

$$\{Airl(idAir_0, name_1, town_1); Flight(idF_1, town_1, town_2, idAir_1)\}$$
$$and\ \{Airl(idAir_0, name_1, town_1); Airp(idAp, name_2, town_2)\}$$

*Assuming that the user does not validate these sets, they will be finally challenged about the last available sets on the next level of the quasi-lattice, namely on the sets:*

$\{Airl(idAir_0, name_1, town_1); Airl(idAir_1, name_2, town_2);$
$TA(idAg, name_3, town_1)\},$
$\{Airl(idAir_0, name_1, town_1); Airl(idAir_1, name_2, town_2);$
$Flight(idF_1, town_1, town_2, idAir_1)\},$
$\{Airl(idAir_0, name_1, town_1); Flight(idF_1, town_1, town_2, idAir_1);$
$TA(idAg, name_3, town_1)\},$
$\{Airl(idAir_0, name_1, town_1); Flight(idF_1, town_1, town_2, idAir_1);$
$Airp(idAp, name_2, town_2)\}$

*and*

$\{Airl(idAir_0, name_1, town_1); Airp(idAp, name_2, town_2); TA(idAg, name_3, town_1)\}$

*which are also labelled as invalid. In the end, for the combination of tgds (2.2) and (2.4), Algorithm 1 will output the single tgd (2.6).*

*Analogously, the tgd:*

$$\phi_1 \rightarrow \exists idC_2, Co(idC_2, name_3, town_1); \tag{2.1}$$

*which is exemplified in the exemplar tuple* $(E_S^1, E_T^1)$ *of our scenario (but not in* $(E_S^2, E_T^2)$*) will lead to the following tgd after refinement:*

$$TA(idAg, name_3, town_1) \rightarrow \exists idC_2, Co(idC_2, name_3, town_1)$$

*Thus, the application of Algorithm 1 over the normalized mapping* $\Sigma_{norm}$ *will result in the following mapping* $\Sigma_{atRef}$*:*

$$\begin{aligned}
\Sigma_{atRef} &= \textsc{TgdsAtomRefinement}(\Sigma_{norm}) \\
&= \{ \\
&\quad Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \\
&\qquad \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr(town_1, idF_2, idC_0) \\
&\qquad\quad \wedge Co(idC_0, name_1, town_1); \qquad (2.6) \\
&\quad TA(idAg, name_3, town_1) \rightarrow \exists idC_2, Co(idC_2, name_3, town_1) \qquad (2.7) \\
&\} \\
\end{aligned}$$

### 2.3.4 Questioning about atoms set validity

In the atom refinement algorithm, the user is challenged on the validity of the left-hand side atoms of the canonical mapping at line 10 of Algorithm 1 on page 60 with the use of questions as those shown in Example 2.10 on page 66.

We recall that the questions asked have been formally defined in the Definition 1.24 on page 34. In order to evaluate the validity of a given conjunction $\phi$ during the exploration of a quasi-lattice, we build on the correspondence between atoms in $\phi$ and the tuples that appear in the sources ${E_S}^i$. To apply this correspondence, the $\textsc{AskAtomSetValidity}(e, \psi_b)$ procedure of Algorithm 1 constructs a pair $({E_S}^{e,\psi_b}, {E_T}^{e,\psi_b})$ by transforming the candidate subset $e$ into an instance ${E_S}^{e,\psi_b}$. Recalling that the bijection $\bar{\theta}$ has been defined in Definition 1.16 on page 27, ${E_S}^{e,\psi_b}$ is formally defined as follows:

$${E_S}^{e,\psi_b} = \left\{ \bar{\theta}^{-1}(a) | a \in e \right\}$$

Then the chase procedure is used to compute ${E_T}^{e,\psi_b}$, formally:

$${E_T}^{e,\psi_b} = \textsc{chase}(e \rightarrow \psi_b, {E_S}^{e,\psi_b})$$

We illustrate the construction of such a data example in the following example:

*Example* 2.11. *This example focuses on the generation of the exemplar tuples underlying the questions of Example 2.10 while refining the tgds (2.2) and (2.4). We are challenging the user about the validity of the set of atoms:*

$$e = \{Flight(idF_0, town_2, town_1, idAir_0); Airl(idAir_0, name_1, town_1)\}$$

*which is a subset of the left-hand side of the tgds (2.2) and (2.4). For each tgd, these atoms are built from the sets:*

$E_S^{1'} = \{Flight(\texttt{flight0}, \texttt{Miami}, \texttt{L.A.}, \texttt{airline0}); Airl(\texttt{airline0}, \texttt{AAir}, \texttt{L.A.})\}$
$E_S^{2'} = \{Flight(\texttt{flight0}, \texttt{Lyon}, \texttt{Paris}, \texttt{airline0}); Airl(\texttt{airline0}, \texttt{AirF}, \texttt{Paris})\}$

*which are subset of the instances $E_S^1$ and $E_S^2$, respectively. We want to challenge the user whether the following generalization of the tgds (2.2) and (2.4) is sufficient:*

$$\begin{aligned}\sigma = & Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \\ & \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ & \qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1)\end{aligned}$$

*The chase procedure applies $\sigma$ on $E_S^{1'}$ ($E_S^{2'}$, resp.) to obtain the following instance $E_T^{1'}$ ($E_T^{2'}$, resp.), from which the first question appearing in Example 2.10 is derived:*

$$\begin{aligned}E_T^{1'} = \{ & Dpt(\texttt{Miami}, \texttt{flight2}, \texttt{comp0}); Arr(\texttt{L.A.}, \texttt{flight2}, \texttt{comp0}); \\ & Co(\texttt{comp0}, \texttt{AAir}, \texttt{L.A.})\} \\ E_T^{2'} = \{ & Dpt(\texttt{Lyon}, \texttt{flight2}, \texttt{comp0}); Arr(\texttt{Paris}, \texttt{flight2}, \texttt{comp0}); \\ & Co(\texttt{comp0}, \texttt{AirF}, \texttt{Paris})\}\end{aligned}$$

### 2.3.5 Formal guarantees of the atom refinement algorithm

In this section, we first show that when shifting from the initial canonical mapping to its refined form as given by Algorithm 1, we obtain a *more general* set of tgds. Then, we show that the mapping output by the atom refinement step is always *split-reduced* and does not contain *$\sigma$-redundant* tgds.

Lemma 2.1.
*Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a canonical mapping.*
*Let $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ be a mapping obtained from atom refinement of $\mathcal{M}$.*
*Then, for all source instances $E_S$, there exists a morphism $\mu$ such that:*

$$\mu(\text{CHASE}(\Sigma, E_S)) \subseteq \text{CHASE}(\Sigma', E_S)$$

*By the correctness of the chase procedure, the logical entailment $\mathcal{M}' \models \mathcal{M}$ holds.*

*Proof.* For each tgd $\sigma = \phi \rightarrow \psi \in \mathcal{M}$ there exists at least one tgd $\sigma' = \phi' \rightarrow \psi \in \mathcal{M}'$ that is an atom refinement of $\sigma$. Then, $\phi'$ must correspond to a node in the quasi-lattice, such that $\phi' \subseteq \phi$. We introduce a function *ref*: $\mathcal{M} \rightarrow \mathcal{M}'$ that associates to each $\sigma$ in $\mathcal{M}$ one of its refinements (that may be arbitrarily chosen if there are several such tgds in $\mathcal{M}'$).

Let $\nu$ be an instantiation mapping to compute $\text{CHASE}(\mathcal{M}, E_S)$. That is, there exists a tgd $\sigma = \phi \rightarrow \psi \in \mathcal{M}$ such that $\nu(\phi) \subseteq E_S$ and $\nu(\psi) \subseteq \text{CHASE}(\mathcal{M}, E_S)$. Moreover each existential variable in $\psi$ is mapped by $\nu$ to a fresh labelled null, which means that $\nu^{-1}$ is defined for such values. Since $\phi' \subseteq \phi$, $\nu(\phi') \subseteq E_S$. Therefore, there exists an instantiation mapping $\nu'$ such that (1) $\nu'(\phi') \subseteq E_S$ (2) $\nu'(\psi') \subseteq \text{CHASE}(\mathcal{M}', E_S)$ and (3) for all variables x in $\phi'$, $\nu'(\text{x}) = \nu(\text{x})$. However, $\nu'$ and $\nu$ can differ in two ways: the domain of $\nu'$ can be smaller than the domain of $\nu$ and the labelled nulls that are assigned to existential variables in $\psi$ can be different because the chase generates fresh null values at each tgd application. By construction of $\mathcal{C}_{cand}$ in Algorithm 1, any variable x in $\psi$ is either an existential variable or a universal variable in $\phi'$. Thus, every variable x in $\psi$ is either mapped to fresh null values by $\nu$ and $\nu'$ or, alternatively, $\nu(\text{x}) = \nu'(\text{x})$. We introduce $\mu_\nu$ a morphism from $\nu(\psi) \subseteq \text{CHASE}(\mathcal{M}, E_S)$ to $\nu'(\psi) \subseteq \text{CHASE}(\mathcal{M}', E_S)$, defined as $\mu_\nu(c) = c$ if there exists x in $\phi'$ such that $\nu(\text{x}) = c$ and $\mu_\nu(c) = \nu'(\nu^{-1}(c))$ otherwise (that if $c$ is a fresh value generated by $\text{CHASE}(\mathcal{M}, E_S)$).

Let us consider two instantiation mappings $\nu_1$ and $\nu_2$ used in $\text{CHASE}(\mathcal{M}, E_S)$ and their associated morphisms $\mu_{\nu_1}$ and $\mu_{\nu_2}$. Let $c$ be a value in $dom(\mu_{\nu_1}) \cap dom(\mu_{\nu_2})$. If $c$ is fresh and in $dom(\mu_{\nu_1})$, it means that it is the image of an existential variable by $\nu_1$, which means that it cannot be the image of any variable by $\nu_2$, and thus $c \notin dom(\mu_{\nu_2})$ which contradicts $c \in dom(\mu_{\nu_1}) \cap dom(\mu_{\nu_2})$. Thus $c$ is not fresh, thus $\mu_{\nu_1}(c) = c = \mu_{\nu_2}(c)$. We define $\mu_{\{\nu_1,\nu_2\}}$ as $\mu_{\{\nu_1,\nu_2\}}(c) = \mu_{\nu_1}(c)$ if $c \in dom(\mu_{\nu_1})$ and $\mu_{\{\nu_1,\nu_2\}}(c) = \mu_{\nu_2}(c)$ otherwise. One can remark that $\mu_{\{\nu_1,\nu_2\}} \mid_{dom(\mu_{\nu_1})} = \mu_{\nu_1}$ and $\mu_{\{\nu_1,\nu_2\}} \mid_{dom(\mu_{\nu_2})} = \mu_{\nu_2}$. By iterating this construction on the finite set $\Lambda$ of all instantiation mappings $\nu$ used in $\text{CHASE}(\mathcal{M}, E_S)$, we can build a morphism $\mu = \mu_\Lambda$.

Let $t$ be a tuple in $\text{CHASE}(\mathcal{M}, E_S)$. There exists an instantiation morphism $\nu$ used in $\text{CHASE}(\mathcal{M}, E_S)$ and a tgd $\phi \rightarrow \psi$ such that $t \in \nu(\psi)$. Since $\mu_\nu(\nu(\psi)) \subseteq \text{CHASE}(\mathcal{M}', E_S)$ and $\mu \mid_{dom(\mu_\nu)} = \mu_\nu$ we deduce $\mu(t) \in \text{CHASE}(\mathcal{M}', E_S)$. □

This lemma comes from the assumption on our practical framework that no new existentially quantified variable is created. However, it should be noted that this lemma does not hold if the framework is allowed to create new existential variables, which is feasible as shown in Chapter 1 on the formal framework for the resolution of the *IMS* problem.

In the following example, we illustrate that this lemma does not hold if the creation of existentially quantified variables is allowed:

*Example* 2.12. *Given a pair* $(E_S, E_T)$ *such that:*

$$E_S = \{R(\mathtt{x}, \mathtt{y}); S(\mathtt{z})\} \text{ and } E_T = \{T(\mathtt{x})\}$$

*then the canonical mapping corresponding to* $(E_S, E_T)$ *is:*

$$\Sigma = \{R(x, y) \wedge S(z) \rightarrow T(x)\}$$

*Suppose that atom refinement allows the creation of existentially quantified variables. By applying this refinement on* $\Sigma$*, we may obtain the mapping:*

$$\Sigma' = \{S(z) \rightarrow \exists x\ T(x)\}$$

*and thus, chasing* $E_S$ *under* $\Sigma$ *and* $\Sigma'$ *will lead to following results:*

$$\textsc{chase}(E_S, \Sigma) = \{T(\mathtt{x})\} \qquad \textsc{chase}(E_S, \Sigma') = \{T(\mathtt{x_1})\}$$

*for which there is no morphism* $\mu$ *such that* $\mu(\textsc{chase}(E_S, \Sigma)) \subseteq \textsc{chase}(E_S, \Sigma')$*, because the constant* `x` *has to be preserved.*

Moreover, we show in following Lemma 2.2 that the mapping output by the atom refinement step is always *split-reduced* and has no $\sigma$*-redundant* tgds:

Lemma 2.2.
*Given a normalized canonical mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$*, application of atom refinement on the tgds in* $\Sigma$ *always produces a mapping which is split-reduced and without* $\sigma$*-redundancy.*

*Proof.* As $\mathcal{M}$ is already normalized, it is *split-reduced*. During the refinement step, only atoms in the left-hand side are suppressed, so there is no way to break joins between existentially quantified variables as they are located only in the right-hand side. This means that $\mathcal{M}'$ is *split-reduced*.

Also, the refinement uses one quasi-lattice for each block of $\psi$-equivalent tgds. So the only way to create equivalent tgds is to validate two equivalent left-hand side conjunctions in a same quasi-lattice, and there is no equivalent nodes in such quasi-lattice. This means that $\mathcal{M}'$ has no $\sigma$*-redundant* tgds. □

Thus, no additional normalization is required after the atom refinement step, prior to the launch of the join refinement step.

In the next section, we will provide the worst case complexity of the atom refinement step, in terms of the number of asked questions.

### 2.3.6 Complexity of the quasi-lattice exploration in terms of the number of asked questions

Depending on the mapping provided as input of Algorithm 1, the size of the explored quasi-lattices will fluctuate noticeably.

In the worst-case scenario, all blocks in the partition of $\psi$-equivalent tgds are singletons, and all sets in the lower bounds are singletons too. This leads to explore the complete quasi-lattices of all possible atom conjunctions on the left-hand side for every tgd in the pre-processed mapping.

Thus, given a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, for each tgd $\sigma$ in the set of tgds $\Sigma$ the complete quasi-lattice of atom sets for $\sigma$ is explored. Thus, if $n_\sigma$ is the number of atoms in the left-hand side of a tgd $\sigma \in \Sigma$, the size of the explored lattice will be $2^{n_\sigma}$ elements. Since the empty set is not considered as a solution, and the upper bound is always valid with respect to the expected mapping under our assumption, we can subtract two questions to the previous results. Thus, the maximum number of questions that can be asked for a tgd $\sigma$ is $(2^{n_\sigma} - 2)$. Finally, we can define an upper bound of the number of questions that can be asked during execution of TgdsAtomRefinement($\Sigma$) as:

$$\sum_{\forall \sigma \in \Sigma_{in}} (2^{n_\sigma} - 2)$$

It should be noted that this worst-case where no pruning can be performed serves as an upper bound of the number of questions but is not likely to occur in practice. An experimental study of the number of questions asked during the specification of realistic mappings is provided in the Section 4.1.

In the next section, we will describe the join refinement step that follows the atom refinement step in our mapping specification framework (Figure 2.1).

## 2.4 Join refinement step

In this section, we describe the join refinement step that follows the atom refinement step in our framework (Figure 2.1 on page 54). The intuition behind this step is that, in relational data, multiple occurrences of a same value do not necessarily imply a semantic relationship between the attributes containing such value. An example from our running example (Figure 2.2 (i) and (iii) on page 57) is the occurrence of the constant `L.A.` to represent both the city where the headquarters of `AAir` company is located, and the arrival and departure city of two flights.

In such a case, the canonical mapping imposes co-occurrences that may be due to spurious use of the same variable. Thus, the canonical mapping

may introduce irrelevant joins in the left-hand side of the tgds, preventing the triggering of tgds as illustrated in the following example:

*Example* 2.13. *We recall the tgd (2.6) from the mapping $\Sigma_{atRef}$ output by the atom refinement step, as illustrated in Example 2.10 on page 66:*

$$\begin{aligned} &Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ &\qquad\qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \end{aligned} \tag{2.6}$$

*The ambiguity conveyed by the initial exemplar tuples leads to only trigger the tgds for source instances with a structure similar to the following instance:*

$$\begin{aligned} E_S = \{&Flight(\texttt{flight0}, \texttt{Lyon}, \texttt{London}, \texttt{airline0}); \\ &Airl(\texttt{airline0}, \texttt{BritAir}, \texttt{London})\} \end{aligned}$$

*where the headquarters of the company and the town of arrival of the flight are identical.*

*At the opposite, if the same company provides a flight from Lyon to another town than London, which is likely to arrive, such as in the following source instance:*

$$\begin{aligned} E_S{}' = \{&Flight(\texttt{flight0}, \texttt{Lyon}, \texttt{Berlin}, \texttt{airline0}); \\ &Airl(\texttt{airline0}, \texttt{BritAir}, \texttt{London})\} \end{aligned}$$

*then, despite the fact that the instance is perfectly valid in a real-world context, the tgd will not be triggered and consequently the information about this flight will not be exported to the target instance.*

Consequently, in order to produce the mapping the users have in their minds, we primarily need to distinguish relevant joins from irrelevant ones. To do so, this section presents the join refinement step and details a join refinement algorithm that explores the candidate joins in each tgd by inquiring the user about the validity of such joins.

### 2.4.1 Join partitions

As joins in tgds are encoded by multiple occurrences of a variable, we refer to these variables as *join variables*. Refining a join corresponds to replace some occurrences of these join variables with fresh variables, in order to break the join. In order to represent the possible breaking of the joins between the occurrences of a variable, we use the notion of *join partition* which is defined as follows:

**Algorithm 2** TgdsJoinRefinement($\Sigma$)

**Input:** A set of tgds $\Sigma$ to be join refined.
**Output:** A set of tgds $\Sigma'$ where each tgd is join refined.

1: $\Sigma' \leftarrow \emptyset$
2: **for all** $\sigma \in \Sigma$ **do**
3: **let** $\sigma = \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$
4: $\Sigma_t \leftarrow \{\sigma\}$
5: **for all** x $\in \overline{x}$ **do**
6: **if** variable $x$ occurs more than once in $\phi$ **then**
7: $\Sigma_{explored} \leftarrow \Sigma_t$
8: $\Sigma_t \leftarrow \emptyset$
9: **for all** $\sigma' \in \Sigma_{explored}$ **do**
10: $\Sigma_t \leftarrow \Sigma_t \cup$ VarJoinsRefinement($\Sigma_t, \sigma'$, x)
11: **end for**
12: **end if**
13: **end for**
14: $\Sigma' \leftarrow \Sigma' \cup \Sigma_t$
15: **end for**
16: **return** $\Sigma'$

Definition 2.8 (Join partition for a variable).
*Let $\sigma$ be a tgd.*
*Let $x$ be a variable of $\sigma$.*
*Let $\{x_1; \ldots; x_n\}$ be the set of the $n$ occurrences of $x$ in $\sigma$.*
*Then, a* join partition for x in $\sigma$ *is a partition $\mathcal{P}$ of the set $\{x_1; \ldots; x_n\}$ such that each block in $\mathcal{P}$ represents the variables to be unified together.*

In the following, for ease of exposition we assimilate initial variable occurrences with their fresh variable counterpart.

The notion of join partition is exemplified in the following example:

*Example* 2.14. *Recall tgd (2.6) from Example 2.10 obtained after the atom-refined mapping below:*

$$\begin{aligned} &Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ &\qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \end{aligned} \tag{2.6}$$

*There is an ambiguity on the use of the same town as the town of arrival and departure of flights and the town where a travel agency's headquarters is located, as shown by the multiple occurrences of the join variable $town_1$ at four different positions. Thus, if we replace the occurrences of $town_1$ by the fresh variables ${town_1}'$, ${town_1}''$, ${town_1}'''$ and ${town_1}''''$, a possible* join partition for $\text{town}_1$ *can be the partition:*

$$\{\{{town_1}'\}; \{{town_1}''\}; \{{town_1}'''\}; \{{town_1}''''\}\}$$

*for which no unification is done between variable occurrences, yielding the following candidate tgd:*

$$\begin{aligned} &Flight(idF_0, town_2, {town_1}', idAir_0) \wedge Airl(idAir_0, name_1, {town_1}'') \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ &\qquad \wedge Arr({town_1}''', idF_2, idC_0) \wedge Co(idC_0, name_1, {town_1}'''') \end{aligned} \tag{2.8}$$

*Another example is the partition in which occurrences ${town_1}'$, ${town_1}'''$ and ${town_1}''''$ are unified:*

$$\{\{{town_1}'; {town_1}'''; {town_1}''''\}; \{{town_1}''\}\}$$

*yielding the following candidate tgd:*

$$\begin{aligned} &Flight(idF_0, town_2, {town_1}', idAir_0) \wedge Airl(idAir_0, name_1, {town_1}'') \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ &\qquad \wedge Arr({town_1}', idF_2, idC_0) \wedge Co(idC_0, name_1, {town_1}') \end{aligned} \tag{2.9}$$

Since we do not wish to introduce new existentially quantified variables, we define a particular class of join partitions called *well-formed join partitions* that do not lead to introduce such variables. Thus, a *well-formed join partition* is defined as follows:

Definition 2.9 (Well-formed join partition).
*Let $\sigma = \phi \rightarrow \psi$ be a tgd.*
*Let $x$ be a variable occurring in $\phi$.*
*Let $\mathcal{E}^{\phi}$ and $\mathcal{E}^{\psi}$ be the set of occurrences of $x$ in $\phi$ and $\psi$, respectively.*

*Then a join partition $\mathcal{P}$ for $x$ is* well-formed *if:*

$$\forall x_i \in \mathcal{E}^{\psi}, \exists x_j \in \mathcal{E}^{\phi} \text{ such that } x_i \equiv_{\mathcal{P}} x_j$$

We illustrate the notion of *well-formed partition* in the following example:

*Example* 2.15. *In the first partition from Example 2.14:*

$$\{\{town_1'\}; \{town_1''\}; \{town_1'''\}; \{town_1''''\}\}$$

*we see that the occurrences $town_1'''$ and $town_1''''$ of $town_1$ belong to the right-hand side of the considered tgd. However they are not unified with occurrences belonging to the left-hand side of this tgd (i.e., in a same block). Consequently, this partition is not well-formed for the variable $town_1$ in tgd (2.6).*

*At the opposite, in the second partition from Example 2.14:*

$$\{\{town_1'; town_1'''; town_1''''\}; \{town_1''\}\}$$

*we see that $town_1'''$ and $town_1''''$ are unified with the occurrence $town_1'$ of $town_1$ which belong to the left-hand side of the tgd. Thus, this partition is well-formed for the variable $town_1$ in tgd (2.6).*

Well-formed partitions are equipped with a quasi-lattice structure: given two partitions $\mathcal{P}$ and $\mathcal{P}'$, if $\mathcal{P} \leq \mathcal{P}'$ and $\mathcal{P}$ is well-formed, then $\mathcal{P}'$ is well-formed as well. In particular, if $\mathcal{P} \leq \mathcal{P}'$ then all unification encoded by $\mathcal{P}$ is also encoded in $\mathcal{P}'$. This means that if $\mathcal{P}$ is acceptable for the user, then it is also the case for $\mathcal{P}'$. Conversely, if $\mathcal{P}'$ is not acceptable for the user (i.e., some joins are missing), then neither is $\mathcal{P}$. We employ these criteria to prune the search space during the exploration of the quasi-lattice of occurrences of $x$. This quasi-lattice structure allows us to avoid challenging our users about partitions leading to redundant tgds.

In the next section, we will show how the quasi-lattices of partitions are explored by Algorithm 2.

### 2.4.2 Join refinement algorithm

In this section, we detail the exploration of the quasi-lattices of partitions performed by Algorithm 2 on page 74.

Using these join partitions, for each tgd in $\Sigma_{atRef}$ Algorithm 2 iterates over the possible join partitions for each variable with multiple occurrences. More precisely, for each evaluated variable the validity of its possible join partitions is conducted by the procedure VarJoinsRefinement which is detailed in Algorithm 3 on page 78. As we do not consider the possibility of creating new joins, but only the suppression of joins which already exists, each original variable is considered separately. However, since each call to VarJoinsRefinement may generate multiple refined tgds for each refined variable, we need to combine these refinements.

In line 1 of Algorithm 3, VarJoinsRefinement first generates a new tgd in which occurrences of the considered variable $x$ are replaced with fresh variables yielding a tgd $\sigma'$. Then it generates the set of possible candidate partitions for the considered variable (line 4 of Algorithm 3). Next to that, the partitions in the upper bound of the quasi-lattice formed by the set of candidate partitions is added to the set of valid partitions. This step comes from the fact that the partitions in the upper bound should be valid to suit to our assumptions, otherwise this would mean that the exemplar tuples provided as input of the framework are ill-defined. Then the SelectPartition procedure selects a partition in the set of partitions and encodes the specific exploration strategy on top of the quasi-lattice. Any suitable exploration strategy can be plugged in here. Function UnifyVariables$(\sigma, \mathcal{P})$ (lines 8 and 19 of Algorithm 3) returns a tgd corresponding to $\sigma$, where variables from the same block of partition $\mathcal{P}$ are unified. Before asking the user about the validity of the tgd generated by UnifyVariables, the procedure verifies if this newly generated tgd is not prunable or redundant with a previously evaluated tgd. This verification is done in line 9 of Algorithm 3, by checking if there exist another tgd $\sigma_t \in \Sigma_t$ which logically entails the currently evaluated tgd (i.e., the procedure check if $\sigma_t$ is logically equivalent or more general than the evaluated tgd).

Then, if the evaluated tgd is neither prunable nor redundant, the user is asked about the validity of this unification in line 9 of Algorithm 3 and the search space and results are pruned according to the answer in lines 11, 12 and 14 of Algorithm 3.

In the following example, we illustrate the join refinement step over our running example:

**Algorithm 3 procedure** VarJoinsRefinement($\Sigma_t, \sigma, \mathrm{x}$)

**Input:** A set of previously join refined tgds $\Sigma_t$.
**Input:** A tgd $\sigma$.
**Input:** A variable $x \in \sigma$ on which the refinement is made.
**Output:** A set of tgds $\Sigma_{out}$ of join refinements of $\sigma$ for variable $x$.

$\sigma' \leftarrow$ generate from $\sigma$ a tgd where occurrences of $x$ are renamed with fresh variables
$\mu_{orig} \leftarrow$ generate a morphism $\mu$ such that $\mu(\sigma') = \sigma$
**let** $\sigma' = \phi' \rightarrow \psi'$
$\mathcal{J}_{cand} \leftarrow$ generate set of possible candidates join partitions from $\sigma'$
$\mathcal{J}_v \leftarrow$ generate upper bound of the join lattice from $\sigma'$
**while** $\mathcal{J}_{cand} \neq \emptyset$ **do**
    $\mathcal{P} \leftarrow$ SelectPartition($\mathcal{J}_{cand}, \mathcal{J}_v$)
    $\sigma'' \leftarrow$ UnifyVariables($\sigma', \mathcal{P}$)
    **if** $(\nexists \sigma_t \in \Sigma_t, \sigma_t \models \sigma'') \wedge$ AskJoinsValidity($\sigma''$) **then**
        add $\mathcal{P}$ to $\mathcal{J}_v$
        remove upper partitions of $\mathcal{P}$ from $\mathcal{J}_v$
        remove $\mathcal{P}$ and its upper partitions from $\mathcal{J}_{cand}$
    **else**
        remove $\mathcal{P}$ and its lower partitions from $\mathcal{J}_{cand}$
    **end if**
**end while**
$\Sigma_{out} \leftarrow \emptyset$
**for all** $\mathcal{P} \in \mathcal{J}_v$ **do**
    $\sigma'' \leftarrow$ UnifyVariables($\sigma', \mathcal{P}$)
    add $\sigma''$ to $\Sigma_{out}$
**end for**
**return** $\Sigma_{out}$

*Example* 2.16. *We recall the tgd (2.6) from Example 2.10:*

$$\begin{aligned} &Flight(idF_0, town_2, town_1, idAir_0) \wedge Airl(idAir_0, name_1, town_1) \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ &\qquad\qquad \wedge Arr(town_1, idF_2, idC_0) \wedge Co(idC_0, name_1, town_1) \end{aligned} \tag{2.6}$$

*The set of universally quantified variables in this tgd is the set:*

$$\overline{x} = \{idF_0, town_2, town_1, idAir_0, name_1\}$$

*Moreover, as we do not create joins between variables but only break joins between occurrences of a same variable, our algorithm only needs to consider*

*variables with more than one occurrence (which is ensured in Algorithm 2 at line 6). Consequently, for the join refinement of tgd (2.6) we only consider the variables* $town_1$ *and* $idAir_0$ *of* $\overline{x}$*.*

*Considering first the* $idAir_0$ *variable, a renaming of each of its occurrences into the fresh variables* ${idAir_0}'$ *and* ${idAir_0}''$ *leads to the following tgd:*

$$\begin{aligned} &Flight(idF_0, town_2, town_1, {idAir_0}') \wedge Airl({idAir_0}'', name_1, town_1) \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0)) \\ &\qquad\qquad \wedge Arr(town_1, idF_2, idC_0 \wedge Co(idC_0, name_1, town_1) \end{aligned} \tag{2.10}$$

*The quasi-lattice for this tgd only contains two partitions:*

$$\{\{{idAir_0}'\}; \{{idAir_0}''\}\} \text{ and } \{\{{idAir_0}'; {idAir_0}''\}\}$$

*Recalling that the upper bound is always valid, the user will not be asked about the validity of the supremum* $\{\{{idAir_0}'; {idAir_0}''\}\}$*. Then, the user is only asked about the validity of the tgd produced from the partition* $\{\{{idAir_0}'\}; \{{idAir_0}''\}\}$*, i.e., the tgd in which occurrences* ${idAir_0}'$ *and* ${idAir_0}''$ *are not unified, meaning that the identifier of an airline company is unrelated to the company identifier of its flight.*

*The user will likely answer 'No' to the above question, thus leading to keep the upper bound* $\{\{{idAir_0}'; {idAir_0}''\}\}$ *of the quasi-lattice as the best valid partition. Since this partition corresponds to the case where every occurrence of* $idAir_0$ *is kept joined, the tgd (2.6) is not modified after the join refinement over variable* $idAir_0$*.*

*Next, we consider the* $town_1$ *variable. A renaming of each of its occurrences with the fresh variables* ${town_1}'$*,* ${town_1}''$*,* ${town_1}'''$ *and* ${town_1}''''$ *leads to the following tgd (2.8) previously shown in Example 2.14:*

$$\begin{aligned} &Flight(idF_0, town_2, {town_1}', idAir_0) \wedge Airl(idAir_0, name_1, {town_1}'') \\ &\rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \wedge Arr({town_1}''', idF_2, idC_0) \\ &\qquad \wedge Co(idC_0, name_1, {town_1}'''') \end{aligned} \tag{2.8}$$

*For this tgd, there exist five partitions that are well-formed (i.e., that do not create new existential variables), namely:*

$$\begin{aligned} &\mathcal{P}_1 = \{\{{town_1}'; {town_1}'''\}; \{{town_1}''; {town_1}''''\}\}, \\ &\mathcal{P}_2 = \{\{{town_1}'; {town_1}''''\}; \{{town_1}''; {town_1}'''\}\}, \\ &\mathcal{P}_3 = \{\{{town_1}'; {town_1}'''; {town_1}''''\}; \{{town_1}''\}\}, \\ &\mathcal{P}_4 = \{\{{town_1}'\}; \{{town_1}''; {town_1}'''; {town_1}''''\}\} \\ &\text{and } \mathcal{P}_5 = \{\{{town_1}'; {town_1}''; {town_1}'''; {town_1}''''\}\}. \end{aligned}$$

*The user is asked about the validity of the candidate partition $\mathcal{P}_1$ with the following question:*

"Are the tuples:
$Flight(\texttt{flight0}, \texttt{Miami}, \texttt{L.A.}', \texttt{airline0})$ and $Airl(\texttt{airline0}, \texttt{AAir}, \texttt{L.A.}'')$
enough to produce:
$Dpt(\texttt{Miami}, \texttt{flight2}, \texttt{comp0})$, $Arr(\texttt{L.A.}', \texttt{flight2}, \texttt{comp0})$
and $Co(\texttt{comp0}, \texttt{AAir}, \texttt{L.A.}'')$?"

*Since this partition is acceptable for the user, they will probably answer 'Yes'. Therefore, the upper bound $\mathcal{P}_5$ of the quasi-lattice is pruned and the following tgd is added to the output:*

$$Flight(idF_0, town_2, {town_1}', idAir_0) \wedge Airl(idAir_0, name_1, {town_1}'') \\ \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ \wedge Arr({town_1}', idF_2, idC_0) \wedge Co(idC_0, name_1, {town_1}'')$$

*The exploration continues with the remaining candidate partitions. However, as the remaining partitions either relate an airline's headquarters to an arrival or a flight to a company's headquarters, the user will consistently answer 'No' to these questions.*

In the join refinement step, the suppression of joins can generate additional tuples in the target instance. For such reason, similarly to the generation of questions in the atom refinement step (Section 2.3.4), the source instance is chased to generate the target instance of the question [1]. Similarly to the procedure described in Section 2.3.4 for atom refinement, the AskJoinsValidity procedure that appears in Algorithm 2 constructs a pair $({E_S}^{\sigma}, {E_T}^{\sigma})$ by instantiating the left-hand side of a candidate tgd $\sigma$ to obtain a source instance ${E_S}^{\sigma}$ and then chasing it to build ${E_T}^{\sigma}$.

*Example* 2.17. *We illustrate the questions asked to the user in Example 2.16. We challenge the user on the validity of the partition:*

$$\mathcal{P}_1 = \{\{{town_1}'; {town_1}'''\}; \{{town_1}''; {town_1}''''\}\}$$

*in the following tgd:*

$$\sigma = Flight(idF_0, town_2, {town_1}', idAir_0) \wedge Airl(idAir_0, name_1, {town_1}'') \\ \rightarrow \exists idC_0, idF_2, Dpt(town_2, idF_2, idC_0) \\ \wedge Arr({town_1}', idF_2, idC_0) \wedge Co(idC_0, name_1, {town_1}'')$$

[1] We recall that the chase is polynomial for $\Sigma$ consisting of only s-t tgds. Thus, repeating it several times, as additional tuples come, is appropriate.

*The instance* ${E_S}^\sigma$ *obtained from the left-hand side of* $\sigma$ *through the bijection* $\bar{\theta}^{-1}$ *is the following:*

$${E_S}^\sigma = \{Flight(\texttt{flight0}, \texttt{Miami}, \texttt{L.A.}', \texttt{airline0}); Airl(\texttt{airline0}, \texttt{AAir}, \texttt{L.A.}'')\}$$

*Chasing* ${E_S}^\sigma$ *with* $\sigma$ *leads to:*

$${E_T}^\sigma = \{Dpt(\texttt{Miami}, \texttt{flight2}, \texttt{comp0}); Arr(\texttt{L.A.}', \texttt{flight2}, \texttt{comp0}); \\ Co(\texttt{comp0}, \texttt{AAir}, \texttt{L.A.}'')\}$$

*Those exemplar tuples are finally rewritten into questions as shown in Example 2.16.*

At the end of this step, the mapping $\mathcal{M}_{final}$ is returned to the user as the result of the framework execution.

### 2.4.3 Formal guarantees

In this section we give some formal guarantees about the mapping output by our framework.

At first, we provide the counterpart lemma of Lemma 2.1 for join refinement, in which we establish the logical entailment of the join-refined mapping:

Lemma 2.3.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping (typically output by the atom refinement step).*
*Let* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *be a mapping obtained from* $\mathcal{M}$ *after join refinement.*
*Then* $\mathcal{M}' \models \mathcal{M}$.

*Proof.* Let $\sigma = \phi \rightarrow \psi$ be a tgd and $x$ be a universal variable in $\sigma$. First, we prove that for all $\sigma'' \in \textsc{VarJoinsRefinement}(\sigma, \mathrm{x})$, $\sigma'' \models \sigma$.

Let $\sigma' = \phi' \rightarrow \psi'$ be the tgd obtained from $\sigma$ by replacing occurrences of $x$ with a fresh variable, and $\mu_{orig}$ be the morphism such that $\mu_{orig}(\sigma') = \sigma$. Let $\sigma'' = \phi' \rightarrow \psi'$. As $\sigma''$ results from the unification of fresh variables in $\sigma'$, there is a morphism $\mu_{unif}$ such that $\mu_{unif}(\sigma') = \sigma''$. Let $\mu_{\sigma''}$ be the morphism defined by: $\mu_{\sigma''}(\mathrm{y}) = \mathrm{x}$ if y results from the unification of fresh variables in $\sigma'$, $\mu_{\sigma''}(\mathrm{y}) = \mathrm{y}$ otherwise. By construction, $\mu_{\sigma''}(\sigma'') = \sigma$. One can remark that existential variables in $\psi''$ are the same as the ones in $\psi$, thus $\mu_{\sigma''}$ is injective for these variables.

In Algorithm 2, $\Sigma_t$ contains tgds that are either element of $\Sigma$ or obtained by applying VarRefinement to previous elements of $\Sigma_t$. Because of line 9, VarRefinement always returns at least one tgd. Thus, for each initial tgd $\sigma$

in $\Sigma$, there is a tgd $\sigma'$ in $\Sigma'$ coming from successive calls of VarRefinement starting with $\sigma$. By transitivity of $\models$ we deduce that $\sigma' \models \sigma$. Thus, $\Sigma' \models \sigma$. Since this holds for all tgds in $\Sigma$, we conclude that $\Sigma' \models \Sigma$ and, by extension, $\mathcal{M}' \models \mathcal{M}$. □

We also prove in the following lemma that the join refinement step preserves the *split-reduction* property of mappings and does not introduce *$\sigma$-redundant* tgds:

Lemma 2.4.
*Given a normalized mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, the application of join refinement on the tgds in $\Sigma$ always produces a mapping which is normalized.*

*Proof.* By definition, if a tgd $\sigma$ is *split-reduced* and contains more than one atom in its right-hand side, these atoms (at least two) are joined using existentially quantified variables. Since join refinement only focuses on universal variables, existential variables are preserved. Thus, all atoms in the right-hand side of join refined tgds are joined together using these existential variables, which means that join refined tgds are also *split-reduced*.

As $\Sigma$ is normalized, each of its tgd is *split-reduced*. Since for each tgd in $\Sigma$, the application of the join refinement step results in new tgds that are also *split-reduced*. Thus, the set $\Sigma'$ of all these refined tgds is a *split-reduced* mapping.

As each tgd is produced only if there is no logically equivalent tgd previously produced (Algorithm 3 at line 9), then no additional step of *$\sigma$-redundancy suppression* is needed.

As the mapping produced is *split-reduced* and does not contain *$\sigma$-redundancy*, then it is normalized. □

Hence, similarly to the atom refinement step and its associated Lemma 2.2, a normalization step following join refinement is not necessary.

### 2.4.4 Complexity of the quasi-lattice exploration in terms of the number of asked questions

For Algorithm 2, the worst case scenario for the join refinement occurs when the user has provided exemplar tuples using the same constant for every attribute. In such a case, the input mapping of Algorithm 2 will contain tgds in which each attribute corresponds to the same variable, as illustrated in the following example:

*Example* 2.18. *Suppose an expected mapping with a set of constraints:*

$$\Sigma_{exp} = \{S(x) \wedge U(y,z) \rightarrow T(x,y,z)\}$$

*An exemplar tuple for* $\Sigma_{exp}$ *can be:*

$$(E_S, E_T) = (\{S(\mathtt{a}); U(\mathtt{a},\mathtt{a})\}, \{T(\mathtt{a},\mathtt{a},\mathtt{a})\})$$

*leading to the canonical mapping:*

$$\Sigma_{can} = \{S(x) \wedge U(x,x) \rightarrow T(x,x,x)\}$$

*The atom refinement step does not lead to a modification of this mapping, thus:*

$$\{S(x) \wedge U(x,x) \rightarrow T(x,x,x)\}$$

*is used as input of the join refinement step.*

In the following, we employ this worst case scenario to compute the maximum number of questions that can be asked during the join refinement of a mapping.

However, even in such a scenario, during the join refinement of a variable $v$ of a tgd $\sigma$ we can still prune partitions with a number of blocks greater than the number of occurrences of $v$ in the left-hand side of $\sigma$. This is due to the fact that we will not produce new existential variables, which will occur if the number of blocks is greater than the number of occurrences of $v$ in the left-hand side.

We recall that the Stirling number of the second kind $\left\{ {n \atop k} \right\}$ corresponds to the number of ways to partition a set of $n$ elements into $k$ blocks (Knuth [Knu97]). Thus, given $\sigma \in \Sigma_{in}$ a mapping, given $n_\sigma$ the total number of occurrences of variables in the tgd $\sigma$, given $n_{\phi,\sigma}$ the number of variable occurrences in the left-hand side of $\sigma$ and knowing that the number of blocks of the partitions we consider cannot be greater than $n_{\phi,\sigma}$, we can define an upper bound of the number of questions asked during execution of TgdsJoinRefinement($\Sigma_{in}$) as:

$$\sum_{\sigma \in \Sigma_{in}} \left( \left( \sum_{k=0}^{n_{\phi,\sigma}} \left\{ {n_\sigma \atop k} \right\} \right) - 1 \right)$$

In the above formula, we have subtracted one question in order to consider the fact that the partition with only one block is always valid.

As for the worst-case complexity of the quasi-lattice exploration during the atom refinement step, this worst-case scenario serves as an upper bound of the number of questions but is not likely to occur in practice. An experimental study of the number of questions asked during the specification of realistic mappings is provided in the Section 4.1.

## 2.5 Output mapping properties

In this section, we provide proofs of good properties of the mapping output by our framework (Figure 2.1 on page 54).

**Conservation of the formal guarantees with our practical approach.** First, we show that the Theorem 1.2 on page 36 over the theoretical framework of Chapter 1 still holds with the optimizations presented in the current chapter:

THEOREM 2.1.
*Let $\mathcal{M}_{can}$ be a canonical mapping.*
*Let $\mathcal{M}_{final}$ be a refinement of $\mathcal{M}_{can}$ output by our framework.*
*Then $\mathcal{M}_{final} \models \mathcal{M}_{can}$.*

*Proof.* This directly follows from the lemmas proved for the atom and join refinement steps.

From Lemma 2.1 on page 69, we have that the atom refinement of $\mathcal{M}_{can}$ will output a mapping $\mathcal{M}_{atRef}$ such that $\mathcal{M}_{atRef} \models \mathcal{M}_{can}$.

From Lemma 2.3 on page 81, we have that the join refinement of $\mathcal{M}_{atRef}$ will output a mapping $\mathcal{M}_{final}$ such that $\mathcal{M}_{final} \models \mathcal{M}_{atRef}$.

Consequently, the refinement of $\mathcal{M}_{can}$ (i.e., the application of an atom refinement step followed by a join refinement step) produces a mapping $\mathcal{M}_{final}$ such that $\mathcal{M}_{final} \models \mathcal{M}_{can}$ □

Next we show that if the user provides *fully informative exemplar tuples*, the pruning performed by our refinement steps does not break the completeness proved in Section 1.3.3 as stated by the following theorem:

THEOREM 2.2.
*Let $\mathcal{M}_{exp}$ be the mapping expected by the users.*
*Let $\mathcal{E}_{FI}$ be a fully informative exemplar tuples set for $\mathcal{M}_{exp}$.*
*Let $\mathcal{M}_{can}$ be the canonical mapping computed from $\mathcal{E}_{FI}$.*
*Let $\mathcal{M}_{final}$ be a refinement of $\mathcal{M}_{can}$ output by our framework, with the use of pruning.*
*Then:*

$$\mathcal{M}_{final} \equiv \mathcal{M}_{exp}$$

*Proof.* Theorem 1.6 state that if no pruning is performed, then our framework leads to a mapping $\mathcal{M}'_{final}$ such that $\mathcal{M}'_{final} \equiv \mathcal{M}_{exp}$. To complete the proof, in the following, we show that the introduction of the pruning only suppress irrelevant candidates.

| Considered schema | Primary keys | Foreign keys |
|---|---|---|
| Source | Not applicable | Section 2.6.2 |
| Target | Section 2.6.3 | Beyond scope |

Figure 2.4: Applicable integrity constraints.

Given a candidate tgd $\sigma_i$ during the execution of our framework, the pruning works in two ways:

- if $\mathcal{M}_{exp} \models \sigma$, then we prune each question about a tgd $\sigma'$ such that $\sigma \models \sigma'$. Trivially, there is no need to explore implied tgds of an already validated tgd as they can be validated by transitivity. Also, there is no need to add them to the final mapping, as they can only create redundant tuples.

- if $\mathcal{M}_{exp} \not\models \sigma$, trivially we can prune each question about a tgd $\sigma'$ such that $\sigma' \models \sigma$.

Consequently, the pruning performed only leads to suppress irrelevant tgds, and thus $\mathcal{M}_{final} \equiv \mathcal{M}'_{final} \equiv \mathcal{M}_{exp}$. □

Thus, the output mapping of our framework is always logically equivalent to the expected mapping $\mathcal{M}_{exp}$ if users provide a fully informative exemplar tuples set for $\mathcal{M}_{exp}$ and if they give correct answers to our questions.

## 2.6 Introducing integrity constraints in the process

In the previous sections, we have described the core of our approach. We now describe how a user can introduce integrity constraints to help the quasi-lattice pruning. Integrity constraints provide a way to define guidelines over a database schema, and ensure that the instances over this schema will comply with these guidelines. In practice, the most commonly used integrity constraints are primary keys and foreign keys. Such constraints are classic tools of database schema design, and therefore might be available in real world integration scenarios.

The introduction of integrity constraints constitutes an extension of the (IMS) problem stated in Definition 1.21 on page 32. This *Interactive Mapping Specification with Integrity Constraints* approach ($\text{IMS}_{\text{IC}}$) can be stated as follows:

Definition 2.10 ($\text{IMS}_{\text{IC}}$).
*Let $\mathcal{M}_{exp}$ be a mapping expected by the user.*
*Let $\mathcal{E}$ be a set of exemplar tuples for $\mathcal{M}_{exp}$.*
*Let $\Sigma_{IC}$ be a (possibly empty) set of integrity constraints composed of source constraints ($\Sigma_{IC\_\mathbf{S}}$) and target constraints ($\Sigma_{IC\_\mathbf{T}}$).*
*Then, the* Interactive Mapping Specification with Integrity Constraints *problem is to discover, by means of boolean interactions, a mapping $\mathcal{M}'$ such that:*

- $\forall (E_S, E_T) \in \mathcal{E}, (E_S, E_T) \models \mathcal{M}'$
- $\mathcal{M}_{exp} \models \mathcal{M}'$
- *$\mathcal{M}'$ is valid with respect to $\Sigma_{IC}$, i.e., for each tgd $(\phi \rightarrow \psi) \in \mathcal{M}'$ we have $\phi \models \Sigma_{IC\_\mathbf{S}}$ and $\psi \models \Sigma_{IC\_\mathbf{T}}$.*

### 2.6.1 Applicable integrity constraints

The studied cases of integrity constraints are summarized in Figure 2.4. We address source foreign keys and target primary keys in Section 2.6.2 and Section 2.6.3, respectively. Notice that we disregard source primary keys that are not pertinent in our framework due to the fact that the user-provided source instances should already satisfy them and that these constraints cannot be violated during the execution of our framework, at the opposite of the source foreign keys. Moreover, we do not consider target foreign keys that, albeit meaningful, would lead to non-trivial extensions beyond the scope of this work.

### 2.6.2 Using source foreign keys

The introduction of foreign key constraints informs us about which tuple (containing a foreign key) can only occur in the presence of another tuple (referenced by the foreign key). These constraints are defined as follows:

Definition 2.11 (Foreign key constraint).
*Let $\mathbf{R}$ be a database schema.*
*Let $S$ and $T$ be two relation symbols such that $S, T \in \mathbf{R}$.*
*Let $X$ and $Y$ be two distinct sequences of attributes over $S$ and $T$, respectively.*
*Then a* foreign key constraint *is a constraint such that:*

$$S[X] \subseteq T[Y] \text{ and } Y \text{ is a key of } T$$

In our algorithm, we use *dependency graphs* to represent the constraints conveyed by the provided foreign keys over a conjunction of atoms. In such a

graph, given each pair of atoms in the conjunction, there exists a directed edge between these atoms if they satisfy a provided foreign key. More formally:

Definition 2.12 (Dependency graph).
*Let $\phi$ be a conjunction of atoms over a schema $\mathbf{S}$.*
*Let $\Sigma_{IC\_\mathbf{S}}$ be a set of integrity constraints over $\mathbf{S}$.*
*The dependency graph over $\phi$ is the directed graph:*

$$\mathcal{G}_\phi = (atoms(\phi), E)$$

*with:*

$$E = \{\langle a_1, a_2\rangle \mid a_1 \in \phi, a_2 \in \phi, \exists \sigma \in \Sigma_{IC\_\mathbf{S}}, \langle \bar{\theta}^{-1}(a_1), \bar{\theta}^{-1}(a_2)\rangle \models \sigma\}$$

We make use of this graph during the atom refinement step as illustrated in the following example:
*Example* 2.19. *Given two schemas:*

$$\mathbf{S} = \{S(x,y); U(x,y,z); V(z,x); W(z,x)\}$$
$$\mathbf{T} = \{T(x)\}$$

*Given an exemplar tuple $(E_S, E_T)$ over $\mathbf{S}$ and $\mathbf{T}$ such that:*

$$E_S = \{S(\mathtt{a},\mathtt{b}), U(\mathtt{a},\mathtt{b},\mathtt{c}), V(\mathtt{c},\mathtt{a}), W(\mathtt{c},\mathtt{a}), S(\mathtt{d},\mathtt{e})\}$$
$$E_T = \{T(\mathtt{a})\}$$

*Given the corresponding conjunctions:*

$$\phi_{E_S} = S(a,b) \wedge U(a,b,c) \wedge V(c,a) \wedge W(c,a) \wedge S(d,e)$$
$$\psi_{E_T} = T(a)$$

*and the set of source foreign keys $\mathbf{S}$:*

$$\Sigma_{fk} = \{U.x, U.y \subseteq S.x, S.y; V.z \subseteq U.z; W.z \subseteq U.z\}$$

*Then we can draw the dependency graph of the atoms in $\phi_{E_S}$ shown in Figure 2.5 (for the sake of clarity, edges are labelled with the corresponding foreign key even if not used in our algorithm).*

*We can see that $S(d,e)$ is not linked to any other atom. At the opposite, atom $V(c,a)$ is linked to atom $U(a,b,c)$, and this atom $U(a,b,c)$ is linked to atom $S(a,b)$. Therefore, we are sure that a tuple triggering atom $V(c,a)$ will always occur with tuples corresponding to atoms $U(a,b,c)$ and $S(a,b)$. As a consequence, we can skip exploring conjunctions like $V(c,a)$ and $U(a,b,c) \wedge V(c,a)$ during the atom refinement step.*

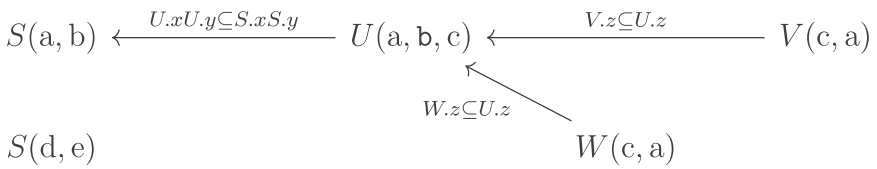



Figure 2.5: Dependency graph of the atoms in $\phi_{E_S}$ (Example 2.19).

To make use of this, we propose Algorithm 4 in order to apply this optimization during atom refinement. In this algorithm, for the sake of clarity, we abuse the notation of $\mathcal{G}_\phi = (atoms(\phi), E)$ by simply writing $\mathcal{G}$ when it is clear from the context. To use it in Algorithm 1 (page 60), the line:

$$\mathcal{C}_{cand} \leftarrow \textsc{SourceFk\_PruneUselessConjunction}(\mathcal{C}_{cand}, \mathcal{C}_{valid}, \Sigma_{sourceFk})$$

needs to be inserted just after line 6 of this algorithm.

This algorithm takes the set of candidate conjunctions that can be explored and prunes it with respect to foreign keys. To achieve that, the algorithm begins with the construction of a dependency graph for each upper bound of the quasi-lattices. Then, for each dependency graph over an upper bound, the algorithm checks if the candidates that are subsets of this upper bound respect all the dependencies of the graph. If such a candidate does not respect every dependency, it is pruned from the set of candidates output by the algorithm. In the following, we provide an example that substantiates the informal description of the algorithm.

*Example* 2.20 (Pruning of quasi-lattice: the need of evaluating each supremum separately). *Given two schemas:*

$$\mathbf{S} = \{S(x, y); S'(x, z); U(x, z)\}$$
$$\mathbf{T} = \{T(x)\}$$

*Given two exemplar tuples over* $\mathbf{S}$ *and* $\mathbf{T}$ *such that:*

$$(E_S^1, E_T^1) = (\{S(\mathtt{a}, \mathtt{b}), U(\mathtt{a}, \mathtt{c})\}, \{T(\mathtt{a})\})$$
$$(E_S^2, E_T^2) = (\{S'(\mathtt{a}, \mathtt{b}), U(\mathtt{a}, \mathtt{c})\}, \{T(\mathtt{a})\})$$

*and the set of foreign keys over schema* $\mathbf{S}$*:*

$$\Sigma_{fk} = \{U.x \subseteq S.x; U.x \subseteq S'.x\}$$

*During atom refinement, as these tgds are $\psi$-equivalent, we will explore the atom sets quasi-lattice shown in Figure 2.6a. If we do not produce a separate*

**Algorithm 4** SourceFk_pruneUselessConjunction($\mathcal{C}_{cand}, \Sigma_{fk}$)

**Input:** A set $\mathcal{C}_{cand}$ of candidate conjunctions to evaluate (as produced by Algorithm 1 on page 60, line 5)
**Input:** A set $\mathcal{C}_{up}$ of the upper bound of the quasi lattice over $\mathcal{C}_{cand}$ (as produced by Algorithm 1, line 6)
**Input:** A set of source foreign keys $\Sigma_{fk}$.
**Output:** A set $\mathcal{C}'_{cand}$ of the pruned set of candidates.

▷ Generation of dependency graphs for each upper bound

1: $\mathcal{F}_{\mathcal{G}} \leftarrow \emptyset$
2: **for all** $\phi_{up} \in \mathcal{C}_{up}$ **do**
3: $\quad E_{\phi_{up}} \leftarrow \emptyset$
4: $\quad$ **for all** $(R[X] \subseteq S[Y]) \in \Sigma_{fk}$ **do**
5: $\quad\quad E_t \leftarrow$ extract the pairs of atoms $\langle a_1, a_2 \rangle$ such that $a_1, a_2 \in \phi_{up}$ and $\bar{\theta}^{-1}(a_1)[X] \subseteq \bar{\theta}^{-1}(a_2)[Y]$
6: $\quad\quad E_{\phi_{up}} \leftarrow E_{\phi_{up}} \cup E_t$
7: $\quad$ **end for**
8: $\quad$ Let $\mathcal{G} = (atoms(\phi_{up}), E_{\phi_{up}})$
9: $\quad \mathcal{F}_{\mathcal{G}} \leftarrow \mathcal{F}_{\mathcal{G}} \cup \{\mathcal{G}\}$
10: **end for**

▷ Pruning of candidates

11: $\mathcal{C}'_{cand} \leftarrow \mathcal{C}_{cand}$
12: **for all** $\mathcal{G} \in \mathcal{F}_{\mathcal{G}}$ **do**
13: $\quad$ Let $\mathcal{G} = (atoms(\phi_{up}), E_{\phi_{up}})$
14: $\quad$ **for all** $c \in \mathcal{C}'_{cand}$ such that $c \subseteq \phi$ **do**
15: $\quad\quad$ **if** $\exists \langle a_1, a_2 \rangle \in E_{\phi_{up}}$ such that $a_1 \in c \wedge a_2 \notin c$ **then**
16: $\quad\quad\quad \mathcal{C}'_{cand} \leftarrow \mathcal{C}'_{cand} \setminus c$
17: $\quad\quad$ **end if**
18: $\quad$ **end for**
19: **end for**
20: **return** $\mathcal{C}'_{cand}$

*dependency graph for each element in the upper bound, we will obtain the graph in Figure 2.6b (for the sake of clarity, edges are labelled with the corresponding foreign key even if not used in our algorithm). This graph will lead to the pruning of each conjunction except $S(a, b)$ and $S'(a, c)$. This is due to the fact that the perfectly acceptable atom conjunctions $S(a, b) \wedge U(a, c)$ and $S'(a, c) \wedge U(a, c)$ do not contain the whole set of dependencies expressed in the graph, and will be pruned by the condition line 15. In other words, this graph is only usable if the conjunction $S \wedge S' \wedge U$ can be accessed during atom refinement.*

*To avoid such a case, our algorithm constructs a dependency graph for each element in the upper bound of the quasi-lattice. This allows to check, for each dependency graph of an element in the upper bound, if a candidate subset of this element does not express each of its dependencies. In our example, this leads to generate the two small dependency graphs shown in Figure 2.6c.*

*These graphs lead to prune conjunction $U(a, c)$ but not the conjunction $S(a, b) \wedge U(a, c)$ as it respects the dependency of the graph at the left and is not included in the other upper bound element $S'(a, c) \wedge U(a, c)$ (this prevents to evaluate this conjunction with the dependency graph at the right, which should have led to its pruning). The exact same principle leads to avoid the pruning of the conjunction $S'(a, c) \wedge U(a, c)$.*

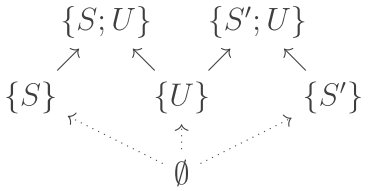
In the following lemma, we show that the introduction of our optimization over source foreign keys only prunes invalid candidates:

Lemma 2.5.
*Let $\mathcal{C}_{valid}$ be the upper bound of a quasi-lattice of atom conjunctions as produced by our atom refinement step.*
*Let $\mathcal{G}_e$ be the dependency graphs that are generated separately for each element $e$ of $\mathcal{C}_{valid}$.*
*If each graph $\mathcal{G}_e$ is checked on a subset of $e$ (lines 12–19 of Algorithm 4), then Algorithm 4 will only suppress candidates that either violates a foreign key or are triggered as often as another candidates in the output set.*

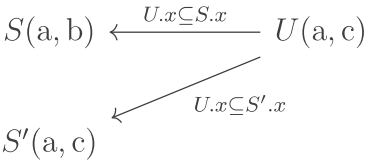
*Proof.* Given an element $e$ and its corresponding graph $\mathcal{G}_e$, then our algorithm will suppress only candidates that are subsets of $e$ and that violate at least one foreign key represented in $\mathcal{G}_e$.

Moreover, given an atom $\delta \in e$ such that $e \setminus \delta$ violates a foreign key in $\mathcal{G}_e$, this means that there is an atom $\gamma \in e$ such that there is a foreign key from $\gamma$ to $\delta$, i.e., $\gamma$ will always occur with the corresponding atom $\delta$. Thus, there is no need to explore conjunction $e \setminus \delta$ as this conjunction will be triggered as often as conjunctions $e$.

□




(a) Explored atoms sets quasi-lattice.




(b) Dependency graph without separate upper bound elements


(c) Dependency graphs in the presence of separate upper bound elements

Figure 2.6: Explored quasi-lattice and dependency graphs of Example 2.20

**Algorithm 5** TargetPk_invalidTgd($\sigma, \Sigma, \{E_S^1; \ldots; E_S^n\}$)

**Input:** A tgd $\sigma$ to evaluate.
**Input:** A set of target primary keys $\Sigma_{Pk}$.
**Input:** A set of source instance $\{E_S^1; \ldots; E_S^n\}$ provided by the user (sources of the exemplar tuples and/or other sources).
**Output:** return `true` if the conjunction can be pruned, else return `false`.

1: **for all** $E_S^i \in \{E_S^1; \ldots; E_S^n\}$ **do**
2: **let** $E_T^i = \text{CHASE}(\sigma, E_S^i)$
3: $t_{bool} \leftarrow$ evaluate if $E_T^i$ violates a primary key in $\Sigma_{Pk}$
4: $res \leftarrow res \vee t_{bool}$
5: **end for**
6: **return** $res$

### 2.6.3 Using target primary keys

During the steps of our framework, exploration can lead to evaluate tgds which are inconsistent with respect to the primary key constraints on the target schema. Such a case is illustrated in the following example:

*Example* 2.21. *Given exemplar tuples:*

| $A$ | *Att1* | *Att2* | *Att3* |
|---|---|---|---|
| | a | b | a |
| | a | b | c |
| | c | b | c |

$\rightarrow$

| $B$ | *Att4* | *Att5* | *Att6* |
|---|---|---|---|
| | a | b | a |

*The join refinement of variable a will explore the following possibilities:*

| | |
|---|---|
| $\sigma : A(a,b,a) \rightarrow B(a,b,a)$ | $\text{CHASE}(\sigma, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$ |
| $\sigma_1 : A(a,b,a') \rightarrow B(a,b,a')$ | $\text{CHASE}(\sigma_1, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{a},\mathtt{b},\mathtt{c}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$ |
| $\sigma_2 : A(a,b,a') \rightarrow B(a',b,a)$ | $\text{CHASE}(\sigma_2, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$ |
| $\sigma_3 : A(a,b,a') \rightarrow B(a,b,a)$ | $\text{CHASE}(\sigma_3, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$ |
| $\sigma_4 : A(a,b,a') \rightarrow B(a',b,a')$ | $\text{CHASE}(\sigma_4, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$ |

*Knowing that the pair of attributes* $(B.Att4, B.Att5)$ *is a target primary key allows us to prune* $\sigma_1$ *and* $\sigma_2$ *as the result of chasing the source instance A with* $\sigma_1$ *and* $\sigma_2$ *will lead to instances:*

$$\text{CHASE}(\sigma_1, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{a},\mathtt{b},\mathtt{c}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$$
$$\text{CHASE}(\sigma_2, E_S) = \{B(\mathtt{a},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{a}); B(\mathtt{c},\mathtt{b},\mathtt{c})\}$$

*which violate the primary key constraint.*

To handle this problem, we propose Algorithm 5 which, given a set of target primary key constraints provided by a user, allows to avoid exploration of candidates which can lead to break these constraints. To use it in Algorithm 3 (page 78), the condition line 9 needs to be changed with:

$$(\nexists \sigma_t \in \Sigma_t, \sigma_t \models \sigma'') \wedge \text{AskJoinsValidity}(\sigma'') \wedge \neg\text{TargetPk\_InvalidTgd}(\sigma'', \Sigma, \{E_S^1; \ldots; E_S^n\})$$

In the following lemma, we show that the introduction of our optimization over target primary keys only prunes invalid candidates:

Lemma 2.6.
*Let* $\sigma : \phi \rightarrow \psi$ *be a candidate tgd during join refinement steps.*
*Let* $\Sigma_{Pk}$ *be a set of target primary keys.*
*Let* $\{E_S^1; \ldots; E_S^n\}$ *a set of source instances.*

*Then Algorithm 5 will prune $\sigma$ only if it leads to the violation of a target primary key, i.e., if $\sigma$ is such that:*

$$\exists E_S \in \{E_S^1; \ldots; E_S^n\}, \textsc{chase}(\sigma, E_S) \not\models \Sigma_{Pk}$$

.

*Proof.* Our optimization leads to pruning candidate tgds which will lead to violate the user's constraint if such tgds are applied to the user's examples. Thus, the invalidity of such candidates is trivially seen. □

## 2.7 Conclusion

In this section, we have provided a practical framework to solve the *interactive mapping specification problem* described in Chapter 1 in an efficient way. During the resolution of such a problem, our approach organizes the possible valid mappings into imbricated quasi-lattices in order to allow an efficient pruning of the space of explored mappings, and thus the reduction of the number of interactions with the users.

Then, we have provided an optimization of our approach through the use of integrity constraints in order to reduce the number of interactions with our users.

Along with the description of our algorithms, we have proved that the good properties proved for our formal framework in Chapter 1 still hold with the introduction of the optimizations presented in this chapter.

# Chapter 3

# Mapping under policy views

In this chapter, we consider the privacy-aware variant of the data exchange problem. In this setting, the source comes with a set of constraints called policy views, 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 schema, the target schema and the s-t tgds in the mapping.

Under these assumptions, we propose the following contributions:

- given privacy restrictions on the source schemas under form of *policy views*, we provide a definition of mapping safety under these privacy restrictions,
- we provide a way to assess mapping safety with respect to the privacy restrictions defined over its source schema,
- finally, in case of privacy violations, we provide repairing methods allowing to rewrite an input mapping in a mapping which is safe with respect to the privacy restrictions.

To the best of our knowledge, our work is the first to provide practical algorithms for a logical privacy-preservation paradigm, a subject which is described as an open research challenge in Nash *et al.* [ND07] and Benedikt *et al.* [BGK17]. The actual version of our repairing process is focused on the rewriting of *GAV schema mappings* (Definition 1.5 page 22), thus only a subset of the mappings that can be output by our specification process can be rewritten[1].

[1]It should be noted that it is easy to force the mapping specification process described in Chapters 1 and 2 to output *GAV schema mappings* instead of *GLAV schema mappings* by using exemplar tuples without labelled nulls, which will lead to an output mapping with atomic right-hand sides and without existentially quantified variables.

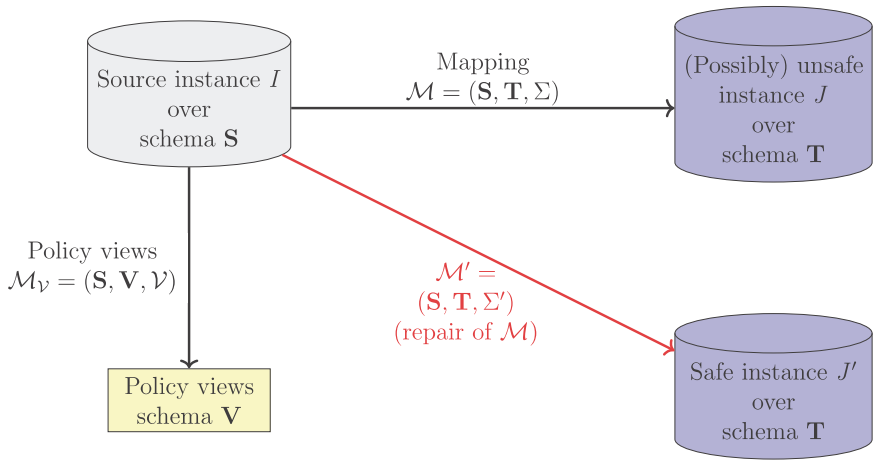



Figure 3.1: A data exchange setting with mappings and policy views.

**Chapter organization** In Section 3.1, we introduce notions used in this chapter that have not been exposed in previous chapters. In Section 3.2, we provide an overview of our approach and the running example used through this chapter. In Section 3.3, we provide formal definition and properties of the privacy preservation ensured by a set of policy views. In Section 3.4, we provide algorithms to repair a mapping in order to ensure that it only exposes information allowed by a reference set of policy views. In Section 3.5, we provide a simple approach to learn the preference function used by our repairing algorithms according to the users' previous choices. In Section 3.6, we discuss related work on the privacy preservation in data exchange problems.

## 3.1 Basic notions

In this section, we provide some additional definitions about notions that have not been already exposed and will be useful throughout this chapter.

**Equality generating dependencies** In order to define the notion of *equality generating dependency* (egd for short), we previously define the notion of equality atom as follows:

Definition 3.1 (Equality atom).
*Let $t$ and $t'$ be two terms, where a term can be a constant, a variable or a labelled null. Then an equality atom is an atom of the form $t = t'$.*

From this definition, we define an egd as follows:

Definition 3.2 (Equality generating dependency).
*Let* $\mathbf{S}$ *be a relational schema.*
*Then an* equality-generating dependency *over* $\mathbf{S}$ *is a first-order logical formula:*

$$\forall \overline{x}, \phi(\overline{x}) \rightarrow \bigwedge_{i,j \in \mathbb{N}} t_i = t_j$$

*such that :*

- $\phi$ *is an atom conjunction over relations in* $\mathbf{S}$
- $\bigwedge_{i,j \in \mathbb{N}} t_i = t_j$ *is a conjunction of equality atoms.*

**Chase procedure with tuple and equality generating dependencies**
With the introduction of egds, we extend our previous definition of the *chase procedure* to handle such dependencies (Fagin *et al.* [FKMP05], Benedikt *et al.* [BKM+17]). To this extent, we first define two separate chase steps for the tgds and the egds as follows:

Definition 3.3 (Chase step, adapted from Fagin *et al.* [FKMP05]).
*Let* $I$ *be a source instance over a schema* $\mathbf{S}$.

- **tgd chase step.**
  *Let* $\sigma$ *be a tgd of the form* $\forall \overline{x}, \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$.
  *Let* $\mu$ *be a homomorphism from* $\phi(\overline{x})$ *into* $I$*, such that there does not exist an extension of* $\mu$ *to a homomorphism* $\mu'$ *from* $\psi(\overline{x}, \overline{y})$ *into* $I$*. Such a homomorphism is called an* active trigger.

  *Applying the tgd chase step for* $\sigma$ *and* $\mu$ *to* $I$ *results in a new instance* $I'$ *obtained by:*

  - *extending the homomorphism* $\mu$ *to a homomorphism* $\mu'$ *such that:*
    *(a) for each variable* $x_i \in \overline{x}$*, then* $\mu'(x_i) = \mu(x_i)$,
    *(b) for each variable* $y_i \in \overline{y}$*, then* $\mu'(y_i)$ *is a fresh labelled null*
  - *taking the image of the atoms of* $\psi$ *under* $\mu'$.

  *This application of* $\sigma$ *to* $I$ *with the homomorphism* $\mu$ *is denoted by the notation* $I \xrightarrow{\sigma,\mu} I'$.

- **egd chase step.**
  *Let* $\sigma$ *be an egd of the form* $\forall \overline{x}, \phi(\overline{x}) \rightarrow \bigwedge_{i,j \in \mathbb{N}} t_i = t_j$.

*Let $\mu$ be a homomorphism from $\phi(\overline{x})$ into $I$ such that there exists an equality atom $t_i = t_j$ in the right-hand side of $\sigma$ such that $\mu(t_i) \neq \mu(t_j)$. Such a homomorphism is called an* active trigger*.*
*This leads to two cases:*

- *if there exists an equality atom $t_i = t_j$ in the right-hand side of $\sigma$ both $\mu(t_i)$ and $\mu(t_j)$ are constants, then the application of $\sigma$ to $I$ with $\mu$ results in* "failure"*, denoted by the notation $I \xrightarrow{\sigma,\mu} \perp$.*
- *otherwise, instance $I'$ is the instance $I$ in which the following replacement has been made for all equality atoms $t_i = t_j$ in the right-hand side of $\sigma$: (a) if $\mu(t_i)$ or $\mu(t_j)$ is a constant, then it replaces the other everywhere, else (b) as both $\mu(t_i)$ and $\mu(t_j)$ are labelled nulls, then one replaces the other everywhere. This application of $\sigma$ to $I$ with the homomorphism $\mu$ is denoted by the notation $I \xrightarrow{\sigma,\mu} I'$.*

Using these steps, the chase procedure can be defined as follows:

Definition 3.4 (Chase procedure, adapted from Fagin *et al.* [FKMP05]).
*Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a mapping.*
*Let $\Sigma_{egd}$ be a set of egds over $\mathbf{S}$.*
*Let $I$ be a source instance over $\mathbf{S}$.*
*Then, to produce an output instance $J$ over $\mathbf{T}$, the chase procedure will execute a* chase sequence *defined as follows:*
*A* chase sequence *of $I$ with the set of dependencies $\Sigma \cup \Sigma_{egd}$ is a sequence of chase steps:*

$$I_i \xrightarrow{\sigma_i,\mu_i} I_{i+1} \quad (i \geq 0)$$

*with $I = I_0$ and $\sigma_i \in \Sigma \cup \Sigma_{egd}$.*

Definition 3.5 (Finite chase procedure, adapted from Fagin *et al.* [FKMP05]).
*Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a mapping.*
*Let $\Sigma_{egd}$ be a set of egds over $\mathbf{S}$.*
*Let $I$ be a source instance over $\mathbf{S}$.*
*Then the chase procedure ends if there is a chase sequence:*

$$I_i \xrightarrow{\sigma_i,\mu_i} I_{i+1} \quad (0 \leq i \leq m)$$

*such that either:*

- $I_i \xrightarrow{\sigma_i,\mu_i} \perp$ *(*failing finite chase*)*
- *or there is no dependency $\sigma_m \in \Sigma \cup \Sigma_{egd}$ such that there exists a homomorphism $\mu_m$ for which we can apply $\sigma_i$ to $I_m$ with $\mu_m$ (*successful finite chase*)*

*In case of a finite chase sequence, the result output by the chase procedure is the instance $I_m$.*

**Inverse of a set of tgds** In the following, the *inverse of a set of s-t tgds* is defined as follows:

Definition 3.6 (Inverse of a set of s-t tgds).
*Let $\Sigma$ be a set of* s-t tgds *from* $\mathbf{S}$ *to* $\mathbf{T}$.
*Then the inverse $\Sigma^{-1}$ of $\Sigma$ is the set :*

$$\Sigma^{-1} = \{\sigma^{-1} : \forall \overline{x}, \forall \overline{z}, \psi(\overline{x}, \overline{z}) \rightarrow \exists \overline{y}, \phi(\overline{x}, \overline{y}) | (\sigma : \forall \overline{x} \forall \overline{y}, \phi(\overline{x}, \overline{y}) \rightarrow \exists \overline{z}, \psi(\overline{x}, \overline{z})) \in \Sigma\}$$

*with $\overline{x}$ the vector of variable shared between $\phi$ and $\psi$.*

Notice that this definition is different from the definitions of inverse from Fagin [Fag07],Fagin *et al.* [FKPT08] and Arenas *et al.* [APRR09].

**Conjunctive query and certain answers** We rely on the notion of *conjunctive queries*, whose definition is as follows:

Definition 3.7 (Conjunctive query).
*A* conjunctive query *is a first order formula of the form:*

$$\exists \overline{x}, \phi(\overline{x}, \overline{y})$$

*where $\phi(\overline{x}, \overline{y})$ is a conjunction of relational atoms and $\overline{y}$ is a vector of free variables.*

We use the notation $p(I)$ to denote the answers to a *conjunctive query* $p$ over an instance $I$. We also define a subclass of the conjunctive queries:

Definition 3.8 (Boolean conjunctive queries).
*A* boolean conjunctive queries *is a* conjunctive query *without free variables (i.e., a conjunctive query of the form $\exists \overline{x}, \phi(\overline{x})$).*

As we have already illustrated in Example 1.2 on page 23, if there exists a solution of an instance under a given mapping $\mathcal{M}$ then there exists an infinity of solutions of this instance under $\mathcal{M}$. If there exists an infinity of solutions, this raises the problem of answering a query over the target schema of these solutions. To solve this problem we rely on the notion of *certain answers*, which correspond to answers that are true in every possible solution. More formally, we define the notion of *certain answer* as follows:

Definition 3.9 (Certain answer).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Let* $I$ *be a source instance over* $\mathbf{S}$.
*Let* $sol(\mathcal{M}, I)$ *be the set of all solutions for instance* $I$ *under* $\mathcal{M}$.
*Let* $p$ *be a conjunctive query over* $\mathbf{T}$.
*Then the certain answer of* $p$ *with respect to* $I$ *under* $\mathcal{M}$ *is:*

$$certain(\mathcal{M}, p, I) = \bigcap \{p(J) | J \in sol(\mathcal{M}, I)\}$$

It should be noted that the *certain answer* of a conjunctive query $p$ with respect to a source instance $I$ under a mapping $\mathcal{M}$ can be obtained by computing the solution of $p$ over a *universal solution* for $I$ under $\mathcal{M}$ (Fagin *et al.* [FFF+05]).

**Critical instance** In the following, the *critical instance* (Marnette *et al.* [MG10], Grau *et al.* [GHK+13]) of a source schema is used to propagate the information about the visibility of attributes through mappings. In our context, *critical instances* are defined as follows:

Definition 3.10 (Critical instance).
*Let* $\mathbf{S}$ *be a database schema.*
*Let* $*$ *be a special constant called the* critical constant.
*Then the* critical instance *of* $\mathbf{S}$ *is the instance* $\mathsf{Crt}_\mathbf{S}$ *such that:*

- *for each n-ary relation* $R \in \mathbf{S}$, *there exists a tuple* $R\underbrace{(*, \ldots, *)}_{n} \in \mathsf{Crt}_\mathbf{S}$
- *there is no tuple* $R'(t_1, \ldots, t_n) \in \mathsf{Crt}_\mathbf{S}$ *such that* $R' \notin \mathbf{S}$ *or* $\exists t_i, t_i \neq *$

## 3.2 Problem overview and running example

In this section, we provide an overview of our problem setting as well as the running example that is used through the rest of the chapter. The notions of privacy preservation and the repairing algorithms that are exposed in this chapter are based on the setting illustrated in Figure 3.1 on page 96. In this setting, we consider a set of *policy views* $\mathcal{V}$ defined over a source schema $\mathbf{S}$. This set of views takes the form of a *GAV tuples generating dependencies* (Garcia-Molina *et al.* [GMUW08]), forming the *GAV mapping* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ where the target schema $\mathbf{V}$ is the schema of the views occurring in $\mathcal{V}$ (we recall that the definition of a *GAV mapping* is given in Definition 1.5 on page 22).

These policy views are a representation of the information that is considered safe to expose in the source instance $I$ over $\mathbf{S}$. When a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$

is provided by a user, there is no guarantee that this mapping does not provide information that is hidden by the set of policy views $\mathcal{V}$.

We first need to identify what is an information leakage, and how to detect it. Inspired by prior work on privacy-preservation from Nash *et al.* [ND07] and Benedikt *et al.* [BGK17], we define a set of s-t tgds to be safe with respect to the policy views if *every positive information* that is kept secret by the policy views is also kept secret by the s-t tgds. We also build upon these works to propose our approach to detect information leaks. One should note that the privacy-preservation protocol proposed in this chapter relies only on the schema and consequently is *data-independent*. This allows our privacy-preservation guarantees to be valid over all instances over the source schema, and to remain valid when updates are performed over these instances.

In a second time, if the mapping $\mathcal{M}$ exposes forbidden information (i.e., if this mapping is *unsafe*), then we need to rewrite $\mathcal{M}$ into a new mapping $\mathcal{M}'$ that is *safe* with respect to the set of policy views $\mathcal{V}$.

In the rest of this chapter, we will illustrate our approach by means of the following running example which has been inspired by a real world scenario from a hospital in the UK:

*Example* 3.1. *We consider a source schema* $\mathbf{S}$ *containing the following relation symbols:*

$$\mathbf{S} = \{Patient; NorthHospital; SouthHospital; Oncology; Student\}$$

*The relation symbol Patient is associated with attributes:*

$$Patient(idIns, name, ethnicity, county)$$

*and stores for each person registered with the NHS: her/his insurance number (idIns), her/his name (name), her/his ethnicity group (ethnicity), and her/his county (county).*

*The relation symbols NorthHospital and SouthHospital are associated with attributes:*

$$NorthHospital(idIns, disease) \text{ and } SouthHospital(idIns, disease)$$

*and store for each patient who has been admitted to some hospital in the north (NorthHospital) or the south (SouthHospital) of the UK: her/his insurance number (idIns) and the reason for being admitted to the hospital (disAtt).*

*The relation symbol Oncology is associated with attributes:*

$$Oncology(idIns, treat, progr)$$

*and stores for each patient in oncology departments: her/his insurance number (idIns), her/his treatment (treat), and their progress (progr).*

*Finally, relation Student is associated with attributes:*

$$Student(idIns, name, ethnicity, county)$$

*and stores for each student in the UK: her/his insurance number (idIns), her/his name (name), her/his ethnicity group (ethnicity), and her/his county (county).*

*In order to serve as reference* policy views *in our running example, we consider the following set of policy views $\mathcal{V}$:*

$$\begin{aligned}
\mathcal{V} = \{ \\
&\sigma_{V_1} = Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\
&\qquad \rightarrow V_1(ethn, disease); \qquad (3.1) \\
&\sigma_{V_2} = Patient(idIns, name, ethn, county) \wedge SouthHospital(idIns, disease) \\
&\qquad \rightarrow V_2(county, disease); \qquad (3.2) \\
&\sigma_{V_3} = Oncology(idIns, treat, progr) \\
&\qquad \rightarrow V_3(treat, progr); \qquad (3.3) \\
&\sigma_{V_4} = Student(idIns, name, ethn, county) \\
&\qquad \rightarrow V_4(ethn) \qquad (3.4) \\
\}
\end{aligned}$$

*These policy views define the information that is safe to make available to public. View $\sigma_{V_1}$ projects the ethnicity groups and the hospital admittance reasons for patients in the north of the UK; $\sigma_{V_2}$ projects the counties and the hospital admittance reasons for patients in the north of the UK; $\sigma_{V_3}$ projects the treatments and the progress of patients of oncology departments; $\sigma_{V_4}$ projects the ethnicity groups of the school students.*

*These policy views are safe with respect to the NHS privacy preservation protocol. Indeed, the NHS 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 $V_2$ give access to results concerning patients from a very large geographical area and, thus, do not leak any sensitive information as the probability of de-anonymizing a patient is significantly small. Analogously, in views $V_3$ there is no way to link a patient to her/his treatment or her/his progress, thus the view is considered to be safe with respect to the NHS privacy preservation protocol. The last view $V_4$ projects an attribute which is not considered sensitive in the NHS privacy preservation protocol.*

*Considering the target schema:*

$$\mathbf{V} = \{ V_1; V_2; V_3; V_4\}$$

*of the set of views $\mathcal{V}$, we define the* GAV mapping*:*

$$\mathcal{M}_{\mathcal{V}} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$$

*that will serve as reference for our privacy-preservation protocol (Figure 3.1).*

*Finally, we consider the* set of s-t tgds $\Sigma$ *such that:*

$$\begin{aligned}
\Sigma = \{ & \\
& \sigma_e = Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\
& \qquad \rightarrow EthnicityDisease(ethn, disease); \qquad (3.5) \\
& \sigma_c = Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\
& \qquad \rightarrow CountyDisease(county, disease); \qquad (3.6) \\
& \sigma_s = Student(idIns, name, ethn, county) \wedge Oncology(idIns, treat, progr) \\
& \qquad \rightarrow StudentOncology(ethn) \qquad (3.7) \\
\} &
\end{aligned}$$

*The tgds $\sigma_e$ and $\sigma_c$ project information that is similar to the information projected by the views $\sigma_{V_1}$ and $\sigma_{V_2}$, respectively. However, both of these views focus on patients admitted in hospitals in the north of the UK, when view $\sigma_{V_2}$ focuses on patients admitted in hospitals in the south of the UK. Finally, the tgd $\sigma_s$ projects the ethnicity groups of students who have been in some oncology department. Considering the target schema:*

$$\mathbf{T} = \{EthnicityDisease; CountyDisease; StudentOncology\}$$

*of the set of tgds $\Sigma$, we define the* GAV mapping*:*

$$\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$$

*that will serve as mapping whose safety we want to assess with respect to the mapping $\mathcal{M}_{\mathcal{V}}$(Figure 3.1 on page 96).*

*In the rest of this chapter, we will address the following questions over our running example:*

- *Are the s-t tgds in $\Sigma$ safe with respect to the policy views in $\mathcal{V}$?*
- *Are there any formal guarantees for privacy preservation in the context of policy views?*
- *If the s-t tgds in $\Sigma$ are not safe with respect to the policy views in $\mathcal{V}$, how could we repair them and provide formal privacy preservation guarantees?*

In the next section, we will define our notions of safety and we will provide a method to assess a mapping safety with respect to a set of policy views.

## 3.3 Privacy preservation

In this section, we introduce our notion of privacy preservation based on the setting illustrated in Figure 3.1 (page 96). Our goal in this section is to verify if a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ provided by a user is safe with respect to a set of policy views $\mathcal{V}$ over the same source schema $\mathbf{S}$. In other words, our goal is to check if the mapping $\mathcal{M}$ does not expose more information than the mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$.

At first, we will provide a formal privacy-preservation protocol. Next, we provide a simple way to check whether a mapping preserves the privacy (i.e., is safe) with respect to a set of policy views.

### 3.3.1 A formal privacy-preservation protocol

We build our privacy preservation protocol upon the protocol introduced in Benedikt *et al.* [BGK17]. However, their privacy preservation protocol is limited to boolean conjunctive queries, whereas in the following we formalize and extend the notion of privacy preservation to non-boolean conjunctive queries.

**Indistinguishability of instances.** First, we want to express the condition for which source instances cannot be distinguished through a mapping $\mathcal{M}$. Informally, this is the case if, for all of these instances, every conjunctive query has the same certain answers over the mapping $\mathcal{M}$. This is done through the notion of *indistinguishability of two source instances* which is formally defined as follows:

Definition 3.11 (Indistinguishability of instances, adapted from Benedikt *et al.* [BGK17]).
*Let $I$ and $I'$ be two instances over a schema $\mathbf{S}$.*
*Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a mapping.*
*Then $I$ and $I'$ are indistinguishable with respect to $\mathcal{M}$ if, for every conjunctive query $p$ over $\mathbf{T}$:*

$$certain(p, I, \mathcal{M}) = certain(p, I', \mathcal{M})$$

We use the notation $I \equiv_\mathcal{M} I'$ to denote that two instances $I$ and $I'$ are indistinguishable with respect to a mapping $\mathcal{M}$. It should be noted that this notion of indistinguishability is not sufficient to guarantee any information privacy, as the indistinguishable source instances can expose the same sensitive information. We formalize our notion of information privacy in the next paragraph.

**Non-disclosure of information.** By leveraging the notion of indistinguishability of instances, we now provide the notion of *non-disclosure of information* by a mapping. Informally, we represent the information that is unsafe to expose with the use of a conjunctive query $p$. A mapping is considered safe if, for every source instance, there exists an indistinguishable source instance over which $p$ returns an empty result. This is formally defined as follows:

Definition 3.12 (Non disclosure of information by a mapping, adapted from Benedikt *et al.* [BGK17]).
*Let $p$ be a conjunctive query over the source schema $\mathbf{S}$.*
*Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a mapping.*
*Then we say that $\mathcal{M}$* does not disclose the query $p$ over $\mathbf{S}$ on any instance of $\mathbf{S}$ *if, for each instance $I$ over $\mathbf{S}$, there exists an instance $I'$ over $\mathbf{S}$ such that:*

$$I \equiv_{\mathcal{M}} I' \text{ and } p(I') = \emptyset$$

Benedikt *et al.* [BGK17] have shown that the problem of checking whether a mapping $\mathcal{M}$ over $\mathbf{S}$ does not disclose a *boolean and constant-free conjunctive query* $p$ on any source instance of $\mathbf{S}$ is decidable for GAV mappings, i.e., mappings with a set of source-to-target tgds consisting of CQ views (Definition 1.5 on page 22).

**Foundations of the visible chase with bags.** In order to obtain the information exposed by a given mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, we rely on the *visible chase procedure* $\mathsf{visChase}(\mathcal{M})$ which outputs an instance based on the *critical instance* $\mathsf{Crt}_{\mathbf{S}}$ for $\mathbf{S}$. In this output instance, the only constant occurring is the critical constant $*$ corresponding to the tuples' positions that are visible, i.e., the positions for which constants are exported into the target instances. In this same instance, the non-exported variables are represented using labelled nulls. We illustrate such an instance output by the *visible chase procedure* in the following example:
*Example* 3.2. *Given a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that:*

$$\Sigma = \{S(x, y) \wedge U(y, z) \rightarrow T(x, z)\}$$

*and the corresponding critical instance:*

$$\mathsf{Crt}_{\mathbf{S}} = \{S(*, *); U(*, *)\}$$

*then the instance output by the visible chase will be an instance:*

$$\mathsf{visChase}(\mathcal{M}) = \{S(*, n_1); U(n_1, *)\}$$

*In this instance, it can be seen that the values of variables $x$ and $z$, which are exported through $\mathcal{M}$, are represented using the critical constant $*$. Moreover, the exported information about the join between the occurences of variable $y$ are represented through the null value $n_1$.*

The principle of the *visible chase procedure* has been described in Benedikt *et al.* [BGK17]. However, for the purpose of efficiently repairing mappings, we propose a new variant of the *visible chase procedure* in which the produced tuples are organized into subinstances called *bags* that are defined as follows:

Definition 3.13 (Bags over a chase result).
*Let $I \xrightarrow{\sigma,\mu} I'$ be a chase step (Definition 3.3 on page 97).*
*Then the* bag *corresponding to this chase step is the set of tuples $\beta_{\sigma,\mu}$ containing each tuple generated by the chase step $I \xrightarrow{\sigma,\mu} I'$.*

The steps of our proposed variant of the *visible chase procedure* are given in Algorithm 6.

Before explaining in details the steps of Algorithm 6, we introduce some additional notions. At first, in order to explain our algorithm, we also need to define the notion of *derived egds* as follows:

Definition 3.14 (Derived egds from a tgd).
*Let $\mathbf{S}$ and $\mathbf{T}$ be a source and a target schema, respectively.*
*Let $I$ be an instance over the schema $\mathbf{S}$.*
*Let $\sigma$ be a source-to-target tgd from $\mathbf{S}$ to $\mathbf{T}$.*
*Let $\mathsf{exported}(\sigma)$ be the set of variables occurring in both the left-hand side and the right-hand side of $\sigma$.*
*Let $h$ be a homomorphism from $\mathsf{body}(\sigma)$ into $I$ such that $h(x) \in \mathsf{Nulls}$ for some $x \in \mathsf{exported}(\sigma)$.*
*Then, the* derived egd from $\sigma$ in $I$ *is the egd:*

$$\mathsf{body}(\sigma) \rightarrow \bigwedge \{x = * | x \in \mathsf{exported}(\sigma) \text{ and } h(x) \in \mathsf{Nulls}\} \tag{3.8}$$

We use the notation $\mathsf{tgd}(\sigma_{deriv})$ to denote the s-t tgd from which an egd $\sigma_{deriv}$ is derived, and the notation $\mathsf{egd}(\sigma, I)$ to denote the egd derived from the s-t tgd $\sigma$ for an instance $I$.

*Example* 3.3. *Given an instance:*

$$I = \{S(\mathtt{n}_1, *)\}$$

*where $n_1$ is a labelled null and $*$ the critical constant. Then, if we have a tgd:*

$$\sigma : S(x, y) \rightarrow T(x, y)$$

**Algorithm 6** VISCHASE($\mathcal{M}$)

**Input:** A mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$.
**Output:** A set of bags $B$ illustrating the information exposed by $\mathcal{M}$.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$
$B_0 \leftarrow \mathsf{bagChaseTGDs}(\Sigma, \mathsf{Crt}_{\mathbf{S}})$
$B_1 \leftarrow \mathsf{bagChaseTGDs}(\Sigma^{-1}, \bigcup_{\beta \in B_0} \beta)$
$\Sigma_{\approx} \leftarrow \{\mathsf{egd}(\sigma, \bigcup_{\beta \in B_1} \beta) | \sigma \in \Sigma\}$
**return** $\mathsf{bagChaseEGDs}(\Sigma_{\approx}, B_0 \cup B_1)$

**procedure** BAGCHASETGDS($\Sigma, I$)
    $B \leftarrow \emptyset$
    **for each tgd** $\sigma \in \Sigma$ **do**
        **for each active trigger** $h : \mathsf{body}(\sigma) \rightarrow I$ **do**
            **create** a fresh bag $\beta$ with tuples $h'(\mathsf{head}(\sigma))$
            **add** $\beta$ to $B$
        **end for**
    **end for**
    **return** $B$
**end procedure**

**procedure** BAGCHASEEGDS($\Sigma_{\approx}, B$)
    $i \leftarrow 0$
    $I_i \leftarrow \bigcup_{\beta \in B} \beta$
    **do**
        $i \leftarrow i + 1$
        **for each egd** $\sigma \in \Sigma_{\approx}$ **do**
            **for each active trigger** $h : \mathsf{body}(\sigma) \rightarrow I_{i-1}$ **do**
                **if** $\exists x \in \mathsf{exported}(\sigma)$ such that $h(x) \neq *$ **then**
                    Let $\beta$ be the derived bag for $\sigma$ and $h$ in $I_{i-1}$
                    **add** $\beta$ to $B$
                    $I_i \leftarrow I_i \cup \beta$
                **end if**
            **end for**
        **end for**
    **while** $I_{i-1} \neq I_i$
    **return** $B$
**end procedure**

*we can find a homomorphism:*

$$h : \{S(x, y) \mapsto S(\mathtt{n}_1, *)\}$$

*such that* $h(x) \in \mathsf{Nulls}$ *and* $x$ *occurs in both left and right-hand sides of* $\sigma$*. This leads to the following* derived egd from $\sigma$ in $I$*:*

$$\epsilon : S(x, y) \rightarrow x = *$$

We now define the *set of derived egds for a mapping* $\mathcal{M}$ *in an instance* $I$:

Definition 3.15 (Set of derived egds from a mapping).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Let* $I$ *be an instance over* $\mathbf{S}$*.*
*Then the set of derived egds from the set of s-t tgds* $\Sigma$ *for an instance* $I$ *is the set of egds* $\Sigma_{\approx}$ *such that:*

$$\Sigma_{\approx} = \bigcup_{\sigma \in \Sigma} \mathsf{egd}(\sigma, I)$$

From this notion of *derived egds*, in order to formalize the provenance of the tuples generated during an execution of the *visible chase procedure*, we define the notion of relevance of a bag. Intuitively, a bag is said *relevant* for a derived egd $\sigma$ in an instance $I$ if this bag contains some tuples that can lead to trigger $\sigma$ and to replace a labelled null with a constant. More formally, this is defined as follows:

Definition 3.16 (Relevance of a bag).
*Let* $I$ *be an instance resulting from a* chase sequence*.*
*Let* $B = \{\beta_1; \ldots; \beta_m\}$ *be the set of bags generated during the* chase sequence *that leads to* $I$*.*
*Let* $\sigma$ *be the egd derived from an s-t tgd* $\sigma'$ *for* $I$*.*
*Let* $h$ *be an active trigger*[2] *for* $\sigma$ *in* $I$*.*
*Then, a bag* $\beta_i \in B$ *is* relevant *for the egd* $\sigma$ *and the homomorphism* $h$ *if:*

- *there exist some tuple* $t \in h(\mathsf{body}(\sigma))$ *such that* $t \in \beta_i$
- *there exist some variables* $x \in \mathsf{body}(\sigma)$ *such that:*
  *(a)* $x$ *occurs in an equality atom of the right-hand side of* $\sigma$*,*
  *(b)* $h(x)$ *is a labelled null occurring in* $\beta_i$*.*

From this notion of *relevance of a bag*, we can define the notions of *derived bag* and *predecessors of a bag* as follows:

[2]We recall that the notion of active trigger is defined in the Definition 3.3 of *chase step* on page 97.

Definition 3.17 (Derived and predecessors bags).
*Let $I$ be an instance resulting from a* chase sequence.
*Let $B = \{\beta_1; \ldots; \beta_m\}$ be the set of bags generated during the* chase sequence *that leads to $I$.*
*Let $\sigma$ be the egd derived from an s-t tgd $\sigma'$ for $I$.*
*Let $h$ be an active trigger for $\sigma$ in $I$.*
*Let $\beta_i, \ldots, \beta_{i+n} \subseteq B$ be the set of bags that are relevant for $\sigma$ and $h$ in $B$.*
*Let $\mu$ be a morphism such that, for any equality atom $x_i = x_j$ in $\mathsf{head}(\sigma)$:*

- *$\mu = \{h(x_j) \mapsto h(x_i)\}$ if $h(x_i) = *$*
- *$\mu = \{h(x_i) \mapsto h(x_j)\}$ if $h(x_i) \notin \mathsf{Const}$.*

*Then, the* derived *bag $\beta$ for $\sigma$ and $h$ in $I$ is the set:*

$$\beta = \bigcup_{j=i}^{i+n} \mu(\beta_j)$$

*The bags $\beta_i, \ldots, \beta_{i+n}$ are called the* predecessors *of $\beta$.*
*We use $\beta_j \prec \beta$ to denote that $\beta_j$ is a predecessor of $\beta$, for $i \leq j \leq i+n$.*

**Execution of the visible chase with bags and universal source instance.** We are now ready to proceed with the description of Algorithm 6 on page 107. Given an input mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, Algorithm 6 computes an instance whose tuples are organized into bags.

The first step, at line 2 of Algorithm 6, computes the set of bags $B_0$ by chasing $\mathsf{Crt_S}$ using the set of s-t tgds $\Sigma$ of $\mathcal{M}$. This chase sequence is done through the call to procedure bagChaseTGDs. During this chase sequence, for each s-t tgd $\sigma$ in $\Sigma$ (Algorithm 6 line 8) and for each active trigger $h$ from $\mathsf{body}(\sigma)$ into $\mathsf{Crt_S}$ (Algorithm 6 line 9), we compute a bag $\beta$ containing the tuples in $h'(\mathsf{head}(\sigma))$. It should be noted that the computation of the homomorphism $h$ and its extension $h'$ is done as described in Definition 3.3 on page 97 (tgd chase step paragraph).

Analogously, the procedure bagChaseTGDs is called in the second step of our chase, at line 3. At this step, we chase the set of tuples produced during the first step $I_0 = \bigcup_{\beta \in B_0} \beta$ with the inverse $\Sigma^{-1}$ of the set of s-t tgds $\Sigma$, leading to a new set of bags $B_1$.

Finally, the set of all tuples generated during the two previous steps:

$$I_1 = (\bigcup_{\beta \in B_0} \beta) \cup (\bigcup_{\beta \in B_1} \beta)$$

is chased with the set $\Sigma_{\approx} = \bigcup_{\sigma \in \Sigma} \mathsf{egd}(\sigma, I_1)$ of all egds derived from $\Sigma$ in $I_1$ (Algorithm 6 line 5). During each chase step $i$, for each egd $\sigma \in \Sigma_{\approx}$, the derived bag for $\sigma$ and $h$ in the instance $I_i$ is added to the output set of bags (line 24).

It should be noted that, $\Sigma_{\approx}$ aims at "disambiguating" as many labelled nulls occurring in $I_1$ as possible, by unifying them with the critical constant $*$. Since the critical constant $*$ represents the information that is "visible" to a third-party, chasing with $\Sigma_{\approx}$ computes the *maximal information* which can be retrieved by a third-party from the source instance.

It also should be noted that the visible chase algorithm has been shown to always terminate by Benedikt *et al.* [BGK17].

In the following, given the set of bags $B = \mathsf{visChase}(\mathcal{M})$ output by the visible chase, we will denote by $\mathsf{univSourceInst}(\mathcal{M})$ the instance $\bigcup_{\beta \in B} \beta$. This instance $\mathsf{univSourceInst}(\mathcal{M})$ is called a *universal source instance*, and is formally defined as follows:

Definition 3.18 (Universal source instance).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Then a universal source instance* $I$ *is an instance such that the visible part of any instance of* $\mathbf{S}$ *(i.e., the subinstance that becomes available through the mappings) has an homomorphism into it.*

It should be noted that the *universal source instance* for a given mapping is unique (modulo constant renaming).

Informally, relying on Definition 3.12, to check if a mapping $\mathcal{M}$ over a schema $\mathbf{S}$ does not disclose the *boolean and constant-free conjunctive query* $p$, we verify that there is no homomorphism from $p$ into its *universal source instance*. This is formalized in the following theorem:

Theorem 3.1 (Relation between query disclosure and universal instance).
*Let* $p$ *be a conjunctive query over the source schema* $\mathbf{S}$.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a GAV mapping.*
*Let* $I$ *be the* universal source instance *for* $\mathcal{M}$.
*Then,* $\mathcal{M}$ does not disclose the query $p$ over $\mathbf{S}$ on any instance of $\mathbf{S}$ *if and only if there does not exist a homomorphism* $h : p \to I$.

In the following example, we illustrate on the running example the generation of a *universal source instance* with the visible chase algorithm:

*Example* 3.4. *In this example, we use the policy views and the mapping of our running example provided in Example 3.1 on page 101.*

*We first present the computation of:*

$$\mathsf{univSourceInst}(\mathcal{M}_{\mathcal{V}}) = \bigcup_{\beta \in \mathsf{visChase}(\mathcal{M}_{\mathcal{V}})} \beta$$

*At first, we consider the critical instance* $\mathsf{Crt}_{\mathbf{S}}$ *of the source schema* $\mathbf{S}$ *as defined in Definition 3.10 on page 100. This critical instance contains the following tuples:*

$$\begin{array}{lll} Patient(*,*,*,*) & NorthHospital(*,*) & SouthHospital(*,*) \\ Oncology(*,*,*) & Student(*,*,*,*) & \end{array} \quad (\mathsf{Crt}_{\mathbf{S}})$$

*where* $*$ *is the critical constant. The first step, at line 2 of Algorithm 6, leads to the set of bags* $B_0$ *containing the following elements:*

$$\begin{array}{ll} \{V_1(*,*)\} & \{V_2(*,*)\} \\ \{V_3(*,*)\} & \{V_4(*)\} \end{array} \quad (B_0)$$

*Then, the set of bags* $B_1$ *is computed by chasing the instance* $I_0 = \bigcup_{\beta \in B_0} \beta$ *using* $\mathcal{V}^{-1}$*, leading to the set of bags* $B_1$ *containing the following elements:*

$$\begin{array}{l} \{Patient(n_{idIns}, n_{name}, *, n_{county}); NorthHospital(n_{idIns}, *)\} \\ \{Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *); SouthHospital(n'_{idIns}, *)\} \\ \{Oncology(n'''_{idIns}, *, *)\} \\ \{Student(n''_{idIns}, n''_{name}, *, n''_{county})\} \end{array} \quad (B_1)$$

*where the constants prefixed by n are labelled nulls created while chasing* $\mathsf{Crt}_{\mathbf{S}}$ *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 labelled null, then* $\Sigma_{\approx}$ *will be empty (see Definition 3.14). Thus, we obtain:*

$$\mathsf{univSourceInst}(\mathcal{M}_{\mathcal{V}}) = \bigcup_{\beta \in B_1} \beta$$

*We next present the computation of:*

$$\mathsf{univSourceInst}(\mathcal{M}) = \bigcup_{\beta \in \mathsf{visChase}(\mathcal{M})} \beta$$

*The instance* $I'_1$ *computed by chasing the output of line 2 by* $\Sigma_{st}^{-1}$ *will consist of the tuples:*

$$\begin{array}{ll} Patient(n_{idIns}, n_{name}, *, n_{county}) & NorthHospital(n_{idIns}, *) \\ Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *) & NorthHospital(n'_{idIns}, *) \\ Student(n''_{idIns}, n''_{name}, *, n''_{county}) & Oncology(n''_{idIns}, n''_{treat}, n''_{progr}) \end{array} \quad (I'_1)$$

*Since there exists a homomorphism from the body of $\sigma_e$ into $I'_1$ mapping the exported variable e into the labelled null $n'_{ethn}$, and since there exists another homomorphism from the body of $\sigma_c$ into $I'_1$ mapping the exported variable c into the labelled null $n_{county}$, then $\Sigma_{\approx}$ will contains the egds $\epsilon_1$ and $\epsilon_2$ shown below:*

$$\begin{aligned} &Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\ &\quad \rightarrow ethn \approx * && (\epsilon_1) \\ &Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\ &\quad \rightarrow county \approx * && (\epsilon_2) \end{aligned}$$

*The last step of the visible chase involves chasing $I'_1$ using the derived tgds in $\Sigma_{\approx}$.*

*Without loss of generality, we can assume that the chase will first consider the egd $\epsilon_1$ and then $\epsilon_2$. During the first step of the chase, there exists a homomorphism from* $\mathsf{body}(\epsilon_1)$ *into $I'_1$. Hence, $n'_{ethn} = *$. During the second step of the chase, there exists a homomorphism from* $\mathsf{body}(\epsilon_2)$ *into $I'_1$ and, hence, $n_{county} = *$. Thus, the instance computed at the end of the second round of the chase contains the tuples:*

$$\begin{array}{ll} Patient(n_{idIns}, n_{name}, *, *) & NorthHospital(n_{idIns}, *) \\ Patient(n'_{idIns}, n'_{name}, *, *) & NorthHospital(n'_{idIns}, *) \\ Student(n''_{idIns}, n''_{name}, *, n''_{county}) & Oncology(n''_{idIns}, n''_{treat}, n''_{progr}) \end{array} \qquad (I'_2)$$

*Since there exists no active trigger for $\epsilon_1$ or $\epsilon_2$ in the instance $I'_2$, the chase terminate and we have:* $\mathsf{univSourceInst}(\mathcal{M}) = I'_2$.

*We summarize the construction of the bags $\beta_1, \ldots, \beta_5$ in* $\mathsf{visChase}(\mathcal{M})$ *by providing, for each bag, the dependency (tgd or egd), the homomorphism and the instance from which this bag derive:*

– *the bag:*

$$\beta_1 = \{Student(n''_{idIns}, n''_{name}, *, n''_{county}); Oncology(n''_{idIns}, n''_{treat}, n''_{progr})\}$$

*derives from the tgd $\sigma_s^{-1}$ and the homomorphism:*

$$h_1 = \{ethn \mapsto *\}$$

*in instance $\{StudentOncology(*)\}$*

– *the bag:*

$$\beta_2 = \{Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *); NorthHospital(n'_{idIns}, *)\}$$

*derives from the tgd* $\sigma_c^{-1}$ *and the homomorphism:*

$$h_2 = \{county \mapsto *, disease \mapsto *\}$$

*in instance* $\{CountyDisease(*, *)\}$

– *the bag:*

$$\beta_3 = \{Patient(n_{idIns}, n_{name}, *, n_{county}), NorthHospital(n_{idIns}, *)\}$$

*derives from the tgd* $\sigma_e^{-1}$ *and the homomorphism:*

$$h_3 = \{ethn \mapsto *, disease \mapsto *\}$$

*in instance* $\{EthnicityDisease(*, *)\}$

– *the bag:*

$$\beta_4 = \{Patient(n'_{idIns}, n'_{name}, *, *), NorthHospital(n'_{idIns}, *)\}$$

*derives from the egd* $\epsilon_1$ *and the homomorphism:*

$h_4 = \{idIns \mapsto n'_{idIns}, name \mapsto n'_{name}, ethn \mapsto n'_{ethn}, county \mapsto *, disease \mapsto *\}$

*in instance* $\{Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *); NorthHospital(n'_{idIns}, *)\}$

– *the bag:*

$$\beta_5 = \{Patient(n_{idIns}, n_{name}, *, *), NorthHospital(n_{idIns}, *)\}$$

*derives from the egd* $\epsilon_2$ *and the homomorphism:*

$h_5 = \{idIns \mapsto n_{idIns}, name \mapsto n_{name}, ethn \mapsto *, county \mapsto n_{county}, disease \mapsto *\}$

*in instance* $\{Patient(n_{idIns}, n_{name}, *, n_{county}); NorthHospital(n_{idIns}, *)\}$

### 3.3.2 Preserving the privacy of policy views

Recalling that the notion of non-disclosure of information has been defined in Definition 3.12 on page 105, we consider that a GAV mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is safe with respect to a GAV mapping consisting of policy views $\mathcal{M}_\mathbf{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ if $\mathcal{M}$ does not disclose more information than $\mathcal{M}_\mathbf{V}$.

The following Definition 3.19 formalizes our notion of privacy preservation:

Definition 3.19 (Privacy preservation).
*Let* $\mathcal{M}_1 = (\mathbf{S}, \mathbf{T}_1, \Sigma_1)$ *and* $\mathcal{M}_2 = (\mathbf{S}, \mathbf{T}_2, \Sigma_2)$ *be two mappings over the same source schema* $\mathbf{S}$.
*Then* $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ on all instances of $\mathbf{S}$ *if, for each constant-free CQ* $p$ *over* $\mathbf{S}$*: if* $\mathcal{M}_1$ *does not disclose* $p$ *over* $\mathbf{S}$*, then* $\mathcal{M}_2$ *does not disclose* $p$ *over* $\mathbf{S}$.

Now, we will show in Theorem 3.2 that checking whether a mapping $\mathcal{M}_2$ is safe with respect to a reference mapping $\mathcal{M}_1$ can be done by checking if a homomorphism exists between their *universal source instances*. The proof of our theorem will rely on the two following lemmas:

Lemma 3.1.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Then* $\mathcal{M}$ does not disclose a constant-free CQ $p$ over $\mathbf{S}$ on any instance of $\mathbf{S}$, *if and only if* $(*, \ldots, *) \notin p(\mathsf{univSourceInst}(\mathcal{M}))$.

*Proof.* By adapting the proof technique of Theorem 16 from Benedikt *et al.* [BGK17], we can show that the *universal source instance* $\mathsf{univSourceInst}(\mathcal{M})$ has the following property: for each set $\mathcal{E}$ of indistinguishable instances with respect to the mapping $\mathcal{M}$, for each source instance $I \in \mathcal{E}$, there exists a homomorphism $h$ from $I$ into $\mathsf{univSourceInst}(\mathcal{M})$ mapping each schema constant into the critical constant $*$. Due to the existence of a homomorphism $h$ from $I$ into $\mathsf{univSourceInst}(\mathcal{M})$, for each source instance $I \in \mathcal{E}$, we can see that if $(*, \ldots, *) \notin p(\mathsf{univSourceInst}(\mathcal{M}))$ for a constant-free CQ $p$, then $p(I) = \emptyset$. Due to the above and due to Definition 3.12 on page 105, it follows that $\mathcal{M}$ does not disclose a constant-free CQ $p$ over $\mathbf{S}$ on any instance of $\mathbf{S}$. □

Lemma 3.1 states that, in order to check if a constant-free CQ is safe according to Definition 3.12 on page 105, we need to check if the critical tuple is among the answers to $p$ over the instance computed by $\mathsf{visChase}(\mathcal{M})$.

Next, we prove the following equivalence:

Lemma 3.2.
*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 3.2.* (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. □

Now we can state our theorem:

Theorem 3.2 (Privacy preservation checking).
*Let* $\mathcal{M}_1 = (\mathbf{S}, \mathbf{T}_1, \Sigma_1)$ *and* $\mathcal{M}_2 = (\mathbf{S}, \mathbf{T}_2, \Sigma_2)$ *be two mappings over the same source schema* $\mathbf{S}$.
*Then* $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ on all instances of $\mathbf{S}$, *if and only if there exists a homomorphism* $h$ *from* $\mathsf{univSourceInst}(\mathcal{M}_2)$ *into* $\mathsf{univSourceInst}(\mathcal{M}_1)$, *such that* $h(*) = *$.

*Proof.* Given a CQ $p$ over a source schema $\mathbf{S}$, and a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, we know from Lemma 3.1 that if $\mathcal{M}$ discloses $p$ on some instance of $\mathbf{S}$, then there exists a homomorphism of $p$ into $\mathsf{visChase}(\mathcal{M})$ mapping the free variables of $p$ into the critical constant $*$.

From the contrapositive of Lemma 3.2, we know that $\mathcal{M}_2$ does not preserve the privacy of $\mathcal{M}_1$ if there exists a CQ $p$ over $\mathbf{S}$, such that:

$$(*, \ldots, *) \notin \mathsf{univSourceInst}(\mathcal{M}_1) \text{ and } (*, \ldots, *) \in \mathsf{univSourceInst}(\mathcal{M}_2)$$

We will now prove that $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ if and only if there exists a homomorphism from $\mathsf{univSourceInst}(\mathcal{M}_2)$ into $\mathsf{univSourceInst}(\mathcal{M}_1)$ that preserves the critical constant $*$. This will be referred to as Claim $C$.

($\Rightarrow$) If $\mathcal{M}_2$ preserves the privacy of $\mathcal{M}_1$ then, for all CQ $p$:

$$(*, \ldots, *) \notin p(\mathsf{univSourceInst}(\mathcal{M}_1)) \Rightarrow (*, \ldots, *) \notin p(\mathsf{univSourceInst}(\mathcal{M}_2))$$

From the above and from Lemma 3.2, it follows that there exists a homomorphism:

$$h : \mathsf{univSourceInst}(\mathcal{M}_2) \rightarrow \mathsf{univSourceInst}(\mathcal{M}_1) \text{ such that } h(*) = *$$

($\Leftarrow$) The proof proceeds by contradiction. Assume that there exists a homomorphism $h$ from $\mathsf{univSourceInst}(\mathcal{M}_2)$ into $\mathsf{univSourceInst}(\mathcal{M}_1)$ 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 $\mathbf{S}$ such that:

$$(*, \ldots, *) \notin p(\mathsf{univSourceInst}(\mathcal{M}_1)) \text{ and } (*, \ldots, *) \in p(\mathsf{univSourceInst}(\mathcal{M}_2))$$

Let $h_2$ be the homomorphism from $p$ into $\mathsf{univSourceInst}(\mathcal{M}_2)$ 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 $\mathsf{univSourceInst}(\mathcal{M}_1)$ mapping its free variables into $*$, i.e., $(*, \ldots, *) \in p(\mathsf{univSourceInst}(\mathcal{M}_1))$. This contradicts our original assumption and hence concludes the proof of Claim $C$. Claim $C$ 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:

$$h : \mathsf{univSourceInst}(\mathcal{M}_2) \rightarrow \mathsf{univSourceInst}(\mathcal{M}_1) \text{ such that } h(*) = *$$

□

According to Theorem 3.2, in order to verify that a mapping $\mathcal{M}_2$ is safe with respect to a mapping $\mathcal{M}_1$, we need to check if there exists a homomorphism from $\mathsf{univSourceInst}(\mathcal{M}_2)$ into $\mathsf{univSourceInst}(\mathcal{M}_1)$ that maps the critical constant $*$ into itself. If there exists such a homomorphism, we say that $\mathsf{univSourceInst}(\mathcal{M}_2)$ is *safe* with respect to $\mathsf{univSourceInst}(\mathcal{M}_1)$, and we say that $\mathsf{univSourceInst}(\mathcal{M}_2)$ is unsafe otherwise.

We illustrate this safety checking in the following example:

*Example* 3.5. *Continuing from Example 3.4 on page 110, we want to verify if the mapping $\mathcal{M}$ is safe with respect to the mapping $\mathcal{M}_\mathcal{V}$. We recall that instance* $\mathsf{univSourceInst}(\mathcal{M})$ *contains the tuples:*

$$\begin{array}{ll} Patient(n_{idIns}, n_{name}, *, *) & NorthHospital(n_{idIns}, *) \\ Patient(n'_{idIns}, n'_{name}, *, *) & NorthHospital(n'_{idIns}, *) \\ Student(n''_{idIns}, n''_{name}, *, n''_{county}) & Oncology(n''_{idIns}, n''_{treat}, n''_{progr}) \end{array}$$

*and that instance* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ *contains the tuples:*

$$Patient(n_{idIns}, n_{name}, *, n_{county}) \qquad NorthHospital(n_{idIns}, *)$$
$$Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *) \qquad SouthHospital(n'_{idIns}, *)$$
$$Student(n''_{idIns}, n''_{name}, *, n''_{county}) \qquad Oncology(n'''_{idIns}, *, *)$$

*According to Theorem 3.2, since there does not exist a homomorphism from the instance* $\mathsf{univSourceInst}(\mathcal{M})$ *into the instance* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$*, then* $\mathcal{M}$ *is not safe with respect to the policy views in mapping* $\mathcal{M}_\mathcal{V}$*. In other words, for some instances over schema* $\mathbf{S}$*, the mapping* $\mathcal{M}$ *will disclose information that is not disclosed by* $\mathcal{M}_\mathcal{V}$*.*

*For example, the tuples:*

$$Student(n''_{idIns}, n''_{name}, *, n''_{county}) \qquad Oncology(n''_{idIns}, n''_{treat}, n''_{progr})$$

*in instance* $\mathsf{univSourceInst}(\mathcal{M})$ *show that we can potentially identify a student who has been admitted to the oncology department. Such an information leak can occur, for example, if there exists only one student in the school coming from a specific ethnicity group, and this ethnicity group is returned by* $\sigma_s$*. At the opposite, this information is not disclosed by the policy views* $V_3$ *and* $V_4$ *since it is impossible to link a tuple in Student to a tuple in Oncology.*

*Moreover, the tuples:*

$$Patient(n_{idIns}, n_{name}, *, *) \qquad NorthHospital(n_{idIns}, *)$$

*in instance* $\mathsf{univSourceInst}(\mathcal{M})$ *show that the identity of a patient admitted to a hospital from the north of UK and the disease that has led to this admission can leak from the mapping* $\mathcal{M}$*, if there exists only one patient who relates to the county and the ethnicity group returned by* $\sigma_e$ *and* $\sigma_c$*. At the opposite, this information is not disclosed by the policy views* $V_1$ *and* $V_2$ *since it is impossible to obtain the county and the ethnicity group of an NHS patient at the same time.*

## 3.4 Repairing mappings

In Section 3.3 we have presented our privacy preservation protocol and a technique for verifying whether a mapping is safe with respect to another one over all possible source instances. This section presents an algorithm for repairing an unsafe mapping (with respect to the policy views of a mapping $\mathcal{M}_\mathcal{V}$) into a safe mapping, i.e. a mapping that preserves the privacy of $\mathcal{M}_\mathcal{V}$ (Definition 3.19 on page 114).

The steps of our algorithm are summarized in Algorithm 7 on page 119. It is seen that our algorithm takes as input:

- the mapping to rewrite $\mathcal{M}$;
- the mapping $\mathcal{M}_\mathcal{V}$ containing the policy views that serves as reference for the rewriting;
- a preference function prf used to select the best repair among the possible repairs explored during the process;
- a positive integer $n$ which is used during the last step of our repairing process to limit the depth of the explored rewriting tree.

It should be noted that the mechanisms behind the preference function can range from choices made using basic metrics or a direct questioning of the users, to more complex approaches such as the supervised learning of the preference function based on the user's prior decisions. In Section 3.5 on page 137, we will illustrate such a learning approach using the k-NN classification algorithm (Friedman *et al.* [FHT01]).

Our algorithm is built upon the property we have proved in Theorem 3.2 on page 115. We recall that in this theorem we have shown that a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is safe with respect to a mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ if the instance $\mathsf{univSourceInst}(\mathcal{M})$ has a homomorphism into the instance $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$. Thus, if the input mapping is unsafe with respect to the reference mapping $\mathcal{M}_\mathcal{V}$, then the goal of Algorithm 7 is to rewrite the tgds in $\mathcal{M}$ such that the derived instance has a homomorphism in $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$. During the first step of our rewriting, the mapping $\mathcal{M}$ is rewritten into a *partially safe* mapping $\mathcal{M}_{partSafe}$. As we will explain later on, *partial safety* ensures that each tgd in $\mathcal{M}_{partSafe}$ taken independently are safe with respect to $\mathcal{M}_\mathcal{V}$. However, *partial safety* does not guarantee that the whole mapping $\mathcal{M}_{partSafe}$ is safe with respect to $\mathcal{M}_\mathcal{V}$. This last point is ensured by the second step of Algorithm 7, during which the *partially safe* mapping $\mathcal{M}_{partSafe}$ is rewritten into a mapping which is *safe* with respect to $\mathcal{M}_\mathcal{V}$. The benefit of this two-step approach is that it allows repairing one or a small set of tgds at a time.

In the next section, we will focus on the notion of *partial safety* and on the first step of our algorithm.

### 3.4.1 Computing partially safe mappings

In this section, we define the notion of *partial safety* and we describe how our algorithm ensures that a mapping is *partially safe* with respect to a reference mapping.

**Algorithm 7** $\mathsf{repair}(\mathcal{M}, \mathcal{M}_\mathcal{V}, \mathsf{prf}, n)$

**Input:** A mapping to rewrite $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, a reference mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{T}, \Sigma_\mathcal{V})$, a preference function $\mathsf{prf}$, a positive integer $n$.
**Output:** A rewriting $\mathcal{M}'$ of $\mathcal{M}$ such that $\mathcal{M}'$ is safe with respect to $\mathcal{M}_\mathcal{V}$.

1: $\mathcal{M}_{partSafe} \leftarrow \mathsf{frepair}(\mathcal{M}, \mathcal{M}_\mathcal{V}, \mathsf{prf})$
2: $\mathcal{M}_{safe} \leftarrow \mathsf{srepair}(\mathcal{M}_{partSafe}, \mathcal{M}_\mathcal{V}, \mathsf{prf}, n)$
3: **return** $\mathcal{M}_{safe}$

**Partial safety.** The intuition behind partial safety is the following: the problem of the safety of a mapping $\mathcal{M}$ with respect to a mapping $\mathcal{M}_\mathcal{V}$ is reduced to the problem of checking for a homomorphism from $\mathsf{univSourceInst}(\mathcal{M})$ into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$, a first step towards checking for such a homomorphism is to look if the tgds in $\mathcal{M}$ considered independently would lead to such a homomorphism or not.

For instance, by looking at $\sigma_s$ in Example 3.1 on page 101 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 $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$.

We formally define the *partial safety* as follows:

Definition 3.20 (Partial safety).
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Let* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *be a reference mapping with* $\mathcal{V}$ *being a set of policy views. Then* $\mathcal{M}$ *is* partially safe with respect to $\mathcal{M}_\mathcal{V}$ on all instances of $\mathbf{S}$*, if there exists a homomorphism from* $\textsc{chase}(\Sigma^{-1}, \mathsf{Crt}_\mathbf{T})$ *into* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$.

From Algorithm 6 on page 107, it follows that $\Sigma$ is partially safe iff the intermediate instance $I_1 = \bigcup_{\beta \in B_1}$ computed by $\mathsf{visChase}(\mathcal{M})$ is safe, i.e., has a homomorphism into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$. This leads to the following lemma:

Lemma 3.3.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Let* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *be a reference mapping with* $\mathcal{V}$ *being a set of policy views. Then* $\mathcal{M}$ *is* partially safe with respect to $\mathcal{M}_\mathcal{V}$ on all instances of $\mathbf{S}$*, if for each* $\sigma \in \Sigma$*, there exists a homomorphism from* $\mathsf{body}(\sigma)$ *into* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ *mapping each* $x \in \mathsf{exported}(\sigma)$ *into the critical constant* $*$.

*Proof.* This follows from the construction of Algorithm 6 on page 107. □

We illustrate this lemma in the following example:

*Example* 3.6. *We recall the set of tgds* $\Sigma$ *in mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *from Example 3.1 on page 101:*

$$
\begin{aligned}
\Sigma = \{ \\
\quad \sigma_e = & Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\
& \rightarrow EthnicityDisease(ethn, disease) \\
\quad \sigma_c = & Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \\
& \rightarrow CountyDisease(county, disease) \\
\quad \sigma_s = & Student(idIns, name, ethn, county) \wedge Oncology(idIns, treat, progr) \\
& \rightarrow StudentOncology(ethn) \\
\}
\end{aligned}
$$

*We have seen in Example 3.5 on page 116 that* $\mathcal{M}$ *is unsafe with respect to the reference mapping* $\mathcal{M}_\mathcal{V}$. *We also see that, according to Lemma 3.3, the set of tgds* $\Sigma$ *is not* partially safe *with respect to* $\mathcal{M}_\mathcal{V}$ *due to the fact that there is no homomorphism from the left-hand side of* $\sigma_s$ *into the instance* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ *that contains the tuples:*

$$
\begin{array}{ll}
Patient(n_{idIns}, n_{name}, *, n_{county}) & NorthHospital(n_{idIns}, *) \\
Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *) & SouthHospital(n'_{idIns}, *) \\
Student(n''_{idIns}, n''_{name}, *, n''_{county}) & Oncology(n'''_{idIns}, *, *)
\end{array}
$$

*However, we also see that the set of tgds* $\Sigma$ *would be* partially safe *if we remove the tgd* $\sigma_s$ *from* $\Sigma$. *Indeed, since there exist homomorphisms from the bodies of* $\sigma_e$ *and* $\sigma_c$ *into* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$, *mapping their exported variables into* $*$, *the mapping formed by the triple* $(\mathbf{S}, \mathbf{T}, \{\sigma_e; \sigma_c\})$ *is* partially safe *with respect to* $\mathcal{M}_\mathcal{V}$.

From Lemma 3.3 we also derive the following lemma:

Lemma 3.4.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping.*
*Let* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *be a reference mapping with* $\mathcal{V}$ *being a set of policy views. Then* $\mathcal{M}$ *is* safe with respect to $\mathcal{M}_\mathcal{V}$ on all instances of $\mathbf{S}$, *if and only if it is partially safe with respect to* $\mathcal{M}_\mathcal{V}$ *on all instances of* $\mathbf{S}$.

From Lemma 3.3 we can see that in order to obtain a *partially safe mapping* from an initial mapping $\mathcal{M}$, we need to rewrite each tgd in $\mathcal{M}$ *independently of the others*. Furthermore, the repair of each $\sigma \in \Sigma$ involves breaking joins and hiding exported variables, such that the repaired tgd $\sigma_r$ satisfies the criterion in Lemma 3.3.

**Computing partially safe mappings.** The computation of a *partially safe* mapping is detailed in Algorithm 8 on page 123. The initialization procedure used by Algorithm 8 is detailed in Algorithm 9 on page 126. It should be noted that, for performance reasons, we do not examine rewritings that introduce atoms in the bodies of the rules. However, not considering the introduction of atoms does not affect the *completeness* of Algorithm 7 as it will be shown in the theorems at the end of this section.

The principle used by Algorithm 8 is built on Lemma 3.3. Indeed, for each $\sigma \in \Sigma$, Algorithm 8 computes a set of rewritings $\mathcal{R}_\sigma$ out of which the best rewriting is chosen according to the preference function $\mathsf{prf}$.

We will now detail the different steps of Algorithm 8 before illustrating it on an example.

At first, for each s-t tgd $\sigma$, Algorithm 8 calls the procedure in Algorithm 9. In this procedure, for each atom $B \in \mathsf{body}(\sigma)$, the procedure constructs a fresh atom $C$ and adds this atom to a set $\mathcal{C}$. The set of atoms $\mathcal{C}$ provides us with the means to identify all repairs of $\sigma$ that involve breaking joins between variable occurrences and hiding exported variables. Additionally, a homomorphism $\nu$ is constructed, linking each atom $B$ with its counterpart in the set of atoms $\mathcal{C}$. At the end of this procedure, the pair $(\mathcal{C}, \nu)$ is returned to Algorithm 8.

Then, for each homomorphism $\xi$ from $\mathcal{C}$ into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ (which corresponds to one repair of $\sigma$), lines 8–26 of Algorithm 8 modify each atom $B \in \mathsf{body}(\sigma)$ in order to obtain a conjunction with a homomorphism into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$. It should be noted that given a homomorphism $\xi$, each iteration over an atom in $\mathsf{body}(\sigma)$ takes into account the prior modifications performed during previous iterations. To this extent, these prior modifications are accumulated in the relation $\rho$ and the mapping $\mu$. Given a homomorphism $\xi$ (line 5 of Algorithm 8), the relation $\rho$ keeps for each variable $x$ occurring in $\mathsf{body}(\sigma)$, the fresh variables that were used to replace $x$ during iterations of the repairing process over $\xi$. To this extent, $\rho$ is updated in line 18 of Algorithm 8. The relation $\mu$ is used to keep a morphism from the variables of the partially repaired body into instance $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ (lines 19 and 23 of Algorithm 8). In particular, at the end of the iterations of the loop from line 8–26 of Algorithm 8, $\mu$ holds the substitution from $\mathsf{body}(\sigma)$ into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$.

One can note that an apparently simpler approach is to always replace a variables $x$ in position $p$ by a fresh variable. However, our approach allows to minimize the number of broken joins between variable occurrences, and consequently to preserve the information conveyed by these joins when this information does not lead to an unsafe mapping.

We will now describe how the body atoms of a tgd $\sigma$ are modified by

Algorithm 8 in order to obtain a rewriting of $\sigma$ which is *partially safe* with respect to the reference mapping $\mathcal{M}_\mathcal{V}$. To this extent, given a tgd $\sigma$, for each homomorphism $\xi$ in the loop at lines 5–30 of Algorithm 8, one candidate rewriting $\sigma_r$ of $\sigma$ might be computed. To obtain such a rewriting, for each atom $B \in \mathsf{body}(\sigma)$ and for each variable position $p \in \mathsf{pos}(B)$ in atom $B$, Algorithm 8 detects if the variable in position $p$ of atom $B$ should be hidden.

More precisely, if the variable $y$ in position $p$ of atom $\nu(B)$ is not mapped to the critical constant $*$ via homomorphism $\xi$, and if the variable $B|_p$ is an exported variable, this means that *the variable occuring in position $p$ of atom $B$ should not be exported* (this corresponds to the condition in line 12 of Algorithm 8). Similarly, if the variable $y$ in position $p$ of atom $\nu(B)$ is mapped to a different constant $\xi(y)$ than the constant $\mu(B|_p)$ then this means that *the variable occuring in position $p$ of atom $B$ introduces an unsafe join* (this corresponds to the condition in line 13 of Algorithm 8).

In the presence of these violations, we must replace variable $x$ in position $p$ of atom $B$, either by a variable that was used in a prior step of the repairing process, as it is done in lines 14–15 of Algorithm 8, or by a fresh variable, as done in lines 16–21 of Algorithm 8. Otherwise, in the case where there is no violation detected, the morphism $\{x \mapsto \xi(y)\}$ is added to $\mu$ if this morphism is not already in $\mu$, this is done at lines 22–23 of Algorithm 8.

Finally, the best repair for the evaluated tgd is chosen using the preference function at lines 31–34 of Algorithm 8.

We will now illustrate an execution of Algorithm 8 in the following example:

*Example* 3.7. *In this example, we demonstrate an execution of Algorithm 8 in order to repair a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that $\Sigma$ is the singleton:*

$$\Sigma = \{\sigma_1 : R_1(x, y, z) \wedge S_1(y, z, z) \rightarrow T_1(x, z)\}$$

*In order to repair $\mathcal{M}$ with respect to a mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$, Algorithm 8 only needs to use the instance* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ *as reference. Therefore, instead of providing the set of policy views $\mathcal{V}$ we provide the instance* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ *such that:*

$$\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V}) = \{R_1(*, n_1, n_2); S_1(n_1, n_2, n_2); S_1(n_1, n_3, *); S_1(n_1, *, *)\}$$

*where $n_1$–$n_3$ are labelled nulls.*

*In order to rewrite $\mathcal{M}$ into a* partially safe mapping *with respect to $\mathcal{M}_\mathcal{V}$, Algorithm 8 will compute two possible repairs for the tgd $\sigma_1$ as described below. At first, Algorithm 8 calls the procedure shown in Algorithm 9. This procedure*

**Algorithm 8** $\mathsf{frepair}(\mathcal{M}, \mathcal{M}_\mathcal{V}, \mathsf{prf})$

**Input:** A mapping to rewrite $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$
**Input:** A reference mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{T}, \Sigma_\mathcal{V})$.
**Input:** A preference function $\mathsf{prf}$.
**Output:** A rewriting $\mathcal{M}'$ of $\mathcal{M}$ s.t. $\mathcal{M}'$ is *partially safe* with respect to $\mathcal{M}_\mathcal{V}$.

1: $\Sigma' \leftarrow \Sigma$
2: **for each** $\sigma \in \Sigma$ **do**
3: $(\mathcal{C}, \nu) \leftarrow$ InitFrepairTgd($\sigma$)
4: $\mathcal{R}_\sigma \leftarrow \emptyset$
5: **for each** homomorphism $\xi : \mathcal{C} \rightarrow \mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ **do**
6: $\rho \leftarrow \emptyset$, $\mu \leftarrow \emptyset$
7: $\sigma_r \leftarrow \sigma$
8: **for each** $B \in \mathsf{body}(\sigma_r)$ **do**
9: **for each** $p \in \mathsf{pos}(B)$ **do**
10: $x \leftarrow B|_p$
11: $y \leftarrow \nu(B)|_p$
12: **if** $(x \in \mathsf{exported}(\sigma)$ **and** $* \neq \xi(y))$
13: **or** $(x \in \mathsf{dom}(\mu)$ **and** $\mu(x) \neq \xi(y))$ **then**
14: **if** $\exists x'$ s.t. $(x, x') \in \rho$ and $\mu(x') = \xi(y)$ **then**
15: $B|_p \leftarrow x'$
16: **else**
17: **create** a fresh variable $x'$
18: **add** $(x, x')$ to $\rho$
19: **add** $\{x' \mapsto \xi(y)\}$ to $\mu$
20: $B|_p \leftarrow x'$
21: **end if**
22: **else if** $x \notin \mathrm{dom}\,\mu$ **then**
23: **add** $\{x \mapsto \xi(y)\}$ to $\mu$
24: **end if**
25: **end for**
26: **end for**
27: **if** $\sigma_r \neq \sigma$ **then**
28: **add** $\sigma_r$ to $\mathcal{R}_\sigma$
29: **end if**
30: **end for**
31: **if** $\mathcal{R}_\sigma \neq \emptyset$ **then**
32: **choose** the best repair $\sigma_r$ of $\sigma$ from $\mathcal{R}_\sigma$ based on $\mathsf{prf}$
33: **replace** $\sigma$ with $\sigma_r$ in $\Sigma'$
34: **end if**
35: **end for**
36: **return** $(\mathbf{S}, \mathbf{T}, \Sigma')$

*computes the pair* $(\mathcal{C}, \nu)$ *such that:*

$$\mathcal{C} = \{R_1(x_1, x_2, x_3); S_1(x_4, x_5, x_6)\}$$
$$\nu = \{R_1(x, y, z) \mapsto R_1(x_1, x_2, x_3); S_1(y, z, z) \mapsto S_1(x_4, x_5, x_6)\}$$

*Then, Algorithm 8 identifies the following three homomorphisms from the set of atoms* $\mathcal{C}$ *into instance* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$*:*

$$\xi_1 = \{x_1 \mapsto *; x_2 \mapsto n_1; x_3 \mapsto n_2; x_4 \mapsto n_1; x_5 \mapsto n_2; x_6 \mapsto n_2\}$$
$$\xi_2 = \{x_1 \mapsto *; x_2 \mapsto n_1; x_3 \mapsto n_2; x_4 \mapsto n_1; x_5 \mapsto n_3; x_6 \mapsto *\}$$
$$\xi_3 = \{x_1 \mapsto *; x_2 \mapsto n_1; x_3 \mapsto n_2; x_4 \mapsto n_1; x_5 \mapsto *; x_6 \mapsto *\}$$

*From homomorphism* $\xi_1$*, we see that the joins in the body of* $\sigma_1$ *are safe. For example, the variables in* $\mathcal{C}$ *corresponding to variable* $y$ *in* $\sigma_1$*, i.e., variables* $x_2$ *and* $x_4$ *through* $\nu$*, are mapped to the same labelled null* $n_1$ *through* $\xi_1$*. Analogously, the variables* $x_3$*,* $x_5$ *and* $x_6$ *correspond to all occurrences of variable* $z$ *in* $\sigma_1$*, and are mapped to the same labelled null* $n_2$ *through* $\xi_1$*. From homomorphism* $\xi_1$*, we also see that, with the previous joins, the variable* $x$ *is safe to expose as its corresponding variable* $x_1$ *in* $\mathcal{C}$ *is mapped to the critical constant* $*$ *through* $\xi_1$*. However,* $y$ *and* $z$ *are unsafe to expose as they are not mapped to the critical constant through the homomorphisms* $\nu$ *and* $\xi_1$*.*

*From homomorphism* $\xi_2$*, we can see that it is safe to reveal the third position of* $S_1$ *as* $\xi_1(x_6) = *$*. However, if we reveal this position then it is unsafe to join the third position of* $R_1$ *(corresponding to variable* $x_3$*), and the second and third positions of* $S_1$ *(corresponding to variables* $x_5$ *and* $x_6$*, respectively) as they are mapped to different values through* $\xi_2$ *(*$n_2$*,* $n_3$ *and* $*$*, respectively).*

*Analogously to* $\xi_2$*, we can see from* $\xi_3$ *that it is safe to reveal the second and third positions of* $S_1$ *as* $\xi_1(x_5) = \xi_1(x_6) = *$*. However, if we reveal these positions then it is unsafe to join them third position (*$x_3$*) of* $R_1$ *as* $\xi_3$ *maps* $x_3$ *to a labelled null instead of the critical constant* $*$*.*

*At line 5, Algorithm 8 iterates over the homomorphisms* $\xi_1$*,* $\xi_2$ *and* $\xi_3$*. At first, Algorithm 8 considers atom* $R_1(x, y, z)$*, for the two first positions of* $R_1(x, y, z)$ *Algorithm 8 generate the mapping:*

$$\mu = \{x \mapsto *; y \mapsto n_1\}$$

*since there is no violation according to lines 12 and 13. However, arrived to the third position of* $R_1(x, y, z)$*, a violation is detected. This is due to the fact that the variable in third position of* $R_1(x, y, z)$ *(the variable* $z = \nu^{-1}(x_3)$*) is an exported variable and that we have* $\xi(x_3) = n_2$*. To tackles this violation, at lines 16–21 Algorithm 8 creates a fresh variable* $z_1$*, adds the relation* $(z, z_1)$ *to* $\rho$*, replaces* $z$ *in* $R_1(x, y, z)$ *by* $z_1$ *and adds the mapping* $\{z_1 \mapsto n_2\}$ *to* $\mu$*.*

*Then, Algorithm 8 considers atom* $S_1(y, z, z)$. *At the first position of* $S_1$ *(corresponding to variable* $x_4$ *in* $\mathcal{C}$*), no violation is detected. Since* $\xi_1(x_4) = n_1$, *the mapping* $y \mapsto n_1$ *already exists into* $\mu$. *However, at the second position of* $S_1$, *a homomorphism violation is detected since* $z$ *is an exported variable and is mapped to a labelled null* $n_2$ *through* $\xi_1$. *Since* $(z, z_1) \in \rho$ *and* $\mu(z_1) = \xi_1(x_5)$, *at line 15 Algorithm 8 replaces* $z$ *in the second position of* $S_1(y, z, z)$ *by* $z_1$. *Similarly, the variable* $z$ *siting at the third podition of* $S_1(y, z, z)$ *is also replaced by* $z_1$.

*Hence, the first repair of* $\sigma$ *is:*

$$R_1(x, y, z_1) \wedge S_1(y, z_1, z_1) \rightarrow T_1(x) \tag{$r_1$}$$

*Algorithm 8 then proceeds by repairing* $\sigma_1$ *based on* $\xi_2$. *At first, Algorithm 8 considers atom* $R_1(x, y, z)$ *and proceeds as described above, leading to the computation of an atom* $R_1(x, y, z_1)$, *where* $z$ *has been renamed into* $z_1$, *and of the morphism:*

$$\mu = \{x \mapsto *, y \mapsto n_1, z_1 \mapsto n_2\}$$

*Then, Algorithm 8 considers the first position in atom* $S_1(y, z, z)$ *and no violation is encountered since* $\mu(y) = \xi_2(x_4)$. *However, for the second position of* $S_1(y, z, z)$ *a violation is encountered since* $z = \nu^{-1}(x_5)$ *is an exported variable and since* $\xi_2(x_5) \neq *$, *violating the condition at line 12 of Algorithm 8. To handle this, since the condition in line 14 is not met, Algorithm 8 creates a fresh variable* $z_2$ *and adds the mapping* $\{z_2 \mapsto n_3\}$ *to* $\mu$.

*Then, at the third position of* $S_1$, *no violation is met since for* $z = \nu^{-1}(x_6)$ *we have* $\xi_2(x_6) = *$. *Hence, the second repair of* $\sigma_1$ *is the tgd:*

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

*Finally, with the same reasoning as above, we see that the repair for* $\sigma_1$ *with respect to* $\xi_3$ *is:*

$$R_1(x, y, z_1) \wedge S_1(y, z, z) \rightarrow T_1(x, z) \tag{$r_3$}$$

We now show that our Algorithm 8 always returns a mapping which is *partially safe* with respect to the mapping containing the reference policy views:

Lemma 3.5.
*For any mapping to rewrite* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, *any mapping* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *and any preference function* prf, *Algorithm* frepair *always returns a mapping* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *that is partially safe with respect to* $\mathcal{M}_\mathcal{V}$ *on all instances of* $\mathbf{S}$.

**Algorithm 9 procedure** InitFrepairTgd($\sigma$)

**Input:** A tgd $\sigma$.
**Output:** A pair $(\mathcal{C}, \nu)$ with $\mathcal{C}$ being a set of atoms and $\nu$ a homomorphism from atom in $\mathsf{body}(\sigma)$ to atom in $\mathcal{C}$.

1: $\nu \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$
2: **for each** $B \in \mathsf{body}(\sigma)$, where $B = R(\vec{x})$ **do**
3: **create** a vector of fresh variables $\vec{y}$
4: **create** the atom $C = R(\vec{y})$
5: **add** $(B, C)$ to $\nu$
6: **add** $C$ to $\mathcal{C}$
7: **end for**
8: **return** $(\mathcal{C}, \nu)$

*Proof.* From Lemma 3.3, a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is partially safe with respect to $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ on all instances of $\mathbf{S}$, if for each $\sigma \in \Sigma$, there exists a homomorphism from $\mathsf{body}(\sigma)$ into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ mapping each variable $x \in \mathsf{exported}(\sigma)$ into the critical constant $*$.

Since for each $\sigma \in \Sigma$ the procedure frepair computes a set of repaired tgds $\mathcal{R}_\sigma$, it follows that Lemma 3.5 holds if such a homomorphism exists for each repaired tgd in $\mathcal{R}_\sigma$.

The proof proceeds as follows. Let $\sigma_r^i$ and $\mu^i$ denote the repaired s-t tgd and the homomorphism $\mu$ computed at the end of each iteration $i$ of the steps in lines 8–26 of Algorithm 8. Let also $B^i$ denote the $i$-th atom in $\mathsf{body}(\sigma_r)$. Since each $C \in \mathcal{C}$ is an atom of distinct fresh variables, since $\xi$ is a homomorphism from $\mathcal{C}$ to $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ and since $\mu(B^i) = \sigma_r|_i$, it follows that in order to prove Lemma 3.3, we have to show that the following claim $C_1$ holds, for each $i \geq 0$:

- $C_1$: *$\mu^i$ is a homomorphism from the first $i$ atoms in the body of $\sigma_r$ into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$ mapping each exported variable occurring in $B^0, \ldots, B^i$ into the critical constant $*$.*

For $i = 0$, $C_1$ trivially holds. For $i + 1$ and assuming that $C_1$ holds for $i$ let $C^{i+1} = \nu(B^{i+1})$, which is calculated at line 11 of Algorithm 8. The proof of claim $C_1$ depends upon the proof of the following claim, for each iteration $p \geq 0$ of the steps in lines 9–25:

- $C_2$: $\mu^{i+1}(B^{i+1}|_p) = \xi(y)$, *where* $y = C^{i+1}|_p$.

The claim $C_2$ trivially holds for $p = 0$, while for $p > 0$, it directly follows from the steps in lines 12–24. Since $C_1$ holds for $i$, since the steps in lines 12–24 do

not modify the variable mappings in $\mu^i$ and due to $C_2$, it follows that $C_1$ holds for $i+1$, concluding the proof of Lemma 3.3. □

In the next section, we will show how to compute a *safe mapping* from a *partially safe mapping*.

### 3.4.2 Computing safe mappings

We have shown in the previous section that given a mapping $\mathcal{M}$, we are able to rewrite this mapping into a mapping $\mathcal{M}'$ that is *partially safe* with respect to a reference mapping $\mathcal{M}_\mathcal{V}$, i.e., the output mapping $\mathcal{M}'$ is such that each tgd in $\mathcal{M}'$ taken *independently* is safe with respect to the reference mapping $\mathcal{M}_\mathcal{V}$. In this section, we will first address the limitation of the notion of partial safety by illustrating the information leakage occurrence when the tgds of a *partially safe* mapping are considered altogether. Then, we describe an algorithm allowing to rewrite a *partially safe* mapping with respect to a reference mapping $\mathcal{M}_\mathcal{V}$ into a *safe* mapping with respect to $\mathcal{M}_\mathcal{V}$.

**Limitations of partial safety.** We recall that *partial safety* of a mapping $\mathcal{M}$ with respect to a mapping $\mathcal{M}_\mathcal{V}$ only ensures that, given $B_1$ the set of bags computed by $\mathsf{visChase}(\mathcal{M})$ at line 3 of Algorithm 6, the instance $I_1 = \bigcup_{\beta \in B_1}$ has a homomorphism into $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$. However, the last step of the visible chase over $\mathcal{M}$ (line 5 of Algorithm 6) can lead to the unification of one or more labelled nulls occurring in the bags of $B_1$ with the critical constant $*$, potentially leading to unsafe instances, as illustrated in the following example:

*Example* 3.8. *We consider a mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ which is a simplified variant of mapping $\mathcal{M}$ from Example 3.1 (page 101) where $\Sigma'$ comprises only the tgds $\sigma_e$ and $\sigma_c$ such that:*

$$\sigma_e = Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \rightarrow EthnicityDisease(ethn, disease) \quad (3.5)$$

$$\sigma_c = Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \rightarrow CountyDisease(county, disease) \quad (3.6)$$

*We also reuse the mapping $\mathcal{M}_\mathcal{V}$ from Example 3.1 as our reference mapping. We have seen in Example 3.6 on page 119 that both $\sigma_e$ and $\sigma_c$ are partially safe. However, the computation of the visible chase over the tgds $\sigma_e$ and $\sigma_c$ leads to the unification of the labelled nulls $n_{name}$ and $n_{county}$ with the critical constant,*

**Algorithm 10** $\mathsf{srepair}(\mathcal{M}, \mathcal{M}_\mathcal{V}, \mathsf{prf}, n)$

**Input:** A mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ which is *partially safe* with respect to $\mathcal{M}_\mathcal{V}$.
**Input:** A reference mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{T}, \Sigma_\mathcal{V})$.
**Input:** A preference function $\mathsf{prf}$.
**Input:** A positive integer $n$.
**Output:** A rewriting $\mathcal{M}'$ of $\mathcal{M}$ such that $\mathcal{M}'$ is *safe* with respect to $\mathcal{M}_\mathcal{V}$.

$\Sigma_0 \leftarrow \Sigma$
$B_0 \leftarrow \mathsf{visChase}(\mathcal{M})$
$i \leftarrow 0$
**do**
    $\Sigma_{i+1} \leftarrow \Sigma_i$
    $cont \leftarrow$ **false**
    **if** $\exists$ unsafe $\beta \in B_i$ s.t. $\forall$ unsafe $\beta' \in B_i$,$\mathsf{depth}(\beta) \leq \mathsf{depth}(\beta')$ **then**
        $cont \leftarrow$ **true**
        **if** $i < n$ **then**
            $r_1 \leftarrow \emptyset$
            $r_2 \leftarrow \mathsf{hideExported}(\beta, \mathcal{M}_\mathcal{V}, \mathsf{prf})$
            **if** $\exists \beta_1, \beta_2 \in \beta^\prec$, s.t. $\beta_1, \beta_2$ are candidates for $\mathsf{modifyBody}$ **then**
                $r_1 \leftarrow \mathsf{modifyBody}(\mathsf{tgd}(\beta_1), \mathsf{tgd}(\beta_2), \mathsf{prf})$
            **end if**
            **if** $r_1 \neq \emptyset$ and it is preferred over $r_2$ with respect to $\mathsf{prf}$ **then**
                **remove** $\mathsf{tgd}(\beta_1)$ from $\Sigma_{i+1}$
                **add** $r_1$ to $\Sigma_{i+1}$
            **else**
                **remove** $\mathsf{tgd}(\beta)$ from $\Sigma_{i+1}$
                **add** $r_2$ to $\Sigma_{i+1}$
            **end if**
        **else**
            **if** $\nexists \beta'$, s.t., $\beta \prec \beta' \in B_i$ **then**
                **add** $\mathsf{hideExported}(\beta, \mathcal{M}_\mathcal{V}, \mathsf{prf})$ to $\Sigma_{i+1}$
            **else remove** $\mathsf{tgd}(\beta)$ from $\Sigma_{i+1}$
            **end if**
        **end if**
    **end if**
    **compute** $J_{i+1}$ from $\Sigma_i$, $\Sigma_{i+1}$ and $B_i$
    $i \leftarrow i + 1$
**while** $cont$ **and** $i \leq n$
**return** $(\mathbf{S}, \mathbf{T}, \Sigma_n)$

*as illustrated in Example 3.4 on page 110. This can be seen in Example 3.4 with the computation of bags $\beta_4$ and $\beta_5$ using the egds $\epsilon_1$ and $\epsilon_2$, respectively.*

*Finally we obtain, as output of the visible chase, a set of bags $B$ leading to an instance* $\mathsf{univSourceInst}(\mathcal{M})$ *such that:*

$$\mathsf{univSourceInst}(\mathcal{M}) = \{Patient(n_{idIns}, n_{name}, *, *); \quad NorthHospital(n_{idIns}, *); \\ Patient(n'_{idIns}, n'_{name}, *, *); \quad NorthHospital(n'_{idIns}, *)\}$$

*As this instance* $\mathsf{univSourceInst}(\mathcal{M})$ *does not possess a homomorphism into* $\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V})$*, then $\mathcal{M}'$ in unsafe with respect to $\mathcal{M}_\mathcal{V}$, despite the fact that $\mathcal{M}'$ is partially safe with respect to $\mathcal{M}_\mathcal{V}$.*

Thus, in order to avoid information leakage, we need to rewrite our *partially safe* mappings into a mapping such that no unsafe unification of a labelled null with $*$ takes place during the last step of the visible chase. In the next section, we will describe such a rewriting approach.

**Computing safe mappings.** We first illustrate two approaches to compute a *safe* mapping out of a *partially safe* mapping.

The prevention of the unwanted exportation of values is illustrated in the following example:

*Example* 3.9. *We consider again the simplified variant $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ of $\mathcal{M}$ from Example 3.8 (page 127).*

*Since $\Sigma'$ 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 tuples associated with the bodies of one or more s-t tgds from $\Sigma'$. 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 $n'_{ethn}$ with $*$. This unification takes place due to $\epsilon_1$, which in turn has been created due to the tuple that ethn is an exported variable in $\sigma_e$.*

*By hiding the exported variable ethn from $\sigma_e$, we actually prevent the creation of $\epsilon_1$ and hence, we block the unsafe unification of ethn 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 as illustrated in the following example:

*Example* 3.10. *Consider a set of policy views $\mathcal{V}$ leading to the following instance:*

$$\mathsf{univSourceInst}(\mathcal{M}_\mathcal{V}) = \{R_1(n_1, n_1, *), R_1(*, *, n_2), S_1(*)\}$$

*where $n_1$ and $n_2$ are labelled nulls. Consider also the mapping $\mathcal{M}$ consisting of the following s-t tgds:*

$$\sigma_2 : R_1(x, x, y) \wedge S_1(y) \rightarrow T_1(y)$$
$$\sigma_3 : R_1(x, x, y) \rightarrow T_2(x)$$

*It is easy to see that $\mathcal{M}$ is partially safe with respect to $\mathcal{M}_\mathcal{V}$, but unsafe in general. Indeed,* $\mathsf{univSourceInst}(\mathcal{M})$ *will consist of the following bags (for presentation purposes, we adopt the notation from Example 3.4):*

$$T_1(*) \xrightarrow{\langle \sigma_2^{-1}, \theta_1 \rangle} R_1(n_3, n_3, *), S_1(*)$$
$$T_2(*) \xrightarrow{\langle \sigma_3^{-1}, \theta_2 \rangle} R_1(*, *, n_4)$$
$$R_1(n_3, n_3, *), S_1(*) \xrightarrow{\langle \epsilon_3, \theta_3 \rangle} R_1(*, *, *), S_1(*)$$

*where $\epsilon_3$ is the egd $R_1(x, x, y) \rightarrow x = *$ and:*

$$\theta_1 = \{y \mapsto *\} \qquad \theta_2 = \{x \mapsto *\} \qquad \theta_3 = \{x \mapsto n_3, y \mapsto *\}$$

*Note that $\epsilon_3$ has been created out of $\sigma_3$, since there exists a homomorphism from* $\mathsf{body}(\sigma_3)$ *into $R_1(n_3, n_3, *)$ mapping the exported variable $x$ into $n_3$.*

*For preventing the unsafe unification of $n_3$ with $*$, the approach presented in the previous example would have led to hide the exported variable $x$ from $\sigma_3$. By doing this, the creation of $\epsilon_3$ is blocked, and hence the unsafe unification did not occur.*

*In the approach presented in this example, $x$ is kept as an exported variable in $\sigma_3$, but the body of $\sigma_2$ is modified by breaking the join between the first and the second positions of $R_1$:*

$$R_1(x, z, y) \wedge S_1(y) \rightarrow T_1(y) \qquad (\sigma_2')$$

*By doing this, we prevent the creation of $\epsilon_3$, since the instance computed at line 3 of Algorithm 6 on page 107 would consist of the tuples $R_1(n_3, n_5, *)$, $R_1(*, *, n_4)$ and $S_1(*)$, hence, there would be no homomorphism from* $\mathsf{body}(\sigma_3)$ *into this instance. Note that the modification from $\sigma_2$ to $\sigma_2'$ is safe. Intuitively, this holds, since we break joins, and thus, we export less information.*

Before presenting Algorithm 10 and its use of our previous rewriting approaches, we introduce some new definitions. The depth of a bag is defined as follows:

Definition 3.21 (Depth of a bag).
*The* depth *of a bag $\beta$ is the highest derivation depth of the tuples in $\beta$.*

We denote the depth of a bag $\beta$ with the notation $\mathsf{depth}(\beta)$.
We also define the support of a bag as follows:

Definition 3.22 (Support of a bag).
*The* support $\beta^{\prec}$ *of a bag* $\beta$ *is inductively defined as follows:*

- *if* $\mathsf{depth}(\beta) = 1$, *then* $\beta^{\prec} = \beta$
- *else* $\mathsf{depth}(\beta) > 1$, *and then* $\beta^{\prec} = \bigcup_{\beta' \prec \beta} \beta'^{\prec}$

We define the notion of *candidates bags for algorithm* modifyBody as follows:

Definition 3.23 (Candidates for modifyBody).
*Two bags* $\beta_1$ *and* $\beta_2$ *are candidates for* modifyBody *if:*

- $\beta_1$ *is the predecessor (Definition 3.17) of* $\beta_2$
- $\mathsf{depth}(\beta_1) = 1$ *and* $\mathsf{depth}(\beta_2) = 2$
- *there exists at least one repeated variable in the body of* $\mathsf{tgd}(\beta_1)$

Considering an active trigger $h$ for a tgd $\sigma$ in an instance $I$ leading to the creation of a bag $\beta$, we also use the following notations: $\mathsf{dependency}(\beta) = \delta$, $\mathsf{trigger}(\beta) = h$ and $\mathsf{premise}(\beta) = h(\mathsf{body}(\delta))$.

Algorithm 10 presents an iterative process for repairing a partially safe mapping, by employing the three ideas we described above:

- checking for homomorphism violations within each bag;
- preventing unsafe unifications by hiding exported variables of an s-t tgd;
- preventing unsafe unifications by modifying the bodies of s-t tgds.

During the execution of Algorithm 10, given a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ to repair, the algorithm starts by initializing the set of tgds $\Sigma_0$ to $\Sigma$ at line 1. Then, at each iteration $i$ of the main while loop (lines 4–31 of Algorithm 10), an unsafe bag $\beta$ with the lowest depth is identified at line 7 of Algorithm 10. When $\beta$ has been identified, the algorithm repair the tgd from $\Sigma_i$ that has lead to its creation (lines 7–28 of Algorithm 10). If the number of iterations did not exceeded the maximum depth of the rewriting tree $n$, i.e., if $i < n$, then lines 9–21 of Algorithm 10 will use the two repairing methods we have exposed below: the function hideExported (called line 11 of Algorithm 10) based on hiding exported variables and detailed in Algorithm 11 on page 133; and the second function modifyBody (called line 13 of Algorithm 10) based on

eliminating joins between variable occurrences and detailed in Algorithm 12 on page 134.

It should be noted that Algorithm 10 can try to apply function modifyBody only if there exist at least two bags in the support of $\beta$ that are candidates for modifyBody.

Informally, Algorithm 10 tries to apply Algorithm 12 with the deepest possible bags and when there are one or more repeated variables in the body of $\mathsf{tgd}(\beta_1)$ (Example 3.10 on page 129).

Finally, if the maximum allowed depth of the rewriting tree is reached, i.e., if $i = n$, then Algorithm 10 either applies the function hideExported (line 24), or it eliminates the s-t tgds that are responsible for unsafe unifications (line 25).

*Example* 3.11. *To demonstrate Algorithm 10 we reuse the simplified version of the running example from Example 3.8 on page 127 in which the set of tgds $\Sigma'$ from mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ comprises only the tgds $\sigma_e$ and $\sigma_c$ such that:*

$$\begin{aligned}\sigma_e = &Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease)\\ &\rightarrow EthnicityDisease(ethn, disease) \qquad (3.5)\\ \sigma_c = &Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease)\\ &\rightarrow CountyDisease(county, disease) \qquad (3.6)\end{aligned}$$

*From Example 3.4 on page 110 it is seen that:*

$$\mathsf{visChase}(\mathcal{M}') = \{\beta_2, \beta_3, \beta_4, \beta_5\}$$

*where $\beta_2, \ldots, \beta_5$ are the bags (illustrated in Example 3.4) such that:*

$$\begin{aligned}\beta_1 &= \{Student(n''_{idIns}, n''_{name}, *, n''_{county}); Oncology(n''_{idIns}, n''_{treat}, n''_{progr})\}\\ \beta_2 &= \{Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *); NorthHospital(n'_{idIns}, *)\}\\ \beta_3 &= \{Patient(n_{idIns}, n_{name}, *, n_{county}), NorthHospital(n_{idIns}, *)\}\\ \beta_4 &= \{Patient(n'_{idIns}, n'_{name}, *, *), NorthHospital(n'_{idIns}, *)\}\\ \beta_5 &= \{Patient(n_{idIns}, n_{name}, *, *), NorthHospital(n_{idIns}, *)\}\end{aligned}$$

*We also assume a maximum rewriting tree depth $n$ such that $n = \infty$.*

*During the first iteration of Algorithm 10, the lowest depth bag for which there exists a homomorphism violation is $\beta_4$. Since $i < n$, the algorithm tries to repair $\Sigma'$ by calling functions* $\mathsf{hideExported}(\beta_4, \mathcal{M}_\mathcal{V}, \mathsf{prf})$ *and* $\mathsf{modifyBody}(\beta_2, \beta_4, \mathcal{M}_\mathcal{V}, \mathsf{prf})$.

*At lines 3–5, Algorithm 11 (procedure* hideExported*) first computes the homomorphism:*

$$\nu = \{n'_{idIns} \mapsto x_1, n'_{name} \mapsto x_2, n'_{ethn} \mapsto x_3\}$$

**Algorithm 11** hideExported($\beta, \mathcal{M}_\mathcal{V}$, prf)

**Input:** A bag $\beta$.
**Input:** A reference mapping $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{T}, \Sigma_\mathcal{V})$.
**Input:** A preference function prf.
**Output:** A tgd $\sigma_r$ in which sensible frontier variables are hidden and selected among the possible rewritings using prf.

```
1: J ← premise(β)
2: ν ← ∅
3: for each n ∈ Nulls occurring into J do
4:     add {n ↦ x} to ν, where x is a fresh variable
5: end for
6: R ← ∅
7: for each ξ : ν(J) → univSourceInst(M_V) do
8:     σ ← tgd(β)
9:     for each x ∈ dom ξ do
10:        if ξ(x) ≠ * then
11:            for each y ∈ exported(σ) do
12:                if τ(y) = ν^{-1}(x), where τ = trigger(β) then
13:                    remove y from exported(σ)
14:                end if
15:            end for
16:        end if
17:    end for
18:    if σ ≠ tgd(β) then
19:        add σ to R
20:    end if
21: end for
22: choose the best repair σ_r of σ from R based on prf
23: return σ_r
```

*and then computes all homomorphisms from the instance:*

$$\nu(J) = \{Patient(x_1, x_2, x_3, *), NorthHospital(x_1, *)\}$$

*into the instance containing the tuples in* univSourceInst($\mathcal{M}_\mathcal{V}$).

*We can see that there exists only one such homomorphism:*

$$\xi = \{x_1 \mapsto n'_{idIns}, x_2 \mapsto n'_{name}, x_3 \mapsto n'_{ethn}\}$$

*Recalling that* $\mathsf{tgd}(\beta_4) = \sigma_e$ *and that:*

$$h_4 = \{idIns \mapsto n'_{idIns}, name \mapsto n'_{name}, ethn \mapsto n'_{ethn}, county \mapsto *, disease \mapsto *\}$$

**Algorithm 12** modifyBody($\sigma_1, \sigma_2$, prf)

**Input:** A tgd $\sigma_1$.
**Input:** A tgd $\sigma_2$ with at least one homomorphism from its left-hand side into the left-hand side of $\sigma_1$.
**Input:** A preference function prf.
**Output:** A rewriting $\sigma_r$ of $\sigma_1$ in which some joins have been removed and selected among the possible rewritings using prf.

1: $\mathcal{R} \leftarrow \emptyset$
2: **if** $\exists$ one or more repeated variables in body($\sigma_1$) **then**
3: **for each** $\xi$ : body($\sigma_2$) $\rightarrow$ body($\sigma_1$) mapping some $x_1 \in$ exported($\sigma_1$) into some $x_2 \notin$ exported($\sigma_2$) **do**
4: Let $B \subseteq$ body($\sigma_1$), s.t. $\xi$(body($\sigma_2$)) $= B$
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: $\sigma \leftarrow \sigma_1$
9: **replace** the variables in positions $S$ of $\sigma$ by fresh variables
10: **add** $\sigma$ to $\mathcal{R}$
11: **end for**
12: **end for**
13: **end if**
14: **choose** the best repair $\sigma_r$ of $\sigma$ from $\mathcal{R}$ based on prf
15: **return** $\sigma_r$

*the first two iterations of the loop in lines 9–17 of Algorithm 11 have no effect. Indeed, despite that* $\xi(x_1) = n'_{idIns}$ *and* $\xi(x_2) = n'_{name}$*, the variables idIns and name from* $\sigma_e$ *that are mapped to* $n'_{idIns}$ *and* $n'_{name}$ *via the homomorphism* trigger($\beta_4$) $= h_4$ *are not exported variables.*

*During the last iteration, since* $\xi(x_3) = n'_{ethn}$*, since* $h_4(ethn) = n'_{ethn}$ *and since ethn is an exported variable, Algorithm 11 removes variable ethn from the exported variables of* $\sigma_e$ *and returns a tgd* $\sigma'_e$*:*

$$Patient(idIns, name, ethn, county) \wedge NorthHospital(idIns, disease) \rightarrow EthnicityDisease(disease) \qquad (\sigma'_e)$$

*Algorithm 10 then calls Algorithm 12 (function* modifyBody*). The function does not return any repair, since there does not exist any variable repetition in the body of* $\sigma_e$*. Hence, Algorithm 10 computes the set of tgds* $\Sigma_1 = \{\sigma'_e, \sigma_c\}$

*and proceeds in the next iteration. The computed instance* $\mathsf{univSourceInst}(\mathcal{M}_1)$ *for the current repaired mapping will consist of the following bags* $\beta'_2$ *and* $\beta'_3$*:*

$$\beta'_2 = \{Patient(n'_{idIns}, n'_{name}, n'_{ethn}, *), NorthHospital(n'_{idIns}, *)\}$$
$$\beta'_3 = \{Patient(n_{idIns}, n_{name}, n'_{ethn}, n_{county}), NorthHospital(n_{idIns}, *)\}$$

*such that:*

$$CountyDisease(county, disease) \xrightarrow{\langle \sigma_c^{-1}, h'_2 \rangle} \beta'_2$$
$$with\ h'_2 = \{county \mapsto *, disease \mapsto *\}$$

$$EthnicityDisease(disease) \xrightarrow{\langle \sigma_e^{-1}, h'_3 \rangle} \beta'_3$$
$$with\ h'_3 = \{disease \mapsto *\}$$

*Finally, Algorithm 10 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 labelled nulls to the critical constant $*$ that were not present before the modification of the s-t tgd through modifyBody. In contrast, the modification performed by hideExported is a safe modification, since it does not introduce new unsafe unifications. In particular, if hideExported modifies an s-t tgd $\sigma$ that leads to the unsafe bag $\beta$, and if in $\beta$ the labelled nulls $n_1, \ldots, n_n$ were unified with $*$, then the modified s-t tgd $\sigma_r$ would lead to the derivation of a new bag where no such unification takes place. Furthermore, by construction of Algorithm 10 it is seen that at each depth $i$, the number of bags that are derived for this depth after the repair is lower or equal to the number of bags of this depth before the repair, and a lower or equal number of labelled null unifications takes place in these bags. Using the above, we show the correctness of Algorithm 10:

Lemma 3.6.
*Let* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *be a mapping with a set of policy views* $\mathcal{V}$*.*
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a partially safe mapping with respect to* $\mathcal{M}_\mathcal{V}$*.*
*Let* prf *be a preference function.*
*Let* $n$ *be an integer greater than* 0*.*
*Then* $\mathsf{srepair}(\mathcal{M}, \mathcal{M}_\mathcal{V}, \mathsf{prf}, n)$ *returns a mapping* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *that* preserves the privacy of $\mathcal{M}_\mathcal{V}$ on all instances of $\mathbf{S}$.

*Proof.* First note that since srepair takes as input a partially safe mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, it follows from Definition 3.20 (page 119) that there exists a

homomorphism from $\textsc{chase}(\Sigma^{-1}, \mathsf{Crt}_{\mathbf{T}}) \setminus \mathsf{Crt}_{\mathbf{T}}$ into $\mathsf{univSourceInst}(\mathcal{M}_{\mathcal{V}})$. Furthermore, from Lemma 3.3 (page 119), we know that for each $\sigma \in \Sigma$, there exists a homomorphism from $\mathsf{body}(\sigma)$ into $\mathsf{univSourceInst}(\mathcal{M}_{\mathcal{V}})$ mapping each $x \in \mathsf{exported}(\sigma)$ into the critical constant $*$.

Due to the above, since the steps in lines 21–25 of Algorithm 6 (page 107) do not introduce new labelled 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 $\beta$ is the predecessor of $\beta'$ (Definition 3.17), it follows that $\mathcal{M}'$ preserves the privacy of $\mathcal{M}_{\mathcal{V}}$ on all instances of $\mathbf{S}$, if hideExported prevents dangerous unifications of labelled nulls with the critical constant in line 5 of Algorithm 6.

In particular, assume that we are in the $n$-th iteration of the steps in lines 4–31 of Algorithm 10 (page 128). Let $\beta_n^0, \ldots, \beta_n^M$ be the unsafe bags in $B_n$. Assume also that for each $1 \leq l \leq M$, $\beta_n^l$, 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 22 of Algorithm 6. Let $\sigma^l = \mathsf{tgd}(\varepsilon^l)$, for each $0 \leq l \leq M$ and let $\sigma_r^l$ 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 29 of Algorithm 10.

Based on the above, in order to show that Lemma 3.6 holds, we need to show that:

(i) the number of bags in $B_{n+1}$ is lower or equal to the number of bags in $B_n$;

(ii) the s-t tgds in the set $\left(\Sigma \setminus \bigcup_{l=0}^{M} \sigma^l\right) \cup \bigcup_{l=0}^{M} \sigma_r^l$ are safe.

In order to show (i) and (ii), we consider the steps in Algorithm 11 (page 133): for each $1 \leq l \leq M$, each exported variable $y$ occurring in $\sigma^l$, which leads to an unsafe unification, line 12 of Algorithm 11, is turned into a non-exported variable. □

The correctness of procedure repair (Algorithm 7 on page 119) derives from the combination of Lemma 3.5 (page 125) and Lemma 3.6. This is stated in the following theorem:

Theorem 3.3.
*Let* $\mathcal{M}_{\mathcal{V}} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *be a mapping with a set of policy views* $\mathcal{V}$.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a mapping to rewrite.*
*Let* prf *be a preference function.*
*Let* $n$ *be an integer greater than* 0.
*Then* $\mathsf{repair}(\mathcal{M}, \mathcal{M}_{\mathcal{V}}, \mathsf{prf}, n)$ *returns a mapping* $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ *that* preserves the privacy of $\mathcal{M}_{\mathcal{V}}$ on all instances of $\mathbf{S}$.

*Proof.* This theorem directly follows from the combination of Lemma 3.5 and Lemma 3.6. □

Furthermore, if the preference function always prefers the repairs computed by hideExported from the repairs computed by modifyBody, we can show the following:

Lemma 3.7.
*Let* $\mathcal{M}_\mathcal{V} = (\mathbf{S}, \mathbf{V}, \mathcal{V})$ *be a mapping with a set of policy views* $\mathcal{V}$.
*Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a partially safe mapping with respect to* $\mathcal{M}_\mathcal{V}$.
*Let* prf *be a preference function that always prefers the repairs computed by* hideExported *rather than the repairs computed by* modifyBody.
*Then Algorithm 7 returns a non-empty mapping that is safe with respect to* $\mathcal{M}_\mathcal{V}$*, if such a mapping exists.*

*Proof.* From Algorithm 8, 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 tuple 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 Lemma 3.7 holds. □

## 3.5 Learning user preferences

In the above presentation of the repairing algorithm, we have assumed that the preference function prf is fixed. Such an assumption is reasonable if the preference mechanism is hardcoded, i.e., if we know that the s-t tgd with highest number of exported variables is chosen at each iteration. In this section, we explain how this function can be learned by relying on off-the-shelf machine learning algorithms and on training sets in which former user choices have been recorded.

In order to be able to rank the possible repairs and employ inference on previously specified user's choices, we first need to define suitable metrics for comparing each pair of repairs $\mu_{r_1}$ and $\mu_{r_2}$ for a given s-t tgd $\mu$. In case the number of possible repairs for $\mu$ is greater than two, the metrics will compare the repairs pair by pair. For instance, Example 3.7 on page 122 presents three different repairs for the s-t tgd $\mu_1$. Since our algorithm modifies the exported

and the repeated variables in the bodies of the s-t tgds, we ground our metrics on the following two parameters:

- $\Delta_{FV}$ amounting to the difference between the number of exported variables in $\mu_{r_1}$ and $\mu_{r_2}$;
- $\Delta_J$ corresponding to the difference between the number of joins in the bodies of $\mu_{r_1}$ and $\mu_{r_2}$.

The key intuition behind applying statistical learning to rank possible safe rewritings of s-t tgds is to leverage an available training set of correctly identified observations. Each observation consists of the two metrics $\Delta_{FV}$ and $\Delta_J$ defined above along with the s-t tgd that better fits the values $\Delta_{FV}$ and $\Delta_J$. More formally, we define a vector of variables $\mathbf{X} = \langle \Delta_{FV}, \Delta_J \rangle$ and we use it as input to train the learning model. We define a qualitative output $G$ embodying the s-t tgds $\mu_{r_1}$ and $\mu_{r_2}$ chosen by the preference function $\mathsf{prf}$. We next denote by $\langle \delta_{FV}, \delta_J \rangle$ the observed values of the vector $\mathbf{X}$ and by $g$ the actual value of the qualitative output $G$. Let $\hat{G}$ be the prediction associated to $G$ obtained by the learning model. The goal of the learning model is to obtain, for each observation $\langle \delta_{FV}, \delta_J \rangle$ of $\mathbf{X}$, a predicted value $\hat{g}$ of $\hat{G}$ that fits the ground-truth value $g$ corresponding to $\langle \delta_{FV}, \delta_J \rangle$.

We employ the k-NN classification algorithm (Cover *et al.* [CH06], Friedman *et al.* [FHT01]) as the supervised learning method for learning the preference function $\mathsf{prf}$. This algorithm has been chosen due to its flexibility in comparing a new item to classify with the existing items in the training set, which are the k-nearest to the former in terms of their similarity, for which few structural assumptions are made.

In our setting, a training set consists of measurements:

$$\{(\langle \delta_{FV,i}, \delta_{J,i} \rangle, g_i) | i = 1, \ldots, N\}$$

such that $\langle \delta_{FV,i}, \delta_{J,i} \rangle$ are values of $\mathbf{X}$ on the $i$-th measurement, and $g_i$ is the value of $G$ on the $i$-th measurement.

Intuitively, measurements have been built by choosing the repair that corresponds to the s-t tgd that best fits the ground truth.

*Example* 3.12. *Consider, for example, the s-t tgd $\mu_1$ from Example 3.7 and its three possible repairs $r_1$–$r_3$. In order to compare the possibles pairs of repairs, suppose that the preference goes to the repair with the maximum number of exported variables and, in case of equality (i.e., $\delta_{FV} = 0$), the preference goes to the repair with the maximum number of joins. The above leads to a training set with three measurements $\langle 1, -1, r_2 \rangle$, $\langle 1, 0, r_3 \rangle$ and $\langle 0, 1, r_3 \rangle$ such that:*

- *the first observation, which corresponds to the comparison between* $r_1$ *and* $r_2$ *leads to computation of values 1 and -1, choosing, thus,* $r_2$*;*
- *the second observation, which corresponds to the comparison between* $r_1$ *and* $r_2$*, leads to computation of values 1 and 0, choosing, thus,* $r_3$*;*
- *the third observation, which corresponds to the comparison between* $r_2$ *and* $r_3$*, leads to computation of values 0 and 1, choosing, thus,* $r_3$*.*

Intuitively, as shown in the previous example, the measurements are built by choosing the repair that corresponds to the s-t tgds that are the closest to the golden standard one. Using this training set and given an input $\langle \delta_{FV}, \delta_J \rangle$, a predicted value $\hat{g}$ is computed with the use of the k-NN method. More precisely, the k-NN method finds the k-nearest measurements to $\langle \delta_{FV,j}, \delta_{J,j} \rangle$ among the measurements of the training set. The adopted similarity function is the Euclidean distance. The principles exposed in this section can be adapted to consider other similarity functions as well as other learning methods to learn the preference function prf, however such an extension falls beyond the scope of this work.

## 3.6 Related work

In this section, we examine the literature related to our approach.

At first, we examine papers relating to privacy preservation in data integration and data publishing. Then we discuss papers on controlled query evaluation. Finally, we situate our work within the setting of data privacy.

**Privacy in data integration** The work of Nash *et al.* [ND07] has addressed the problem of checking the safety of secret queries over a global schema from a theoretical standpoint. At first they define the optimal set of queries of an attack, corresponding to a set of queries for which the introduction of any new query does not lead to infer more information. Then, Nash *et al.* [ND07] define the privacy guarantees against the optimal attack by considering the static and the dynamic case. The dynamic case corresponds to modifications of the schemas or of the GLAV tgds of the considered mapping.

The work of Benedikt *et al.* [BGK17] adopts the same definition of secret queries as in Nash *et al.* [ND07]. However, their work focuses on the notion of safety with respect to a given mapping, and uses boolean conjunctive queries as policy views. Their setting takes place in an ontology-based integration scenario in which the target instance is produced via a set of tgds starting from an underlying data source. Whereas they study the complexity of the view compliance problem in both data-dependent and data-independent setting, in

our work we focus on the latter and extend it to non-boolean conjunctive queries as policy views. Compared to the work of Benedikt *et al.* [BGK17], we further consider multiple policy views altogether in the design of practical algorithms 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** In data publishing, a view allows to export the information of an underlying data source. Thus, privacy concerns about data disclosure linger over the problem of avoiding the disclosure of the content of the view under a confidential query. To handle privacy in such a setting, Miklau *et al.* [MS07] propose a theoretical study of the query-view security model built both on logic and probability theory, where they offer a complete treatment of the multi-party collusion and the use of external adversarial knowledge. The work of Miklau *et al.* [MS07] also explores the use of access control policies using cryptography to enforce the authorization to an XML document. Our work differs from the work of Miklau *et al.* [MS07] 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 in which a policy declaratively specifies the sensitive information and where the confidentiality is enforced by a censor. This framework has been introduced in Sicherman *et al.* [SDJVdR83], and then refined in the works from Bonatti *et al.* [BKS95], Biskup *et al.* [BB04] and Biskup *et al.* [BW08].

In the setting of the Controlled Query Evaluation, the censor takes as input the queries executed over the database. When an input query is executed, the censor proceeds by checking if this query leads to a violation of the security policy it ensures and, if this policy is compromised by the query, it returns a distorted answer which does not violate the security policy. This approach has been adapted in the work of Grau *et al.* [GKKZ15] to handle ontologies expressed in the Datalog language or in the lightweight Description Logic language `OWL2`.

In these approaches, the policy views are not supposed to be known by any users except the database administrators, and the queried data has a 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 (Sarfraz *et al.* [SNCB15]), 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 (Dwork *et al.* [DR14]) and k-anonymity (Sweeney[Swe02]).

## 3.7 Conclusion

In this chapter, we have provided formal definitions of the safety of a GAV mapping under privacy restrictions taking the form of policy views. Then we have provided a process to easily assess if such a mapping is safe with respect to these policy views. Finally, we have provided a two-step framework to rewrite an unsafe mapping into a safe one with respect to the policy views used as references, as well as a simple approach to learn the preference function according to user's previous choices.

Along with the description of our algorithms, we have provided proofs that the mappings returned by our framework are always safe with respect to the reference policy views, and that our framework always returns a non empty mapping if such a mapping exists.

# Chapter 4

# Experimental assessment

In this section, we present our experimentations on the framework described in this thesis. As the repairing part of our framework is limited to the class of *GAV mapping*, contrarily to the mapping specification part, we split our experimentations between these two blocks of our framework.

**Chapter organization** In Section 4.1, we focus on the mapping specification. At first, in Section 4.1.1, we detail our experimental setting. In Section 4.1.2, we study the number of asked questions during the specification of a mapping, with an emphasis on the influence of the use of quasi-lattices structures during this process. In Section 4.1.3, we study the benefit of using non-universal exemplar tuples. Finally, in Section 4.1.4, we study the benefit of using an interactive process compared to the approach used in the `EIRENE` system.

Then, in Section 4.2, we focus on the repairing process. In Section 4.2.1, we detail our experimental setting. In Section 4.2.2, we provide experiments over the running time of our repairing algorithms. In Section 4.2.3, we compare the time breakdown between the two steps of our repairing process. Finally, in Section 4.2.4, we evaluate the efficiency of our learning approach of the preference function.

## 4.1 Efficiency of the interactive specification process

In this section, we investigate the efficiency of the steps of our specification framework in terms of the number of interactions needed with users. We also provide a comparison with the `EIRENE` system (Alexe *et al.* [AtCKT11b]).

| Operator | Number of s-t tgds generated by the operator | Max. left-hand side atoms in the generated tgds | Number of operators used for scenarios with: 15 tgds | 30 tgds | 45 tgds | 60 tgds | 75 tgds | 90 tgds |
|---|---|---|---|---|---|---|---|---|
| copy | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| vertical partitioning | 1 | 1 | 2 | 4 | 6 | 8 | 10 | 12 |
| merging | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| fusion | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| self-joins | 2 | 2 | 2 | 4 | 6 | 8 | 10 | 12 |
| add attribute | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| del attribute | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| add+del attribute | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| merge and add attribute | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |

Table 4.1: `iBench` operators used for the generated scenarios.

The source code of the prototype is publicly available at `https://github.com/ucomignani/MapSpec`.

### 4.1.1 Experimental setting

We have implemented the specification part of our framework using OCaml 4.03, and tested it on a 2.6GHz 4-core, 16Gb laptop running Debian 9. We have borrowed mappings from seven real integration scenarios of the `iBench` benchmark (Arocena *et al.* [AGCM15]), as well as generated scenarios using the same benchmark. The generated scenarios range from 15 to 90 tgds by steps of 15 tgds using the configurations of operators listed in Table 4.1. In this table, the first column represent the `iBench` operators used, the second column represent the number of tgds generated by the use of these operators, the third column represent maximum size of the left-hand sides generated by these operators, and the last column represent the number of times each of these operators are applied in order to generate a particular scenario. For each of these configurations, we generate ten mapping scenarios with `iBench` in order to run our experiments.

The left part of Table 4.2 reports the *size* of each considered mapping scenario as the total number of tgds ($|\Sigma|$), as well as the number of relations in the source schema and the target schema of the considered mapping scenario.

**Methodology.** In all experiments, we consider the `iBench` mapping scenarios (both fixed and generated) as the ideal mappings that the user has in mind. Starting from these mapping scenarios, we construct exemplar tuples as follows. Each tgd $\sigma \in \Sigma$ of the form $\phi \rightarrow \psi$ is transformed into a pair of instances $(I^\sigma, J^\sigma)$, with instance $I^\sigma$ ($J^\sigma$, resp.) being generated by replacing each atom in conjunction $\phi$ ($\psi$, resp.) by its tuple counterpart with freshly picked constants for each variable in the tgd. Thus, for each scenario $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$,

we obtain a set of exemplar tuples $E_\Sigma = \{(I^{\sigma_1}, J^{\sigma_1}), \ldots, (I^{\sigma_n}, J^{\sigma_n})\}$.

These exemplar tuples are used as a baseline in our experimental study, as we expect that an "ideal" user, who does not make any mistakes, would actually produce such examples. In order to introduce user ambiguities in the above tuples, we have built alternative test cases, in which the exemplar tuples $E_\Sigma$ are degraded. The degradation procedure is meant to reproduce users' common mistakes while specifying exemplar tuples.

It should be noted that, in order to provide a fair evaluation of the quasi-lattices efficiency, we explicitly avoided to artificially introduce overlapping atoms to favor the use of quasi-lattices over the setting with separate semi-lattices.

The atom degradation procedure is parametrized by the total number of extraneous tuples added to $E_\Sigma$. In this first degradation procedure, an extraneous tuple is generated by randomly choosing a source instance $I^{\sigma_i}$, picking a tuple at random within it, copying it and then replacing one constant of the tuple with a fresh one.

The second degradation affects join paths. The join degradation procedure is parametrized by the total number of unifications between constants in the set $E_\Sigma$, with the constraint that at most one unification is applied within each individual example $(I^{\sigma_i}, J^{\sigma_i})$. An extraneous unification is produced by choosing at random two constants that appear in $I^{\sigma_i}$, and replacing one with the other in all its occurrences in $I^{\sigma_i}$ and in $J^{\sigma_i}$.

*Example* 4.1. *By applying the degradation procedure on the tgd $\sigma$ from Example 2.17 on page 80, the following exemplar tuples may be yielded $(I'^{\sigma}, J'^{\sigma})$. An extraneous Flight atom is added (atom degradation) and constants* `Miami` *and* `L.A.`$''$ *are unified (join degradation), the degradation being underlined:*

$$\begin{aligned} I'^{\sigma} = \{&\mathit{Flight}(\texttt{flight0}, \underline{\texttt{Miami}}, \texttt{L.A.}', \texttt{airline0});\\ &\underline{\mathit{Flight}(\texttt{flight1}, \texttt{Miami}, \texttt{L.A.}', \texttt{airline0})};\\ &\mathit{Airl}(\texttt{airline0}, \texttt{AAirline}, \underline{\texttt{Miami}})\} \end{aligned}$$

$$\begin{aligned} J'^{\sigma} = \{&\mathit{Dpt}(\texttt{L.A.}', \texttt{flight2}, \texttt{comp0});\\ &\mathit{Arr}(\texttt{L.A.}', \texttt{flight2}, \texttt{comp0});\\ &\mathit{Co}(\texttt{comp0}, \texttt{AAirline}, \underline{\texttt{Miami}})\} \end{aligned}$$

In our experimental study, we have deteriorated each initial set of examples $E_\Sigma$ by adding 0, 2, 5, 8, 10, 20 or 30 tuples or by joining 0, 2, 5, 8, 10, 20 or 30 variables. For each of the above configurations, we repeated the degradation procedure 30 times in order to obtain an equivalent number of degraded test cases.

Moreover, we simulate the user's answers during the interactive part of our approach with the following assumption: the user always replies correctly to a given challenge (i.e., an input pair $(E_S, E_T)$) w.r.t. the original mapping $\Sigma$ from the scenario. In order to simulate the user's answer, $I$ is chased to obtain $J'$. 'Yes' is returned as an answer if there exists a substitution $\mu$ from $J$ into $J'$ such that $\mu(J) \subseteq J'$, otherwise 'No' is returned.

### 4.1.2 Number of questions asked during the process and benefit of using quasi-lattices

**Influence of number of atom degradations over the number of asked questions.** In the first experiment, we gauge the effectiveness of using quasi-lattice structures compared to the case where the *ψ-equivalent* tgds are not grouped together. In these experiments, we use a Breadth-First exploration strategy, both in Top-Down and Bottom-Up versions. The use of quasi-lattices is shown to have a statistically significant correlation with the number of questions asked during atom refinement ($p\text{-}value = 4.45 \times 10^{-8}$, tested with the use of a `MANOVA` (Everitt *et al.* [ES02])). In the following, we analyze the results of our experiments presented in Table 4.2 and Table 4.3. These results are also illustrated by Figure 4.1 showing the boxplots of the difference between the number of asked questions with and without quasi-lattices. We recall that, in a boxplot:

- the central black point corresponds to the median;
- the lower and upper edges are the first quartile $Q_1$ and the third quartile $Q_3$, respectively;
- the lower and upper whisker are the minimum and maximum values excluding outliers, respectively;
- the isolated points are the outliers.

We also recall that a data item is considered as an outlier if its value $v$ is such that:

$$v \leq Q_1 - 1.5 \times (Q_3 - Q_1) \text{ or } v \geq Q_3 + 1.5 \times (Q_3 - Q_1)$$

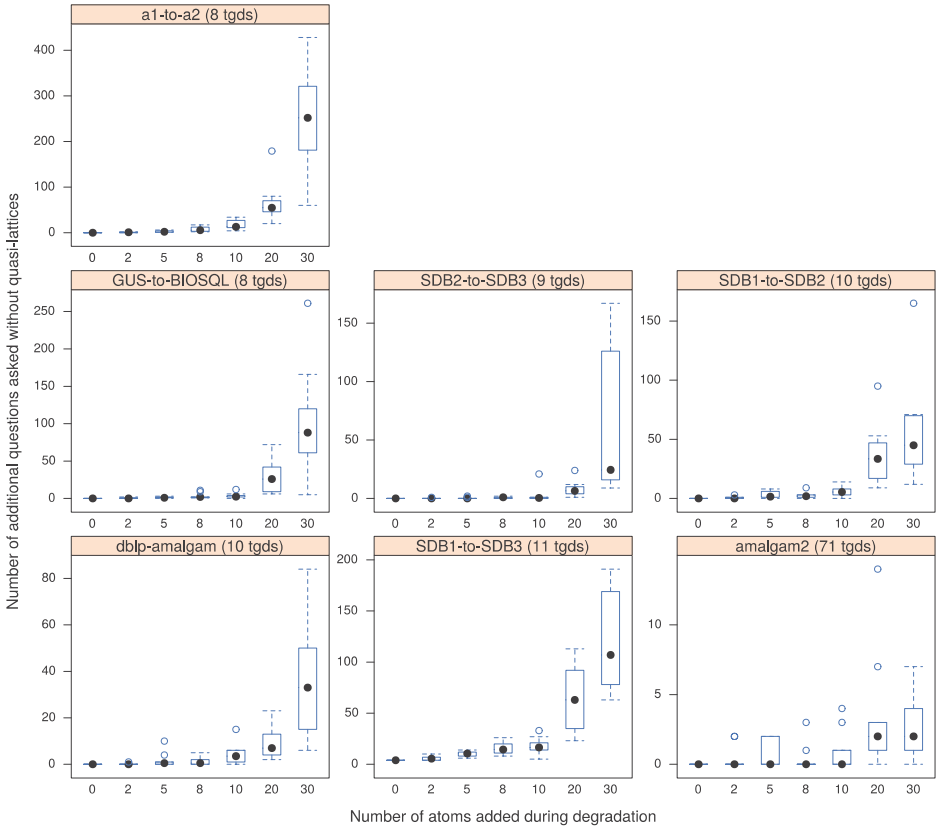
Table 4.2 presents the results of experiments over the real scenarios. It shows the average number of questions asked with the use of quasi-lattices ($\bar{n}_{quasi}$), the average number of additional questions asked without the use of quasi-lattices ($\Delta\bar{n}_{non\_quasi}$), the maximum number of questions asked with quasi-lattices ($max_{quasi}$), and the maximum number of additional questions asked without quasi-lattices ($\Delta max_{non\_quasi}$).

| Scenarios | | | | | Number of asked questions with and without the use of quasi-lattices | | | |
|---|---|---|---|---|---|---|---|---|
| Name | $\|\Sigma\|$ | Source relations | Target relations | Degradations | $\bar{n}_{quasi}$ | $\Delta\bar{n}_{non\_quasi}$ | $max_{quasi}$ | $\Delta max_{non\_quasi}$ |
| a1-to-a2 | 8 | 15 | 27 | 0 | 7 | + 0 | 7 | + 0 |
| | | | | 2 | 11.7 | + 0.9 | 13 | + 2 |
| | | | | 5 | 19.3 | + 2.7 | 22 | + 6 |
| | | | | 8 | 28.3 | + 7.7 | 38 | + 17 |
| | | | | 10 | 36.1 | + 18.4 | 43 | + 34 |
| | | | | 20 | 80.5 | + 67.5 | 175 | + 179 |
| | | | | 30 | 186.7 | + 247 | 269 | + 428 |
| amalgam2 | 71 | 15 | 27 | 0 | 14 | + 0 | 14 | + 0 |
| | | | | 2 | 18 | + 0.4 | 20 | + 2 |
| | | | | 5 | 23.3 | + 0.7 | 26 | + 2 |
| | | | | 8 | 29.1 | + 0.4 | 37 | + 3 |
| | | | | 10 | 32.7 | + 0.9 | 36 | + 4 |
| | | | | 20 | 51.9 | + 3.3 | 65 | + 14 |
| | | | | 30 | 64.3 | + 2.6 | 71 | + 7 |
| dblp-amalgam | 10 | 7 | 9 | 0 | 2 | + 0 | 2 | + 0 |
| | | | | 2 | 5.3 | + 0.1 | 7 | + 1 |
| | | | | 5 | 10.1 | + 1.7 | 14 | + 10 |
| | | | | 8 | 13.4 | + 1.1 | 17 | + 5 |
| | | | | 10 | 19.3 | + 4.1 | 37 | + 15 |
| | | | | 20 | 31.6 | + 8.9 | 50 | + 23 |
| | | | | 30 | 66.7 | + 38.3 | 118 | + 84 |
| GUS-to-BIOSQL | 8 | 7 | 6 | 0 | 6 | + 0 | 6 | + 0 |
| | | | | 2 | 9.8 | + 0.5 | 12 | + 2 |
| | | | | 5 | 16.9 | + 1.3 | 21 | + 3 |
| | | | | 8 | 23.1 | + 3.2 | 28 | + 11 |
| | | | | 10 | 26.8 | + 3.5 | 36 | + 12 |
| | | | | 20 | 67.6 | + 28.3 | 111 | + 72 |
| | | | | 30 | 146.3 | + 100.2 | 270 | + 261 |
| SDB1-to-SDB2 | 10 | 6 | 11 | 0 | 9 | + 0 | 9 | + 0 |
| | | | | 2 | 13.2 | 0.6 | 16 | + 3 |
| | | | | 5 | 19.6 | + 3.1 | 24 | + 8 |
| | | | | 8 | 22.3 | + 2.3 | 25 | + 9 |
| | | | | 10 | 27.2 | + 5.7 | 40 | + 14 |
| | | | | 20 | 66.6 | + 36.7 | 136 | + 95 |
| | | | | 30 | 86.7 | + 56.9 | 127 | + 165 |
| SDB1-to-SDB3 | 11 | 6 | 11 | 0 | 24 | + 4 | 24 | + 4 |
| | | | | 2 | 29.8 | + 5.8 | 34 | + 10 |
| | | | | 5 | 39.6 | + 10.4 | 46 | + 14 |
| | | | | 8 | 50.6 | + 15.5 | 70 | + 26 |
| | | | | 10 | 59.6 | + 17.9 | 79 | + 33 |
| | | | | 20 | 102.4 | + 65 | 166 | + 113 |
| | | | | 30 | 181.3 | + 119.1 | 293 | + 191 |
| SDB2-to-SDB3 | 9 | 11 | 11 | 0 | 3 | + 0 | 3 | + 0 |
| | | | | 2 | 5.7 | + 0.1 | 8 | + 1 |
| | | | | 5 | 10 | + 0.3 | 12 | + 2 |
| | | | | 8 | 12.4 | + 0.8 | 17 | + 2 |
| | | | | 10 | 18.4 | + 2.5 | 50 | + 21 |
| | | | | 20 | 33.2 | + 7.9 | 55 | + 24 |
| | | | | 30 | 88.7 | + 59.5 | 195 | + 167 |

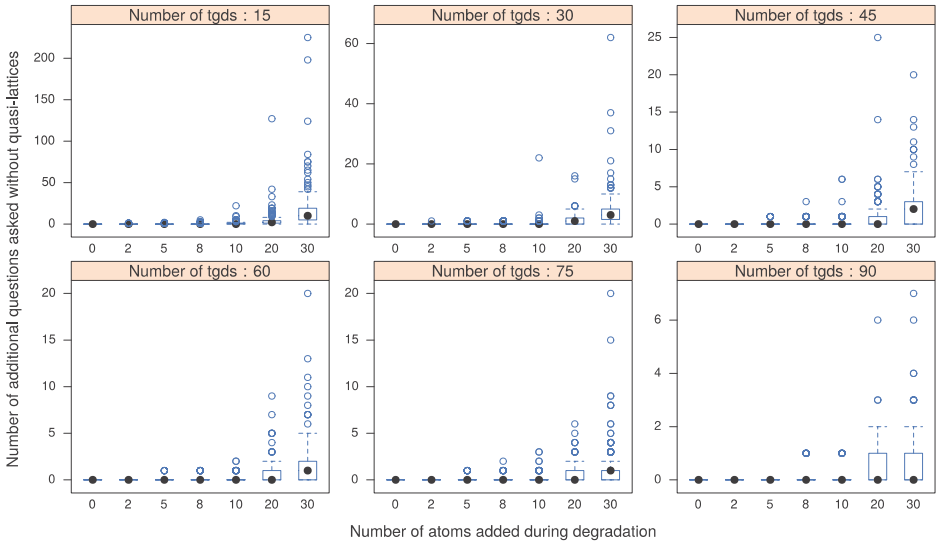
Table 4.2: Experimental results for the real scenarios : average number of asked questions with use of quasi-lattices ($\bar{n}_{quasi}$), average number of additional questions asked without quasi-lattices ($\Delta\bar{n}_{non\_quasi}$), maximum number of asked questions with use of quasi-lattices ($max_{quasi}$) and maximum number of additional questions asked without quasi-lattices ($\Delta max_{non\_quasi}$).

| Scenarios | | | | Number of asked questions with and without the use of quasi-lattices | | | |
|---|---|---|---|---|---|---|---|
| Number of tgds $\|\Sigma\|$ | Source relations | Target relations | Degradations | $\bar{n}_{quasi}$ | $\Delta\bar{n}_{non\_quasi}$ | $max_{quasi}$ | $\Delta max_{non\_quasi}$ |
| 15 tgds | 16 | 12 | 0 | 0 | + 0 | 0 | + 0 |
| | | | 2 | 2.8 | + 0.1 | 5 | + 1 |
| | | | 5 | 6.9 | + 0.1 | 11 | + 2 |
| | | | 8 | 10.9 | + 0.3 | 25 | + 5 |
| | | | 10 | 11.97 | + 0.8 | 48 | + 22 |
| | | | 20 | 31 | + 4.4 | 38 | + 22 |
| | | | 30 | 60.9 | + 17 | 327 | + 225 |
| 30 tgds | 32 | 24 | 0 | 0 | + 0 | 0 | + 0 |
| | | | 2 | 2.7 | + 0.1 | 4 | + 1 |
| | | | 5 | 6.8 | + 0.1 | 9 | + 1 |
| | | | 8 | 11 | + 0.2 | 15 | + 1 |
| | | | 10 | 14.1 | + 0.5 | 38 | + 22 |
| | | | 20 | 27.8 | + 1.7 | 56 | + 16 |
| | | | 30 | 45.4 | + 5.2 | 147 | + 62 |
| 45 tgds | 48 | 36 | 0 | 0 | + 0 | 0 | + 0 |
| | | | 2 | 2.9 | + 0 | 5 | + 0 |
| | | | 5 | 7 | + 0.1 | 11 | + 1 |
| | | | 8 | 11.1 | + 0.1 | 17 | + 3 |
| | | | 10 | 14 | + 0.2 | 39 | + 6 |
| | | | 20 | 27 | + 1 | 68 | + 25 |
| | | | 30 | 41.3 | + 2.2 | 64 | + 20 |
| 60 tgds | 64 | 48 | 0 | 0 | + 0 | 0 | + 0 |
| | | | 2 | 2.8 | + 0 | 4 | + 0 |
| | | | 5 | 7.1 | + 0 | 11 | + 1 |
| | | | 8 | 11 | + 0 | 15 | + 1 |
| | | | 10 | 13.8 | + 0.1 | 22 | + 2 |
| | | | 20 | 27.7 | + 0.6 | 46 | + 9 |
| | | | 30 | 41.6 | + 1.6 | 95 | + 20 |
| 75 tgds | 80 | 60 | 0 | 0 | + 0 | 0 | + 0 |
| | | | 2 | 2.8 | + 0 | 4 | + 0 |
| | | | 5 | 7.1 | + 0 | 10 | + 1 |
| | | | 8 | 11.1 | + 0 | 15 | + 2 |
| | | | 10 | 14.1 | + 0.1 | 23 | + 3 |
| | | | 20 | 27.5 | + 0.5 | 50 | + 6 |
| | | | 30 | 41.5 | + 1.3 | 89 | + 20 |
| 90 tgds | 96 | 72 | 0 | 0 | + 0 | 0 | + 0 |
| | | | 2 | 2.8 | + 0 | 4 | + 0 |
| | | | 5 | 7.2 | + 0 | 10 | + 0 |
| | | | 8 | 11.4 | + 0 | 16 | + 1 |
| | | | 10 | 14 | + 0 | 19 | + 1 |
| | | | 20 | 27.4 | + 0.4 | 35 | + 6 |
| | | | 30 | 41.4 | + 0.8 | 56 | + 7 |

Table 4.3: Experimental results for the scenarios generated with iBench : average number of asked questions with use of quasi-lattices ($\bar{n}_{quasi}$), average number of additional questions asked without quasi-lattices ($\Delta\bar{n}_{non\_quasi}$), maximum number of asked questions with use of quasi-lattices ($max_{quasi}$) and maximum number of additional questions asked without quasi-lattices ($\Delta max_{non\_quasi}$).




(a) real scenarios




(b) generated scenarios

Figure 4.1: Benefit of quasi-lattices during mapping specification

It can be seen that the reduction of the average number of questions by the use of quasi-lattices ranges from 0 questions for the simplest scenarios to a reduction of 247 questions for the scenario `a1-to-a2` with 30 degradations. In this last scenario, it could be noticed that the average number of questions asked with the use of quasi-lattices is 186.7 questions, thus if quasi-lattices are not used the number of questions is more than doubled. Also, the reduction of the maximal number of questions by the use of quasi-lattices ranges from 0 to 428 questions in the most complex scenario. This is illustrated in Figure 4.1a where it can be seen that, when the number of degradation increases, the reduction of the number of questions increases as well. Moreover, when the median number of additional asked questions increases, the first quartile comparably grows as well.

The efficiency of the optimization is not directly correlated with the number of tgds in a scenario. This is illustrated with scenarios `amalgam2` and `SDB1-to-SDB3`, where the biggest one (`amalgam2`) leads to a small amelioration, when the other one leads to high reductions of the number of questions asked. This can be explained by the structure of the tgds contained by the scenarios. When scenarios contain numerous but non-overlapping tgds (i.e., our degraded exemplar tuples sets lead to few $\psi$-equivalent tgds), most of the quasi-lattices cover one tgd at a time and consequently are equivalent to the case without use of semi-lattices. In the other case, even with fewer tgds than in `amalgam2`, the use of quasi-lattices during refinement of scenario `SDB1-to-SDB3` leads to an important reduction of the number of asked questions. Indeed, such a scenario contains tgds which are differentiated by more subtle differences than those in `amalgam2`. We can thus conclude that such scenarios with numerous $\psi$-equivalent tgds lead to exemplar tuples sets which are efficiently handled by the use of quasi-lattices.

Table 4.3 presents the results of the experiments over the generated scenarios with the same information as in Table 4.2.

This table shows that the number of asked questions decreases with the size of the mapping. Indeed, during degradation of a mapping, extraneous atoms are more prone to be added in a same tgd if the mapping size is low, thus leading to an increased complexity of the atom refinement. Moreover, the tgds generated with `iBench` are typically small with only one atom in the left-hand side. Hence, for many of them our framework does not ask any questions as the supremum of the quasi-lattice is the only possible choice. This can also be seen in all cases where no degradations are applied over the generated scenarios, for which our system does not ask any questions in order to infer the correct mapping.

However, in the scenario with 15 tgds and 30 degradations, the use of quasi-

lattices leads to ask an average number of 60.9 questions, which corresponds to an average reduction of 17 questions in comparison with the case without quasi-lattices. For the same scenario,the use of quasi-lattices leads to a reduction up to 112 questions compared to the case without quasi-lattices.

Figure 4.1b illustrates these results while showing that, although the overall reduction is low as discussed previously, the high number of outliers leads to an important reduction of the number of asked questions. This confirms that the use of quasi-lattices is effective for numerous particular cases.

*Overall, these results show that the use of quasi-lattices leads to a noticeable reduction of the number of asked questions. This is especially the case with the most complex scenarios (i.e., the scenarios leading to the greatest number of questions).*

**Influence of join degradation number over the number of asked questions.** In this experiment, we evaluate the influence of the number of join degradations on the number of asked questions. As in our previous experiment, we use a Breadth-First exploration strategy, both in Top-Down and Bottom-Up versions.

The use of quasi-lattices does not show to have a statistically significant correlation with the number of questions asked during join refinement ($p\text{-}value > 0.05$, tested with the use of a `MANOVA` (Everitt *et al.* [ES02])) thus, contrarily to the previous section, we do not provide a comparative analysis with the case without the use of quasi-lattices.

Table 4.4 presents the results of experiments over the real scenarios. It shows the average number of questions asked ($\bar{n}_{quasi}$) and the maximum number of questions asked ($max_{quasi}$).

This table shows that with the real scenarios, the number of questions ranges from 5 questions in average for the scenario `dblp-amalgam` to 313.25 questions in average for scenario `SDB1-to-SDB3` with 30 degradations. This last scenario also comes with the maximal highest number of questions (3206 questions), which can be explained by a fewer number of variables occurrences than in the other scenarios, leading to bigger quasi-lattices to explore after the join degradation phase. However, as illustrated in Figure 4.2a, the cases where more than 40 questions are asked per tgds corresponds to outlier values, and in most of the cases less than 40 questions per tgds are asked, regardless of the scenario.

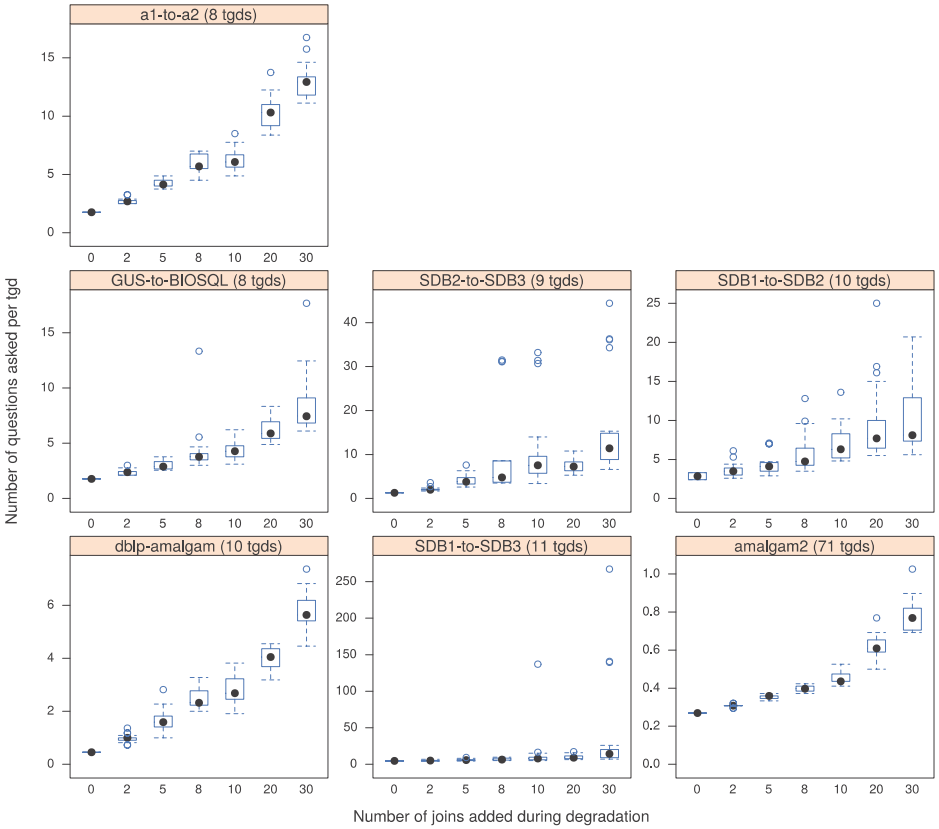
As for the experiments on atom degradations, it can be seen that the number of asked questions is not directly correlated with the number of tgds in a scenario. This is illustrated with scenarios `dblp-amalgam` and `SDB1-to-SDB3`, where the biggest one (`dblp-amalgam`) leads to a small amelioration, when

| Name | Scenarios \|Σ\| | Source relations | Target relations | Degradations | Number of asked questions $\bar{n}_{quasi}$ | $max_{quasi}$ |
|---|---|---|---|---|---|---|
| a1-to-a2 | 8 | 15 | 27 | 0 | 14 | 14 |
| | | | | 2 | 21.55 | 26 |
| | | | | 5 | 33.625 | 39 |
| | | | | 8 | 46.4 | 56 |
| | | | | 10 | 49.375 | 68 |
| | | | | 20 | 81.225 | 110 |
| | | | | 30 | 101.8 | 134 |
| amalgam2 | 71 | 15 | 27 | 0 | 21 | 21 |
| | | | | 2 | 24 | 25 |
| | | | | 5 | 27.7 | 29 |
| | | | | 8 | 31.1 | 33 |
| | | | | 10 | 35.15 | 41 |
| | | | | 20 | 47.825 | 60 |
| | | | | 30 | 60.3 | 80 |
| dblp-amalgam | 10 | 7 | 9 | 0 | 5 | 5 |
| | | | | 2 | 10.725 | 15 |
| | | | | 5 | 18.1 | 31 |
| | | | | 8 | 27.075 | 36 |
| | | | | 10 | 30.425 | 42 |
| | | | | 20 | 43.8 | 50 |
| | | | | 30 | 63.4 | 81 |
| GUS-to-BIOSQL | 8 | 7 | 6 | 0 | 16 | 16 |
| | | | | 2 | 21.2 | 27 |
| | | | | 5 | 26.925 | 34 |
| | | | | 8 | 37 | 120 |
| | | | | 10 | 38.65 | 56 |
| | | | | 20 | 54.825 | 75 |
| | | | | 30 | 73.15 | 159 |
| SDB1-to-SDB2 | 10 | 6 | 11 | 0 | 27.75 | 33 |
| | | | | 2 | 34.275 | 61 |
| | | | | 5 | 41.625 | 71 |
| | | | | 8 | 52.2 | 128 |
| | | | | 10 | 64.25 | 136 |
| | | | | 20 | 85.15 | 250 |
| | | | | 30 | 93.125 | 207 |
| SDB1-to-SDB3 | 11 | 6 | 11 | 0 | 54.25 | 61 |
| | | | | 2 | 58.625 | 81 |
| | | | | 5 | 69.875 | 112 |
| | | | | 8 | 79.55 | 116 |
| | | | | 10 | 135.75 | 1645 |
| | | | | 20 | 111.125 | 207 |
| | | | | 30 | 313.35 | 3206 |
| SDB2-to-SDB3 | 9 | 11 | 11 | 0 | 13 | 13 |
| | | | | 2 | 21.05 | 36 |
| | | | | 5 | 41.05 | 76 |
| | | | | 8 | 91.8 | 315 |
| | | | | 10 | 108.9 | 332 |
| | | | | 20 | 75.2 | 108 |
| | | | | 30 | 158.95 | 444 |

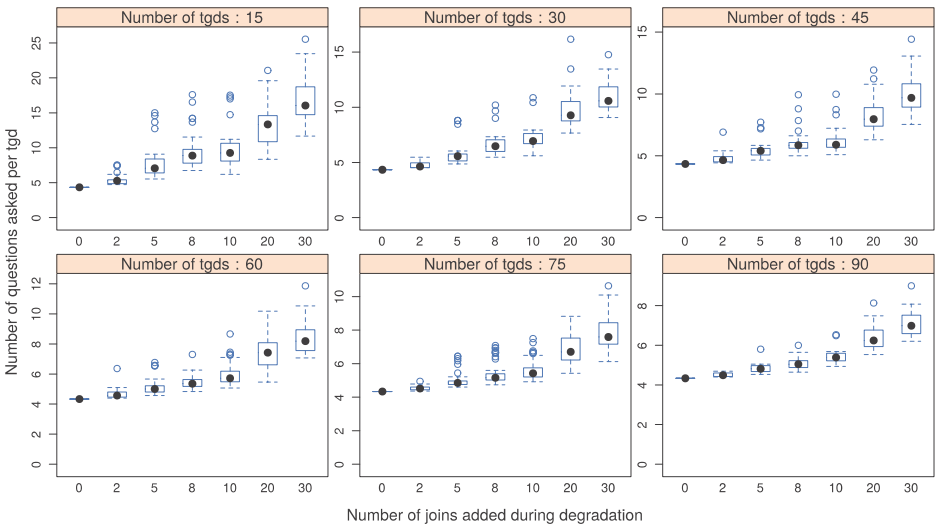
Table 4.4: Experimental results on join degradations for the real scenarios : average number of asked questions with use of quasi-lattices ($\bar{n}_{quasi}$) and maximum number of asked questions with use of quasi-lattices ($max_{quasi}$).

| Scenarios | | | | Number of asked questions | |
|---|---|---|---|---|---|
| Number of tgds $\|\Sigma\|$ | Source relations | Target relations | Degradations | $\bar{n}_{quasi}$ | $max_{quasi}$ |
| 15 tgds | 16 | 12 | 0 | 65 | 65 |
| | | | 2 | 81 | 113 |
| | | | 5 | 115.95 | 225 |
| | | | 8 | 141.275 | 264 |
| | | | 10 | 146.775 | 263 |
| | | | 20 | 200.95 | 316 |
| | | | 30 | 254.325 | 383 |
| 30 tgds | 32 | 24 | 0 | 130 | 130 |
| | | | 2 | 143.4 | 164 |
| | | | 5 | 177.4 | 264 |
| | | | 8 | 204.65 | 306 |
| | | | 10 | 220.4 | 326 |
| | | | 20 | 297.3 | 485 |
| | | | 30 | 330.75 | 443 |
| 45 tgds | 48 | 36 | 0 | 195 | 195 |
| | | | 2 | 214.2 | 311 |
| | | | 5 | 246.1 | 347 |
| | | | 8 | 270.65 | 447 |
| | | | 10 | 278.25 | 449 |
| | | | 20 | 374.3 | 537 |
| | | | 30 | 446.2 | 649 |
| 60 tgds | 64 | 48 | 0 | 260 | 260 |
| | | | 2 | 280.725 | 382 |
| | | | 5 | 308.25 | 406 |
| | | | 8 | 325.175 | 438 |
| | | | 10 | 358.15 | 520 |
| | | | 20 | 449.8 | 611 |
| | | | 30 | 505.45 | 712 |
| 75 tgds | 80 | 60 | 0 | 325 | 325 |
| | | | 2 | 340.3 | 371 |
| | | | 5 | 373.4 | 483 |
| | | | 8 | 403.9167 | 531 |
| | | | 10 | 418.9333 | 562 |
| | | | 20 | 513.9 | 662 |
| | | | 30 | 588.8 | 798 |
| 90 tgds | 96 | 72 | 0 | 390 | 390 |
| | | | 2 | 405.5 | 422 |
| | | | 5 | 435.55 | 522 |
| | | | 8 | 457.4 | 540 |
| | | | 10 | 490.45 | 588 |
| | | | 20 | 576.6 | 732 |
| | | | 30 | 636.55 | 810 |

Table 4.5: Experimental results on join degradations for the scenarios generated with iBench : average number of asked questions with use of quasi-lattices ($\bar{n}_{quasi}$) and maximum number of asked questions with use of quasi-lattices ($max_{quasi}$).




(a) real scenarios




(b) generated scenarios

Figure 4.2: Number of questions asked during mapping specification in function of join degradations number

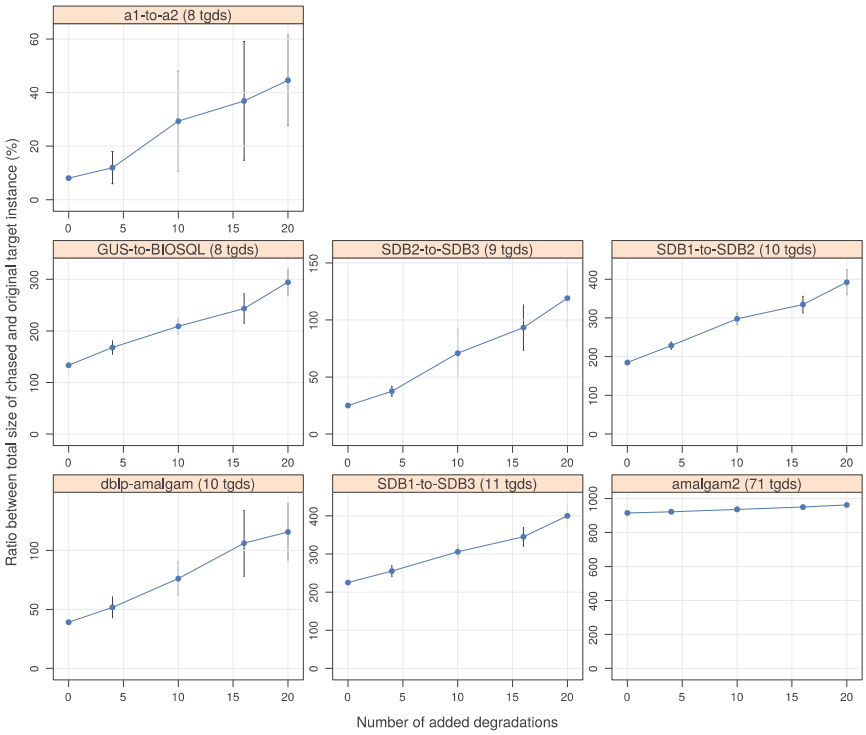
`SDB1-to-SDB3` leads to high reductions of the number of questions asked. As in the previous experiments, this can be explained by the structure of the tgds contained by the scenarios. A first factor is the number of ambiguous joins in a scenario before the application of join degradations. For example, this can be seen on the scenarios `SDB1-to-SDB3` and `amalgam2`: without join degradations scenario `SDB1-to-SDB3` (11 tgds) leads to ask 54.25 questions in average, when in the same setting scenario `amalgam2` (71 tgds) leads to ask 21 questions in average. The other factor is directly imputable to the degradation procedure. Indeed, during degradation of a mapping, extraneous joins are more prone to be added in a same tgd if the mapping size is low, thus leading to an increased complexity of the join refinement.

The effect of the mapping size over the influence of the join degradation procedure is more visible on our experiments with the generated scenarios as we will illustrate now. The results over these scenarios are presented in Table 4.5 with the same information as in Table 4.4. It is seen that the total number of asked questions increases both with the number of tgds in the mappings and with the number of join degradations. However, Figure 4.2b illustrates that the number of questions asked to specify one tgd in a mapping decrease with the size of this mapping. For example, it is seen that scenarios with 15 tgds and 30 degradations lead to ask at least 11 questions to specify a tgd, when scenarios with 90 tgds and 30 degradations lead to ask a maximum of 9 questions to specify a tgd.

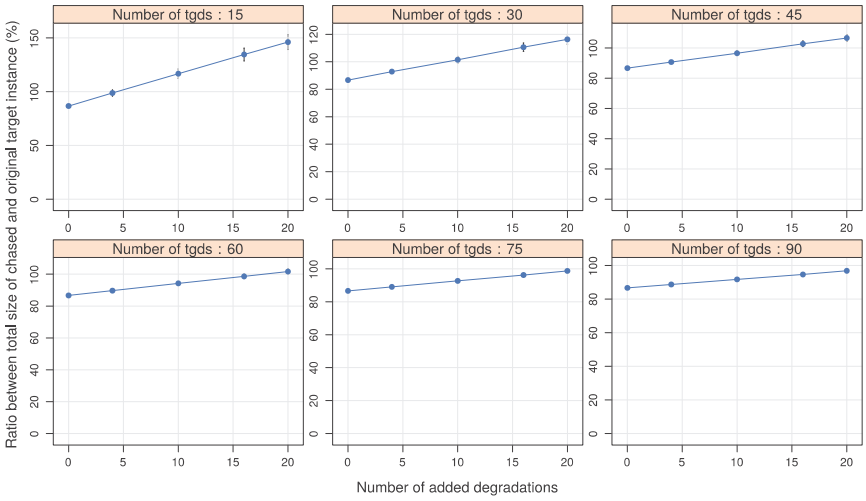
*Overall, these results show that despite rare cases where a high number of questions needs to be asked in order to retrieve the expected mapping, in average the number of questions is kept low despite the introduction of numerous join degradations.*

### 4.1.3 Benefit of (non-universal) exemplar tuples

Our second experiment aims to evaluate the benefit of using exemplar tuples as opposed to universal examples adopted in Alexe *et al.* [AtCKT11b] for the mapping inference process. For each scenario, we apply the chase to all the source instances $E_S^i$ to obtain $\textsc{chase}(\mathcal{M}, E_S^i)$. This lets us compute the number of tuples in the target instance of the universal solution produced, which we compare with the number of tuples in the target instances used in our approach. Concretely, for exemplar tuples $\{(E_S^1, E_T^1); \ldots; (E_S^n, E_T^n)\}$ and




(a) Real scenarios




(b) Scenarios generated with `iBench`

Figure 4.3: Growth of the ratio $r$ with respect to number of degradations.

the corresponding expected mapping $\mathcal{M}$, we calculate the ratio:

$$r = \frac{\sum_{i=1}^{n} |\,\textsc{chase}(\mathcal{M}, E_S^i)|}{\sum_{i=1}^{n} |E_T^i|} - 1$$

In order to get a comprehensive view of the effects of atom and join degradations, both degradations occur together in this experiment. Precisely, in Figure 4.3a and Figure 4.3b we present the results where an equal number of atoms and joins degradations are used for, respectively, the real and the generated mapping scenarios. The $x$ axis corresponds to the total number of degradations (e.g., the value 20 corresponds to the case with 10 atoms and 10 join degradations), while the $y$ axis corresponds to the aforementioned ratio $r$.

In all the employed scenarios, we can observe the effectiveness and practicality of using exemplar tuples as opposed to the universal data examples of `EIRENE`: universal exemplar tuples are from 8% to 962% larger than the non-universal ones used in our approach. Moreover, in all scenarios, we can observe a strong linear correlation between the number of degradations and the number of additional target tuples needed by universal examples. Hence, the more degradations the exemplar tuples have, the larger is the benefit of using our approach. Notice that the scenario that is the least sensitive to the variation of the number of degradations is `amalgam2`, which is also the real scenario with the greatest number of tgds.

Such a scenario is also among those that exhibited the maximum benefit of using fewer exemplar tuples rather. Although the precise amount of gain is clearly dependent on the dataset and on the number of degradations, *we can observe that, in all scenarios, the advantage of using non-universal exemplar tuples is non-negligible, thus making our approach a practical solution for mapping specification.*

### 4.1.4 Relative benefit of interactivity

A key contribution of our mapping specification method is that it helps the user to interactively correct errors (e.g., unnecessary atoms during atom refinement, collisions of constants during join refinement) that may appear in the exemplar tuples. In this section, we aim at quantifying this benefit via a comparison with a baseline approach, i.e., the one in which refinement steps are disabled. As a baseline, we adopted the canonical GLAV generation performed in `EIRENE`[1]

[1]For the sake of fairness, `EIRENE`'s canonical GLAV mappings are *split-reduced* and $\sigma$-*redundant* tgds are suppressed.

| Scenarios | Number of extraneous atoms added | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 2 | 5 | 8 | 10 | 20 | 30 |
| a1-to-a2 | 0 | 8.3 | 18.5 | 26.2 | 31 | 44.5 | 52.9 |
| amalgam2 | 0 | 2.1 | 5.1 | 8 | 9.8 | 17.8 | 24.6 |
| dblp-amalgam | 0 | 11.1 | 23.9 | 32.1 | 37 | 53.6 | 61.3 |
| GUS-to-BIOSQL | 0 | 11.7 | 24.6 | 34.5 | 40 | 55.2 | 65.3 |
| SDB1-to-SDB2 | 0 | 11.1 | 23.8 | 33.3 | 37.9 | 54.9 | 63.8 |
| SDB1-to-SDB3 | 0 | 6.7 | 14.9 | 22 | 26.3 | 41.4 | 50.9 |
| SDB2-to-SDB3 | 0 | 11.8 | 24.2 | 34.4 | 40 | 54.4 | 64.8 |
| Average | 0 | 8.9 | 19.2 | 27.2 | 31.7 | 45.9 | 54.8 |

(a) Real scenarios

| Scenarios | Number of extraneous atoms added | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 2 | 5 | 8 | 10 | 20 | 30 |
| 15 tgds | 0 | 7.7 | 17.1 | 24.9 | 29.1 | 44.7 | 54.5 |
| 30 tgds | 0 | 4 | 9.4 | 14.2 | 17.1 | 29.1 | 38 |
| 45 tgds | 0 | 2.7 | 6.5 | 10 | 12.2 | 21.5 | 29 |
| 60 tgds | 0 | 2 | 4.9 | 7.7 | 9.4 | 17.1 | 23.6 |
| 75 tgds | 0 | 1.6 | 4 | 6.2 | 7.7 | 14.2 | 19.8 |
| 90 tgds | 0 | 1.4 | 3.3 | 5.2 | 6.5 | 12.1 | 17.1 |
| Average | 0 | 3.2 | 7.5 | 11.3 | 13.6 | 23.1 | 30.3 |

(b) Scenarios generated with iBench

Table 4.6: Relative difference (in percent) between EIRENE and our framework.

(Alexe *et al.* [AtCKT11b]). As EIRENE is not intended to handle errors in its input data examples, we had to make sure that exemplar tuples in our case are an acceptable input for EIRENE, in particular that they pass the so-called "homomorphism extension test". In other words, we bootstraps our algorithms on universal exemplar tuples $(E_S, E_T)$ in order to warrant such comparison.

We use the sum of the number of left-hand side atoms of the tgds as the comparison criterion: the larger it is, the more "complex" is the mapping for the end user. This optimality criterion is inspired by a compound measure proposed in Gottlob *et al.* [GPS11]. Notice that this comparison only deals with extraneous atoms during atom refinement and does not consider collision of values, which is done during join refinement. For such reason, and also due to the fact that here we are compelled to use universal data examples instead of few arbitrary exemplar tuples in order to compare with EIRENE, this comparison should be taken with a grain of salt.

The obtained results are presented in Table 4.6a and Table 4.6b for real and

| | 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 4.7: Properties of the generated iBench scenarios.

generated scenarios, respectively. If no extraneous atom is added to the left-hand sides of mappings, then there is no qualitative difference between the two approaches. However, when extraneous atoms are introduced, a remarkable difference can be observed: across real scenarios, `EIRENE`'s canonical mapping is about 27% larger on average when 8 such atoms are introduced, and goes up to 54% on average with 30 atoms. For the generated scenarios, which lead to fewer asked questions during refinement, `EIRENE`'s canonical mapping is still about 11% larger on average when 8 such atoms are introduced, and goes up to 30% on average with 30 atoms. *Hence, our mappings are noticeably simpler than* `EIRENE`*'s ones. Such improvement is both beneficial for the readability of mappings as well as for their efficiency because spurious atoms are eliminated.*

## 4.2 Efficiency of the repairing process

In this section, we investigate the efficiency of our repairing framework both with the use of hard-coded preference function and with a preference function based on a simple learning approach. The source code is publicly available at `https://github.com/ucomignani/MapRepair.git`.

### 4.2.1 Experimental setting

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`. We considered relations of up to five attributes and we created GAV mappings using the `iBench` configuration recommended by Arocena *et al.* [AGCM15]. 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 4.7. 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 [EHW16] that provides an off-the-shelf implementation of the k-NN algorithm.

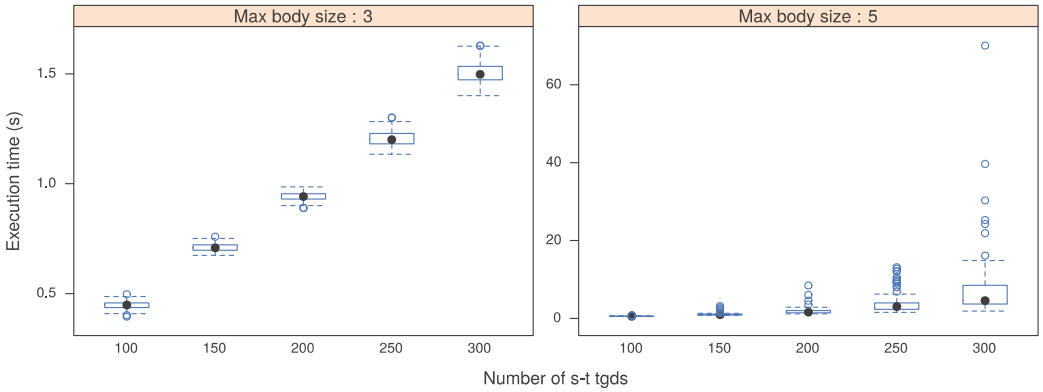



Figure 4.4: Repairing times.

We ran our experiments on a 2.6GHz 4-core, 16Gb laptop running Debian 9.

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

### 4.2.2 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 4.4. Figure 4.4 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 4.5 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 tine 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*

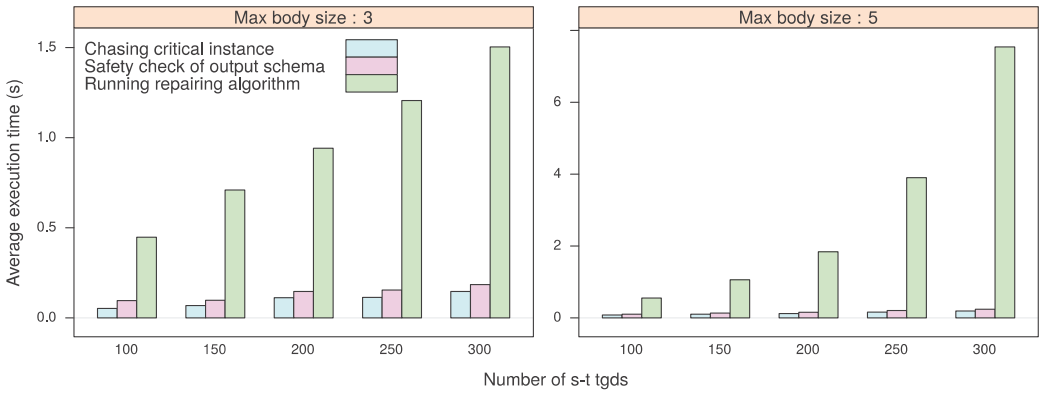



Figure 4.5: Time comparisons.

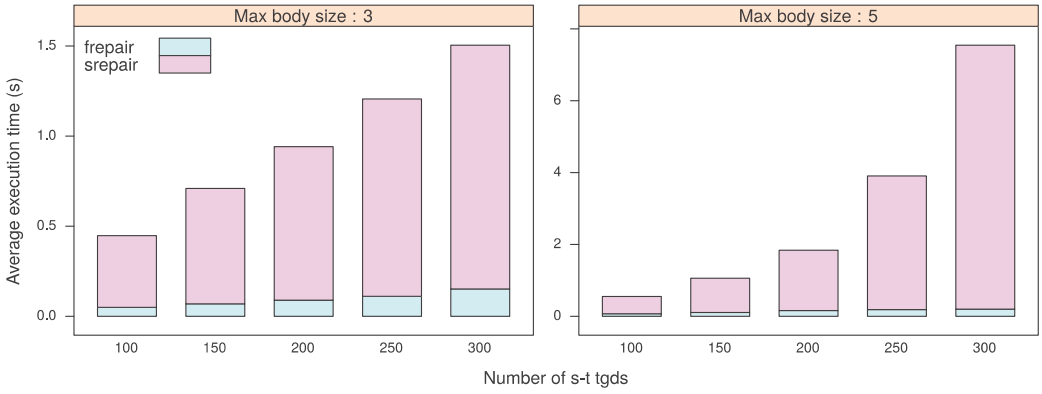



Figure 4.6: Time breakdown between frepair and srepair.

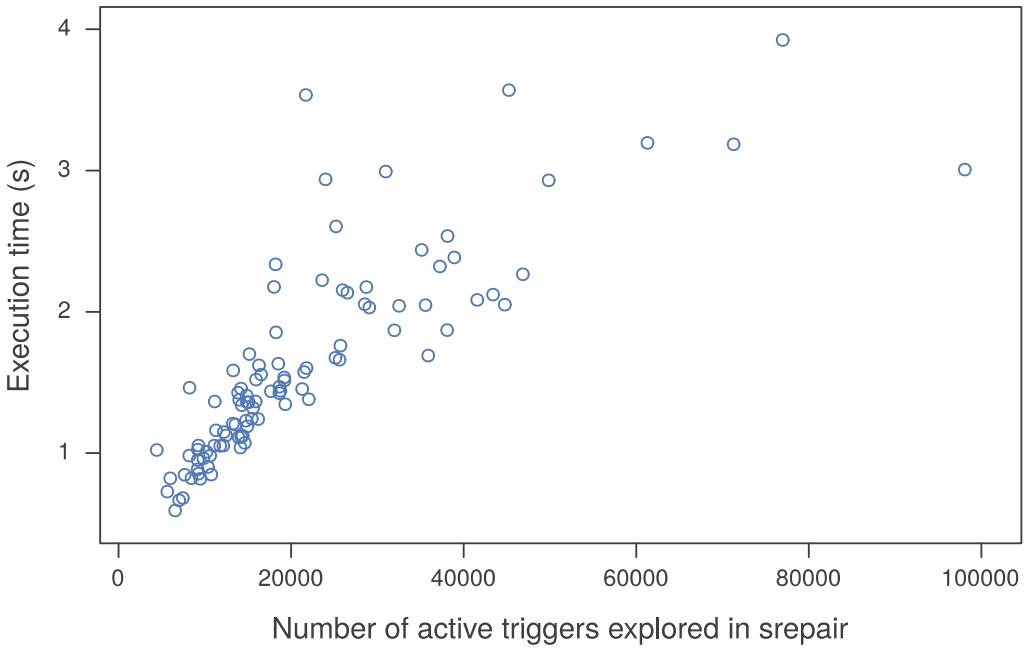



Figure 4.7: Running time of srepair over 100 s-t tgds.

*in their bodies.*

### 4.2.3 Time breakdown between frepair and srepair

Figure 4.6 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 overhead that is incurred during the incremental computation of the visual chase after repairing an s-t tgd (line 29 of Algorithm 10). Figure 4.7 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 ($p\text{-}value < 2.2e^{-16}$). Figure 4.7 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.*

### 4.2.4 Evaluating learning accuracy and efficiency

We adopted the following steps in order to evaluate the performance of our learning approach. First, we recall that, given two possible repairs $\mu_{r_1}$ and $\mu_{r_2}$, $\Delta_{FV}$ corresponds to the difference between the number of exported variables in $\mu_{r_1}$ and $\mu_{r_2}$, and that $\Delta_J$ corresponds to the difference between the number of joins in the bodies of $\mu_{r_1}$ and $\mu_{r_2}$. We defined the following two golden standard preference functions that we 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. It should be noted that higher values of this parameter led to comparable predictions. Finally, we used the trained k-NN classifier as a preference function in srepair, rerun the scenarios from Section 4.2.2 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 4.8a 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 4.8a 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) [BBC+00] 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 $\mu_2$). Given the values:

- $N_{1,1}$ 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

this measure is calculated as follows:

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

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 4.8b 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, the learning approach still leads to an acceptable fit of our preference function in the case of the learning of $P_{avg}$.*

**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 4.4) and when adopting a learned preference function. Figure 4.8 shows the running times for the same scenarios used in Figure 4.4. *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 performance of our algorithm.*

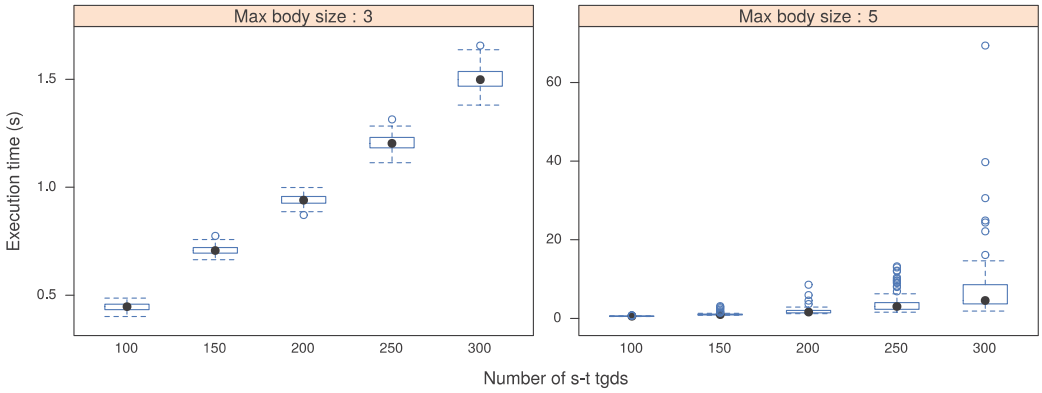



Figure 4.8: Repairing times with ML.

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

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

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

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

Table 4.8: Confusion matrix for the golden standards.

## 4.3 Conclusion

In this section, we have provided the results of our experiments over the two prototypes implemented from the framework exposed in the previous chapters.

At first, we have exposed the results over the *interactive mapping specification* part of our framework by showing its efficiency at keeping reasonably low the number of interactions needed to specify a mapping. We have also shown that the use of quasi-lattices structures to prevent the exploration of equivalent tgds is an important factor of reduction of the number of interactions during the atom refinement step. Next, we have shown that the use of exemplar tuples instead of universal data examples allows users to provide smaller examples and, coupled with the interactivity, leads to produce mappings allowing to retrieve a higher number of tuples of interest than the `EIRENE` system.

In a second time, we have shown that our rewriting approach maintains short rewriting times even for complex mappings with a high number of tgds and numerous active triggers explored. We have also shown that simple preference functions can be efficiently learned, even with a minimal amount of metrics.

# Conclusions

In this thesis, we have presented a novel framework allowing non-expert users to handle data exchange tasks.

At first, we have studied the problem of specifying mappings in an interactive way with inputs that are easy to produce for non-expert users, as well as simple interactions.

To do so, we have formally defined the *Interactive Mapping Specification problem* and the *exemplar tuples sets* it takes as input in order to allows the specification of mappings. Then, we have studied a formal approach to the resolution of this problem, as well as a class of *fully informative exemplar tuples sets* allowing to retrieve a mapping logically equivalent to the one expected by users. Next, we have proved the correctness of this formal approach, as well as its convergence to a unique output mapping given a set of users interactions. We have also proved the completeness of our approach when a *fully informative exemplar tuples set* is provided as input of the framework. Finally, we have provided the worst-case complexity of our formal framework in terms of the maximal number of interactions with the users.

Then, built over our formal framework, we have studied a practical framework for the resolution of the *Interactive Mapping Specification problem*. In this approach, we have provided algorithms allowing to explore efficiently the set of candidates output mapping by ordering them into interleaved quasi-lattices structure. We have shown that these structures can be used to efficiently prune large sets of candidate mappings, in order to reduce the number of interactions with our users. Next, we have provided algorithms allowing to improve the pruning with the use of integrity constraints. Finally, we have provided the worst-case complexity of the two steps of our approach, as well as proofs that the optimizations introduced by our practical framework do not break the good properties proved over our formal framework.

Furthermore, we have shown the efficiency of our approach in an extensive set of experiments.

In the second part of this thesis, we have studied the problem of *rewriting a mapping with respect to a set of policy views defined over its source schema*

in order to prevent forbidden information leakage.

To do so, we have provided a definition of the safety of a mapping given privacy restrictions taking the form of *policy views*, and a simple way to assess this safety. Next, we have provided a rewriting algorithm allowing to rewrite an unsafe mapping to obtain a mapping which is safe with respect to a set of policy views. We have proved that this rewriting algorithm will always output a mapping that is safe with respect to the set of policy views that has served as reference. We have also proved that our rewriting algorithm always return a non-empty mapping if such a mapping exists. Finally, as our rewriting algorithm can lead to multiple rewritings, we have provided a simple approach to learn the user's preference in order to automatize this choice.

We have shown the efficiency of our approach in an extensive set of experiments in which we have evaluated the rewriting time needed in various configurations as well as the efficiency of our learning approach.

## Future directions of investigation

Regarding the mapping specification part of our work, an interesting direction to explore would be to introduce a provenance tracking of user's interactions in order to spot incoherent answers, and being able to handle errors both in their answers or in the *exemplar tuples sets* they provide.

Another interesting direction would be to introduce new optimizations during the mapping specification such as the use of additional integrity constraints during the pruning (e.g., the inclusion dependencies over the target schema of the specified mapping).

We also want to extend the use of learning approaches in our approach. During the specification of mapping, we think that such approaches could be fruitfully introduced in order to improve the quasi-lattices exploration strategies and to completely or partially replace users answering. During the reparation of mappings, we want to extend the exploratory work done to more complex metrics to learn users' preferences functions and provide a comparative work over multiple learning methods.

Finally, recalling that the reparation framework works with GAV mappings, its extension to other classes of mappings is also an important direction of investigation.

# Bibliography


[AAP+13] Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of PODS*, pages 49–60, 2013.

[ABLM14] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of data exchange*. Cambridge University Press, 2014.

[ABPT19] Paolo Atzeni, Luigi Bellomarini, Paolo Papotti, and Riccardo Torlone. Meta-mappings for schema mapping reuse. *Proceedings of the VLDB Endowment*, 12(5):557–569, 2019.

[ACKT11] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, 2011.

[ACMT08] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, and Wang Chiew Tan. Muse: Mapping understanding and design by example. In *Proceedings of ICDE*, pages 10–19, 2008.

[AGCM15] Patricia C Arocena, Boris Glavic, Radu Ciucanu, and Renée J Miller. The iBench integration metadata generator. *Proceedings of the VLDB Endowment*, 9(3):108–119, 2015.

[AHS12] Azza Abouzied, Joseph M. Hellerstein, and Avi Silberschatz. Playful query specification with dataplay. *Proceedings of the VLDB Endowment*, 5(12):1938–1941, 2012.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[Alt] Altova. Mapforce v.2019. `https://www.altova.com/mapforce`.

[Ang87] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[APRR09] Marcelo Arenas, Jorge Pérez, Juan Reutter, and Cristian Riveros. Composition and inversion of schema mappings. *arXiv preprint arXiv:0910.3372*, 2009.

[AtCKT11a] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Designing and refining schema mappings via data examples. In *Proceedings of SIGMOD*, pages 133–144, 2011.

[AtCKT11b] Bogdan Alexe, Balder ten Cate, Phokion G Kolaitis, and Wang-Chiew Tan. Eirene: Interactive design and refinement of schema mappings via data examples. *Proceedings of VLDB*, 2011.

[Bar09] Pablo Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.

[BB04] Joachim Biskup and Piero Bonatti. Controlled query evaluation for enforcing confidentiality in complete information systems. *International Journal of Information Security*, 3(1), 2004.

[BBC+00] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5), 2000.

[BCCT17] Angela Bonifati, Ugo Comignani, Emmanuel Coquery, and Romuald Thion. Interactive mapping specification with exemplar tuples. In *Proceedings of SIGMOD*, pages 667–682, New York, NY, USA, 2017. ACM.

[BCCT19] Angela Bonifati, Ugo Comignani, Emmanuel Coquery, and Romuald Thion. Interactive mapping specification with exemplar tuples. *ACM Trans. Database Syst.*, 44(3):10:1–10:44, 2019.

[BCS16] Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24:1–24:38, 2016.

[BCT19a] Angela Bonifati, Ugo Comignani, and Efthymia Tsamoura. Maprepair: Mapping and repairing under policy views. In *Proceedings of SIGMOD*, pages 1873–1876, New York, NY, USA, 2019. ACM.

[BCT19b] Angela Bonifati, Ugo Comignani, and Efthymia Tsamoura. Repairing mappings under policy views. *arXiv e-prints*, page arXiv:1903.09242, 2019.

[BGK17] M. Benedikt, B. Cuenca Grau, and E. Kostylev. Source Information Disclosure in Ontology-Based Data Integration. In *Proceedings of AAAI*, 2017.

[BKM+17] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In *Proceedings of PODS*, 2017.

[BKS95] Piero A. Bonatti, Sarit Kraus, and VS Subrahmanian. Foundations of secure deductive databases. *IEEE Trans. Knowl. Data Eng.*, 7(3):406–422, 1995.

[BMPV11] Angela Bonifati, Giansalvatore Mecca, Paolo Papotti, and Yannis Velegrakis. Discovery and correctness of schema mapping transformations. In *Schema matching and mapping*, pages 111–147. Springer, 2011.

[BW08] Joachim Biskup and Torben Weibert. Keeping secrets in incomplete databases. *International Journal of Information Security*, 7(3):199–217, 2008.

[CDK13] Balder Ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28:1–28:31, 2013.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogenous information sources. In *Information Processing Society of Japan*, 1994.

[CH06] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, 2006.

[CK90] Chen Chung Chang and H Jerome Keisler. *Model theory*, volume 73. Elsevier, 1990.

[CKQT17] Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. Approximation Algorithms for Schema-Mapping Discovery from Data Examples. *ACM Trans. Database Syst.*, 42(2):12:1–12:41, 2017.

[Cod70] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[CT06] Laura Chiticariu and Wang-Chiew Tan. Debugging schema mappings with routes. In *Proceedings of the 32nd international conference on Very large data bases*, pages 79–90. VLDB Endowment, 2006.

[DNR08] Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 149–158. ACM, 2008.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[DT03] Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. In *Proceedings of ICDT*, pages 225–241. Springer, 2003.

[EHW16] Frank Eibe, MA Hall, and IH Witten. The weka workbench. online appendix for" data mining: Practical machine learning tools and techniques. *Morgan Kaufmann*, 2016.

[ES02] Brian Everitt and Anders Skrondal. *The Cambridge dictionary of statistics*, volume 106. Cambridge University Press, 2002.

[Fag80] Ronald Fagin. Horn clauses and database dependencies. In *Proceedings of STOC*, pages 123–134. ACM, 1980.

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[Fag07] Ronald Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(4), 2007.

[FFF+05] Ronald Fagin, Ronald Fagin, Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: Getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

[FHH+09] Ronald Fagin, Laura M Haas, Mauricio Hernández, Renée J Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236. Springer, 2009.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics, 2001.

[FKMP05] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[FKPT08] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. Quasi-inverses of schema mappings. *ACM Trans. Database Syst.*, 33(2):11:1–11:52, 2008.

[FLM+99] Marc Friedman, Alon Y Levy, Todd D Millstein, et al. Navigational plans for data integration. *Proceedings of AAAI*, 1999:67–73, 1999.

[GAMH10] Boris Glavic, Gustavo Alonso, Renée J. Miller, and Laura M. Haas. Tramp: Understanding the behavior of schema mappings through provenance. *Proceedings of the VLDB Endowment*, 3(1-2):1314–1325, 2010.

[GDM+11] Boris Glavic, Jiang Du, Renée J. Miller, Gustavo Alonso, and Laura M. Haas. Debugging data exchange with vagabond. *Proceedings of the VLDB Endowment*, 4(12):1383–1386, 2011.

[GHK+13] B Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research*, 47:741–808, 2013.

[GKKZ15] Bernardo Cuenca Grau, Evgeny Kharlamov, Egor V. Kostylev, and Dmitriy Zheleznyakov. Controlled Query Evaluation for Datalog and OWL 2 Profile Ontologies. In *IJCAI*, 2015.

[GMUW08] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, second edition, 2008.

[GPS11] Georg Gottlob, Reinhard Pichler, and Vadim Savenkov. Normalization and optimization of schema mappings. *The VLDB Journal*, 20(2):277–302, 2011.

[Grä02] George Grätzer. *General lattice theory*. Springer Science & Business Media, 2002.

[GS10] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2):6:1–6:37, 2010.

[Hit] Hitachi. Pentaho data integration. `https://www.pentaho.com/`.

[JBC+18] Zhongjun Jin, Christopher Baik, Michael Cafarella, HV Jagadish, and Yuze Lou. Demonstration of a multiresolution schema mapping system. *arXiv preprint arXiv:1812.07658*, 2018.

[JBCJ18] Zhongjun Jin, Christopher Baik, Michael Cafarella, and H. V. Jagadish. Beaver: Towards a declarative schema mapping. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 10:1–10:4, New York, NY, USA, 2018. ACM, ACM.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[Kol05] Phokion G Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 61–75. ACM, 2005.

[Kol18] Phokion G Kolaitis. Reflections on schema mappings, data exchange, and metadata management. In *Proceedings of PODS*, pages 107–109. ACM, 2018.

[KPQ19] Phokion G Kolaitis, Lucian Popa, and Kun Qian. Knowledge refinement via rule selection. *arXiv preprint arXiv:1901.10051*, 2019.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of PODS*, pages 233–246. ACM, 2002.

[Lev00] Alon Y Levy. Logic-based techniques in data integration. In *Logic-based artificial intelligence*, pages 575–595. Springer, 2000.

[LRO96] Alon Y Levy, Anand Rajaraman, and Joann J Ordille. The world wide web as a collection of views: Query processing in the information manifold. In *VIEWS*, pages 43–55, 1996.

[Man17] Federica Mandreoli. A framework for user-driven mapping discovery in rich spaces of heterogeneous data. In *OTM Confederated International Conferences*, pages 399–417. Springer, 2017.

[MG10] Bruno Marnette and Floris Geerts. Static analysis of schema-mappings ensuring oblivious termination. In *Proceedings of ICDT*, pages 183–195. ACM, 2010.

[MHH+01] Renée J Miller, Mauricio A Hernández, Laura M Haas, Ling-Ling Yan, CT Howard Ho, Ronald Fagin, and Lucian Popa. The clio project: managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[MLVP14] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: Give me an example of what you need. *Proceedings of the VLDB Endowment*, 7(5):365–376, 2014.

[MLVP17] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. New trends on exploratory methods for data analytics. *Proceedings of the VLDB Endowment*, 10(12):1977–1980, 2017.

[MMS79] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[MS07] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *Journal of Computer and System Sciences*, 73(3), 2007.

[ND07] Alan Nash and Alin Deutsch. Privacy in GLAV information integration. In *Proceedings of ICDT*, 2007.

[One13] Adrian Onet. The chase procedure and its applications in data exchange. In *Dagstuhl Follow-Ups*, volume 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[Pat19] Norman W Paton. Automating data preparation: Can we? should we? must we? *Workshop Proceedings of the EDBT/ICDT*, 2019.

[PVH+02] Lucian Popa, Yannis Velegrakis, Mauricio A. Hernández, Renée J. Miller, and Ronald Fagin. Translating web data. In *Proceedings of VLDB*, pages 598–609, 2002.

[QCJ12] Li Qian, Michael J Cafarella, and HV Jagadish. Sample-driven schema mapping. In *Proceedings of SIGMOD*, pages 73–84. ACM, 2012.

[SDJVdR83] George L. Sicherman, Wiebren De Jonge, and Reind P. Van de Riet. Answering queries without revealing secrets. *ACM Trans. Database Syst.*, 8(1):41–59, 1983.

[SME+17] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Synthesizing entity matching rules by examples. *Proceedings of the VLDB Endowment*, 11(2):189–202, 2017.

[SNCB15] Muhammad I. Sarfraz, Mohamed Nabeel, Jianneng Cao, and Elisa Bertino. Dbmask: Fine-grained access control on encrypted relational databases. In *Proceedings of CODASPY*, pages 1–11, 2015.

[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[Tal] Talend. Talend data integration. `https://www.talend.com/products/data-integration/`.

[tCKQT18] Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. Active learning of gav schema mappings. In *Proceedings of PODS*, SIGMOD/PODS '18, pages 355–368, New York, NY, USA, 2018. ACM.

[TCKT10] Balder Ten Cate, Phokion G Kolaitis, and Wang-Chiew Tan. Database constraints and homomorphism dualities. In *International Conference on Principles and Practice of Constraint Programming*, pages 475–490. Springer, 2010.

[Val84] Leslie Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[Val13] Leslie Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World.* Basic Books, Inc., 2013.

[YMHF01] Ling-Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In *Proceedings of SIGMOD*, pages 485–496, 2001.