Semantic-Based Mappings

Giansalvatore Mecca[1], Guillem Rull[2],
Donatello Santoro[1,3], and Ernest Teniente[2]

[1] Università della Basilicata – Potenza, Italy
[2] Universitat Politècnica de Catalunya – Barcelona, Spain*
[3] Università Roma Tre – Roma, Italy

Abstract. Data translation consists of the task of moving data from a source database to a target database. This task is usually performed by developing mappings, i.e., executable transformations from the source to the target schema. However, it is often the case that a richer description of the target database semantics is available under the form of a conceptual schema. We investigate how the mapping process changes when such a rich conceptualization of the target database is available. As a major contribution, we develop a translation algorithm that automatically rewrites a mapping from the source database schema to the target conceptual schema into an equivalent mapping from the source schema to the underlying target database schema. Experiments show that our approach scales nicely to complex conceptual schemas and large databases.

1 Introduction

Integrating data coming from disparate sources is a crucial task in many applications. An essential requirement of any data integration task is that of manipulating *mappings* between sources. Mappings are executable transformations which define how an instance of a source repository should be translated into an instance of a target repository. Traditionally, mappings are developed to exchange data between two relational database schemas [15]. A rich body of research has been devoted to developing algorithms to simplify the specification of the mapping [19], formalizing the semantics of the translation process [8], and improving the quality of results [23, 13, 12].

This paper investigates how the mapping process changes in presence of richer *conceptual schemas* of the two data sources. In fact, the repositories used in the organization by the various processes and applications often undergo modifications during the years, and may lose their original design. As a consequence, it is often the case that an additional description of the domain of interest is available under the form of a (*mediator*) conceptual schema. The global unified view given by the mediator schema is constructed independently from the representation adopted for the data stored at the sources. Also, these semantic conceptual schemas are particularly important in the context of information integration since they are used to provide a transparent access through a global

* This work has been partially supported by the Ministerio de Ciencia y Tecnología under project TIN2011-24747

schema to a collection of data stored in multiple heterogeneous data sources [11]. It is therefore important to study how the mapping process changes in this setting.

We assume that a mediator conceptual schema is provided for the target and, possibly, for the source data repository. The relationship between the domain concepts in this conceptual schema and the data sources is given by a set of views, which define the conceptual constructs in terms of the logical database tables using a relational language of conjunctive queries, comparisons and negations, as discussed in the following example.

Motivating Example Assume we have the two relational schemas below and we need to develop a mapping to translate data among them (from the source to the target).

Source schema:	*Emp*(*id*, *name*, *dept*, *rating*)	*Dept*(*name*, *manager*)
Target schema:	*Employee*(*id*, *name*, *dept*)	*Dept*(*name*)
	HasPassedTest(*emp*, *testcode*)	*DisciplinaryProcedure*(*pnum*, *emp*)
	Manager(*mgr*, *dept*)	

Clearly, both schemas rely on the same domain, which includes data from employees and the departments they work in. However, it is not evident at all how the source-to-target mapping should be defined since it is difficult to establish a clear correspondence between the tables in the source schema and those in the target. This is so mainly because the target schema contains some tables, such as *HasPassedTest* or *DisciplinaryProceedure*, whose contents is difficult to relate to the information stored in the tables *Emp* and *Dept* from the source schema.

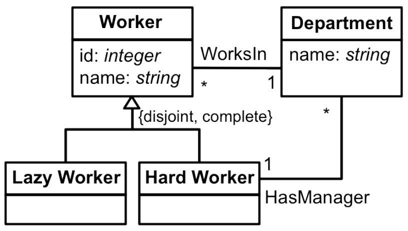

Fig. 1. Target conceptual schema.

Suppose now that the mediator conceptual schema corresponding to the target relational schema is the one shown in Figure 1. The views defining each class and association in the conceptual schema in terms of the database tables are reported in Figure 2 (keys are underlined). Notice how, to improve the expressibility of the language, negated literals are used in the body of view definitions, that is, the view-definition language is non-recursive Datalog [6] with negation. The semantics of this mediator conceptual schema is closer to the way the information is stored in the source schema than the one provided by the physical target tables. Therefore, the mapping designer will find it easier to define the mapping from the source schema to the target conceptual schema. For instance, he could easily realize that lazy workers in the target conceptual schema are those employees in the source with a rating lower than 5, while the rest of employees are hard work-

$$\begin{aligned}
&\mathsf{Worker}(\underline{\mathsf{id}}, \mathsf{name}) \Leftarrow \mathit{Employee}(id, name, dept)\\
&\mathsf{Promoted}(\mathsf{id}) \Leftarrow \mathit{HasPassedTest}(id, tc), \neg \mathit{DisciplinaryProcedure}(pnum, id)\\
&\mathsf{LazyWorker}(\mathsf{id}, \mathsf{name}) \Leftarrow \mathit{Employee}(id, name, dept), \neg \mathsf{Promoted}(\mathsf{id})\\
&\mathsf{HardWorker}(\mathsf{id}, \mathsf{name}) \Leftarrow \mathit{Employee}(id, name, dept), \neg \mathsf{LazyWorker}(\mathsf{id}, \mathsf{name})\\
&\mathsf{Department}(\mathsf{name}) \Leftarrow \mathit{Dept}(name)\\
&\mathsf{WorksIn}(\underline{\mathsf{id}}, \mathsf{dept}) \Leftarrow \mathit{Employee}(id, name, dept)\\
&\mathsf{HasManager}(\underline{\mathsf{dept}}, \mathsf{mgr}) \Leftarrow \mathit{Manager}(mgr, dept)
\end{aligned}$$

Fig. 2. View definitions for the target conceptual schema.

ers. As it is common [8], we shall use *tuple generating dependencies (tgds)* and *equality-generating dependencies (egds)* [3] to express the mappings. In our case, the translation of source tuples into the LazyWorkers and HardWorkers target concepts can be expressed by using the following tgds with comparison atoms:

$$m_0 : \forall id, name, dept, rat : Emp(id, name, dept, rat), rat < 5 \rightarrow \mathsf{LazyWorker}(\mathsf{id}, \mathsf{name})$$
$$m_1 : \forall id, name, dept, rat : Emp(id, name, dept, rat), rat \geq 5 \rightarrow \mathsf{HardWorker}(\mathsf{id}, \mathsf{name})$$

Intuitively, mapping expression m_0 specifies that, for each tuple in the source table *Emp* such that the *rating* attribute is lower than 5, there should be a lazy worker in the target. Similarly for m_1. Mappings m_2 below relates the *Dept* source table to the class Department and the association HasManager, and m_3 relates table *Emp* to association WorksIn (from now on, we omit universal quantifiers):

$$m_2 : Dept(name, mgr) \rightarrow \mathsf{Department}(\mathsf{name}), \mathsf{HasManager}(\mathsf{name}, \mathsf{mgr})$$
$$m_3 : Emp(id, name, dept, rating) \rightarrow \mathsf{WorksIn}(\mathsf{id}, \mathsf{dept})$$

Suppose now we want to perform the exchange and actually bring data from the source database to the target. Unfortunately, mappings $m_0 - m_3$ above are not directly executable. In fact, they refer to virtual entities – the constructs in the conceptual schema – and not to the actual tables in the target. We therefore need to devise a way to translate such source-to-semantic mapping into a classical source-to-target one (which copies data directly from the source into the target database) in order to be able to execute it. That is, given a source-to-semantic mapping, the target conceptual schema, and the views defining this schema in terms of the underlying database, we want to obtain the corresponding source-to-target mapping.

Contributions This paper develops a number of techniques to solve this kind of *semantic-based mapping problems.* More specifically:
(i) we develop rewriting algorithms to automatically translate mappings over the semantic schema into mappings over the underlying databases; we first discuss the case in which a semantic schema is available for the target database only; then, we extend the algorithm to the case in which a semantic schema is available both for the source and the target;
(ii) the algorithm that rewrites a source-to-semantic mapping into a classical and executable source-to-target mapping is based on the idea of unfolding views in mapping conclusions; however, in our setting, this unfolding is far from being straightforward; in the paper, we show that the problem is made significantly more complex by the expressibility of the view definition language, and more precisely, by the presence of negated atoms in the body of view definitions; we investigate the implications of adopting such an expressive language, and propose a solution that represents a good compromise between expressibility and complexity;
(iii) the classical approach to executing a source-to-target exchange consists in running the given mappings using a *chase* engine [8]; we develop a chase engine for this task, and integrate it within the working prototype of our semantic-based mapping system; using the prototype, we conduct several experiments to show that our approach scales nicely to large databases and mapping scenarios.

We believe this paper represents a significant step forward towards the goal of incorporating richer descriptions, under the form of conceptual schemas, into the data translation process. Given the evolution of the Semantic Web, and the increased adoption of ontologies, we believe this represents an important problem that may open up further research directions.

The paper is organized as follows. Section 2 recalls some basic notions and definitions. Section 3 introduces the semantic-based mapping problem. The rewriting algorithm and formal results are in Section 4. Experiments are in Section 5. Related works are in Section 6.

2 Background

Conceptual Schemas A *conceptual schema* in information systems is the general knowledge that the system needs about its domain in order to perform its functions [17]. In this paper, we focus on the part of a conceptual schema that deals with static aspects, i.e., the *structural schema*. In particular, we consider structural schemas that consist of a taxonomy of entity types (which may have attributes), a taxonomy of relationship types (defined among entity types), and a set of integrity constraints (which affect the state of the domain). The integrity constraints are expressed by means of *dependencies* (see subsection below). Throughout the paper, whenever we use the term conceptual schema, we are referring to a structural schema that conforms to this language.

Databases We focus on the relational setting. A *schema* $\mathbf{S}$ is a set of relation symbols $\{R_1, \ldots, R_n\}$ each with an associated relation schema $R(A_1, \ldots, A_m)$. Given schemas $\mathbf{S}, \mathbf{T}$ with disjoint relations symbols, $\langle \mathbf{S}, \mathbf{T} \rangle$ denotes the schema corresponding to the union of $\mathbf{S}$ and $\mathbf{T}$. An *instance* of a schema is a set of tuples in the form $R(v_1, \ldots, v_m)$, where each v_i denotes either a constant, typically denoted by $a, b, c, \ldots$, or a *labeled null*, denoted by $N_1, N_2, \ldots$. Constants and labeled nulls form two disjoint sets. Given instances I and J, a homomorphism $h : I \to J$ is a mapping from $dom(I)$ to $dom(J)$ such that for every $c \in \textsc{const}$, $h(c) = c$, and for all tuple $t = R(v_1, \ldots, v_n)$ in I, it is the case that $h(t) = R(h(v_1), \ldots, h(v_n))$ belongs to J. Homomorphisms immediately extend to formulas, since atoms in formulas can be seen as tuples whose values correspond to variables.

Views To bridge the gap between the conceptual schema and the underlying database, we assume a set of GAV views (Global-As-View) is given, i.e., we assume there is one view for each entity and relationship type which defines this type in terms of the underlying database (see Figure 2). A *view* V is a derived relation defined over a schema S. The view definition for V over S is a non-recursive rule in the form of $V(\overline{x}) \Leftarrow R_1(\overline{x}_1), \ldots, R_p(\overline{x}_p), \neg R_{p+1}(\overline{x}_{p+1}), \ldots, \neg R_{p+g}(\overline{x}_{p+g})$, with $p \geq 1$ and $g \geq 0$, where the variables in $\overline{x}$ are taken from $\overline{x}_1, \ldots, \overline{x}_p$. Atoms in a view definition can be either base or derived. An atom $V(\overline{x})$ is a *derived atom* if V denotes a view; otherwise, it is a *base atom*. A view definition specifies how the extension of the view is computed from a given instance of the underlying schema, that is, given a homomorphism h from the definition of V to an instance

I, $h(V(\overline{x}))$ belongs to the extension of V iff $h(R_1(\overline{x})) \wedge \ldots \wedge \neg h(R_{p+g}(\overline{x}_{p+g}))$ is true on I.

Dependencies A *tuple-generating dependency (tgd)* over $\mathbf{S}$ is a formula of the form $\forall \overline{x}, \overline{z}\big(\varphi(\overline{x}, \overline{z}) \to \exists \overline{y} \psi(\overline{x}, \overline{y})\big)$, where $\varphi(\overline{x}, \overline{z})$ and $\psi(\overline{x}, \overline{y})$ are conjunctions of atoms. We allow two kinds of atoms in the premise: (a) relational atoms over $\mathbf{S}$; (b) comparison atoms of the form v *op* c, where *op* is a comparison operator $(=, >, <, \geq, \leq)$, v is a variable that also appears as part of a relational atom, and c is a constant. Only relational atoms are allowed in the conclusion. A *denial constraint* is a special form of tgd of the form $\forall \overline{x}\big(\varphi(\overline{x}) \to \bot\big)$, in which the conclusion only contains the $\bot$ atom, which cannot be made true. An *equality generating dependency (egd)* over $\mathbf{S}$ is a formula of the form $\forall \overline{x}(\phi(\overline{x}) \to x_i = x_j)$ where $\phi(\overline{x})$ is a conjunctions of relational atoms over $\mathbf{S}$ and comparison atoms as defined above, and x_i and x_j occur in $\overline{x}$.

Mapping Scenarios A *mapping scenario*, $\mathcal{M} = \{\mathbf{S}, \mathbf{T}, \Sigma_{ST}, \Sigma_T\}$, is a quadruple consisting of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, and a set of *source-to-target tgds* Σ_{ST} – i.e., tgds such that the premise is a formula over $\mathbf{S}$ and the conclusion a formula over $\mathbf{T}$ –, and Σ_T is a set of *target tgds* – tgds over $\mathbf{T}$ – and *target egds* – egds over $\mathbf{T}$. Given a source instance I, a solution for I under $\mathcal{M}$ is a target instance J such that I and J satisfy Σ_{ST}, and J satisfies Σ_T. A solution J for I and $\mathcal{M}$ is called a *universal solution* if, for all other solution J' for I and $\mathcal{M}$, there is a homomorphism from J to J'. The chase is a well-known algorithm for computing universal solutions [8].

3 The Semantic-Based Mapping Problem

In this section we shall introduce our mapping problem. As we discussed above, let us first assume that a conceptual schema is only available for the target database. Later on we shall discuss how things can be extended to handle a source conceptual schema as well.

Source-to-Semantic Mappings The inputs to our source-to-semantics mapping problem are: (i) a source relational schema, $\mathbf{S}$, and a target relational schema $\mathbf{T}$; (ii) a target conceptual schema, $\mathbf{T}'$, defined by means of a set of view definitions, Σ_V, over $\mathbf{T}$, as in Figure 2. View definitions may involve negations over derived atoms; (iii) a set of egds, $\Sigma_{T'}$, to encode key constraints and functional dependencies over the conceptual schema; (iv) finally, as an additional input of crucial importance, we assume the definition of the source-to-semantic mapping, $\Sigma_{ST'}$ as a set of s-t tgds over $\mathbf{S}$ and $\mathbf{T}'$.

In the following, we always assume that the input mapping captures all of the semantics from the conceptual level. To do this, we assume the graphical integrity constraints of the conceptual schema are properly encoded into the mapping [9]. We restrict the textual integrity constraints to be key constraints and functional dependencies, and assume they are expressed as logic dependencies over the views (an automatic OCL-to-logic translation is proposed in [20]). Therefore, the complete set of dependencies $\Sigma_{ST'}$ and $\Sigma_{T'}$ for our running example is reported in Figure 3.

m_0 : $\mathit{Emp}(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{rating} < 5 \rightarrow \mathsf{LazyWorker}(\mathsf{id}, \mathsf{name}), \mathsf{Worker}(\mathsf{id}, \mathsf{name}),$
$\mathsf{WorksIn}(\mathsf{id}, \mathsf{dept}), \mathsf{Department}(\mathsf{dept})$
m_1 : $\mathit{Emp}(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{rating} \geq 5 \rightarrow \mathsf{HardWorker}(\mathsf{id}, \mathsf{name}), \mathsf{Worker}(\mathsf{id}, \mathsf{name})$
$\mathsf{WorksIn}(\mathsf{id}, \mathsf{dept}), \mathsf{Department}(\mathsf{dept})$
m_2 : $\mathit{Dept}(\mathit{name}, \mathit{mgr}) \rightarrow \exists \mathit{mname} : \mathsf{Department}(\mathsf{name}), \mathsf{HasManager}(\mathsf{name}, \mathsf{mgr}),$
$\mathsf{HardWorker}(\mathsf{mgr}, \mathsf{mname}), \mathsf{Worker}(\mathsf{mgr}, \mathsf{mname})$

m_3 : $\mathsf{Worker}(\mathsf{id}, \mathsf{name}), \mathsf{Worker}(\mathsf{id}, \mathsf{name'}) \rightarrow \mathsf{name} = \mathsf{name'}$
m_4 : $\mathsf{WorksIn}(\mathsf{id}, \mathsf{dept}), \mathsf{WorksIn}(\mathsf{id}, \mathsf{dept'}) \rightarrow \mathsf{dept} = \mathsf{dept'}$
m_5 : $\mathsf{HasManager}(\mathsf{dept}, \mathsf{mgr}), \mathsf{HasManager}(\mathsf{dept}, \mathsf{mgr'}) \rightarrow \mathsf{mgr} = \mathsf{mgr'}$

Fig. 3. Source-to-semantic mapping.

Based on these, our intention is to rewrite the dependencies in $\Sigma_{ST'} \cup \Sigma_{T'}$ as a new set of dependencies, from the source to the target database. More specifically, we shall generate:

(a) a new set of source-to-target tgds, Σ_{ST};

(b) a set of target dependencies, Σ_T. This latter set will contain: $(b.1)$ a set of target egds, as it is expected, in order to model egds over the conceptual schema. However, it may also incorporate other constraints that were not in the input. More precisely: $(b.2)$ target tgds, i.e., tgds defined over the symbols in the target only; $(b.3)$ denial constraints. Recall from Section 2 that a denial constraint is a dependency of the form $\forall \overline{x}(\varphi(\overline{x}) \rightarrow \perp)$. In our approach, these are used to express the fact that some tuple configurations in the the target are not compatible with the view definitions, and therefore should cause a failure in the mapping process.

The process is illustrated in Figure 5.a, where solid lines refer to inputs, and dashed lines to outputs produced by the rewriting. We shall require that the result of executing the source-to-target mapping is "equivalent" to the one that we would obtain if the source-to-semantic mapping was to be executed. By equivalent we mean that a universal solution produced by the source-to-target mapping induces a universal solution for the source-to-semantic mapping when applying the view definitions. To be more precise, we first consider the source-to-semantic mapping scenario: $\mathcal{M}_{ST'} = \{\mathbf{S}, \mathbf{T}', \Sigma_{ST'}, \Sigma_{T'}\}$. For each source instance I, assume there exist a universal solution $J_{T'}$ for I and $\mathcal{M}_{ST'}$ that complies with the view definitions in Σ_V (i.e., there exist an instance J_T of schema $\mathbf{T}$ such that $J_{T'} = \Sigma_V(J_T)$). Figure 4.a and b show one example of I and $J_{T'}$.

a. Source instance I

Emp(1, John, Marketing, 5)	*Emp(2, Luke, Shipping, 7)*	*Emp(3, Mike, Shipping, 2)*
Dept(Marketing, 1)	*Dept(Shipping, 2)*	

b. Conceptual schema instance $J_{T'}$

HardWorker(1, John)	Worker(1, John)	WorksIn(1, Marketing)
HardWorker(2, Luke)	Worker(2, Luke)	WorksIn(2, Shipping)
LazyWorker(3, Mike)	Worker(3, Mike)	WorksIn(3, Shipping)
Department(Marketing)	HasManager(Marketing, 1)	
Department(Shipping)	HasManager(Shipping, 2)	

c. Target instance J_T

Employee(1, John, Mark.)	*HasPassedTest*$(1, N_1)$	*Dept(Mark.)*	*Manager(1, Mark.)*
Employee(2, Luke, Ship.)	*HasPassedTest*$(2, N_2)$	*Dept(Ship.)*	*Manager(2, Ship.)*
Employee(3, Mike, Ship.)			

Fig. 4. Source, conceptual, and target instances.

We compute our rewriting, and obtain a new source-to-target scenario: $\mathcal{M}_{ST} = \{\mathbf{S}, \mathbf{T}, \Sigma_{ST}, \Sigma_T\}$. We may run $\mathcal{M}_{ST}$ on I to obtain solutions under the form of target instances. To any target instance J_T of this kind, we may apply the view definitions in Σ_V in order to obtain an instance of $\mathbf{T}'$, $\Sigma_V(J_T)$.

To formalize this notion, given a source-to-semantic scenario $\mathcal{M}_{ST'} = \{\mathbf{S}, \mathbf{T}', \Sigma_{ST'}, \Sigma_{T'}\}$ with view definitions Σ_V, we say that the source-to-target rewritten scenario $\mathcal{M}_{ST} = \{\mathbf{S}, \mathbf{T}, \Sigma_{ST}, \Sigma_T\}$ is *correct* if, for each instance I of the source database, whenever a universal solution J_T for I and $\mathcal{M}_{ST}$ exist, then $\Sigma_V(J_T)$ is also a universal solution for I and the original scenario $\mathcal{M}_{ST'}$.

Figure 4.c reports a correct target solution for I (N_1, N_2 are labeled nulls). It can be seen, in fact, that $\Sigma_V(J_T)$ is exactly the conceptual instance $J_{T'}$ in Figure 4.b. Note that a different font is used for entity and relationship types in the conceptual instance.

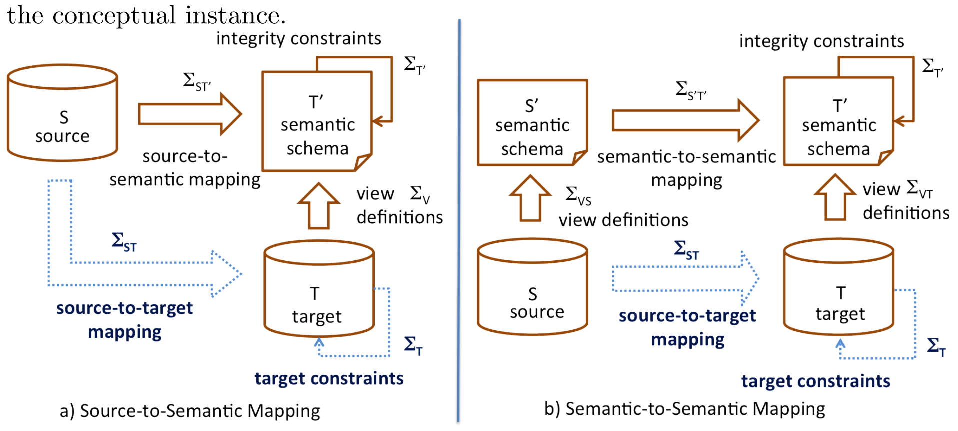

Fig. 5. Semantic Mapping Scenarios.

Semantic-to-Semantic Mappings The following sections are devoted to the development of the mapping rewriting algorithm. Before we turn to that, let us discuss what happens when also a conceptual schema over the source is given, as shown in 5.b. In this case, we assume that both view definitions for the source conceptual schema, Σ_{VS}, and target conceptual schema, Σ_{VT} are given, with the respective egds. We also assume that the mapping, $\Sigma_{S'T'}$, is designed between the two conceptual descriptions.

It can be seen that this case can be reduced to the one above. In fact, we can see the problem as the composition of two steps: (i) applying the source view definitions to materialize the extent of the source conceptual schema; (b) consider the materialized instance as a new source database, and solve the source-to-semantic mapping problem as in Figure 5.a. In light of this, in the following we shall concentrate on the scenario in Figure 5.a only.

4 The Rewriting Algorithm

Given our input source-to-semantic mapping scenario, $\mathcal{M}_{ST'} = \{\mathbf{S}, \mathbf{T}', \Sigma_{ST'}, \Sigma_{T'}\}$, our approach consists in progressively rewriting dependencies in $\Sigma_{ST'}$ and $\Sigma_{T'}$ in order to remove view symbols, and replace them with target relations.

To do this, we shall apply a number of transformations that guarantee that the rewritten mapping yields equivalent results wrt to input one, in the sense discussed in Section 3.

Normalization The first of these transformations consists in *normalizing* the tgds in $\Sigma_{ST'}$. A tgd is said to be in *normal form* [13] if it cannot be split in two or more equivalent tgds with the same premise. In our example, the tgds in Figure 3 are not normalized. To give an example, consider m_2:

$$m_2 : Dept(name, mgr) \rightarrow \exists mname : \mathsf{Department(name)}, \mathsf{HasManager(name, mgr)}, \mathsf{HardWorker(mgr, mname)}, \mathsf{Worker(mgr, mname)}$$

The tgd can be split in several tgds in normal form. The intuition behind the normalization is to break atoms in the conclusion in such a way that two atoms remain together only if they share an existential variable, as follows:

$$m_{2a} : Dept(name, mgr) \rightarrow \mathsf{Department(name)}$$
$$m_{2b} : Dept(name, mgr) \rightarrow \mathsf{HasManager(name, mgr)}$$
$$m_{2c} : Dept(name, mgr) \rightarrow \exists mname : \mathsf{HardWorker(mgr, mname)}, \mathsf{Worker(mgr, mname)}$$

Similarly, we can split m_0 into four tgds, $m_{0a}, m_{0b}, m_{0c}, m_{0d}$, and m_1 into m_{1a}, m_{1b}, m_{1c}, m_{1d}, each with a single atom in the conclusion. In the following, we shall always refer to these normalized tgds.

Unfolding the View Definition Language Once tgds have been normalized, we may proceed with the goal of removing view symbols from mapping conclusions. It can be seen that our problem reduces to a variant of the classical problem of *view unfolding*.

As it is obvious, the complexity of the problem depends quite a lot on the expressibility of the view definition language allowed in our scenarios. In fact, if we used plain conjunctive queries over target relation symbols as a view definition language, the rewriting would pretty much reduce to an application of the standard view unfolding algorithm [22]. To give an example, consider mapping m_{0c} and recall the definition of view $\mathsf{WorksIn}$:

$$m_{0c} : Emp(id, name, dept, rating) \rightarrow \mathsf{WorksIn(id, dept)}$$
$$\mathsf{WorksIn(id, dept)} \Leftarrow Employee(id, name, dept)$$

Standard view unfolding yields the following s-t tgd:

$$m'_{0c} : Emp(id, name, dept, rating) \rightarrow \exists name' : Employee(id, name', dept)$$

However, the main purpose of having a semantic description of the target database stands in its richer nature with respect to the power of the pure selection-projection-join paradigm. In fact, in this paper we allow for a more expressive language than conjunctive queries, i.e, non-recursive Datalog with negation.

Since the standard view-unfolding algorithm cannot handle negated atoms, we need to devise a new and improved algorithm to correctly unfold non-recursive Datalog view definitions with negations. However, before doing that we need to ask ourselves if the new view definition language allows for this kind of unfolding.

Unfortunately, we have the following negative result, stating that the semantic-based mappig problem cannot in general be solved for the full language of non-recursive Datalog with negation:

Proposition 1. *There exists a semantic schema T' defined in terms of non-recursive Datalog rules with negation and a source-to-semantic scenario $\mathcal{M}_{ST'} = \{\boldsymbol{S}, \boldsymbol{T'}, \Sigma_{ST'}, \Sigma_{T'}\}$, such that there exists no correct rewriting in terms of tgds, egds, and denials only.*

The main intuition behind the proof is that the increased expressive power of the view definition language requires a more expressive language to rewrite the tgds. This new tgd language is the language of *disjunctive embedded dependencies* [7], i.e., tgds in which disjunction symbols are allowed in the conclusion. Since computing solutions for DEDs is significantly more challenging than for standard dependencies, in this paper we rather concentrate on standard tgds and egds.

Proposition 1 leaves us with a crucial question: is it possible to find a view definition language that is at the same time more expressive than plain conjunctive queries, and allows in practice to compute correct rewritings in terms of tgds and egds ?

In the following sections, we show that this language exists, and corresponds to non-recursive Datalog with a limited negation. To be more precise, we limit negation in such a way that: (i) only one negated atom is allowed for each view definition; (ii) keys and functional dependencies – i.e., egds – are defined only for views whose definition does not depend on negated atoms.

In the rest of the paper, we concentrate exactly on this language.

The View Unfolding Algorithm The pseudocode of our unfolding algorithm *UnfoldDependencies* is reported in Algorithm 1. The algorithm takes as input the initial set of source-to-semantic dependencies, $\Sigma_{ST'}$, and the semantic egds in $\Sigma_{T'}$, plus the view definitions in Σ_V, and returns a set of source-to-target tgds, Σ_{ST}, plus a set of target egds, tgds and denials Σ_T. To define the algorithm, we use the standard unfolding algorithm for positive views, *unfoldView* [22], as a building block.

The main intuition behind the algorithm is easily stated: it works with a set of dependencies, called Σ, initialized as $\Sigma_{ST'} \cup \Sigma_{T'}$, and progressively transforms this set, until a fixpoint is reached. Note that it always terminates, since we assume the view definitions are not recursive. Three main transformations are employed in order to remove derived atoms from the dependencies of Σ:
(i) first, whenever a positive derived atom $L(\bar{x}_i)$ is found in a dependency d, the algorithm uses the standard view unfolding algorithm as a building block in order to replace $L(\bar{x}_i)$ by its view definition; to see an example, consider tgd m_0 and view $\mathsf{LazyWorker}$:

$$m_{0a} : \mathit{Emp}(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{rating} < 5 \rightarrow \mathsf{LazyWorker}(\mathsf{id}, \mathsf{name})$$
$$\mathsf{LazyWorker}(\mathsf{id}, \mathsf{name}) \Leftarrow \mathit{Employee}(\mathit{id}, \mathit{name}, \mathit{dept}), \neg\mathsf{Promoted}(\mathsf{id})$$

Standard unfolding changes m_0 as follows:

$$m'_{0a} : \mathit{Emp}(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{rating} < 5 \rightarrow \exists \mathit{dept'} : \mathit{Employee}(\mathit{id}, \mathit{name}, \mathit{dept'}), \neg\mathsf{Promoted}(\mathsf{id})$$

(ii) the second, and most important transformation, consists in handling negated view atoms $\neg L(\bar{x}_i)$ in tgd conclusions, as the one about $\mathsf{Promoted}$ employees in m'_{0a}; we cannot directly unfold the atom in the conclusion in order to have

Algorithm 1 *UnfoldDependencies*$(\Sigma_{ST'}, \Sigma_{T'}, \Sigma_V)$

```
$\Sigma := \Sigma_{ST'} \cup \Sigma_{T'}$
repeat
  for all $d \in \Sigma$ do
    if $d$ contains a positive derived atom $L$ then
      $d := \mathit{unfoldView}(L, d, \Sigma_V)$
    end if
    if $d$ is a tgd containing a negative derived atom $\neg L(\bar{x}_i)$ in $\psi(\bar{x}, \bar{y})$ then
      $d := \phi(\bar{x}) \rightarrow \psi(\bar{x}, \bar{y}) - \{\neg L(\bar{x}_i)\}$
      let $TGD_i$ be a new relation symbol
      $d^1 := \phi(\bar{x}) \rightarrow TGD_i(\bar{x})$
      $d^2 := TGD_i(\bar{x}) \wedge L(\bar{x}_i) \rightarrow \bot$
      $\Sigma := \Sigma \cup \{d^1, d^2\}$
    end if
    if $d$ contains a negative atom $\neg L(\bar{x}_i)$ in $\phi(\bar{x})$ then
      $d := \phi(\bar{x}) - \{\neg L(\bar{x}_i)\} \rightarrow \psi(\bar{x}, \bar{y}) \cup \{L(\bar{x}_i)\}$
    end if
  end for
until fixpoint
$\Sigma_{ST}$ := the set of s-t tgds in $\Sigma$
$\Sigma_T$ := the set of egds, target tgds and denials in $\Sigma$
```

an equivalent tgd; we need a way to express more appropriately the intended semantics, i.e, the fact that the tgd should be fired only if it is not possible to satisfy $L(\bar{x}_i)$; to express this, we remove the negated atom from the conclusion

$$m''_{0a} : \mathit{Emp}(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{rating} < 5 \rightarrow \exists \mathit{dept'} : \mathit{Employee}(\mathit{id}, \mathit{name}, \mathit{dept'})$$

and introduce two new dependencies;
– the first one, d^1, uses a new relation symbol, $TGD_i(\bar{x})$, to express the fact that the body of d is fired for a vector of variables $\bar{x}$; in our example, we need to add this new tgd:

$$m^1_{0a} : \mathit{Emp}(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{rating} < 5 \rightarrow \mathit{TGD}_0(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating})$$

– the second one, d^2, states that d should fire only if it is not possible to satisfy $L(\bar{x}_i)$, by means of a denial constraint, as follows:

$$m^2_{0a} : \mathit{TGD}_0(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathsf{Promoted}(\mathsf{id}) \rightarrow \bot$$

(iii) the third, and final transformation, consists of moving negated view atoms of the form $\neg L(\bar{x}_i)$ in the premise of a dependency d to its conclusion, in order to remove the negation. To see an example of this, let's complete the rewriting of m^2_{0a}; transformation (i) needs to be applied again in order to unfold the Promoted atom:

$$m^{2'}_{0a} : \mathit{TGD}_0(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{HasPassedTest}(\mathit{id}, \mathit{tc}), \neg \mathit{DisciplinaryProcedure}(\mathit{pnum}, \mathit{id}) \rightarrow \bot$$

however, the negative atom may be easily moved to the conclusion, to yield the final target tgd:

$$m^{2''}_{0} : \mathit{TGD}_0(\mathit{id}, \mathit{name}, \mathit{dept}, \mathit{rating}), \mathit{HasPassedTest}(\mathit{id}, \mathit{tc}) \rightarrow \exists \mathit{pnum} : \mathit{DisciplinaryProcedure}(\mathit{pnum}, \mathit{id})$$

It is worth discussing why this last step is safe in our setting. Notice, in fact, that while moving a single negated atom from the premise to the conclusion gives a correct rewriting, if the atoms were more than one a disjunction would be needed in the conclusion; in this way, we would end up again with disjunctive embedded dependencies, and therefore go outside the domain of standard dependencies.

However, given the restrictions that we have imposed on our view definition language, this cannot be the case; notice in fact that (a) negated view atoms may appear only in the premise of denials introduced by transformation; the initial tgds only contain source symbols in the premise, and we make the assumption that no egd is defined over a view whose definition involves negation; (b) by construction, these denials are such that only one negation may appear in the body; in fact, for each view symbol in a tgd conclusion only one negation may appear.

We are now ready to state our main result, about the correctness of the rewriting algorithm. Before we do that, we should make it more precise the schemas that are involved in the translation. In fact, we start with a target schema, $\mathbf{T}$, but during the rewriting we enrich it with new relation symbols, $TGD_0, TGD_1, \ldots$, in order to be able to correctly specify denials. We shall call the resulting schema $\mathbf{T}''$.

Theorem 1 (Correctness). *Given a source-to-semantic scenario* $\mathcal{M}_{ST'} = \{\boldsymbol{S}, \boldsymbol{T'}, \Sigma_{ST'}, \Sigma_{T'}\}$ *with view definition* Σ_V*, algorithm UnfoldDependencies computes a correct source-to-target rewritten scenario* $\mathcal{M}_{ST''} = \{\boldsymbol{S}, \boldsymbol{T''}, \Sigma_{ST''}, \Sigma_{T''}\}$*, where* $\boldsymbol{T''}$ *is obtained from* $\boldsymbol{T}$ *by enriching it with a finite set of new relation symbols* $TGD_0, TGD_1, \ldots$.

5 Experiments

We have implemented a chase engine and a prototype of the algorithm presented in this paper, both written in Java. The experiments have been performed on an Intel core i7 machine with a 2.6 GHz processor, 8 GB of RAM, and running MacOSX. The DBMS used has been PostgreSQL 9.2.1 (x64 version).

Datasets We used two datasets. The first one (Employees), correspond to our running example; of this, we have studied two variants, one with egds, the other with no egds. The second one (Synthetic), is a fully synthetic dataset of which we generated variants from 50 to 40K dependencies.

Effectiveness As a first set of experiments, we study the effectiveness of our approach. More specifically, we compared the size of the source-to-semantic mapping that users need to specify for the various scenarios, to the size of the actual source-to-target scenario generated by our rewriting. As a measure of the size of a scenario, we took the number of nodes and edges of the *dependency graph* [8], i.e., the graph in which each atom of a dependency is a node, and there is an edge from node n_1 to node n_2 whenever the corresponding atoms share a variable. Intuitively, the higher is the complexity of this graph, the more complicated is to express the mapping. Figure 6.a reports the results for 4 scenarios. It can be seen that in all scenarios there was a considerable increase in the size

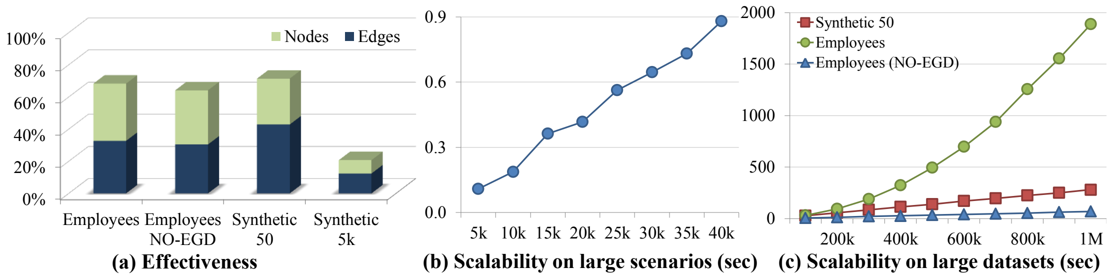

Fig. 6. Results of Experiments.

of the dependency graph (up to 70%). This is a clear indication that in many cases our approach is more effective with respect to manually developing the source-to-target mapping.

Scalability on Large Scenarios The second set of experiments tests the scalability of our unfolding algorithm on mapping scenarios of large size. The results of these experiments are reported on Figure 6.b. We have generated a series of synthetic scenarios of increasing size. All source-to-semantic tgds in these scenarios have two source relations on the premise and two views on the conclusion. Each view definition has two positive target relations and (at most) one negative auxiliary view. For each mapping scenario, 20% of the tgds have no negated atoms, the next 20% have 1 level of negation (i.e., negated atoms that do not depend in turn on other negations), the next 20% have 2 levels of negation, and so on, up to 4 levels of negations. The number of source relations in the mapping scenarios ranges from 10k to 80k, the number of view definitions ranges from 30k to 240k, and the number of target relations ranges from 60k to 480k. The reported times are the running times of the unfolding algorithm running in main memory, and do not include disk read and write times. As it can be seen, the rewriting algorithm scales nicely to large scenarios.

Scalability on Large Datasets The final set of experiments tests the scalability of the chase engine on large databases. This is a very important issue, since previous works [14, 12] have shown that existing chase engines for egds may require several hours on a few thousands tuples, and therefore hardly scale to large datasets. Figure 6.c reports the time needed to compute a solution for three of our scenarios. As it was expected, scenarios with no egds required lower computing times. However, also in the case of egds, our chase engine scaled nicely to databases of 1 million tuples. To the best of our knowledge, this is the first actual scalability result for the chase of egds.

6 Related Works

The standard view unfolding algorithm [22] has been extensively used in data integration as a tool for query answering. In such a setting, users pose queries over a set of heterogeneous sources through a single global schema, which provides a uniform view of all the sources. Mappings between the sources and the global schema are used to rewrite the users' queries in terms of the sources. One way to define these mappings is the so-called global-as-view approach (GAV), in which the global schema is defined as a view over the sources. With this kind of mappings, answering a query posed on the global schema usually reduces to unfolding the view definitions [11] (unless integrity constraints are present in the global schema, which makes answering harder [4]).

Another similar problem is that of accessing data through ontologies, in which users pose queries on an ontology that is defined on top of a set of databases; the ontology plays the role of global schema, and the databases play the role of data sources [18, 5]. The problem we address in this paper, however, is not about using view unfolding to answer queries, but to copy data into a target. As we have discussed in Section 4, standard view unfolding suffices only when the views that define the target conceptual schema in terms of the underlying database are plain conjunctive queries. In the presence of negation, copying data into the target gets more complicated, as negated atoms in mapping conclusions introduce new integrity constraints that standard view unfolding does not handle (intuitively, negated atoms must be kept false during all the process of copying data into the target).

A problem that relates to our use of view unfolding in mappings is that of mapping composition [10, 16]. Composing a mapping between schemas **A** and **B** with a mapping between schemas **B** and **C** produces a new mapping between **A** and **C**. In a sense, our application of view unfolding to the conclusion of a mapping can be seen as a kind of mapping composition; one in which the mapping between the source and the conceptual schema is composed with a second mapping that relates the conceptual schema with the underlying database (i.e., the views). However, mapping composition techniques take into account the direction of the mapping, that is, one can compose a mapping from **A** to **B** only with another mapping that goes from **B** to some **C** in order to get a mapping that goes from **A** to **C**. In our case, we have a mapping from the source to the conceptual schema and another one from the database to the conceptual schema, which cannot be directly composed.

The introduction of conceptual schemas into the mapping process has also been investigated in [1] with respect to a different problem, i.e., that of generating mappings between databases. Since we assume that source-to-semantic mappings are given as inputs, the techniques developed in [1] can be used as a preliminary step to simplify the mapping specification phase.

Another context where mappings involving conceptual schemas have been studied is that of semantic-web ontologies; in particular, [21] proposes a technique that translates a set of correspondences between a source and target ontologies into a set of SPARQL queries that can then be run against the data source to produce the target's data. Comparing with our approach, we assume that the given mapping is not just a set of correspondences, but a complete declarative mapping expressed as tgds, and we also take into account that the target's conceptual schema is a view of the underlying database.

Mappings between conceptual schemas have also been studied in [2], where the authors propose an approach for finding "semantically similar" associations between two conceptual schemas. This similar associations are then used to generate a mapping. This approach is complementary to ours in the sense that it could be used to generate a semantic-based mapping, which would then be rewritten using the algorithm we present in this paper.

References

1. Y. An, A. Borgida, R. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *ICDE*, pages 206–215, 2007.
2. Y. An and I.-Y. Song. Discovering semantically similar associations (sesa) for complex mappings between conceptual models. In *ER*, pages 369–382, 2008.
3. C. Beeri and M. Vardi. A Proof Procedure for Data Dependencies. *J.ACM*, 1984.
4. A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Inf. Syst*, 29(2):147–163, 2004.
5. D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. View-based query answering in description logics: Semantics and complexity. *J. Comput. Syst. Sci.*, 78(1):26–46, 2012.
6. S. Ceri, G. Gottlob, and L. Tanca. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE TKDE*, 1(1):146–166, 1989.
7. A. Deutsch and V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *DBPL*, pages 21–39, 2001.
8. R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.
9. R. Fagin, P. Kolaitis, A. Nash, and L. Popa. Towards a Theory of Schema-Mapping Optimization. In *PODS*, pages 33–42, 2008.
10. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM TODS*, 30(4):994–1055, 2005.
11. M. Lenzerini. Data integration: a Theoretical Perspective. In *PODS*, 2002.
12. B. Marnette, G. Mecca, and P. Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.
13. G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings. In *SIGMOD*, 2009.
14. G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings: Scalable Core Computations in Data Exchange. *Inf. Syst*, 37(7):677–711, 2012.
15. R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB*, pages 77–99, 2000.
16. A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. *ACM Trans. Database Syst.*, 32(1):4, 2007.
17. A. Olivé. *Conceptual modeling of information systems.* Springer, 2007.
18. A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
19. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.
20. A. Queralt and E. Teniente. Reasoning on uml class diagrams with ocl constraints. In *ER*, pages 497–512, 2006.
21. C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. Generating sparql executable mappings to integrate ontologies. In *ER*, pages 118–131, 2011.
22. L. Sterling and E. Y. Shapiro. *The Art of Prolog: Advanced Programming Techniques.* MIT Press, 1994.
23. B. ten Cate, L. Chiticariu, P. Kolaitis, and W. C. Tan. Laconic Schema Mappings: Computing Core Universal Solutions by Means of SQL Queries. *PVLDB*, 2(1):1006–1017, 2009.