Interactive Mapping Specification with Exemplar Tuples

Angela Bonifati, Ugo Comignani, Emmanuel Coquery, Romuald Thion

▶ **To cite this version:**

Angela Bonifati, Ugo Comignani, Emmanuel Coquery, Romuald Thion. Interactive Mapping Specification with Exemplar Tuples. ACM International Conference on Management of Data (SIGMOD 2017), May 2017, Chicago, IL, United States. pp.667-682, 10.1145/3035918.3064028 . hal-01548855

HAL Id: hal-01548855
https://hal.science/hal-01548855

Submitted on 26 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

Interactive Mapping Specification with Exemplar Tuples *

Angela Bonifati Ugo Comignani Emmanuel Coquery Romuald Thion

University Lyon 1 & CNRS Liris
firstname.lastname@univ-lyon1.fr

ABSTRACT

While schema mapping specification 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 paper, we present an interactive framework for schema mapping specification suited for non-expert users. The underlying key intuition is to leverage a few exemplar tuples to infer the underlying mappings and iterate the inference process via simple user interactions under the form of boolean queries on the validity of the initial exemplar tuples. The approaches available so far are mainly assuming pairs of complete universal data examples, which can be solely provided by data curation experts, or are limited to poorly expressive mappings.

We present several exploration strategies of the space of all possible mappings that satisfy arbitrary user exemplar tuples. Along the exploration, we challenge the user to retain the mappings that fit the user's requirements at best and to dynamically prune the exploration space, thus reducing the number of user interactions. We prove that after the refinement process, the obtained mappings are correct. We present an extensive experimental analysis devoted to measure the feasibility of our interactive mapping strategies and the inherent quality of the obtained mappings.

1. INTRODUCTION

Schema mappings [17] are declarative specifications, typically in first-order logic, of the semantic relationship between elements of a source schema and a target schema. They constitute key *data programmability primitives*, leading database users to be empowered with programming facilities on top of large shared databases. Mappings are usually specified and tested in enterprise IT and several other domains by data architects, also known as *developers of engineered mappings* [12]. Several paradigms have been proposed to aid data architects to specify engineered mappings. The first paradigm relies on visual specification of mappings using user-friendly graphical interfaces, as in several mapping designers [27, 11]. Such graphical tools help the data architects design a mapping between schemas in a high-level notation. A major drawback of these approaches is that the generation of mappings in a programming language or in a query language from graphical primitives is dependent of the specific tool. As a consequence, the same graphical specification might be translated into different and incomparable declarative mappings by two different tools, leading to inconsistencies. In order to tackle such impedance mismatch, model management operators have been proposed in [12] to provide a general-purpose mapping designer that can be adapted to a wide variety of tools for data programmability. Model management, however, is also suited for expert users. The third paradigm is to generate the desired mappings from representative data examples [5, 6, 22], i.e., a pair of source and target instances, provided by the expert user. However, such data examples are assumed to be solutions of the mapping at hand and representative of all other solutions. Notwithstanding the progress made in mapping specification thanks to the aforementioned approaches, all the above paradigms have in common the fact that they are intended for expert users. Such users are typically acquainted with mapping specification tools and possess complete knowledge of the mapping domains, the formal semantics of mappings and their solution. Ultimately, they are capable of formulating queries or writing customized code.

As also observed in [12], at the other end of the spectrum lies end-users, who find relationships between data and build mapping examples as they go, as in mining heterogeneous data sources, web search, scientific and personal data management. More and more ordinary users are in fact confronted on a daily basis with user-driven data exploration scenarios, such as those exposed by dataspaces [18]. As a consequence, the problem of mapping specification for such classes of users is even more compelling.

To tackle the above problem, in this paper we set forth a novel approach for *Interactive Mapping Specification* (IMS) that bootstraps with *exemplar tuples*, corresponding to a limited number of tuples provided by non-expert users. Such tuples are employed to challenge the user with simple boolean questions, which are intended to drive the inference process of the mapping that the user has in mind and that is unknown beforehand.

*Corresponding author: Ugo Comignani. The research leading to these results has received funding from the ANR under the DataCert grant (ANR-15-CE39-0009) and the Palse Impulsion grant.

(i) Source instance E_S:

T (*Travel*):

IdOutwardFlight	*IdReturnFlight*	*IdAgency*
f0	f1	a0

TA (*TravelAgency*):

IdAgency	*Name*	*Town*
a0	TC	L.A.

A (*Airline*):

IdAirline	*Name*	*Town*
a1	AF	L.A.

F (*Flight*):

IdFlight	*From*	*To*	*IdAirline*
f0	Miami	L.A.	a1
f1	L.A.	Miami	a1

(ii) Canonical mapping:

$$T(\mathrm{idF}_0, \mathrm{idF}_1, \mathrm{idAg}) \wedge TA(\mathrm{idAg}, \mathrm{name}, \mathrm{t}) \wedge F(\mathrm{idF}_0, \mathrm{t}', \mathrm{t}, \mathrm{idAir}) \wedge F(\mathrm{idF}_1, \mathrm{t}, \mathrm{t}', \mathrm{idAir}) \wedge A(\mathrm{idAir}, \mathrm{name}', \mathrm{t}) \rightarrow \exists \mathrm{idC}_0, \mathrm{idC}_1, \mathrm{idC}_2, \mathrm{idC}_3, \mathrm{idC}_4, Co(\mathrm{idC}_0, \mathrm{name}, \mathrm{t}) \wedge Dpt(\mathrm{t}', \mathrm{idF}_0, \mathrm{idC}_1) \wedge Co(\mathrm{idC}_1, \mathrm{name}', \mathrm{t}) \wedge Arr(\mathrm{t}, \mathrm{idF}_0, \mathrm{idC}_2) \wedge Co(\mathrm{idC}_2, \mathrm{name}', \mathrm{t}) \wedge Dpt(\mathrm{t}, \mathrm{idF}_1, \mathrm{idC}_3) \wedge Co(\mathrm{idC}_3, \mathrm{name}', \mathrm{t}) \wedge Arr(\mathrm{t}', \mathrm{idF}_1, \mathrm{idC}_4) \wedge Co(\mathrm{idC}_4, \mathrm{name}', \mathrm{t})$$

(iii) Target instance E_T:

Dpt (*Departure*):

Town	*IdFlight*	*IdCompany*
Miami	f0	P1
L.A.	f1	P2

Arr (*Arrival*):

Town	*IdFlight*	*IdCompany*
L.A.	f0	P3
Miami	f1	P4

Co (*Company*):

IdCompany	*Name*	*Town*
P0	TC	L.A.
P1	AF	L.A.
P2	AF	L.A.
P3	AF	L.A.
P4	AF	L.A.

(iv) Final mapping after refinement:

$$\Sigma_{final} = \{\, TA(\mathrm{idAg}, \mathrm{name}, \mathrm{t}) \rightarrow \exists \mathrm{idC}_0, Co(\mathrm{idC}_0, \mathrm{name}, \mathrm{t});$$
$$F(\mathrm{idF}_0, \mathrm{t}', \mathrm{t}_1, \mathrm{idAir}) \wedge A(\mathrm{idAir}, \mathrm{name}', \mathrm{t}_2) \rightarrow \exists \mathrm{idC}_1, Dpt(\mathrm{t}', \mathrm{idF}_0, \mathrm{idC}_1) \wedge Co(\mathrm{idC}_1, \mathrm{name}', \mathrm{t}_2);$$
$$F(\mathrm{idF}_0, \mathrm{t}', \mathrm{t}_1, \mathrm{idAir}) \wedge A(\mathrm{idAir}, \mathrm{name}', \mathrm{t}_2) \rightarrow \exists \mathrm{idC}_2, Arr(\mathrm{t}_1, \mathrm{idF}_0, \mathrm{idC}_2) \wedge Co(\mathrm{idC}_2, \mathrm{name}', \mathrm{t}_2)\}$$

Figure 1: Running example: exemplar tuples (E_S, E_T) (i) and (iii), resp.; Canonical mapping (ii) and Final mapping (iv).

(IMS) Given exemplar tuples as an input pair (E_S, E_T) provided by a non-expert user and a mapping $\mathcal{M}$ that the user has in mind, the Interactive Mapping Specification *problem is to discover, by means of boolean interactions, a mapping $\mathcal{M}'$ such that (E_S, E_T) satisfy $\mathcal{M}'$ and $\mathcal{M}'$ generalizes $\mathcal{M}$.*

Notice that the user-provided exemplar tuples may turn to be not well chosen or even ambiguous with respect to the mapping $\mathcal{M}$ that the user has in mind. Moreover, exemplar tuples are not supposed to be solutions nor universal solutions of the mapping that needs to be inferred. Whereas a wealth of research on schema mapping understanding and refinement has been conducted in databases [31, 15, 5, 19, 20] since the pioneering work of Clio [25], these approaches assume more sophisticated input (such as an initial mapping to refine and the schemas and schema constraints) and/or more complex user interactions. Although exemplar tuples reminisce data examples [6], they are fundamentally different in that they are not meant to be universal. Furthermore, the mappings we consider in this paper are unrestricted GLAV mappings. We present a detailed comparison with previous work in Section 5, and a comparative analysis with [7] in Section 4.

Query specification has been recognized as challenging for non-expert users and more time-consuming than executing the query itself [23]. We argue that mapping specification is even more arduous for such users, merely because mappings embody semantic relationships between inherently complex queries. Despite many recent efforts on query specification for non-expert users [2, 1, 24, 16, 13], these works are not applicable to mapping specification for non-expert users, which we address in this paper (for more details, we refer the reader to Section 5).

Figure 1 illustrates our running scenario, where a non-expert user needs to establish a mapping between two databases exhibiting travel information. The source database schema is made of four relations, *Travel*, *TravelAgency*, *Airline* and *Flight* (abbreviated respectively as T, TA, A and F). The target database schema contains three relations *Departure*, *Arrival* and *Company* (resp. *Dpt*, *Arr* and *Co*).

The exemplar tuples provided by the user for the source and target databases are reported in the left-hand and right-hand sides of the Figure 1, as E_S and E_T respectively. We can observe that the number of tuples per each table is small: the user is not intended to provide a complete instance but only a small set of representative tuples. We can also easily identify a few inherent ambiguities within the provided exemplar tuples: the constant L.A. represents both the town where the travel agency is located (in relation *TA*, which contains travel agencies information) and the destination of a flight (in the corresponding relation F). Moreover, both the outward and the inward flights are operated by the same airline company, namely a1. If we would consider these exemplar tuples as the ground truth, we would translate them into a *canonical mapping* illustrated in Figure 1 (ii). Such mapping, however, *reflects the ambiguities* of the provided exemplar tuples, by assuming that *all solutions* must have a round trip operated by a unique airline and that the travel agency selling this trip must be located in the same city of the airline headquarters. Thus, from a logical viewpoint, such a mapping is ways *too specific*. Moreover, such mapping can be quite large and unreadable in real-world scenarios, as it embeds all the exemplar tuples altogether. Our *mapping specification process* builds upon end-user exemplar tuples, which can be ambiguous and ill-defined. Hence, it aims at deriving smaller refined and normalized mappings through simple user interactions, in order to obtain more controllable mappings closer to what the user has in mind (illustrated in Figure 1 (iv)). The rest of the paper is devoted to explain such a transformation.

The main contributions of our paper are summarized as follows:

- We define a mapping specification process for non-expert users that bootstraps with exemplar tuples, and works for general GLAV mappings. The user is challenged with boolean questions over even smaller refinement-driven tuples generated from the initial exemplar tuples. The space of possible solutions is represented as an upper semi-lattice, on top of which a dynamic pruning keeps the number of user interactions reasonably low.

- We prove that the generated mappings have *irreducible right-hand sides*. Combined with redundant mapping elimination, this guarantees that the obtained refined mappings are in normal form [21]. Intuitively, normalized mapping are more self-explanatory and understandable for end-users compared to monolithic canonical mappings.
- We prove that the refinement process always produces *a more general mapping* than the canonical mapping. As an example, an illustration of the obtained mapping for our running example is in Figure 1 (iv), which can be confronted with the canonical mapping of Figure 1 (ii).
- We propose and experimentally gauge several exploration strategies of the upper semi-lattice corresponding to the space of possible mappings, and we identify the ones that entail less interactions with the final user. Moreover, we experimentally gauge the effectiveness of our approach, by comparing the sizes of exemplar tuples with the size of universal solutions.

The rest of this paper is organized as follows. Section 2 introduces the notation used in the rest of our paper. Specific background on the mapping generation from exemplar tuples is detailed in Section 3.1. The bulk of our approach is described in Section 3.2 and an extensive experimental study is presented in Section 4. Related work is devoted to Section 5. We conclude the paper in Section 6.

2. PRELIMINARIES

We briefly introduce various concepts from the data exchange framework [17] that we use in this paper. Given two disjoint countably infinite sets of constants $\mathcal{C}$ and variables $\mathcal{V}$, we assume a bijective function $\bar{\theta}$, such that if $\bar{\theta}(\mathrm{x}) = \mathtt{c}$, then $\mathtt{c}_i \in \mathcal{C}$ is the constant associated to the variable $\mathrm{x}_i \in \mathcal{V}$ and $\bar{\theta}^{-1}(\mathtt{c}) = \mathrm{x}$. A *tuple* over a relation R has the form $R(\mathtt{c}_1, \ldots, \mathtt{c}_n)$ where $\mathtt{c}_i \in \mathcal{C}$, while an *atom* has the form $R(\mathrm{x}_1, \ldots, \mathrm{x}_n)$ where $\mathrm{x}_i \in \mathcal{V}$. The bijection $\bar{\theta}$ naturally extends to a bijection between (conjunctions of) atoms and (sets of) tuples.

A *(schema) mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ with $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema disjoint from $\mathbf{S}$, and Σ is a set of *tuple-generating dependency* (tgd for short) over schemas $\mathbf{S}$ and $\mathbf{T}$. A tgd is a first-order logical formula the form $\phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$ where $\overline{x}$ and $\overline{y}$ are vectors of variables, $\overline{x}$ being universally quantified, and where both ϕ and ψ are conjunctions of atoms. In this paper, we only consider source-to-target tgd (s-t tgds for short), in which atoms in ϕ are over relations in $\mathbf{S}$ and atoms in ψ are over relation in $\mathbf{T}$. We consider *GLAV* mappings where a tgd can contain more than one atom in ϕ and in ψ.

An instance E_T over $\mathbf{T}$ is a *solution* for a source instance E_S over $\mathbf{S}$ under a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ iff $(E_S, E_T) \models \Sigma$. A mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *logically entails* a mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$, denoted by $\mathcal{M} \models \mathcal{M}'$, if for every (E_S, E_T) if $(E_S, E_T) \models \Sigma$ then $(E_S, E_T) \models \Sigma'$. Two mappings $\mathcal{M}$ and $\mathcal{M}'$ are *logically equivalent*, denoted by $\mathcal{M} \equiv_l \mathcal{M}'$, if $\mathcal{M} \models \mathcal{M}'$ and $\mathcal{M}' \models \mathcal{M}$. When comparing mappings, we say that $\mathcal{M}$ is *more general* than $\mathcal{M}'$ if $\mathcal{M} \models \mathcal{M}'$. Informally, this means that tgds in $\mathcal{M}$ are triggered more often than those in $\mathcal{M}'$.

Let E_S and $E_S{}'$ be two instances over the same schema. A *homomorphism* from E_S to E_S is a function h from constants in E_S to constants in $E_S{}'$ such that for any tuple $R(c_1, \ldots, c_n)$ in the instance E_S, the tuple $R(h(c_1), \ldots, h(c_n))$ belongs to E_S. An instance E_T is an *universal solution* for the instance E_S under a mapping $\mathcal{M}$ if E_T is a solution for E_S and if for each solution $E_T{}'$ for E_S under $\mathcal{M}$, there exists a homomorphism $h : E_T \rightarrow E_T{}'$ such that $h(\mathtt{c}) = \mathtt{c}$ for every constant $\mathtt{c}$ appearing both in E_T and $E_T{}'$.

It was shown in [17] that the result of chasing E_S with Σ is a universal solution. The application of the *chase procedure*, denoted by $\mathrm{chase}(\Sigma, E_S)$, is as follows: for each tgd $\phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y}) \in \Sigma$, if there exists a substitution μ of $\overline{x}$ such that all atoms in $\phi(\overline{x})$ can be mapped to tuples in E_S, extend this substitution to μ' by picking a fresh new constant for each variable in $\overline{y}$ and finally add all atoms of $\psi(\overline{x}, \overline{y})$ instantiated to tuples with μ' into E_T. Another key result of the literature that we use in this paper is borrowed from [10] and states that $\Sigma \models \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$ *if and only if* there exists a substitution μ' extending an arbitrary μ such that $\mu'(\psi(\overline{x}, \overline{y})) \subseteq \mathrm{chase}(\Sigma, \mu(\phi(\overline{x})))$.

Finally, we borrow two notions from [21]: *split-reduced* mappings and *σ-redundant* mappings. While *split-reduction* breaks a tgd into a logically equivalent set of tgds with right-hand sides having non overlapping existentially quantified variables, *σ-redundancy* encodes the presence of unnecessary tgds. We report formal definitions below. Let $\sigma : \phi(\overline{x}) \rightarrow \exists \overline{y}, \psi(\overline{x}, \overline{y})$ be a tgd. We say that σ is *split-reduced* if there is no pair of tgds $\sigma_1 : \phi_1(\overline{x}) \rightarrow \exists \overline{y}_1, \psi_1(\overline{x}, \overline{y}_1)$ and $\sigma_2 : \phi_2(\overline{x}) \rightarrow \exists \overline{y}_2, \psi(\overline{x}, \overline{y}_2)$ such that $\overline{y}_1 \cap \overline{y}_2 = \emptyset$ and $\{\sigma\} \equiv_l \{\sigma_1; \sigma_2\}$. A mapping $(\mathbf{S}, \mathbf{T}, \Sigma)$ is *split-reduced* if, for all tgd $\sigma \in \Sigma$, σ is *split-reduced*. According to [21], given a mapping $\mathcal{M}$, it is always possible to find a split-reduced mapping $\mathcal{M}'$ that is equivalent to $\mathcal{M}$. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and $\sigma \in \Sigma$ a tgd. We say that $\mathcal{M}$ is *σ-redundant*, w.r.t. logical equivalence, iff $\Sigma \setminus \{\sigma\} \equiv_l \Sigma$. Such equivalence can be tested using the *chase procedure* as a proof procedure for the implication problem by checking whether $\Sigma \setminus \{\sigma\} \models \sigma$.

3. MAPPING REFINEMENT

In this section, we describe the key components of our interactive mapping specification process, as depicted in Figure 2, along with proving its correctness.

3.1 Exemplar tuples and mappings

Exemplar tuples are defined as a pair of source and target instances (E_S, E_T). Given an input pair (E_S, E_T), we build a *canonical mapping* as follows. More precisely, given a pair (E_S, E_T), the canonical mapping associated to (E_S, E_T) is the tgd $\phi \rightarrow \psi$ where $\phi = \bar{\theta}^{-1}(E_S)$ and $\psi = \bar{\theta}^{-1}(E_T)$. Informally, the left-hand side ϕ is constructed from E_S by replacing all tuples in E_S by their atoms counterparts, with the constants being replaced by variables. The right-hand side of the canonical mapping is obtained in a similar fashion.

EXAMPLE 1. *The canonical mapping corresponding to the exemplar tuples of Figure 1 is represented in Figure 1 (ii).*

However, notice that canonical mapping of Example 1 is extremely rigid. For instance, we can observe that tuples in the source relation *TA* are mandatorily needed in order to obtain tuples in the target relation *Arr*. This is due to the fact that the canonical mapping is the most specific mapping obtained from the exemplar tuples: it contains *all* the atoms corresponding to E_S on its left-hand side. Since exemplar tuples are not universal by definition [1], this mapping is far

[1]If exemplar tuples (E_S, E_T) were *universal*, then $(E_S, E_T) \models \sigma$ where σ is the canonical mapping associated to (E_S, E_T).

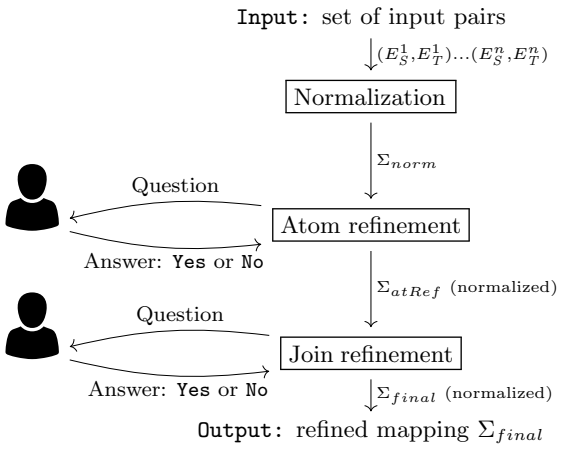


Figure 2: Interactive mapping specification process.

too constrained. The envisioned workaround is to refine the canonical mapping into a less constrained one by leveraging simple user interactions. Intuitively, the refinement of the canonical mapping is done through the following steps: the first is a pre-processing that leads to a normalized mapping, in which the large tgd is divided into an *equivalent* set of smaller ones; the second and the third steps revolve around mapping refinement via user interactions that lets simplify the left-hand sides of the tgds. We devote the rest of this subsection to the first step, while we describe the latter steps in the next subsections.

We define formal criteria that capture the quality of a mapping $\mathcal{M}$ intuitively as follows: each tgd in Σ should have a minimal right-hand side and there should be no spurious tgd in Σ. To that purpose, we rely on the two previously introduced notions, i.e. *split-reduced* mappings and σ-redundant mappings[21].

The splitting of the original mapping into smaller tgds turns out to be convenient for mapping refinement, in that it lets the user focus only on the *necessary* atoms implied in the left-hand sides of each reduced tgd. However, as a side effect of split-reduction, we may get redundant tgds in the set Σ. Such redundant tgds are unnecessary and need to be removed to avoid inquiring the user about useless mappings. Finally, we say that $(\mathbf{S}, \mathbf{T}, \Sigma)$ is *normalized* when each tgd in Σ is *split-reduced* and there is no σ*-redundant* tgd in Σ.

Example 2. *The normalization on the canonical mapping of Figure 1(ii) leads to the following set of tgds* Σ_{norm} *in which all left-hand sides are identical:*

$$\begin{aligned}\phi =& T(idF_0, idF_1, idAg) \wedge TA(idAg, name, t)\\ &\wedge F(idF_0, t', t, idAir) \wedge F(idF_1, t, t', idAir)\\ &\wedge A(idAir, name', t)\end{aligned}$$

$$\begin{aligned}\Sigma_{norm} = \{&\\ &\phi \rightarrow \exists idC_0, Co(idC_0, name, t); && (1)\\ &\phi \rightarrow \exists idC_1, Dpt(t', idF_0, idC_1) \wedge Co(idC_1, name', t); && (2)\\ &\phi \rightarrow \exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t); && (3)\\ &\phi \rightarrow \exists idC_3, Dpt(t, idF_1, idC_3) \wedge Co(idC_3, name', t); && (4)\\ &\phi \rightarrow \exists idC_4, Arr(t', idF_1, idC_4) \wedge Co(idC_4, name', t)\} && (5)\end{aligned}$$

3.2 Refinement of mappings

The previous section has defined the pre-processing step that leads to a normalized canonical mapping. We now introduce the two refinement steps that constitute the core of our proposal. The assumption underlying our approach is that a non-expert user provides a pair of instances (E_S, E_T) which is not universal and interacts via simple boolean answers within the mapping refinement steps. For a given input pair (E_S, E_T), the number of mappings satisfying it may be quite large. Therefore, it is important to provide efficient exploration strategies of the space of mappings in order to reduce the number of questions to ask to the user. In the rest of this section, for the ease of exposition, we assume that the user provides a *unique* pair (E_S, E_T) of exemplar tuples. However, in practice, the user should rather provide a *set* of small instances, say $(E_S^1, E_T^1) \ldots (E_S^n, E_T^n)$. As the questions we ask to the user are centered around one tgd at a time and because we do not assume that each (E_S^j, E_T^j) is universal, our approach is fully applicable to a set of pairs of instances as follows: (i) first compute the canonical mapping σ^i associated to each pair (E_S^j, E_T^j), then (ii) split-reduce each σ^i into Σ^i and finally (iii) remove redundant tgds in $\bigcup_{i=1}^{n} \Sigma^i$.

Two successive steps are applied to each mapping during refinement: the *atom refinement* step and the *join refinement* step. We illustrate such steps in Figure 2, along with the corresponding user interactions required to obtain the final result, i.e., the refined tgds that meet 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 atoms in each tgd as output by the previous 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 first step in Section 3.3 and we postpone the description of the second step to Section 3.4.

In our approach, we use 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 labeled 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. The introduction of such variables would drastically increase the number of mappings to explore and their coverage would entail non-trivial extension of our algorithms, which we leave as future work. Throughout this section, proofs are omitted for space reasons and postponed to Appendix B.

3.3 Atom refinement

As discussed in Section 3.1, the normalization produces a *split-reduced* mapping from the canonical mapping in which each tgd has a large left-hand side, say ϕ. However, some atoms in ϕ may be irrelevant, preventing the triggering of a tgd and causing further ambiguities. Algorithm 1 applies atom refinement on each normalized tgd to alleviate these ambiguities. In the following, we explain its key components and properties.

Algorithm 1 TgdsAtomRefinement(Σ)

Input: A set of tgds Σ to be atom refined.
Output: A set of tgds Σ' where each tgd is atom refined.

```
1: Σ' ← ∅
2: for all σ ∈ Σ do
3:     let σ = φ → ψ
4:     E_cand ← generate set of possibles candidates from φ
5:     E_v ← ∅
6:     repeat
7:         e ← SelectAtomSet(E_cand, E_v)
8:         if AskAtomSetValidity(σ,e)
9:                 ∨(E_v = ∅ ∧ E_cand = ∅) then
10:            add e to E_v
11:            remove supersets of e from E_v
12:            remove e and its supersets from E_cand
13:        else
14:            remove e and its subsets from E_cand
15:        end if
16:    until E_cand = ∅
17:    for all e ∈ E_v do
18:        add the tgd (e → ψ) to Σ'
19:    end for
20: end for
21: return Σ'
```

3.3.1 Semilattice for Atom Refinement

The baseline structure for the atom refinement of a tgd is the upper semilattice $\mathcal{S} = (\mathcal{P}(\phi), \subseteq)$ where $\mathcal{P}(\phi)$ is the set of all subsets of ϕ considered as a set of atoms. For all elements $\mathcal{E}_x$ and $\mathcal{E}_y$ of $\mathcal{P}(\phi)$ the least-upper-bound of the set $\{\mathcal{E}_x, \mathcal{E}_y\}$ is their union.

Example 3. *Considering the tgds in Σ_{norm} of Example 2, each left-hand side is made of an identical conjunction ϕ. The elements of the semilattice we consider are the subsets of the set of atoms $\{T(idF_0,\ idF_1, idAg);\ TA(idAg, name, t);\ F(idF_0, t', t, idAir);\ F(idF_1, t, t', idAir);\ A(idAir, name', t)\}$.*

The atom refinement does not create new existentially quantified variables in the tgds. This restriction takes effect in line 4 of Algorithm 1 where each subset of left-hand side atoms that does not contain the whole set of right-hand side universal variables will be excluded from the set of candidates.

Example 4. *We illustrate the atom refinement on Example 2. As the process stay analogous for each tgd in Σ_{norm}, we focus on the tgd (3) which right-hand side is:*

$$\exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t)$$

The set of universally quantified variables in this conjunction is $\{t, idF_0, name'\}$. A refined tgd needs to contain at least these variables in its left-hand side. The smallest subsets of the set of atoms given in Example 3 for which this assumption is valid are: $\{F(idF_0, t', t, idAir); A(idAir, name', t)\}$ and $\{T(idF_0, idF_1, idAg); A(idAir, name', t)\}$. Each set which is not a superset of one of these two sets is pruned in line 4 of Algorithm 1. The resulting semilattice is shown in Figure 3.

3.3.2 Exploring the semilattice

During the exploration of the space of possible candidates, the user is challenged upon one element of the semilattice at a time, as in line 8 of Algorithm 1. This element can be chosen according to a given exploration strategy, corresponding to the call of SelectAtomSet in line 7. We will experimentally compare four different exploration strategies in Section 4. An important property of the upper semilattice 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.

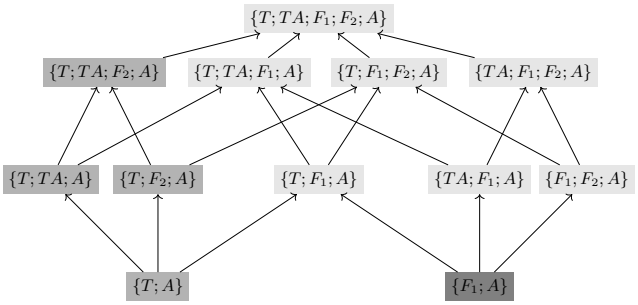


Figure 3: Atom sets semi-lattice on examples 4 and 5. With atoms: $T = T(\text{idF}_0, \text{idF}_1, \text{idAg})$, $A = A(\text{idAir}, \text{name}', \text{t})$, $TA = TA(\text{idAg}, \text{name}, \text{t})$, $F_1 = F(\text{idF}_0, \text{t}', \text{t}, \text{idAir})$, and $F_2 = F(\text{idF}_1, \text{t}, \text{t}', \text{idAir})$.

Example 5. *Assume, for the sake of the example, that we employ a breadth-first bottom-up strategy, starting the exploration of the upper semilattice in Figure 3 at its bottom-up level with $\{F_1; A\}$ and $\{T; A\}$. Following previous Example 4, we are refining tgd (3). The user is asked about the validity of $\{F_1; A\}$ (the dark gray box of Figure 3) with the following question:*

"Are the tuples $F(\texttt{f0}, \texttt{Miami}, \texttt{L.A.}, \texttt{a1})$ and $A(\texttt{a1}, \texttt{AF}, \texttt{L.A.})$ enough to produce $Arr(\texttt{a1}, \texttt{L.A.}, \texttt{f0}, \texttt{P2})$ and $Co(\texttt{P2}, \texttt{AF}, \texttt{L.A.})$?"

Assuming that the user will answer 'Yes' to this question, the supersets of $\{F_1; A\}$ will be pruned (pale gray boxes of Figure 3) and the following tgd will be output by the algorithm:

$$F(idF_0, t', t, idAir) \wedge A(idAir, name', t) \quad (6)$$
$$\rightarrow \exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t)$$

Next, assume now that Algorithm 1 proceeds with $\{T; A\}$ (bottom left gray box of Figure 3). The question asked is:

"Are the tuples $T(\texttt{travel0}, \texttt{f0}, \texttt{f1}, \texttt{a1})$ and $A(\texttt{a1}, \texttt{AF}, \texttt{L.A.})$ enough to produce $Arr(\texttt{A1}, \texttt{L.A.}, \texttt{f0}, \texttt{P2})$ and $Co(\texttt{P2}, \texttt{AF}, \texttt{L.A.})$?"

We can observe that a positive answer implies an ambiguity, namely that the flight company is based in the same town of the destination 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.*

This implies that we need to continue the exploration on the next level of the semilattice, namely on the sets $\{T; TA; A\}$ and $\{T; F_2; A\}$. Assuming that the user does not validate these sets, he will be finally challenged about the last available set $\{T; TA; F_2; A\}$ which he also labels as invalid. In the end, for tgd (3)*, Algorithm 1 will output the single tgd* (6)*.*

We now state 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. The proof as well as an example are found in Appendix B.

LEMMA 1. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a canonical mapping and let Σ' be a mapping obtained from atom refinement of $\mathcal{M}$, then, for all source instances E_S, there exists a morphism μ such that $\mu(\text{chase}(\Sigma, E_S)) \subseteq \text{chase}(\Sigma', E_S)$. By the correctness of the chase procedure, the logical entailment $\Sigma' \models \Sigma$ holds.*

The following Lemma 2 states that the *split-reduction* step is not necessary on the intermediate mappings obtained after the atom refinement step.

LEMMA 2. *Given a normalized canonical mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, application of atom refinement on the tgds in Σ always produces a mapping which is split-reduced.*

3.3.3 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 8 of Algorithm 1. We build on the correspondence between these atoms and the tuples that appear in the source E_S to ask pertinent questions, as those shown in Example 5. The AskAtomSetValidity(σ, e) subroutine that appears in Algorithm 1 constructs a pair $({E_S}^{\sigma,e}, {E_T}^{\sigma,e})$ for each tgd $\sigma = \phi \rightarrow \psi$ by transforming the candidate subset e of its left-hand side ϕ into ${E_S}^{\sigma,e}$, formally ${E_S}^{\sigma,e} = \{\bar{\theta}(a) | a \in e\}$. Then the chase procedure is used to compute ${E_T}^{\sigma,e}$, formally ${E_T}^{\sigma,e} = \text{chase}(\{\sigma\}, {E_S}^{\sigma,e})$.

EXAMPLE 6. *This example focuses on the generation of the exemplar tuples underlying the questions of Example 5 while refining the tgd (3). We are challenging the user about the validity of the set of atoms $e = \{F(idF_0, t', t, idAir); A(idAir, name', t)\}$, which is a subset of the left-hand side of the tgd (3). These atoms are built from the set ${E_S}' = \{F(\texttt{f0}, \texttt{Miami}, \texttt{L.A.}, \texttt{a1}); A(\texttt{a1}, \texttt{AF}, \texttt{L.A.})\}$, a subset the of instance E_S. We want to challenge the user whether the following generalization of the tgd (3) is sufficient:*

$$\sigma = F(idF_0, t', t, idAir) \wedge A(idAir, name', t) \rightarrow \exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t)$$

The chase procedure applies σ on ${E_S}'$ to obtain the following instance ${E_T}'$, from which the first question appearing in Example 5 is derived:

$${E_T}' = \{Arr(\texttt{A1}, \texttt{L.A.}, \texttt{f0}, \texttt{P2}); Co(\texttt{P2}, \texttt{AF}, \texttt{L.A.})\}$$

3.4 Join refinement between variables of a tgd

In relational data, multiple occurrences of the same value do not necessarily imply a semantic relationship between the attributes containing such a value. An example from our running scenario is the occurrence of the constant `L.A.` both as the city where a travel agency is located and as the arrival and departure city of flights booked by that travel agency. However, the canonical mapping imposes such 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. In order to produce the mapping the user has in his mind, we primarily need to distinguish relevant joins from irrelevant ones. This section presets the join refinement step and details the join Algorithm 2 that explores the candidate joins in each tgd by inquiring the user about the validity of such joins.

We briefly recall a few notions on partitions before delving into the actual details and internals of our join refinement algorithm. A partition of a set $\mathcal{W}$ is a set $\mathcal{P}$ of disjoint and non-empty subsets of $\mathcal{V}$ called *blocks*, such that $\bigcup_{b \in \mathcal{P}} b = \mathcal{W}$. The set of all partitions of $\mathcal{W}$ is denoted by $\text{Part}(\mathcal{W})$. Two objects of $\mathcal{W}$ that are in the same block of a partition $\mathcal{P}$ are denoted by $a \equiv_{\mathcal{P}} b$. The set of all partitions of $\mathcal{W}$ form a complete lattice under the partial order $\mathcal{P}_0 \leq \mathcal{P}_1 \Leftrightarrow \forall x, y \in \mathcal{W}, \big(x \equiv_{\mathcal{P}_0} y \Rightarrow x \equiv_{\mathcal{P}_1} y\big)$. This partial order formally captures the intuitive notion of refinement of a partition.

Algorithm 2 TgdsJoinRefinement(Σ)

Input: A set of tgds Σ to be join refined.
Output: A set of tgds Σ' where each tgd is join refined.

```
1: Σ' ← ∅
2: for all σ ∈ Σ do
3:     let σ = φ(x̄) → ∃ȳ, ψ(x̄, ȳ)
4:     Σ_t ← {σ}
5:     for all x ∈ x̄ do
6:         if variable x occurs more than once in φ then
7:             E_explored ← Σ_t
8:             Σ_t ← ∅
9:             for all σ' ∈ E_explored do
10:                Σ_t ← Σ_t ∪ VarJoinsRefinement(σ', x)
11:            end for
12:        end if
13:    end for
14:    Σ' ← Σ' ∪ Σ_t
15: end for
16: return Σ'
```

As in conjunctive queries joins are encoded by multiple occurrences of a variable, we refer to these variables as to *join variables*, refining a join corresponds to replace some occurrences with fresh variables. In Algorithm 2 this replacement of join variables by fresh ones is conducted by the subroutine named VarJoinsRefinement which is detailed in Algorithm 3. The subroutine explores the partitions of these newly introduced variables and questions the user to check if the joins are relevant (some fresh variables are unified) or not (they are kept renamed). A block in the set of all partitions represents the variables to be unified together.

EXAMPLE 7. *Recall tgd (6) from Example 5 obtained after the atom-refined mapping below:*

$$F(idF_0, t', t, idAir) \wedge A(idAir, name', t) \rightarrow \exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t) \quad (6)$$

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 is located, as shown by the multiple occurrences of the join variable t at four different positions. Each occurrence of t is replaced with a fresh variable (namely t_1, t_2, t_3 and t_4) yielding the following candidate tgd:

$$F(idF_0, t', t_1, idAir) \wedge A(idAir, name', t_2) \rightarrow \exists idC_2, Arr(t_3, idF_0, idC_2) \wedge Co(idC_2, name', t_4)$$

In the corresponding semilattice of the set $\{t_1, t_2, t_3, t_4\}$, the supremum corresponds to the case where no refinement is needed, all occurrences being replaced with the original t.

Given a variable x in a tgd $\sigma = \phi \rightarrow \psi$, we consider the set of its occurrences in $\phi \cup \psi$. Since we do not wish to introduce new existentially quantified variables, each variable

occurrence in ψ must be bound to at least one variable occurrence in ϕ. In order to achieve this, we only consider the partitions in which all blocks contain at least one occurrence in ϕ. Those partitions are called *well-formed*

Well-formed partitions are equipped with an upper semilattice 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 unifications encoded by $\mathcal{P}$ are also performed 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 semilattice of occurrences of x.

EXAMPLE 8. *Following Example 7, t_3 and t_4 must be in a partition containing either t_1 or t_2. This means that partitions containing one of the blocks $\{t_3\}$, $\{t_4\}$ or $\{t_3, t_4\}$ are not well-formed and will be excluded.*

Algorithm 3 *Subroutine:*VARJOINSREFINEMENT(σ, x)

Input: A tgd σ.
Input: A variable $\mathrm{x} \in \sigma$ on which the refinement is made.
Output: A set of tgds Σ where each tgd is join refined for variable x.

1: generate from σ a tgd σ' where occurrences of x are renamed with fresh variables and a morphism μ_{orig} such that $\mu_{orig}(\sigma') = \sigma$
2: **let** $\sigma' = \phi' \rightarrow \psi'$
3: $\mathcal{E}_{cand} \leftarrow$ generate set of possibles candidates from σ'
4: $\mathcal{E}_v \leftarrow \emptyset$
5: **repeat**
6: $\quad \mathcal{P} \leftarrow$ SELECTPARTITION$(\mathcal{E}_{cand}, \mathcal{E}_v)$
7: $\quad \sigma'' \leftarrow$ UNIFYVARIABLES$(\sigma', \mathcal{P})$
8: $\quad$ **if** ASKJOINSVALIDITY(σ'')
9: $\quad\quad \vee(\mathcal{E}_v = \emptyset \wedge \mathcal{E}_{cand} = \emptyset)$ **then**
10: $\quad\quad$ add $\mathcal{P}$ to $\mathcal{E}_v$
11: $\quad\quad$ remove upper partitions of $\mathcal{P}$ from $\mathcal{E}_v$
12: $\quad\quad$ remove $\mathcal{P}$ and its upper partitions from $\mathcal{E}_{cand}$
13: $\quad$ **else**
14: $\quad\quad$ remove $\mathcal{P}$ and its lower partitions from $\mathcal{E}_{cand}$
15: $\quad$ **end if**
16: **until** $\mathcal{E}_{cand} = \emptyset$
17: $\Sigma' \leftarrow \emptyset$
18: **for all** $\mathcal{P} \in \mathcal{E}_v$ **do**
19: $\quad \sigma'' \leftarrow$ UNIFYVARIABLES$(\sigma', \mathcal{P})$
20: $\quad$ add σ'' to Σ'
21: **end for**
22: **return** Σ'

Algorithm 2 implements the join refinement by iterating variable refinements on each universal variable of each tgd. 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, one for each refined join variable, we need to combine these refinements. This is done by considering a set Σ_t of tgds to be processed.

Subroutine VARJOINSREFINEMENT(σ, x) explores the upper semilattice of a variable x in a tgd σ, asking questions to the user in order to determine the proper join refinement. In line 1, occurrences of x are replaced with fresh variables yielding a tgd σ' and a morphism μ_{orig} such that $\mu_{orig}(\sigma') = \sigma$. Line 3 initializes the semilattice by excluding malformed partitions as stated above. The SELECTPARTITION subroutine selects a partition in the set of partitions and encodes the specific exploration strategy on top of the semilattice. Any suitable exploration strategy can be plugged in here, as shown in the experimental study presented in Section 4. Function UNIFYVARIABLES$(\sigma, \mathcal{P})$ (lines 7 and 19 of the Algorithm) returns a tgd corresponding to σ where variables from the same block of a partition $\mathcal{P}$ are unified. The user is asked about the validity of this unification in line 8 and the search space and results are pruned according to his answer in lines 11, 12 and 14.

One can easily prove the following Lemma, which is the counterpart of Lemma 1 for join refinement. Hence, Lemma 3 establishes the logical entailment of the join-refined mapping.

LEMMA 3. *Let Σ be a mapping and let Σ' be a mapping obtained from Σ after join refinement, then $\Sigma' \models \Sigma$.*

EXAMPLE 9. *We recall the tgd (6) from Example 5:*

$$F(idF_0, t', t, idAir) \wedge A(idAir, name', t) \quad (6)$$
$$\rightarrow \exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t)$$

Its set of universal variables is $\overline{x} = \{idF_0, t', t, idAir, name'\}$. As Algorithm 2 only considers variables that appears several times (line 6), we only consider t and idAir of $\overline{x}$. Considering first the idAir variable, a renaming of each of its occurrences to $idAir_1$ and $idAir_2$ leads to the following tgd:

$$F(idF_0, t', t, idAir_1) \wedge A(idAir_2, name', t)$$
$$\rightarrow \exists idC_2, Arr(t, idF_0, idC_2) \wedge Co(idC_2, name', t)$$

The semilattice contains two partitions $\{\{idAir_1\}; \{idAir_2\}\}$ and $\{\{idAir_1; idAir_2\}\}$. The user is asked about the validity of $\{\{idAir_1\}; \{idAir_2\}\}$, i.e.*, to have the identifier of an airline company unrelated to its flight. The user will likely answer 'No' to the above question, thus keeping the supremum $\{\{idAir_1; idAir_2\}\}$ of the semilattice valid. Since these join is relevant, the tgd is not modified.*

Then, we consider the t variable. A renaming of each of its occurrences leads to the following tgd previously given in Example 7:

$$F(idF_0, t', t_1, idAir) \wedge A(idAir, name', t_2)$$
$$\rightarrow \exists idC_2, Arr(t_3, idF_0, idC_2) \wedge Co(idC_2, name', t_4)$$

There are five partitions that do not create new existential variables, namely $\{\{t_1; t_3\}; \{t_2; t_4\}\}$, $\{\{t_1; t_4\}; \{t_2; t_3\}\}$, $\{\{t_1; t_3; t_4\}; \{t_2\}\}$, $\{\{t_1\}; \{t_2; t_3; t_4\}\}$ and $\{\{t_1; t_2; t_3; t_4\}\}$. The user is asked about the validity of the candidate partition $\{\{t_1; t_3\}; \{t_2; t_4\}\}$ with the following question:

"Are the tuples $F(\texttt{f0}, \texttt{Miami}, \texttt{L.A.}_1, \texttt{a1})$ and $A(\texttt{a1}, \texttt{AF}, \texttt{L.A.}_2)$ enough to produce $Arr(\texttt{A1}, \texttt{L.A.}_1, \texttt{f0}, \texttt{P2})$ and $Co(\texttt{P2}, \texttt{AF}, \texttt{L.A.}_2)$?"

Since this partition is acceptable for the user, he will probably answer 'Yes'. Therefore, the supremum $\{\{t_1; t_2; t_3; t_4\}\}$ of the semilattice is pruned and the following tgd is added to the output:

$$F(idF_0, t', t_1, idAir) \wedge A(idAir, name', t_2)$$
$$\rightarrow \exists idC_2, Arr(t_1, idF_0, idC_2) \wedge Co(idC_2, name', t_2)$$

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.

The join refinement step preserves the *split-reduction* property of mappings, as formalized in the following Lemma 4. Hence, similarly to the atom refinement step and its associated Lemma 2, a split-reduction step following join refinement is not necessary.

LEMMA 4. *Given a normalized mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, application of join refinement on the tgds in Σ always produces a mapping which is split-reduced.*

However, both the atom and join refinement steps applied to the various tgds do not guarantee a *normalized* mapping, only a *split-reduced* one. As a consequence, similarly to atom refinement, redundant tgds have to be suppressed.

In the join refinement step, the suppression of joins can generate additional tuples in the target instance. For such a reason, similarly to the generation of questions in the atom refinement step, the source instance is again chased to generate such additional tuples [2]. Similarly to the subroutine described in Section 3.3.3 for atom refinement, the AskJoinsValidity subroutine that appears in Algorithm 2 constructs a pair $(E_S{}^{\sigma}, E_T{}^{\sigma})$ by instantiating the left-hand side of a candidate tgd σ to obtain a source instance $E_S{}^{\sigma}$ and then chasing it to build $E_T{}^{\sigma}$.

EXAMPLE 10. *We illustrate the questions asked to the user in Example 9. We challenge the user on the validity of the partition $\mathcal{P} = \{\{t_1; t_3\}; \{t_2; t_4\}\}$ in the following tgd:*

$$\sigma = F(idF_0, t', t_1, idAir) \wedge A(idAir, name', t_2) \rightarrow \exists idC_2, Arr(t_1, idF_0, idC_2) \wedge Co(idC_2, name', t_2)$$

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

$$E_S{}^{\sigma} = \{F(\texttt{f0}, \texttt{Miami}, \texttt{L.A.}_1, \texttt{a1}); A(\texttt{a1}, \texttt{AF}, \texttt{L.A.}_2)\}$$

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

$$E_T{}^{\sigma} = \{Arr(\texttt{A1}, \texttt{L.A.}_1, \texttt{f0}, \texttt{P2}); Co(\texttt{P2}, \texttt{AF}, \texttt{L.A.}_2)\}$$

Those exemplar tuples are finally rewritten into questions as shown in Example 9.

3.5 Final mapping

The pre-processing, the atom and join refinement steps being defined, we are able to summarize the correctness of our approach with the following Theorem 1.

THEOREM 1. *Let (E_S, E_T) be a pair of exemplar tuples sets. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be the canonical mapping corresponding to these exemplar tuples obtained after pre-processing. Let $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ be the refined mapping produced by our framework. Then, for all instances E_S over $\mathbf{S}$, there exists a morphism μ such that $\mu(\text{chase}(\Sigma, E_S)) \subseteq \text{chase}(\Sigma', E_S)$, that is, the refined mapping is a generalization of the canonical one: $\mathcal{M}' \models \mathcal{M}$.*

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

If the user provides universal exemplar tuples at first (which correspond to the data examples used in [7]), that is an ideal pair (E_S, E_T) without extraneous atoms nor irrelevant joins, the mapping he has in his mind is truly the canonical mapping. In this case, the following Theorem 2 shows that the canonical mapping can be obtained by consistently answering 'Yes' only when the questions asked to the user are built from the supremums of the semilattices in the atom and join refinement steps. In other words, Theorem 2 states a form of completeness of our approach: if there is an ideal answer, the refinement process will not miss it.

THEOREM 2. *The refined mapping obtained when user always validates the supremums of the explored semilattices is equivalent to the canonical mapping.*

4. EXPERIMENTS

Our experimental study has three main objectives: (i) to study the effectiveness of interactivity under different exploration strategies of the search space, (ii) to evaluate the benefit of using exemplar tuples with respect to universal solutions for mapping refinement, and (iii) to provide a comparative analysis with [7].

Experimental settings. We have implemented our framework using OCaml 4.03 on a 2.6GHz 4-core, 16Gb laptop running Fedora 24. We have borrowed mappings from seven real integration scenarios of the `iBench` benchmark [9]. The left part of Table 1 reports the *size* of each considered mapping scenario as the total number of tgds ($|\Sigma|$) and the average number of occurrences of their variables ($\bar{N}$) defined as $\bar{N} = \Sigma_{v \in V} N_v / |V|$, with V being the set of distinct variables and N_v the number of occurrences of each v variable within the tgds. Since there exists a bijection between variables and constants, $\bar{N}$ also stands for the average number of occurrences of constants per instance in the exemplar tuples.

Methodology. In all experiments, we consider the `iBench` mapping scenarios 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 $(E_S{}^{\sigma}, E_T{}^{\sigma})$, $E_S{}^{\sigma}$ ($E_T{}^{\sigma}$, resp.) being generated by replacing each atom in ϕ (ψ, 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 $E_\Sigma = \{(E_S{}^{\sigma_1}, E_T{}^{\sigma_1}), \ldots, (E_S{}^{\sigma_n}, E_T{}^{\sigma_n})\}$.

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_Σ are degraded . The degradation procedure is meant to reproduce users' common mistakes while specifying exemplar tuples.

The first degradation is performed on atoms. It is parametrized by the total number of extraneous tuples added to E_Σ, with the constraint that at most one tuple is added to each individual instance $E_S{}^{\sigma_i}$. An extraneous tuple is generated by randomly choosing a source instance $E_S{}^{\sigma_i}$, 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. It is parametrized by the total number of unifications between constants in the set E_Σ, with the constraint that at most one unification is applied within each pair of instances $(E_S{}^{\sigma_i}, E_T{}^{\sigma_i})$.

An extraneous unification is produced by choosing at random two constants that appear in ${E_S}^{\sigma_i}$ and replacing one with the other in all its occurrences in ${E_S}^{\sigma_i}$ and in ${E_T}^{\sigma_i}$.

EXAMPLE 11. *By applying the degradation procedure on the tgd σ from Example 10, the following exemplar tuples may be yielded $({E_S}'^{\sigma}, {E_T}'^{\sigma})$. An extraneous F atom is added (first degradation) and where* `Miami` *and* $\texttt{L.A.}_1$ *are unified (second degradation), the degradations being underlined:*

$$
\begin{aligned}
{E_S}'^{\sigma} &= \{F(\texttt{f0}, \underline{\texttt{L.A.}_1}, \texttt{L.A.}_1, \texttt{a1}); \underline{F(\texttt{f0}, \texttt{L.A.}_1, \texttt{L.A.}_1, \texttt{a1})};\\
&\qquad A(\texttt{a1}, \texttt{AF}, \texttt{L.A.}_2)\}\\
{E_T}'^{\sigma} &= \{Arr(\texttt{A1}, \texttt{L.A.}_1, \texttt{f0}, \texttt{P2}); Co(\texttt{P2}, \texttt{AF}, \texttt{L.A.}_2)\}
\end{aligned}
$$

In our experimental study, we have deteriorated each initial set of examples E_Σ by respectively adding 0, 2, 5, 8 or 10 tuples without joins or by unifying 0, 2, 5, 8 or 10 variables without added tuples. 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 Σ from the scenario. In order to simulate the user answer, E_S is chased to obtain ${E_T}'$. 'Yes' is produced as an answer if there exists a substitution μ from E_T into ${E_T}'$ such that $\mu(E_T) \subseteq {E_T}'$, otherwise 'No' is returned.

Impact of Mapping Refinement. In the first experiment, we gauge the effectiveness of our interactive approach with four exploration strategies. The BUBF (Bottom-Up Breadth-First) strategy corresponds to the classical Apriori algorithm for lattice exploration [3], its Bottom-Up Depth-First variation being BUDF. TDBF and TDDF are the respectively Breadth-First and Depth-First variations of the Top-Down exploration of the lattice.

For atom refinement and join refinement, Table 1 presents the average number of questions per tgd asked to the user ($\bar{x}$), and the maximum number of questions per tgd observed (x_m) for a scenario.

The maximum average number of questions for atom refinement in our experiments is about 6.7 questions per tgd. *Even if TDBF and TDDF strategies seem slightly more efficient than the other ones, all results have the same order of magnitude and do not allow to make a clear choice of the strategy. However, the situation is reverted for the experiments on join refinement, which clearly show that only the TDBF strategy exhibits an acceptable number of questions per tgd* (below 17). Indeed, the remaining strategies can lead to more than one hundred questions per tgd.

This performance analysis on the employed strategies let us conclude that in the remainder of our experimental assessment, *we can only focus on two out of the four strategies, namely TDBF and BUBF*. We thus decided to compare the former, which is well-behaved by keeping low the number of questions, with the latter, which implements the classical Apriori algorithm [3].

Figure 4 summarizes our results on atom refinement for the BUBF and TDBF strategies. The x axis of each scenario is $\bar{N}$, which grows as the degradation introduces more tuples (ranging from 0 to 10). The y axis corresponds to the total number of user interactions divided by the number of tgds in the scenario. The latter provides an estimate of the average number of questions needed to recover the expected tgd from exemplar tuples that contain errors.

In all scenarios but one, we can observe a strong linear correlation between the number of interactions and $\bar{N}$. Intuitively, this illustrates the fact that the search space of Algorithm 1 gracefully grows as the number of occurrences of constants grows. This trend can be observed for all scenarios with the exception of scenario `a1-to-a2`, which doesn't confirm the aforementioned strong correlation. We impute this different behavior to the fact that that the majority of the atoms of this scenario have a comparably lower arity, which leads to a smaller increase of the number of variable occurrences during the degradation process.

Figure 5 (a) and (b) summarize our experiments on join refinement on the two strategies BUBF and TDBF, respectively. For each strategy, a linear correlation between the number of interactions and $\bar{N}$ can be observed on scenarios `a1-to-a2`, `amalgam2` and `dblp-amalgam`. *The relatively small number of required user interactions shows the effectiveness of questioning for join refinement, with comparable results to the ones obtained for atom refinement. Also, with these scenarios, we can observe the superiority of TDBF over BUBF.*

For the remaining scenarios, we can observe on one hand a higher number of user interactions and, on the other hand, the presence of outliers [3]. *This shows a higher burden of join refinement on the final end user compared to atom refinement.*

As an example, we can notice that the mapping scenario `SDB1-to-SDB3` is the most problematic since it inherently embodies an extreme case for Algorithm 2. Through the bias of this scenario, we can observe that the search space of join refinement is quite sensitive to the number of partitions to be explored in Algorithm 2. Thus, the number of partitions (and then the number of joins on the same variable) heavily depends on the involved scenario.

As also highlighted previously, in this extreme case it can be seen that TDBF algorithm (Figure 5b) allows to obtain a considerably lower number of questions than BUBF algorithm (Figure 5a), leading to an acceptable number of questions (less than 20 for the worst case, when BUBF algorithm leads to more than one hundred questions per example).

Finally, we measured the running time of TDBF and BUBF as the time between two questions (i.e., the sum of the time for lattice exploration between two questions and the time to generate a new question). Notice that the total time does not include the time to answer the question itself, which is primarily user dependent. In all experiments such runtime steadily stays below 26 ms per question, with an average of 3.3 ms across all questions. This confirms that *our approach is fast enough for an interactive experience as a real user would not have to wait between questions.*

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 [7] for the mapping inference process. For each scenario, we apply the chase to all the source instances E_S^i to obtain chase$(\mathcal{M}, E_S^i)$. This lets us compute the number of universal exemplar tuples, which we compare with the number of targets (non-universal) exemplar tuples used in our approach.

[3]In Figure 5, to better illustrate the intervals of the maximum values of the number of occurrences variables and thus to better describe the outliers, we have grouped such values as shown in the upper right legend.

Scenarios			Atom refinement								Join refinement							
			TDBF		TDDF		BUBF		BUDF		TDBF		TDDF		BUBF		BUDF	
name	$\lvert\Sigma\rvert$	$\bar{N}$	$\bar{x}$	x_m	$\bar{x}$	x_m	$\bar{x}$	x_m	$\bar{x}$	x_m	$\bar{x}$	x_m	$\bar{x}$	x_m	$\bar{x}$	x_m	$\bar{x}$	x_m
a1-to-a2	8	2.5	1.6	2.4	2.2	4.1	2.2	4.1	2.0	2.9	4.1	6.5	3.8	6.5	3.9	6.5	3.9	8.0
amalgam2	71	1.3	0.4	0.5	0.3	0.6	0.3	0.6	1.0	4.1	0.3	0.5	0.4	0.6	0.4	0.6	1.9	7.7
dblp-amalgam	10	1.4	1.0	1.9	0.9	1.9	1.0	2.2	0.7	2.1	1.6	3.5	1.7	3.8	1.8	3.9	1.6	4.3
GUS-to-BIOSQL	8	1.5	1.2	2.3	1.7	3.1	2.0	3.7	1.9	3.8	2.6	5.0	2.8	5.9	2.8	5.6	2.5	5.9
SDB1-to-SDB2	10	1.5	1.0	3.2	2.0	2.9	2.3	3.3	2.5	4.4	3.9	6.1	6.5	50.2	6.5	50.2	4.3	13.8
SDB1-to-SDB3	11	1.5	2.3	3.2	3.1	5.4	4.0	6.6	4.1	6.7	5.3	16.1	11.1	135.9	11.4	136.2	13.1	136.8
SDB2-to-SDB3	9	2.1	0.7	1.4	0.8	1.9	0.9	1.9	1.0	1.7	3.2	6.2	5.6	43.6	5.6	43.6	3.8	8.4

Table 1: Scenarios characteristics; average ($\bar{x}$) and maximum (x_m) number of questions per tgd for each dataset refinement.

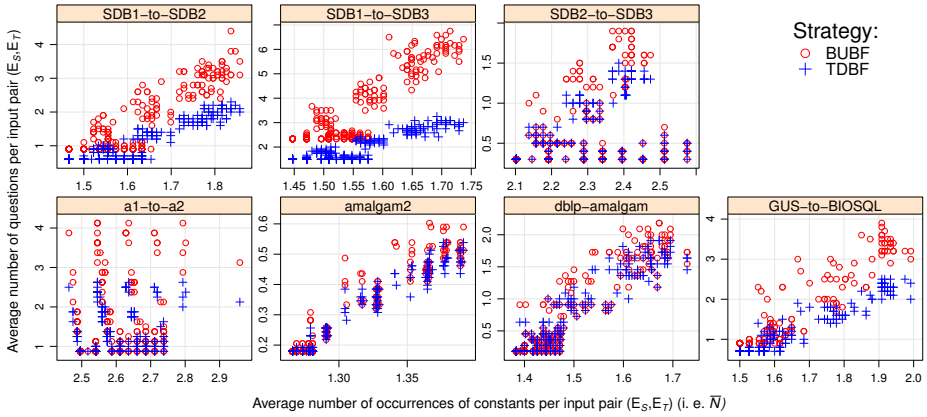
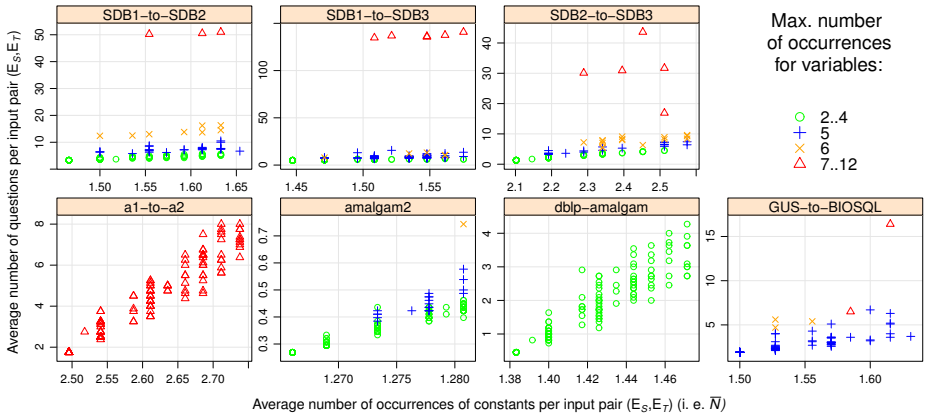
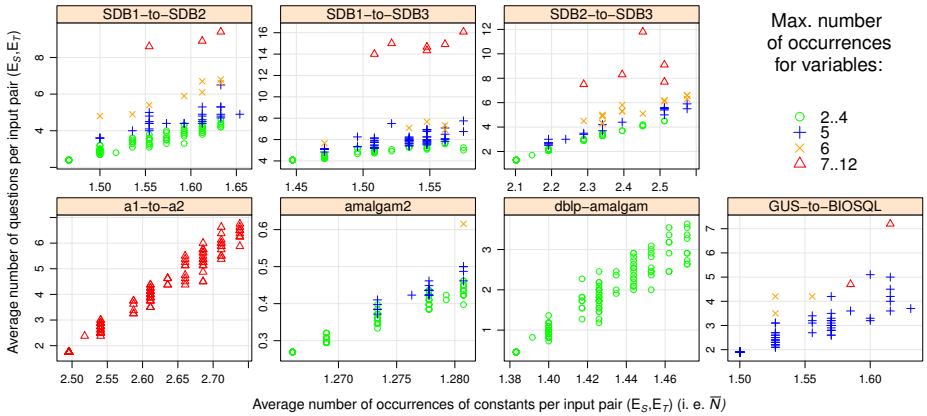


Figure 4: Average number of questions per pair (E_S, E_T) versus $\bar{N}$ on atom refinement using breadth-first strategies.



(a) Using the bottom-up breadth-first (BUBF) strategy



(b) Using the top-down breadth-first (TDBF) strategy

Figure 5: Average number of questions per pair (E_S, E_T) versus $\bar{N}$ on join refinement using breadth-first strategies.

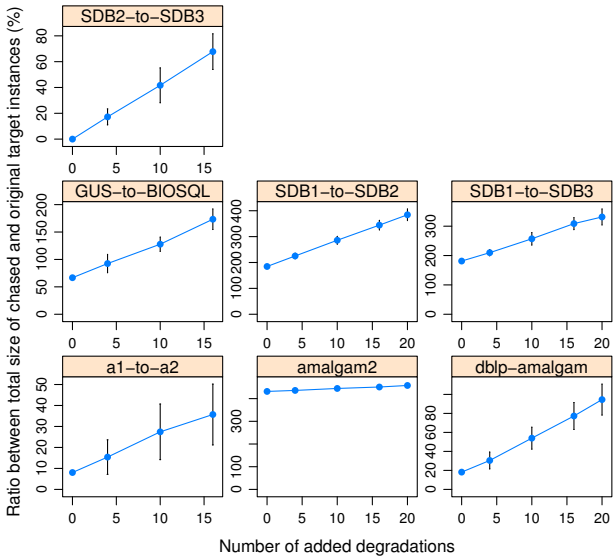


Figure 6: Growth of the ratio r wrt. number of degradations.

Concretely, for exemplar tuples $\{(E_S^1, E_T^1); \ldots; (E_S^n, E_T^n)\}$ and the corresponding expected mapping $\mathcal{M}$, we calculate the ratio $r = \frac{\Sigma_{i=1}^{n} |\,\text{chase}(\mathcal{M}, E_S^i)|}{\Sigma_{i=1}^{n} |E_T^i|} - 1$ of additional atoms in the generated solution. 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 6, we present the results where an equal number of atom and join degradations are used. 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 30% to 458% 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 less sensitive to the variation of the number of degradations is amalgam2, which is also the 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.*

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

Scenarios	Number of extraneous atoms added				
	0	2	5	8	10
SDB2-to-SDB3	0	12.4	27.0	36.9	-
SDB1-to-SDB2	0	11.1	23.8	33.3	38.5
SDB1-to-SDB3	0	7.3	16.2	23.6	28.0
dblp-amalgam	0	12.2	25.3	35.5	40.7
GUS-to-BIOSQL	0	11.7	25.8	35.7	-
a1-to-a2	0	8.3	18.5	26.6	-
amalgam2	0	3.1	7.5	10.9	12.8
Average	0	9.5	20.6	28.9	30

Table 2: Relative difference (in percent) between EIRENE and our system.

GLAV generation performed in EIRENE[4]. 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 bootstrap 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 [21]. 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 a 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 2. 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: EIRENE's canonical mapping is about 20% larger on average (across all scenarios) when 5 such atoms are introduced, and goes up to 30% on average with 10 atoms. *Hence, our mappings are noticeably simpler than* EIRENE*'s ones. Such an improvement is both beneficial for the readability of mappings as well as for their efficiency because spurious atoms are eliminated.*

5. RELATED WORK

A pioneering work on the usage of data examples in mapping understanding and refinement [31] relies on Clio's [25] schema correspondences as specified in a graphical user interface. By leveraging such correspondences, Yan et al. [31] propose alternative data associations among relevant source instances leading to construct mappings in an incremental fashion with the intervention of the mapping designer. The dichotomy between the expected user instance and the generated instance has been further investigated in Routes [15]. The input required by Routes consists of both a source instance and a mapping that the user readily intends to debug. The user then builds test cases for the mapping at hand by probing values in the target instance, and the system returns a provenance trace to explain how and why the probed values are computed. This approach closely resembles testing as done for software development. By opposite, our method requires as inputs a source and target exemplar tuples and

[4]For the sake of fairness, EIRENE's canonical GLAV are *split-reduced* and *σ-redundant* tgds are suppressed.

no prior mapping connecting them. The final objective of our approach, which especially targets users unfamiliar with schema mappings, is to build the mapping that the user had in mind via simple boolean user interactions. To draw a comparison with software development, our method generates a specification (i.e. a mapping expressed in first-order logic) starting solely from supplied unit test cases (i.e. small exemplar tuples).

As in Yan et al. [31], Muse [5] leverages data examples to differentiate between alternative mapping specifications of the designer and drives the mapping design process based on the designer's actions. However, the techniques proposed in [5] are more sophisticated than the ones in [31], in that they address the problem of the grouping semantics of mappings and their alternative semantics in case of ambiguity. Muse also poses a number of yes/no questions to the designer to clarify the grouping semantics. However, the number of questions are 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. `TRAMP` [19] and `Vagabond` [20] focus on the understandability of user errors in mappings by using provenance. However, explanations returned by `Vagabond` are to be interpreted by users who are familiar with the mapping language and its underlying semantics.

The use of data examples as evaluation tools has begun in [4, 29], which investigated the possibility of uniquely characterizing a schema mapping by means of a set of data examples. Hence, such unique characterization, up to logical equivalence of the obtained mappings, 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 [4, 29]. As a negative result, it was shown in [4] that already simple s-t tgds mappings, such as 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 the impossibility of uniquely characterizing GLAV mappings in real settings, [6, 7] made the choice of being less specific. Precisely, they decided to characterize, for a given schema mapping, the set of valid "non-equivalent" mappings with respect to the class of GLAV. To achieve that, they rely on the notion of "most general mapping". It was shown that, given a schema mapping problem, a most general mapping always exists in the class of GLAV mappings if there exists at least one valid mapping for the considered problem [6]. In `EIRENE` [7], 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 have shown in Section 4, universal target instances tend to be *significantly larger* than our exemplar tuples. The only previous work targeting non-expert users is MWeaver [26], 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.

All the aforementioned approaches are meant to produce the best exact mapping. However, one can use data examples to produce approximate mappings. Gottlob and Senellart propose a cost-based method to estimate the best approximate mapping given a set of possible repairs of the initial mapping [22]. The cost function takes account for the length of the generated tgds and the number of *repairs* that are needed to obtain a tgd that fully explain the instance E_T. Approximation of schema mappings has been considered recently in [28] by considering more expressive fragments of GLAV and GAV.

Cate et al. [14] show how computational learning (i.e., the *exact learning model* introduced by D. Angluin [8] and the Probably Approximately Correct model introduced by L. Valiant [30]) can be used to infer mappings from data examples. Their analysis is restricted to GAV schema mappings.

Besides mapping specification and learning, researchers have investigated the problem of inferring relational queries [2, 1, 24, 13]. The work in [2, 1] focuses 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 [13], 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, [24] presents the exemplar query evaluation paradigm, which relies 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.

6. CONCLUSIONS

We have addressed the problem of interactive schema mapping inference starting from arbitrary sets of exemplar tuples, as provided by non-expert users. We have shown that simplification of the mappings is possible by alternating normalization and refinement steps, the latter under the form of simple boolean questions.

This paper lays the foundations of an envisaged practical framework. Much work is left to be done in order to make mapping specification an activity for non-expert users, for instance by adding features like error acceptance in user responses. Also, after such improvements, a user study with a carefully chosen statistical protocol is needed in order to evaluate the practicality of the approach and capture the user behavior when using our system. A further direction of future work is devoted to enhance the lattice exploration, for instance by leveraging machine learning methods.

APPENDIX

A. REFERENCES

[1] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of PODS*, pages 49–60, 2013.

[2] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Playful query specification with dataplay. *PVLDB*, 5(12):1938–1941, 2012.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94*, pages 487–499, 1994.

[4] B. Alexe, B. T. Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *TODS*, 36(4):23:1–23:48, 2011.

[5] B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. Muse: Mapping understanding and design by example. In *Proceedings of the ICDE*, pages 10–19, 2008.

[6] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Designing and refining schema mappings via data examples. In *Proceedings of SIGMOD*, pages 133–144, 2011.

[7] B. Alexe, B. Ten Cate, P. G. Kolaitis, and W.-C. Tan. Eirene: Interactive design and refinement of schema mappings via data examples. *Proceedings of VLDB*, 2011.

[8] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[9] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. The ibench integration metadata generator. *Proceedings of VLDB*, 9(3):108–119, 2015.

[10] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *JACM*, 31(4):718–741, 1984.

[11] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.

[12] P. A. Bernstein and S. Melnik. Model management 2.0: Manipulating richer mappings. In *SIGMOD*, 2007.

[13] A. Bonifati, R. Ciucanu, and S. Staworko. Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24:1–24:38, Jan. 2016.

[14] B. T. Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. *ACM TODS*, 38(4):28, 2013.

[15] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *Proceedings of the 32nd international conference on Very large data bases*, pages 79–90. VLDB Endowment, 2006.

[16] G. I. Diaz, M. Arenas, and M. Benedikt. Sparqlbye: Querying RDF data by example. *PVLDB*, 9(13):1533–1536, 2016.

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[18] M. J. Franklin, A. Y. Halevy, and D. Maier. A first tutorial on dataspaces. *PVLDB*, 1(2):1516–1517, 2008.

[19] B. Glavic, G. Alonso, R. J. Miller, and L. M. Haas. Tramp: Understanding the behavior of schema mappings through provenance. *Proc. VLDB Endow.*, 3(1-2):1314–1325, Sept. 2010.

[20] B. Glavic, J. Du, R. J. Miller, G. Alonso, and L. M. Haas. Debugging data exchange with vagabond. *PVLDB*, 4(12):1383–1386, 2011.

[21] G. Gottlob, R. Pichler, and V. Savenkov. Normalization and optimization of schema mappings. *VLDB J.*, 20(2):277–302, 2011.

[22] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *Journal of the ACM (JACM)*, 57(2):6, 2010.

[23] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *Proceedings of SIGMOD*, pages 13–24, 2007.

[24] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.

[25] L. Popa, Y. Velegrakis, M. A. Hernández, R. J. Miller, and R. Fagin. Translating web data. In *Proceedings of VLDB*, pages 598–609, 2002.

[26] L. Qian, M. J. Cafarella, and H. Jagadish. Sample-driven schema mapping. In *Proceedings of SIGMOD*, pages 73–84. ACM, 2012.

[27] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, pages 146–171, 2005.

[28] B. ten Cate, P. G. Kolaitis, K. Qian, and W.-C. Tan. Approximation algorithms for schema-mapping discovery from data examples. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, page 24, 2015.

[29] B. Ten Cate, P. G. Kolaitis, and W.-C. Tan. Database constraints and homomorphism dualities. In *CP*. Springer, 2010.

[30] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, Nov. 1984.

[31] L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *Proceedings of SIGMOD*, pages 485–496, 2001.

B. PROOFS

We report in this section the proofs omitted in the bulk of the paper, along with an additional auxiliary lemma. For the formal development, we assume a countably infinite set $\mathcal{N}$ of labeled nulls.

LEMMA 1. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a canonical mapping and let Σ' be a mapping obtained from atom refinement of $\mathcal{M}$, then, for all source instances E_S, there exists a morphism μ such that $\mu(\text{chase}(\Sigma, E_S)) \subseteq \text{chase}(\Sigma', E_S)$. By the correctness of the chase procedure, the logical entailment $\Sigma' \models \Sigma$ 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 σ. Then, ϕ' must correspond to a node in the semilattice, such that $\phi' \subseteq \phi$. We introduce an function $ref : \mathcal{M} \rightarrow \mathcal{M}'$ that associate to each σ in $\mathcal{M}$ one of its refinements (that may be arbitrarily chosen if there are several such tgds in $\mathcal{M}'$).

Let ν 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 ψ is mapped by ν to a fresh labeled

null, which means that is ν^{-1} is defined for such values. Since $\phi' \subseteq \phi$, $\nu(\phi') \subseteq E_S$. Therefore, there exists an instantiation mapping ν' 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 ϕ', $\nu'(\text{x}) = \nu(\text{x})$. However, ν' and ν can differ in two ways: the domain of ν' can be smaller than the domain of ν and the labeled nulls that are assigned to existential variables in ψ can be different because the chase generate fresh null values at each tgd application. By construction of $\mathcal{E}_p$ in Algorithm 1, any variable x in ψ is either an existential variable or a universal variable in ϕ'. Thus, every variable x in ψ is either mapped to fresh null values by ν and ν' or, alternatively, $\nu(\text{x}) = \nu'(\text{x})$. We introduce μ_ν 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 ϕ' 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 ν_1 and ν_2 used in $\text{chase}(\mathcal{M}, E_S)$ and their associated morphisms μ_{ν_1} and μ_{ν_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 than it is the image of an existential variable by ν_1, which means that it cannot be the image of any variable by ν_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 Λ of all instantiation mappings ν 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 ν used in $\text{chase}(\mathcal{M}, E_S)$ and a tgd $\phi \to \psi$ 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 $\mu(t) \in \text{chase}(\mathcal{M}', E_S)$. □

The following Example 12 shows that the previous property would not hold if Algorithm 1 is allowed to create new existential variables.

EXAMPLE 12. *Given a pair (E_S, E_T) such that $E_S = \{R(\mathtt{x}, \mathtt{y}); S(\mathtt{z})\}$ and $E_T = \{T(\mathtt{x})\}$. The canonical mapping corresponding to (E_S, E_T) is $\Sigma = \{R(x, y) \wedge S(z) \to T(x)\}$. Suppose that atom refinement allows the creation of existentially quantified variables. By applying this refinement on Σ, we may obtain the mapping $\Sigma' = \{S(z) \to \exists x, T(x)\}$. Chasing E_S under Σ and Σ' will lead to following results:*

$$\text{chase}(E_S, \Sigma) = \{T(\mathtt{x})\} \qquad \text{chase}(E_S, \Sigma') = \{T(\mathtt{x1})\}$$

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

LEMMA 2. *Given a normalized canonical mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, application of atom refinement step on the tgds in Σ always produce a mapping which is split-reduced.*

PROOF (SKETCH). 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 also *split-reduced*. □

EXAMPLE 13. *Given a normalized canonical mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, application of atom refinement step on the tgds in Σ does not allow to avoid σ-redundant tgds.*

We can exhibit a counter-example of σ-redundancy generation, given a normalized set of tgds Σ as follows:

$$\Sigma = \{R(u, v) \wedge R(x, y) \wedge S(y, z) \to T(u, v);\\ R(u, v) \wedge R(x, y) \wedge S(y, z) \to T(x, y)\}$$

Applying atom refinement on Σ allows to produce the following refined set of tgds Σ', where tgds are σ-redundant:

$$\Sigma' = \{R(u, v) \to T(u, v); R(x, y) \to T(x, y)\}$$

LEMMA 3. *Let Σ be a mapping and let Σ' be a mapping obtained from Σ after join refinement, then $\Sigma' \models \Sigma$.*

PROOF. Let $\sigma = \phi \to \psi$ be a tgd and x be a universal variable in σ. First, we prove that for all $\sigma'' \in \text{VARJOINSREFINEMENT}(\sigma, \text{x})$, $\sigma'' \models \sigma$.

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

In Algorithm 2, Σ_t contains tgds that are either elements of Σ or obtained by applying VARREFINEMENT to previous elements of Σ_t. Because of line 9, VARREFINEMENT always returns at least one tgd. Thus, for each initial tgd σ in Σ, there is a tgd σ' in Σ' coming from successive calls of VARREFINEMENT starting with σ. By transitivity of $\models$ we deduce that $\sigma' \models \sigma$. Thus, $\Sigma' \models \sigma$. Since this holds for all tgds in Σ, we conclude that $\Sigma' \models \Sigma$. □

LEMMA 4. *Given a normalized mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, application of join refinement on the tgds in Σ always produces a mapping which is split-reduced.*

PROOF. By definition, if a tgd σ is *split-reduced* and contain 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 Σ is normalized, each of its tgd is *split-reduced*. Since for each tgd in Σ, the application of the join refinement step results in new tgds that are also *split-reduced*. Thus, the set Σ' of all these refined tgds is a *split-reduced* mapping. □

THEOREM 1. *Let (E_S, E_T) be a pair of exemplar tuples sets. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be the canonical mapping corresponding to these exemplar tuples obtained after pre-processing. Let $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ be the refined mapping produced by our framework. Then, for all instances E_S over $\mathbf{S}$, there exists a morphism μ such that $\mu(\text{chase}(\Sigma, E_S)) \subseteq \text{chase}(\Sigma', E_S)$, that is, the refined mapping is a generalization of the canonical one: $\mathcal{M}' \models \mathcal{M}$.*

PROOF (SKETCH). This theorem follows from Lemma 1 and Lemma 3. □

THEOREM 2. *The refined mapping obtained when user always validates the supremums of the explored semilattices is equivalent to the canonical mapping.*

Scenarios	BUBF		BUDF		TDBF		TDDF	
	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
a1-to-a2	0.38	0.06	0.43	0.06	0.43	0.07	0.50	0.13
amalgam2	19.19	3.41	18.23	5.06	20.55	3.35	17.70	4.84
dblp-amalgam	0.55	0.25	0.54	0.25	0.71	0.34	0.63	0.31
GUS-to-BIOSQL	0.29	0.04	0.30	0.08	0.40	0.11	0.37	0.15
SDB1-to-SDB2	0.15	0.02	0.30	0.84	0.23	0.07	0.35	0.84
SDB1-to-SDB3	0.16	0.02	1.98	11.81	0.26	0.10	2.30	12.51
SDB2-to-SDB3	0.18	0.05	0.52	1.70	0.24	0.08	0.76	2.21

Table 3: Average executions times to produce a question ($\bar{x}$) in milliseconds and their standard deviation (s).

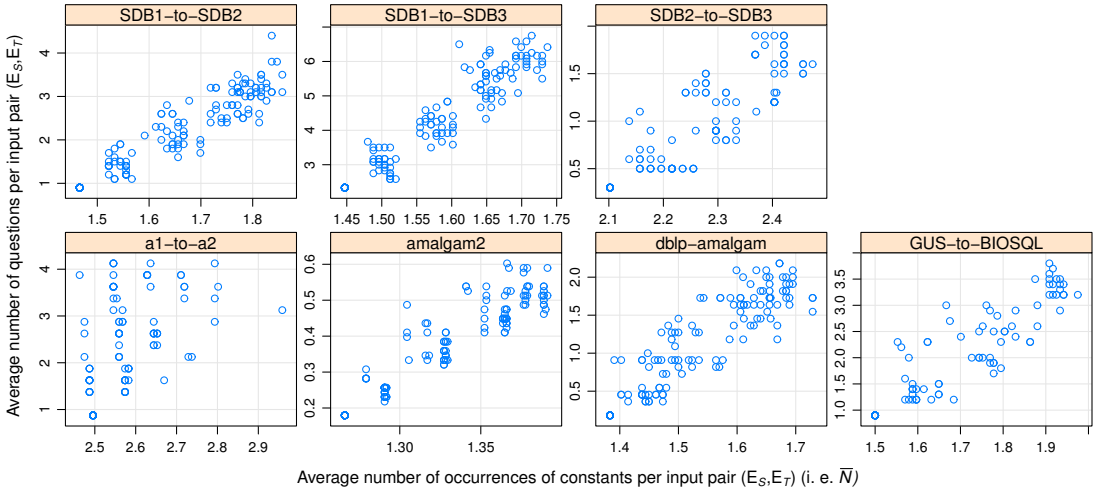


Figure 7: Average number of questions per per pair (E_S, E_T) versus $\bar{N}$ on atom refinement using top-down depth-first (BUDF) strategy.

Proof (sketch). Our framework first produces a normalized version of the canonical mapping, which is logically equivalent to the corresponding canonical mapping generated for universal exemplar tuple. In addition, the underlying assumption of our algorithms asserts that new constraints cannot be added and, by construction of the semilattices, the supremum of each semilattice corresponds to the case where no constraints are suppressed in both the atom and join refinement. Therefore, if a user chooses to keep only the supremum, the produced mapping will be logically equivalent to the normalized canonical mapping. As the normalized mapping is logically equivalent to the initial canonical mapping, by transitivity the mapping produced by our framework by choosing only the semilattices' supremum is also logically equivalent to this canonical mapping. □

C. COMPLETE EXECUTION TIMES

We report in Table 3 the running times for all four exploration strategies.

D. EXPERIMENTS: COMPLEMENTARY FIGURES

As supplementary material, we report the results for the strategies TDDF and BUDF (omitted in the paper due to space constraints) in Figures 7 and 8 for atom refinement and Figure 9 for join refinement.

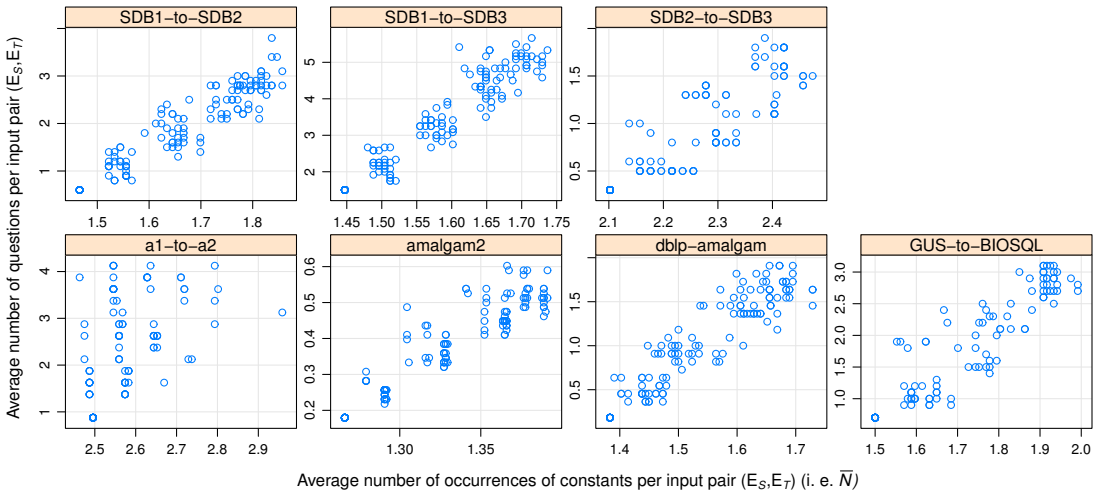
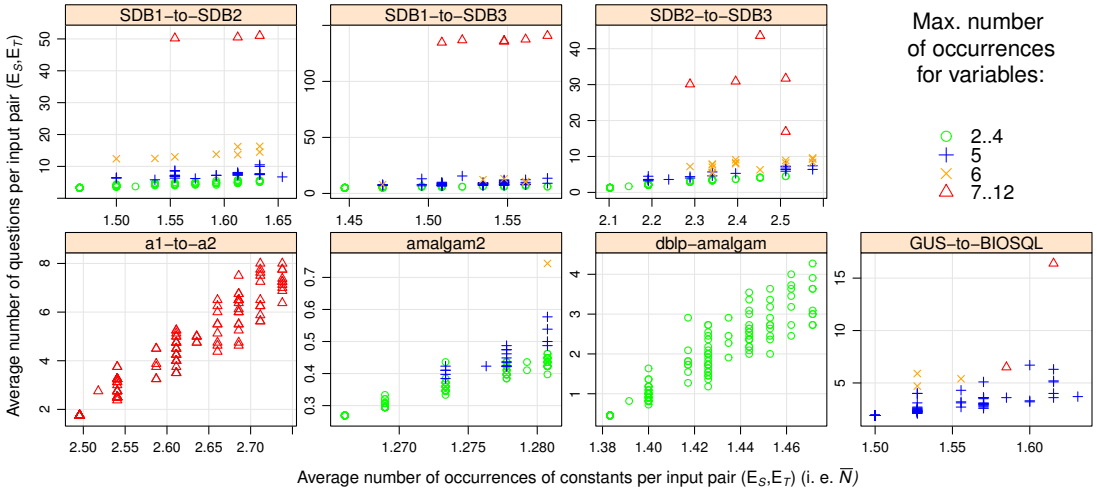
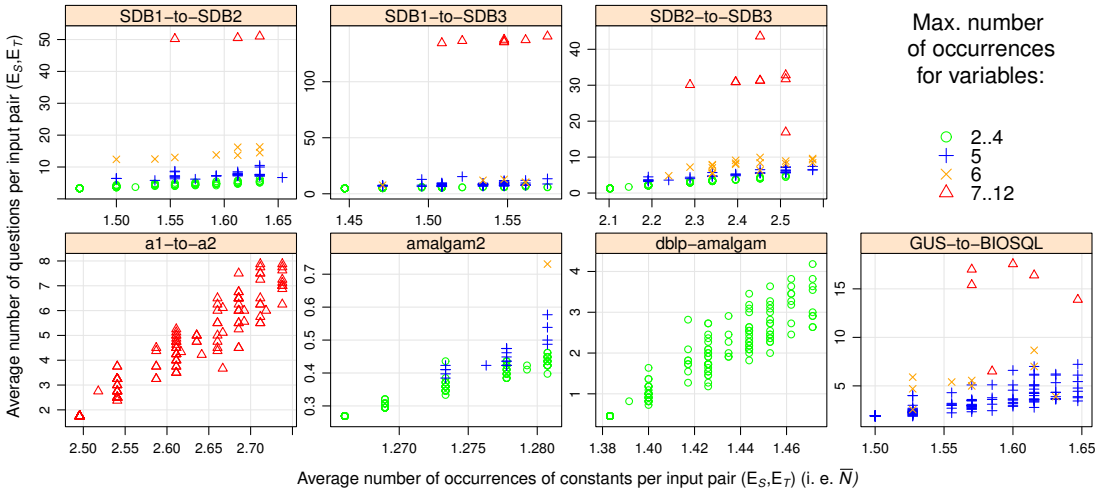


Figure 8: Average number of questions per pair (E_S, E_T) versus $\bar{N}$ on atom refinement using top-down depth-first (TDDF) strategy.



(a) Using the bottom-up depth-first (BUDF) strategy



(b) Using the top-down depth-first (TDDF) strategy

Figure 9: Average number of questions per pair (E_S, E_T) versus $\bar{N}$ on join refinement using depth-first strategies.