Validation of Mappings between Schemas

Guillem Rull[1,2] Carles Farré[2] Ernest Teniente[2] Toni Urpí[2]

Universitat Politècnica de Catalunya
Jordi Girona 1-3
08034 - Barcelona
phone: +34934137887

{grull | farre | teniente | urpi}@lsi.upc.edu

ABSTRACT

Mappings between schemas are key elements in several contexts such as data exchange, data integration, peer data management systems, etc. In all these contexts, the process of designing a mapping requires the participation of a mapping designer that needs a way to validate the mapping being defined, i.e., to check whether the mapping is in fact what the designer intended. However, to date very little work has directly focused on the effective validation of schema mappings. In this paper, we present a new approach for validating schema mappings that allows the mapping designer to ask questions about the accomplishment of certain desirable properties of these mappings. We consider four properties of mappings: mapping satisfiability, mapping inference, query answerability and mapping losslessness. We reformulate these properties in terms of the problem of checking the liveliness of a derived predicate. We emphasize that this approach is independent of any particular method for liveliness checking and, to show the feasibility of our approach, we use an implementation of the CQC Method and provide some experimental results.

1. INTRODUCTION

Mappings between schemas are key elements for any system that requires an interaction of heterogeneous data and applications. Data exchange [10, 19] and data integration [20] are two well known contexts in which mappings play a central role. Another context in which mappings are also extensively used is that of peer data management systems [6, 17]. In all these cases, schema mappings are specifications that model a relationship between schemas. Several formalisms are used for defining them. In data exchange, tuple-generating dependencies (tgds) and equality-generating dependencies (egds) are widely used [10]. In the context of data integration, global-as-view (GAV), local-as-view (LAV), and global-and-local-as-view (GLAV) [14, 20] are the most common approaches. A further formalism that has recently emerged is that of nested mappings [15], which extends previous formalisms for relational and nested data. Model management [3, 25] is also a widely known approach which establishes a conceptual framework for handling schemas and mappings in a generic way, providing a set of generic operators, such as the composition of mappings [4, 11, 23].

Data transformation languages like XSLT, XQuery and SQL are also used for specifying mappings in several existing tools that help engineers to build mappings [5, 24, 29]. One example of a system for producing mappings is Clio [16], which can be used to semi-automatically generate a schema mapping from a set of correspondences between schema elements (e.g. field names). This set of uninterpreted correspondences is usually called a matching. Hence, finding a matching is the first step towards developing a schema mapping. A significant amount of work on schema matching techniques can be found in the literature, among which we could cite [9, 21, 27].

Nevertheless, the process of designing a mapping requires the feedback of a human engineer. The designer guides the generating process by choosing between alternative candidates, and successively refining the mapping. The designer thus needs a way to check whether the mapping produced is in fact what was intended, that is, the developer must validate the mapping. However, to date very little work has directly focused on the effective validation of schema mappings.

In this paper, we present a new approach for validating schema mappings that allows the mapping designer to ask questions about the accomplishment of certain desirable properties of these mappings. Answers to these questions will provide information on whether the mapping correctly and adequately matches the intended needs and requirements.

We consider the two properties identified in [22] as important properties of mappings: mapping inference and query answerability. We likewise introduce two new properties: mapping satisfiability and mapping losslessness.

Mapping satisfiability is aimed at detecting incompatibilities, either within the mapping or between the mapping and the constraints of the schemas. Mapping inference can be used to check for redundancies in the mapping or to check the equivalence of two mappings.

In general, mappings lose some information and can be partial or incomplete, since they rarely map all the concepts from one schema to the other. Query answerability and mapping losslessness are aimed at detecting whether these lossy mappings still preserve some relevant information, which is represented in terms of some queries defined over one of the schemas. Specifically, query answerability informs the designer whether two mapped schemas are equivalent with respect to these queries, in the sense that they would obtain the same answers in both sides. Instead, mapping losslessness is aimed at detecting whether the mapping does not lose information from one schema to the other, in the sense that the queries over the first schema can also be computed over the second one. Note that it is not required for the answers obtained from the second schema to be the exact ones obtained from the former.

[1] *This work was supported in part by Microsoft Research through the European PhD Scholarship Programme.*

[2] *This work was partially supported by the Spanish Ministerio de Educación y Ciencia under project TIN2005-05406.*

In summary, the main idea of our approach is to define, for each particular property, a distinguished derived predicate which describes the fulfillment of the property. This predicate will be defined over a new schema that integrates the two mapped schemas and a set of integrity constraints that make the relationship modeled by the mapping explicit. Thus, if the distinguished predicate is lively [13, 18, 32] over this schema, the property is satisfied.

Hence, the four properties we consider in this paper are expressed in terms of the problem of checking whether a derived predicate is lively in a database schema. This allows us to use one single method to effectively check all the properties. It is worth noting that any existing method for checking the liveliness of a predicate can be used, provided that it is able to deal with the classes of queries and integrity constraints that appear in the database schema.

To show the feasibility of our approach, we will use our CQC Method [12], which is able to deal with a broad class of database schemas and, thus, we are able to consider schemas and mappings with a high degree of expressiveness.

In this paper, we consider mappings defined as binary relationships between two relational schemas, which are specified as sets of mapping formulas in a similar way to the GLAV approach.

Contributions

- We propose a new approach for validating schema mappings. This approach consists in defining a set of desirable properties of mappings and reformulates them in terms of predicate liveliness. The approach is independent of the method used for checking liveliness.
- We show how to use our approach for checking two important properties of mappings already identified in the literature [22]: mapping inference and query answerability.
- We propose two additional properties—mapping satisfiability and mapping losslessness—to capture validation information not covered by the two other properties.
- We show that our mapping losslessness property is in fact an extension of query answerability. We can use losslessness to try to obtain useful validation information when query answerability does not hold.
- We show how to use our CQC Method to effectively check the four mapping properties. The CQC Method is able to deal with schemas that have integrity constraints, safe-negated EDB and IDB literals, and comparisons.

The paper is organized as follows. Section 2 introduces the basic concepts and notation to be used throughout the paper. Section 3 presents our approach, the properties and their formulation in terms of predicate liveliness. Section 4 presents the experimental evaluation using the CQC Method. In Section 5, we address related work and, finally, in Section 6 we present our conclusions and suggestions for further work.

2. SCHEMAS AND MAPPINGS

In the following sections, we consider relational schemas, and mappings between two relational schemas. These mappings define a binary relationship on the sets of instances of the two schemas. We express schemas and mappings in a logic notation, in a similar way as is done in many other formalisms (e.g. tgds and GLAV). Now, we introduce the basic concepts.

Schemas A *schema S* is a tuple (*DR*, *IC*) where *DR* is a finite set of deductive rules and *IC* a finite set of constraints.

A *deductive rule* has the form:

$$p(\bar{X}) \leftarrow r_1(\bar{X}_1) \wedge \ldots \wedge r_n(\bar{X}_n) \wedge \\ \neg r_{n+1}(\bar{Y}_1) \wedge \ldots \wedge \neg r_m(\bar{Y}_s) \wedge C_1 \wedge \ldots \wedge C_t$$

and a *constraint* (or *condition*) the denial form:

$$\leftarrow r_1(\bar{X}_1) \wedge \ldots \wedge r_n(\bar{X}_n) \wedge \\ \neg r_{n+1}(\bar{Y}_1) \wedge \ldots \wedge \neg r_m(\bar{Y}_s) \wedge C_1 \wedge \ldots \wedge C_t.$$

The symbols p and r_1, ..., r_m are *predicates*. The tuples $\bar{X}$, $\bar{X}_1$, ..., $\bar{X}_n$, $\bar{Y}_1$, ..., $\bar{Y}_s$ contains *terms*, which are either variables or constants. Each C_i is a *built-in literal* in the form of $t_1\ \theta\ t_2$, where t_1 and t_2 are terms and operator θ is <, ≤, >, ≥, = or ≠. The atom $p(\bar{X})$ is the *head* of the rule, and $r_1(\bar{X}_1)$, ..., $r_n(\bar{X}_n)$, $\neg r_{n+1}(\bar{Y}_1)$, ..., $\neg r_m(\bar{Y}_s)$ are positive and negative *ordinary literals* (those that are not built-in). We require every rule and constraint be *safe*, that is, every variable occurring in $\bar{X}$, $\bar{Y}_1$, ..., $\bar{Y}_s$, C_1, ..., C_t must also appear in some $\bar{X}_i$. Note that differing from other formalisms like tgds and egds, we express constraints stating what may not happen instead of what should happen.

Predicates that appear in the head of a deductive rule are *derived predicates* also called *intensional database* (IDB) predicates. They correspond to views or queries. The rest are *base predicates* also called *extensional database* (EDB) predicates. They correspond to tables.

For a schema *S* = (*DR*, *IC*), a schema *instance D* is an EDB, that is, a set of ground facts about the base predicates of *S* (the tuples stored in the database). We denote by *DR*(*D*) the whole set of ground facts about base and derived predicates that are inferred from an instance *D*, i.e., the fix-point model of $DR \cup D$.

An instance *D violates* a constraint $\leftarrow L_1 \wedge ... \wedge L_k$ if $(L_1 \wedge ... \wedge L_k)\sigma$ is true on *DR*(*D*) for some ground substitution σ. An instance *D* is *consistent* with the schema *S* if it violates no constraint in *IC*.

A *query* over a schema is a finite set of deductive rules that define the same n-ary predicate. Given a schema *S* = (*DR*, *IC*) and an instance *D*, the *answer* to a query *Q*, defining the predicate *q*, over *S* on *D*, written $A_Q(D)$, is the set of all ground facts about *q* obtained evaluating the deductive rules from both *Q* and *DR* on *D*, i.e., $A_Q(D) = \{q(\bar{a}) \mid q(\bar{a}) \in (Q \cup DR)(D)\}$.

Mappings A *mapping M* between two schemas $A = (DR_A, IC_A)$ and $B = (DR_B, IC_B)$ is a triplet (*F*, *A*, *B*) where *F* is a finite set of formulas $\{f_1, ..., f_u\}$. Each *formula* f_i is either of the form $Q^A{}_i = Q^B{}_i$ or $Q^A{}_i \subseteq Q^B{}_i$, where $Q^A{}_i$ and $Q^B{}_i$ are queries over the schemas *A* and *B*, respectively. Obviously, both queries must be compatible, that is, the predicates they define must have the same number and type of arguments. We will assume that the deductive rules for these predicates are either on DR_A or on DR_B.

In this paper, we consider queries used to define the mapping formulas as well as IDB predicates under the exact view assumption [20], i.e., the extension of a view or query is exactly the set of tuples satisfying their definition in the database.

We say that two schema instances D_A and D_B are *consistent under a mapping M* = (*F*, *A*, *B*) if all formulas in *F* are true. We say that a formula $Q^A{}_i = Q^B{}_i$ is true if the tuples obtained as an answer to $Q^A{}_i$ on D_A are the same that the ones obtained as an answer to $Q^B{}_i$ on D_B. More formally, the formula holds when for all tuple of constants $\bar{a}$, $q^A{}_i(\bar{a}) \in AQ^A{}_i(D_A)$ if and only if $q^B{}_i(\bar{a}) \in$

$AQ^{B}_{i}(D_B)$, where q^{A}_{i} and q^{B}_{i} are the predicates defined by the two queries in the formula. Similarly, a formula $Q^{A}_{i} \subseteq Q^{B}_{i}$ is true if the tuples obtained as an answer to Q^{A}_{i} on D_A are a subset of those obtained as an answer to Q^{B}_{i} on D_B.

3. CHECKING DESIRABLE PROPERTIES OF MAPPINGS

In this section, we describe our approach to validate mappings: (1) we formalize the desirable properties that we consider, and (2) we explain how to express the fulfillment of each property in terms of predicate liveliness [13, 18, 32].

A predicate p is *lively* in a schema S if there is some consistent instance of S where at least one fact about p is true. In our case, schema S is defined in such a way that considers the two mapped schemas, A and B, and the mapping M, together. We assume that the two original schemas have different predicate names; otherwise, we can just rename the predicates. In general, the schema S is built by grouping the deductive rules and the integrity constraints from the two schemas, and then adding new constraints to make explicit the relationship stated by the mapping. Formally, $S = (DR_A \cup DR_B, IC_A \cup IC_B \cup IC_M)$ where IC_M is the set of additional constraints enforcing the mapping formulas. For each mapping formula like $Q^{A}_{i} = Q^{B}_{i}$, two constraints are needed: $\leftarrow Q^{A}_{i}(\bar{X}) \wedge \neg Q^{B}_{i}(\bar{X})$ and $\leftarrow Q^{B}_{i}(\bar{X}) \wedge \neg Q^{A}_{i}(\bar{X})$. They state that queries Q^{A}_{i} and Q^{B}_{i} may not give different answers, that is, cannot be true neither $Q^{A}_{i} \not\subseteq Q^{B}_{i}$ nor $Q^{B}_{i} \not\subseteq Q^{A}_{i}$. When formula is like $Q^{A}_{i} \subseteq Q^{B}_{i}$, just the first constraint is required: $\leftarrow Q^{A}_{i}(\bar{X}) \wedge \neg Q^{B}_{i}(\bar{X})$.

Having schema S defined, we will define for each property a new derived predicate p, such that p will be lively in S if and only if the property holds.

Next, we detail each property and its concrete reformulation in terms of predicate liveliness.

3.1. Mapping Satisfiability

As stated in [20], when constraints are considered in the global schema in a data integration context, it may be the case that the data retrieved from the sources cannot be reconciled in the global schema in such a way that both the constraints of the global schema and the mapping are satisfied.

In general, whenever we have a mapping between two schemas that have constraints there may be incompatibilities between the constraints and the mapping, or even among the mapping formulas. Therefore, checking whether there is at least one case when the mapping and the constraints are satisfied simultaneously is clearly a validation task that should be done, and this is precisely what this first desirable property of mapping does.

DEFINITION 3.1. We say that a mapping $M = (F, A, B)$ is *satisfiable* if there exists a pair of non-empty consistent instances D_A and D_B of schemas A and B, respectively, such that they are also consistent under M.

Note that the above definition explicitly avoids the trivial case when D_A and D_B are both the empty set. However, the formulas in F can still be satisfied trivially. We say that a formula $Q^{A}_{i} = Q^{B}_{i}$ is satisfied trivially when both $AQ^{A}_{i}(D_A)$ and $AQ^{B}_{i}(D_B)$ are empty sets. A formula $Q^{A}_{i} \subseteq Q^{B}_{i}$ is satisfied trivially when $AQ^{A}_{i}(D_A)$ is the empty set. Therefore, to really validate the mapping, we may ask if M is either strongly satisfiable or weakly satisfiable.

DEFINITION 3.2. A mapping $M = (F, A, B)$ is *strongly satisfiable* if all formulas in F are satisfied non-trivially. The mapping is *weakly satisfiable* if at least one formula in F is satisfied non-trivially.

EXAMPLE 3.3. In this example, we consider schemas A and B, shown in Figure 1, and the following mapping:

$M = (F, A, B)$,
where $F = \{Qa1 = Qb1, Qa2 = Qb2\}$.

Note that the queries in the mapping formulas are those also shown in Figure 1.

SCHEMA A
Tables:
employee(*emp-id*, *category*, *happiness-degree*)
category(*cat-id*, *salary*)
Constraints:
$\leftarrow category(C, S) \wedge S > 100$
$\leftarrow employee(E, C, H) \wedge \neg auxFkEmpToCat(C)$
Deductive rules:
$auxFkEmpToCat(C) \leftarrow category(C, S)$
$Qa1(E, H) \leftarrow employee(E, C, H) \wedge H > 10$
$Qa2(E, S) \leftarrow employee(E, C, H) \wedge category(C, S)$

SCHEMA B
Tables:
emp(*id*, *salary*)
happy-emp(*emp-id*, *happiness-degree*)
Constraints:
$\leftarrow happy\text{-}emp(E, H) \wedge emp(E, S) \wedge S \leq 200$
$\leftarrow happy\text{-}emp(E, H) \wedge \neg auxFkHapToEmp(E)$
Deductive rules:
$auxFkHapToEmp(E) \leftarrow emp(E, S)$
$Qb1(E, H) \leftarrow happy\text{-}emp(E, H)$
$Qb2(E, S) \leftarrow emp(E, S)$

Figure 1: Two example schemas.

Schema A has two tables: *employee*(*emp-id*, *category*, *happiness-degree*) and *category*(*cat-id*, *salary*). Table *employee* is related with table *category* via a foreign key where *employee*.*category* references *category*.*cat-id*. Table *category* has a constraint on the salary which may not be greater that 100. In schema B there are also two tables: *emp*(*id*, *salary*) and *happy-emp*(*emp-id*, *happiness-degree*). There is a foreign key from *happy-emp*.*emp-id* to *emp*.*id* and a constraint stating that all happy employees have a salary greater than 200. Mapping M relates instances of A and B where all *employee*s with a happiness degree greater than 10 on A are *happy-emp*s on B, and all *employee*s on A are *emp*s on B (together with their salary).

Note in Figure 1 that expressing a foreign key constraint requires the use of a derived predicate (*auxFkEmpToCat* and *auxFkHapToEmp*) because we require negated literals to be safe.

In this example, we can see that mapping M is not strongly satisfiable. The first formula, $Qa1 = Qb1$, can only be satisfied trivially. First, in schema B all *happy-emp*s must have a salary over 200 while, in schema A, all *employee*s, no matter their happiness degree, have a maximum salary of 100. At the same time, formula $Qa2 = Qb2$, enforces that employees should have the same salary in both sides.

In contrast, the second formula is non-trivially satisfiable and, thus, M is weakly satisfiable. That is because there may be *employee*s on A with a happiness degree of 10 or lower, and *emp*s on B that are not *happy-emp*s.

Notice also that if we remove the second formula in the mapping, i.e. $Qa2 = Qb2$, and kept the first one alone, then the resulting mapping would be strongly satisfiable. This is an example of how satisfiability of one mapping formula may be affected by the rest of formulas in the mapping. □

To formulate this mapping satisfiability property in terms of checking predicate liveliness, we build a new schema S grouping schemas A, B, and the mapping M:

$$S = (DR_A \cup DR_B,\ IC_A \cup IC_B \cup IC_M),$$

and define a new derived predicate in order to check whether it is lively in S. Given that there are two types of satisfiability, strong and weak, we define two new predicates, one for each case.

Assuming that $F = \{f_1, ..., f_n\}$ and that we want to check strong satisfiability, we define the new derived predicate *strong_sat*, as follows:

$$strong\_sat \leftarrow Q^A{}_1(\bar{X}_1) \wedge ... \wedge Q^A{}_n(\bar{X}_n)$$

where no variable in $\bar{X}_i$ appears in $\bar{X}_j$, $j \neq i$.

In a similar way, for the case of weak satisfiability, we would define the new derived predicate *weak_sat*, as follows: $weak\_sat \leftarrow Q^A{}_1(\bar{X}_1) \vee ... \vee Q^A{}_n(\bar{X}_n)$, but due to the fact that rules' bodies must be conjunctions of literals, it should be defined with the following deductive rules:

$$weak\_sat \leftarrow Q^A{}_1(\bar{X}_1),$$
$$...,$$
$$weak\_sat \leftarrow Q^A{}_n(\bar{X}_n).$$

Figure 2 shows the schema we obtain when expressing Example 3.3 in terms of predicate liveliness. Note that the schema contains the deductive rule for the derived predicate *strong_sat*.

PROPOSITION 3.4. *The derived predicate strong_sat/weak_sat is lively in schema $S = (DR_A \cup DR_B,\ IC_A \cup IC_B \cup IC_M)$ if and only if the mapping $M = (F, A, B)$ is strongly/weakly satisfiable.*

PROOF. Let us suppose strong satisfiability. Therefore, we know that there exist a pair of instances not violating the constraints on A and B, making true all mapping formulas. Strong satisfiability implies that all formulas are satisfied non-trivially, so we can say that all $Q^A{}_i$ queries on F have at least one tuple on its answer. Then, by definition *strong_sat* will be true. Therefore, we have found an instance of schema S where *strong_sat* is true and that proves that *strong_sat* is lively in S. On the other hand, to get *strong_sat* lively in S, it is required by the definition of *strong_sat* the existence of a consistent instance of S where each query $Q^A{}_i$ contains at least one tuple in its answer. By definition of S, that means that there exists a consistent instance of A and a consistent instance of B such that these two instances are also consistent under the mapping, that is, the mapping is satisfiable. Moreover, all $Q^A{}_i$ queries have at least one answer so all mapping formulas are satisfied non-trivially. Thus, we can conclude that the mapping is strongly satisfiable.

The proof for the case of weak satisfiability is very similar to the previous one. □

SCHEMA *S*

Deductive rules:

$$strong\_sat \leftarrow Qa1(X, Y) \wedge Qa2(Z, U)$$

$$\left.\begin{array}{l} auxFkEmpToCat(C) \leftarrow category(C, S) \\ Qa1(E, H) \leftarrow employee(E, C, H) \wedge H > 10 \\ Qa2(E, S) \leftarrow employee(E, C, H) \wedge category(C, S) \end{array}\right\} DR_A$$

$$\left.\begin{array}{l} auxFkHapToEmp(E) \leftarrow emp(E, S) \\ Qb1(E, H) \leftarrow happy\text{-}emp(E, H) \\ Qb2(E, S) \leftarrow emp(E, S) \end{array}\right\} DR_B$$

Constraints:

$$\left.\begin{array}{l} \leftarrow category(C, S) \wedge S > 100 \\ \leftarrow employee(E, C, H) \wedge \neg auxFkEmpToCat(C) \end{array}\right\} IC_A$$

$$\left.\begin{array}{l} \leftarrow happy\text{-}emp(E, H) \wedge emp(E, S) \wedge S \leq 200 \\ \leftarrow happy\text{-}emp(E, H) \wedge \neg auxFkHapToEmp(E) \end{array}\right\} IC_B$$

$$\left.\begin{array}{l} \leftarrow Qa1(A, B) \wedge \neg Qb1(A, B) \\ \leftarrow Qb1(A, B) \wedge \neg Qa1(A, B) \\ \leftarrow Qa2(A, B) \wedge \neg Qb2(A, B) \\ \leftarrow Qb2(A, B) \wedge \neg Qa2(A, B) \end{array}\right\} IC_M$$

Figure 2: Example 3.3 in terms of predicate liveliness

3.2. Mapping Inference

The mapping inference property was identified in [22] as an important property of mappings. It consists on checking whether a mapping formula is entailed by a given mapping, that is, whether it does not add new mapping information. One application of the property would be that of checking whether a formula in the mapping is redundant, that is, entailed by the other formulas. Another application would be that of checking the equivalence of two different mappings. We can say that two mappings are equivalent if the formulas in the first mapping entail all the formulas in the second, and the vice versa.

The results presented in [22] are in the setting of conjunctive queries without considering integrity constraints in the schemas. The proof for the complexity result shows that the property can be checked in this setting by means of finding a maximally contained rewriting and query containment. Here, we consider a broader class of queries along with the presence of constraints in the schemas.

DEFINITION 3.5. (see [22]) Let a formula f be defined over schemas A and B. Formula f is *inferred* from a mapping $M = (F, A, B)$ if all pair of instances of A and B consistent under M also satisfy the formula f.

EXAMPLE 3.6. We take schemas A and B (on Figure 1), but without the salary constraints. We define a new mapping:

$M_2 = (F_2, B, A)$,
where F_2 contains just one mapping formula, $Qb2 = Qa2$,

and the queries on this formula have the same definition than in Example 3.3.

We consider also a new formula $f1$, $q1 \subseteq q2$, where $q1$ and $q2$ are queries over B and A, respectively, with the following definitions:

$q1(E) \leftarrow happy\text{-}emp(E, H)$
$q2(E) \leftarrow employee(E, C, H)$

The foreign key on schema *B*, from *happy-emp* to *emp*, guarantees that all *happy-emp*s are also *emp*s so, if the mapping formula in M_2 holds, that means they are also *employee*s on the corresponding instance of schema *A*. Thus, it is true what formula *f1* states: the employees' identifiers on table *happy-emp* are a subset of those on table *employee*. Therefore *f1* is entailed by mapping M_2.

Let us consider now another formula *f2*, $q3 \subseteq q4$, having *q3* and *q4* the following definitions:

$q3(E, H) \leftarrow happy\text{-}emp(E, H)$
$q4(E, H) \leftarrow employee(E, C, H)$

This time, *f2* is not entailed by mapping M_2. The difference is that we are not projecting just the employee's identifier as before, but also the happiness degree, and given that the formula on the mapping does not care about the happiness degree, we can build a counterexample showing that entailment of *f2* does not hold. This counterexample consists of a pair of instances of *A* and *B, D_A* and D_B, that satisfy the mapping formula in M_2 but that do not satisfy *f2* like, for instance, the following EDBs: D_A = {*employee*(0, 0, 5), *category*(0, 50)} and D_B = {*emp*(0, 50), *happy-emp*(0, 10)}. It is not difficult to show that in this counterexample the formula in mapping M_2 holds: $A_{qb2}(D_B) = \{qb2(0, 50)\}$ and $A_{qa2}(D_A) = \{qa2(0, 50)\}$, but *f2* does not: $A_{q3}(D_B) = \{q3(0, 10)\}$ and $A_{q4}(D_A) = \{q4(0, 5)\}$. □

Expressing the mapping inference property in terms of predicate liveliness is best done by checking the negation of the property (i.e., the lack of inference) instead of checking the property directly. The negated property states that a certain formula *f* is *not inferred* from a mapping *M* if there exists a pair of schema instances *A* and *B*, consistent under *M*, such that they do not satisfy *f*. In this way, the derived predicate for checking liveliness must be the negation of *f*. When *f* has the form $qa = qb$, then we define the new derived predicate *map_inf* with the following two deductive rules:

$map\_inf \leftarrow qa(\bar{X}) \wedge \neg qb(\bar{X})$
$map\_inf \leftarrow qb(\bar{X}) \wedge \neg qa(\bar{X})$.

Otherwise, when *f* has the form $qa \subseteq qb$, just the following deductive rule is needed:

$map\_inf \leftarrow qa(\bar{X}) \wedge \neg qb(\bar{X})$.

We define the new schema *S* as usual, putting together the deductive rules and constraints from schemas *A* and *B*, and considering additional constraints to enforce mapping formulas:

$S = (DR_A \cup DR_B, IC_A \cup IC_B \cup IC_M)$.

The schema that results from the reformulation of Example 3.6 in terms of predicate liveliness is similar to the one shown in Figure 2 but removing the constraints about salary and adding the rules defining the predicate *map_inf*.

PROPOSITION 3.7. *The derived predicate map_inf will be lively in S if and only if the formula f is not inferred from the mapping M.*

PROOF. Let us assume *f* has the form $qa = qb$. By the definition of lively, when *map_inf* is lively in *S*, there exists a consistent instance of *S* where *map_inf* is true. By construction of *S*, the existence of a consistent instance of *S* leads to the existence of a pair of consistent instances of *A* and *B* such that they are also consistent under the mapping. By the definition of *map_inf*, and given that *map_inf* is true, there exists a tuple either belonging to the answer of *qa* but not belonging to the answer of *qb*, or belonging to the answer of *qb* but not to the answer of *qa*. Thus, by definition of *f*, this pair of instances does not satisfy *f*.

On the other hand, let us suppose that there exist a pair of instances of *A* and *B* consistent under the mapping but that do not satisfy the formula *f*. By construction of *S*, there is a consistent instance of *S* for which the formula *f* does not hold. That means that $qa \neq qb$, i.e., either $qa \nsubseteq qb$ or $qb \nsubseteq qa$. Then, by definition of *map_inf*, *map_inf* is true in this instance of *S*, i.e., *map_inf* is lively in *S*.

The proof for the case when *f* is like $qa \subseteq qb$ can be immediately obtained from the previous one. □

3.3. Query Answerability

In this section we consider the query answerability property, introduced in [22] as an important property of mappings. The intuition behind this property is that a mapping that is partial or incomplete can be, however, adequate for a certain task. This task will be represented by means of a certain query. The property will check whether the mapping enables answering of this query over the schemas being mapped. While the previous two properties are intended to validate the mapping without considering its context, this property validates the mapping with respect to the use for which it is designed.

As with mapping inference, the results presented in [22] are in the setting of conjunctive queries without constraints in the schemas. The proof for the complexity result showed that the property can be checked by means of the existence of an equivalent rewriting. As in the previous case, we consider a broader class of queries along with the presence of constraints in the schemas.

DEFINITION 3.8. (see [22]) Let be *Q* a query over a schema *A*. The mapping *M* = (*F*, *A*, *B*) *enables query answering* of *Q* if the following holds. Let D_B be a consistent instance of *B*. Let $\mathcal{D}_A$ be the set of instances of schema *A* consistent under *M* with D_B. Then, $A_Q(D_A) = A_Q(D_A')$ for every pair of consistent instances D_A, $D_A' \in \mathcal{D}_A$.

Intuitively, the mapping enables answering the query when, given a consistent instance of schema *B*, the answer for *Q* over schema *A* gets uniquely determined. In other words, when the query answerability property holds for a given query *Q* over a schema *A*, the exact answer for *Q* over an instance D_A can be computed having only access to the tuples in the corresponding mapped instance D_B.

EXAMPLE 3.9. We consider again schemas *A* and *B* shown in Figure 1, but without the salary constraints. We consider also the mapping *M* from Example 3.3 and a query *Q* defined over schema *A* as follows:

$q(E) \leftarrow employee(E, C, H) \wedge H > 5$

It can be seen that mapping *M* does not enable answering of this query *Q*. The intuition is that the mapping only takes care of those employees of schema *A* who have a happiness degree greater than 10, while the evaluation of the query *Q* needs also access to the employees with a happiness degree between 5 and 10. Thus, we can build a counterexample consisting of three consistent instances: one instance D_B of schema *B* and two instances of schema *A*, D_A and D_A'. The instances D_A (D_A') and D_B of the counterexample are consistent under the mapping *M* but the answer of the query *Q* is not the same in both instances of the schema *A*, i.e., $A_Q(D_A) \neq A_Q(D_A')$. These criteria are satisified, for instance, by the following EDBs: D_B = {*emp*(0, 150), *emp*(1, 200), *happy-emp*(1, 15)}, D_A = {*employee*(0, 0, 6), *employee*(1, 1, 15), *category*(0, 150), *category*(1, 200)} and D_A' = {*employee*(0, 0, 4), *employee*(1, 1, 15), *category*(0, 150), *category*(1, 120)}. It is easy to see that those instances are consistent and that they satisfy the mapping formulas. We can verify that this is certainly a counterexample because $A_Q(D_A)$ = {*q*(0), *q*(1)} but $A_Q(D_A')$ = {*q*(1)}, that is, the answers to the query do not match. □

SCHEMA *S*

Like the one in Figure 2 but removing the two constraints about salary and adding the following rules and constraints.

Additional deductive rules:

$q\_answer \leftarrow Q(X) \wedge \neg Q'(X)$

$Q(E) \leftarrow employee(E, C, H) \wedge H > 5$

$$\left.\begin{array}{l} Q'(E) \leftarrow employee'(E, C, H) \wedge H > 5 \\ auxFkEmpToCat'(C) \leftarrow category'(C, S) \\ Qa1'(E, H) \leftarrow employee'(E, C, H) \wedge H > 10 \\ Qa2'(E, S) \leftarrow employee'(E, C, H) \wedge category'(C, S) \end{array}\right\} DR_{A'}$$

Additional constraints:

$$\left.\begin{array}{l} \leftarrow employee'(E, C, H) \wedge \neg auxFkEmpToCat'(C) \end{array}\right\} IC_{A'}$$

$$\left.\begin{array}{l} \leftarrow Qa1'(A, B) \wedge \neg Qb1(A, B) \\ \leftarrow Qb1(A, B) \wedge \neg Qa1'(A, B) \\ \leftarrow Qa2'(A, B) \wedge \neg Qb2(A, B) \\ \leftarrow Qb2(A, B) \wedge \neg Qa2'(A, B) \end{array}\right\} IC_{M'}$$

Figure 3: Example 3.9 in terms of predicate liveliness

As seen in the example, the query answerability property is also easier to check by means of its negation.

First, we need to define the new schema *S*. In this case, given that the counterexample requires finding two instances of a same schema, we need to extend the definition of schema *S*, as follows:

$S = (DR_A \cup DR_{A'} \cup DR_B, IC_A \cup IC_{A'} \cup IC_B \cup IC_M \cup IC_{M'})$

where $A' = (DR_{A'}, IC_{A'})$ is a copy of schema $A = (DR_A, IC_A)$ just renaming all predicates (e.g. renaming predicate *q* by predicate *q'*) and, similarly, *M'* is a copy of mapping *M* renaming those predicates in the mapping formulas that are predicates from schema *A*.

Then, we define the derived predicate *q_answer* with the following deductive rule:

$q\_answer \leftarrow q(\bar{X}) \wedge \neg q'(\bar{X})$,

where *q* is the predicate defined by *Q* and *q'* is its renamed version over schema *A'*.

Figure 3 shows the additional deductive rules and constraints that we get when reformulating Example 3.9 in terms of database schema validation, with respect to the schema shown in Figure 2.

PROPOSITION 3.10. *The derived predicate q_answer is lively in S if and only if the mapping M does not enable answering of query Q.*

PROOF. Let us assume that *q_answer* is lively in *S*. That means that there exists a consistent instance of *S* where *q_answer* is true. By construction of *S*, there exists an instance D_B of *B*, an instance D_A of *A*, and an instance $D_{A'}$ of *A'*, such that they are all consistent and, D_A and $D_{A'}$ are both consistent with D_B under mappings *M* and *M'*, respectively. By definition of *q_answer*, there exists a ground fact $q(\bar{a})$ such that $q(\bar{a}) \in A_Q(D_A)$ but $q'(\bar{a}) \notin A_{Q'}(D_{A'})$. Given that schema *A'* is in fact a copy of schema *A*, for each instance of schema *A'* there exists an identical one in schema *A*. Thus, $D_{A'}$ can be seen as an instance of schema *A*, D_A', in such a way that, if it was previously consistent with D_B under *M'* it is also now consistent with D_B under *M*, and for all $q'(\bar{a})$ previously in the answer of *Q'* exists now a $q(\bar{a})$ in the answer of *Q*. Therefore, we found two instances of schema *A*, D_A and D_A', consistent under the mapping *M* with a given instance D_B of schema *B*, such that $A_Q(D_A) \not\subseteq A_Q(D_A')$. The conclusion is that *M* does not enable answering of *Q*. The other direction can be easily proved by inverting the reasoning. □

3.4. Mapping Losslessness

As we have seen, query answerability determines whether two mapped schemas are equivalent with respect to a certain set of queries in the sense that they obtain the same answers on both sides. However, in certain contexts this may be too restrictive. Consider data integration [20] for instance and assume that for security reasons it must be known whether some sensitive local data is exposed by the integrator system. Clearly, in such a situation, the answer to a query intended to retrieve such sensitive data from one of the sources is not expected to obtain the exact answer that would be obtained if the query were directly executed over the global schema. Therefore, such a query is not answerable under the terms specified above. Nevertheless, sensitive local data are in fact exposed if the query can really be computed over the global schema. Thus, a new property that captures this idea is needed.

In fact, when a mapping contains formulas like $Q^A_i \subseteq Q^B_i$, checking query answerability does not always provide the designer with useful information. Let us illustrate this with an example.

EXAMPLE 3.11. We consider schemas *A* and *B* shown in Figure 1, and a new mapping:

$M_3 = (F_3, A, B)$,
where $F_3 = \{Qa1 \subseteq Qb1, Qa2 \subseteq Qb2\}$
and *Qa1*, *Qb1*, *Qa2*, *Qb2* are the queries also in shown in Figure 1.

Note that mapping M_3 is similar to the previous mapping *M* (see Example 3.3), but the = operator is replaced by the ⊆ operator.

We also consider the query *Q*, which is defined as follows:

$q(E) \leftarrow employee(E, C, H)$

It is not difficult to see that the mapping M enables answering of query Q. The query selects all the employees in schema A and M states that all employees on A are also emps in B. Thus, when an instance of schema B is given, the extension of the table *employee* in schema A is uniquely determined and, consequently, also the answer to Q.

However, if we consider mapping M_3, the extension of the table employee is not uniquely determined when an instance of B is given. In fact, it can be any subset of the emps in the given instance of B. For example, let D_B be an instance of schema B such that $D_B = \{emp(0, 70), emp(1, 40)\}$, and let D_A and D_A' be two instances of schema A such that $D_A = \{employee(0, 0, 5), category(0, 70)\}$ and $D_A' = \{employee(1, 0, 5), category(0, 40)\}$. Therefore, we have come up with a counterexample and, thus, M_3 does not enable answering of Q. □

The previous example shows that when the mapping contains formulas like $Q^A_i \subseteq Q^B_i$, giving an instance of schema B does not result, in general, on uniquely determining the answer of a certain query Q over schema A. This is because for a given instance of B, there is just one possible answer for each query $Q^B_1,...,Q^B_n$ in the mapping. However, due to the $\subseteq$ operator, there is more than one possible answer for the queries $Q^A_1,...,Q^A_n$. Something similar would happen if Q were defined over schema B. Thus, query answerability does not hold in general for mappings of this type. The intuitive reason for this is that query answerability holds when we are able to compute the exact answer for Q over an instance D_A in which we only have access to the tuples in the corresponding mapped instance D_B. However, if any of the mapping formulas have the $\subseteq$ operator, we cannot know just by looking at D_B which tuples are also in D_A and we are therefore unable to compute an exact answer.

DEFINITION 3.12. We say that a mapping $M = (F, A, B)$ is *lossless* with respect to a query Q defined over schema A if, for every pair of consistent instances of schema A, D_A and D_A', the following holds: if there exists an instance D_B of schema B consistent under M with both D_A, and D_A', and also $AQ^A_i(D_A) = AQ^A_i(D_A')$ for $1 \le i \le$ n, then, it is true that $A_Q(D_A) = A_Q(D_A')$.

Informally, when the *mapping losslessness* property holds for a given query Q over a schema A, an answer to Q can be computed by only accessing the tuples in the corresponding mapped instance D_B. Notice that if the query selects tuples from a table that doesn't participate in the mapping, the property clearly does not hold.

EXAMPLE 3.13. We again consider the mapping M_3 and the query Q from Example 3.11. We saw that the query answerability property does not hold for this mapping. Let us now check the mapping losslessness property.

Let us assume that we have two consistent instances of schema A, D_A and D_A', and an instance D_B of schema B consistent under M with both, such that the answers to *Qa2* and *Qa1* are exactly the same over D_A and D_A'. Assume now that Q obtains $q(0)$ over D_A but not over D_A'. According to the definition of Q, it follows that D_A contains a least a tuple of employee, say *employee*(0, 0, 12), that D_A' does not contain. Since D_A is consistent with the integrity constraints, it must also contain its corresponding tuple of predicate category, say *category*(0, 20). Thus, according to the definition of *Qa2* the answer *Qa2*(0, 20) would be obtained over D_A but no over D_A'. This clearly contradicts our initial assumption and, thus, M_3 is lossless with respect to Q. □

To formulate the mapping losslessness property in terms of checking the liveliness of a derived predicate, we define the new schema S in a similar way as we did for query answerability:

$$S = (DR_A \cup DR_{A'} \cup DR_B,\ IC_A \cup IC_{A'} \cup IC_B \cup IC_M \cup IC_L),$$

where schema A' is a copy of A in which the predicates are renamed, and IC_M is the set of constraints that enforce the mapping formulas in M. By IC_L we denote the set of constraints that force A and A' to share the same answers for the queries in the mapping:

$$\begin{aligned} IC_L = \{ & \leftarrow Q^A{}_1(\bar{X}_1) \wedge \neg Q^A{}_1{}'(\bar{X}_1),\ \leftarrow Q^A{}_1{}'(\bar{X}_1) \wedge \neg Q^A{}_1(\bar{X}_1), \\ & \ldots, \\ & \leftarrow Q^A{}_n(\bar{X}_n) \wedge \neg Q^A{}_n{}'(\bar{X}_n),\ \leftarrow Q^A{}_n{}'(\bar{X}_n) \wedge \neg Q^A{}_n(\bar{X}_n)\} \end{aligned}$$

Let Q be the query we wish to use for checking losslessness and let Q' be a copy of Q in which the predicates are renamed, as we did with A'. We define a derived predicate *map_loss* as follows:

$$map\_loss \leftarrow Q(\bar{X}) \wedge \neg Q'(\bar{X}).$$

Figure 4 shows the additional deductive rules and constraints that we obtain when reformulating Example 3.13 in terms of database schema validation, with respect to the schema shown in Figure 2.

<u>SCHEMA *S*</u>

Like the one in Figure 2 but removing the salary constraints, and adding the following rules and constraints. Note that we replace IC_M by IC_{M3}.

Additional deductive rules:

$map\_loss \leftarrow Q(X) \wedge \neg Q'(X)$

$Q(E) \leftarrow employee(E, C, H)$

$$\left.\begin{array}{l} Q'(E) \leftarrow employee'(E, C, H) \\ auxFkEmpToCat'(C) \leftarrow category'(C, S) \\ Qa1'(E, H) \leftarrow employee'(E, C, H) \wedge H > 10 \\ Qa2'(E, S) \leftarrow employee'(E, C, H) \wedge category'(C, S) \end{array}\right\} DR_{A'}$$

Additional constraints:

$$\left.\begin{array}{l} \leftarrow employee'(E, C, H) \wedge \neg auxFkEmpToCat'(C) \end{array}\right\} IC_{A'}$$

$$\left.\begin{array}{l} \leftarrow Qa1(A, B) \wedge \neg Qa1'(A, B) \\ \leftarrow Qa1'(A, B) \wedge \neg Qa1(A, B) \\ \leftarrow Qa2(A, B) \wedge \neg Qa2'(A, B) \\ \leftarrow Qa2'(A, B) \wedge \neg Qa2(A, B) \end{array}\right\} IC_L$$

$$\left.\begin{array}{l} \leftarrow Qa1(A, B) \wedge \neg Qb1(A, B) \\ \leftarrow Qa2(A, B) \wedge \neg Qb2(A, B) \end{array}\right\} IC_{M3}$$

Figure 4: Example 3.13 in terms of predicate liveliness

PROPOSITION 3.14. *The derived predicate map_loss is lively in S if and only if the mapping M is not lossless with respect to Q.*

Note that we check for the existence of a counterexample.

PROOF. Let us assume that *map_loss* is lively in *S*. Hence, there exists an instance of *S* in which *map_loss* is true. This means that the answer to *Q* has a tuple that is not in the answer to *Q'*. By construction of schema *S*, we can build from the instance of *S*, three instances, D_A for *A*, $D_{A'}$ for *A'* and D_B for *B*. Given that *A* and *A'* are in fact the same schema, just with different predicate names, and also queries *Q* and *Q'* are the same query, we can conclude that $D_{A'}$ is also an instance for *A*, D_A*'*, and that the query *Q* evaluated over D_A returns a tuple that is not returned when evaluated over D_A*'*. Thus, we have two instances for schema *A*, both consistent under mapping *M* with a third instance of schema *B*, in such a way that both instances have the same answer for the queries in the mapping but not for the query *Q*. According to the definition of mapping losslessness, *M* is not lossless w.r.t. *Q*. The other direction can be proved by following the reasoning backwards. □

The mapping losslessness property is the result of adapting the property of *view losslessness* or *determinacy* [7, 28]. A set of views *V* is lossless with respect to a query *Q*, under the exact view assumption, if for every pair of database instances having the same extensions for the views in *V*, they also have the same answer for *Q*.

In our case, the mapping losslessness property checks whether the set of queries $V = \{Q^A_1,...,Q^A_n\}$ is lossless with respect to the query *Q*, but with the additional requirement that the extensions for the queries in *V* must also ensure the existence of a consistent instance for schema *B*.

Finally, it can be seen that in the cases in which all the formulas in the mapping have the form $Q^A_i = Q^B_i$, the mapping losslessness and the query answerability properties, are equivalent.

PROPOSITION 3.15. *Let Q be a query over schema A, and let M = (F, A, B) be a mapping where F =* $\{f_1,...,f_n\}$ *and* f_i *is* $Q^A_i = Q^B_i$ *for* $1 \leq i \leq n$. *Mapping M is lossless w.r.t Q if and only if M enables answering of Q.*

PROOF. Let us assume that a mapping *M* is lossless w.r.t a query *Q*, and let us suppose that mapping *M* does not enable answering the query *Q*. By the negation of query answerability, there exists an instance D_B of *B* and a pair of instances, D_A and D_A*'*, of *A*, such that D_A and D_A*'* are both consistent under *M* with D_B but $A_Q(D_A) \neq A_Q(D_A')$. Given that all mapping formulas are like $Q^A_i = Q^B_i$, it's true that $AQ^A_i(D_A) = AQ^A_i(D_A')$ for $1 \leq i \leq$ n. Hence, instances D_A, D_A*'* and D_B are a counterexample for mapping losslessness and we reached a contradiction. In the other way, let us assume now that mapping *M* enables answering the query *Q*, and let us suppose *M* is not lossless w.r.t *Q*. By the negation of losslessness, there are two instances of *A*, D_A and D_A*'*, in which $AQ^A_i(D_A) = AQ^A_i(D_A')$ for $1 \leq i \leq$ n, there exists a third instance D_B of *B* consistent under *M* with both instances, and such that $A_Q(D_A) \neq A_Q(D_A')$. In this case, the three instances are directly a counterexample for query answerability. Thus, we reached a contradiction again. □

4. EXPERIMENTAL EVALUATION

The main goal of this section is to show the feasibility of our approach by means of some experiments. We have used the CQC Method [12], more precisely its implementation in the Schema Validation Tool prototype [30], to perform the previous tests in different situations.

We first provide a brief overview of the CQC Method and the SVT. and we explain how to use them to perform liveliness tests and, therefore, to effectively check the desirable properties of mappings we defined in the previous section. Then, we describe some experiments we have performed using SVT for validating mappings and comment on the results.

4.1. CQC Method and SVT

The CQC (Constructive Query Containment) Method [12], originally defined for query containment, performs a validation test by trying to build a consistent instance for a database schema in order to satisfy a given goal (a conjunction of literals). It is able to deal with database schemas having integrity constraints, safe-negated EDB and IDB literals, and comparisons.

The method starts with the empty instance and uses different *Variable Instantiation Patterns* (VIPs), according to the syntactic properties of the views/queries and constraints in the schema, to generate only the relevant facts to be added to the instance under construction. If the method is able to build an instance that satisfies all literals in the goal and does not violate any constraint, then that instance is a solution and it shows that the goal is satisfiable. The key point is that the VIPs guarantee that if instantiating the variables in the goal using the constants they provide the method does not find any solution, then no solution exists.

As proved in [12], the CQC Method always terminates when there is a finite consistent instance satisfying the goal, or when the goal is unsatisfiable.

SVT (Schema Validation Tool) [30] is a prototype tool designed to perform some validation tests on database schemas, in particular the liveliness test in which we are interested here. It accepts the following subset of the SQL language:

- Primary key, foreign key, boolean check constraints.
- SPJ views, negation, subselects (exists, in), union.
- Data types: integer, real, string.

The current implementation of SVT assumes a set semantics of views and queries and it does not allow null values neither aggregate nor arithmetic functions.

SVT implements the CQC Method as a backtracking algorithm. It adds facts to the EDB under construction in order to make true the literals in the goal. After adding a new fact, it checks if the EDB violates some constraint. When it detects that some constraint is violated or some literal in the goal is evaluated to false (e.g. a comparison), it backtracks and reconsiders the last decision. Some constraints, like foreign keys, can be repaired by adding new literals to the goal and thus no backtracking is required in these cases.

Using the CQC Method, and thus SVT, for checking the properties of mappings is easy once we have redefined them in terms of predicate liveliness. We just have to provide the schema and the goal. The schema will be the schema *S* we explained how to construct in Section 3. The goal will be only one literal corresponding to the derived predicate we defined in Section 3 for each property.

4.2. Experiments

We have experimentally evaluated the behavior of our approach for validating mappings by means of some experiments using SVT. We executed our experiments on an Intel Core 2 Duo, 2.16 GHz machine with Windows XP (SP2) and 2GB RAM. Each

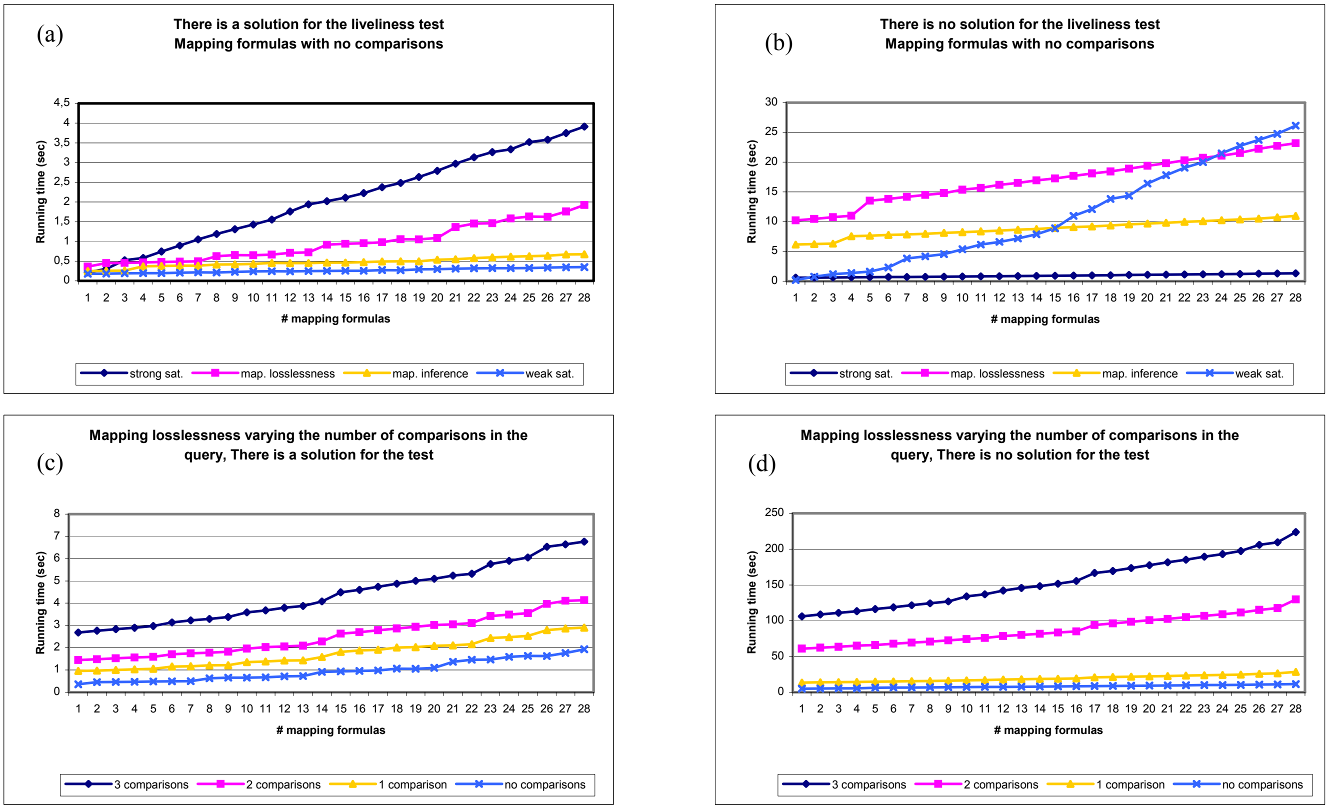

Figure 5: (a-b) Comparison in performance between the properties and (c-d) study of the effect in mapping losslessness of increasing query complexity by means of comparisons.

experiment was repeated three times and we report the average of these three trials.

The experiments were designed with the goal of measuring the influence of two parameters: (1) the size of the mapping, and (2) its complexity.

We focused on the setting where the two sides of the mapping are of similar complexity. Therefore, if we increase the complexity of the mapping, we should increase the complexity in both sides.

We designed the scenario for the experiments using the relational schema of the **Mondial** database [26], which models geographic information. The schema consists in 28 tables with 38 foreign keys. We consider each table with their primary key, unique and foreign key constraints.

The scenario consists in two "copies" of the Mondial database schema that play the roles of schema *A* and schema *B*, respectively. The mapping between the two schemas contains one formula for each table in the Mondial database. The mapping states that each table of one schema is equal to its copy in the other schema. Thus, the mapping consists in 28 formulas all with the = operator, and all the queries in the mapping follow the pattern: $q(\bar{X}) \leftarrow R(\bar{X})$, where R is a table from the corresponding schema.

Note that in this scenario the reformulation in terms of predicate liveliness of both query answerability and mapping losslessness is identical, i.e., $IC_{M'} = IC_L$. The results shown in the graphics for mapping losslessness are thus also applicable to query answerability (not shown in the graphics). In fact, it is expected that when the complexity of the schemas is similar, and also the complexity of the queries in both sides of the mapping, the schemas resulting from the reformulation of query answerability and mapping losslessness be also of similar complexity.

In Figure 5(a), we show the results of the first experiment, which consists on the execution of the liveliness tests for the properties: strong satisfiability, mapping losslessness, mapping inference and weak satisfiability, with an increasing number of mapping formulas. We started the experiment considering just one of the 28 formulas, which we selected randomly. Then, we executed the test again but with two formulas, the previous one and another also selected randomly among the remaining ones, and so on. In this experiment, we consider mapping formulas with queries that have no order comparisons. We also focus in this experiment on the case where the liveliness tests have a solution. Note that finding a solution means something different depending on the property we are checking. Finding a solution for mapping satisfiability means that the property holds, and the found solution is an EDB showing that. For the other properties, finding a solution means that a counterexample has been found and so the property does not hold.

In Figure 5(a), the two variants of mapping satisfiability, strong and weak, can be checked without any change in the mapping because both already hold. Instead, to check mapping inference and mapping losslessness, it is required to find a formula and a query, respectively, for which the properties do not hold. In the case of mapping inference we used the following formula: $q1 = q2$ where $q1(\bar{X}) \leftarrow R(\bar{X}) \wedge X_i \neq K1$, $q2(\bar{X}) \leftarrow R'(\bar{X}) \wedge X_i \neq K2$, and *R* is a selected randomly table from schema *A*, *R'* is the corresponding copy of *R* in schema *B*, $X_i \in \bar{X}$, and *K1* and *K2* are two different fresh constants. We used this formula, as a parameter for checking the mapping inference property because is very similar to the formulas already in the mapping. We add the inequalities to make the property fail while keeping the changes in the mapping at minimum. We add an inequality in both queries with the goal of keeping the two sides of the formula with the same complexity. In the case of mapping losslessness, we generated a parameter query that selects all the tuples from one random table from schema *A*. We also modified the corresponding formula that maps this table with its copy in schema *B*, in such a way that now the formula maps all the columns in the table except one.

We can see in Figure 5(a) that the strong version of mapping satisfiability is slower than the weak one. This is expected since strong satisfiability requires checking all formulas to be sure that all of them can be satisfied non-trivially, but weak satisfiability can stop the checking after finding one. We can also see that strong satisfiability is clearly slower than mapping losslessness and mapping inference. That is because these two properties have an additional parameter: a query and a formula respectively, and in order to check the properties SVT has to deal only with the fragment of the schema that is "affected" by the parameter query or formula. On the other hand, strong satisfiability has to deal with the whole part of the schema that participates in the mapping, which in our scenario is the entire schema.

Comparing mapping inference and mapping losslessness in Figure 5(a), we can see that losslessness is slower. This is the expected behavior since if we remind the reformulation of each property in terms of liveliness we will see how the schema *S* for losslessness is formed by grouping three schemas, while in the case of mapping inference it is formed by grouping only two. Thus, there is a significant difference of size between the resulting schemas, which explains the gap between the computing times.

In Figure 5(b), we can see the same experiment as in the previous figure but for the case when there is no solution for the liveliness tests. To make the two satisfiability properties fail in this second experiment, we added to each table in schema *A*, a check constraint requiring that one of the columns must be greater than some constant. We add the same constraint to the corresponding copy of the table in schema *B*. We also modified the mapping formulas in such a way that the two queries in the formula are forced to select those tuples that violate the check constraints. In the case of mapping inference, we used one of the formulas already in the mapping as a parameter. In this way, the property does not hold. In the case of mapping losslessness, to make the property fail we used a query that selects all tuples from a randomly selected table (we did not make any modification in the mapping).

The first thing we can observe in Figure 5(b) is the global increment of all computing times. This is not unexpected since the SVT must try all the relevant instances provided by the VIPs before concluding that no solution exists. In the previous experiment, the search stopped when a solution was found. It is worth to note that strong satisfiability and weak satisfiability has exchanged their roles. Now the weak version of the property is slower than the strong one. The intuitive explanation is that strong satisfiability can stop as early as it finds a formula that cannot be satisfied non-trivially, while weak satisfiability must continue the searching until all the formulas have been considered.

In Figure 5(c) and Figure 5(d), we show the results of two experiments where it is measured the effect of increasing the complexity of the parameter query when checking mapping losslessness. The same experiments for the case of mapping inference would be similar (graphics not shown). Figure 5(c) shows the case where the there is a solution for the liveliness test and Figure 5(d) shows the case where no solution exists. In each experiment the query we used to check mapping losslessness with respect to, is the same than in Figure 5(a) and 5(b) respectively. The increment of complexity consists on adding comparisons to the query definition. These comparisons are like $X > K$, where *X* is a variable corresponding to one column of the underlying table, and *K* is a fresh constant.

In Figure 5(c), the increasing in the number of comparisons results in a greater computation time. This is expected since more constants in the definitions of the queries means a greater number of constants provided by the VIPs. We can see a significant gap between the cases of 2 and 3 comparisons. In Figure 5(d) a gap already appears between the cases of 1 and 2 comparisons. It is not unexpected that the increment of computing time grows when we add more comparisons since the VIPs provide more constants and thus there are more combinations to try when instantiating a literal from the goal (Figure 5(c)). If in addition to that, there is no solution, computing time increases even faster with respect to the number of comparisons (Figure 5(d)).

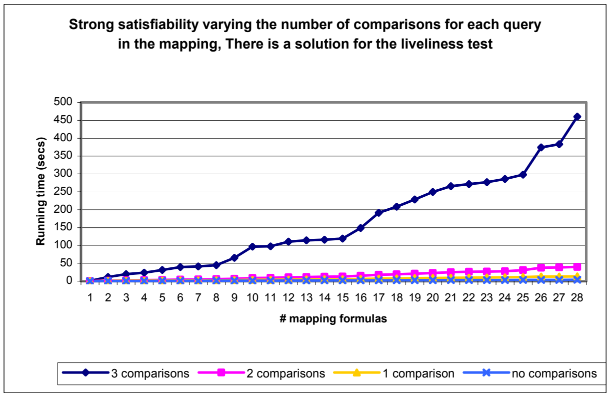

Figure 6: Study of the effect in strong satisfiability of increasing mapping complexity by means of comparisons.

In Figure 6, we study the effect that adding comparisons to the queries in the mapping has on strong satisfiability checking. We focus here in the case when there is a solution for the liveliness test. In the case when there is no solution, the results would be similar but with lower times (remind Figure 5(b)). The comparisons we add to the queries are like $X > K$, where *X* is a variable corresponding to some column of the underlying table, and *K* is a fresh constant. Note that we vary the number of comparisons for each query in the mapping. That means, for example, that when the mapping has 28 formulas, in the case of 3 comparisons per query there is an amount of 168 comparisons in the whole mapping. The graphic shows a great increment of computing time when going from the case of 2 comparisons per query to the case of 3. The reason for that increment is the growth in the number of combinations that can be made using the constants provided by the VIPs. That number of combinations increments exponentially as the graphic already shows. That exponential cost cannot be avoided because the complexity of the problem, but we believe that further work on optimizing the implementation of the CQC Method can reduce computing time considerably.

5. RELATED WORK

A debugger for schema mappings is presented in [8]. The approach is based on the idea of routes. These routes describe the relationships between source and target data with the schema mapping. The authors present this feature to allow a user to explore and understand a schema mapping. It is the first debugging tool developed for schema mappings [1]. The main difference with our approach is that we do not need a source instance and a target instance in order to do the validation. In our approach is only required to provide the mapping and the mapped schemas, and therefore we are able to reason over the mapping itself rather than relying on a concrete instance that may not expose all the potential pitfalls.

However, the work in [8] can be seen as complementary to our approach. Indeed, since we could get a counterexample when validating a certain property, it would be interesting to use the routes to allow the designer understand the counterexample and discover why the checked property does not hold.

As another difference we could mention that we deal with a class of mappings more general than tgds. We consider mappings defined by means of queries that may have negations and comparisons, and schemas that may have views and check constraints.

In [31], the authors propose a framework for understanding and refining mappings in Clio. It consists on making easier to the user the task of building the mapping by means of examples. This examples are samples of a given data source carefully selected to help the user to understand the mapping and choose between alternative mappings. Our approach can be used in conjunction with this framework because it is always difficult for the user to be aware of all possible questions so it would be useful if she were able to confirm that some desirable properties hold for her mapping. The main difference of our approach with respect to this work is that we focus on checking some concrete desirable properties, but not concretely on data exchange context so we do not assume that there is a source data available.

In [2], the authors address the problem of information preservation in XML-to-relational mapping schemes. A mapping schema consists, basically, on a procedure for storing XML documents into a relational database and a procedure for recovering the documents back. Compared with our mappings, these mapping schemas would be mappings between models (the XML model and the relational model in this case) while our mappings are between two concrete relational schemas. Our approach is related with this in the sense that the authors define two properties of information preservation for mapping schemas. They define validating mapping schemas as those in which valid documents can be mapped into legal databases and all legal databases are mappings of valid documents. They also define lossless mapping schemas as those that preserve the structure and content of the XML documents. Note that despite of its name this property is not the same as our mapping losslessness property. The authors show decidability results for these two properties and propose a XML-to-relational mapping schema that satisfies both.

The related work for each validation property has been stated in the corresponding section.

6. CONCLUSIONS AND FUTURE WORK

We have proposed a new approach for validating schema mappings which relies on determining the accomplishment of certain desirable properties of those mappings. We have considered two properties already identified in the literature [22]: mapping inference and query answerability, and we have introduced two new properties: mapping satisfiability and mapping losslessness.

We have also shown how all these properties may be established by means of checking liveliness of a distinguished derived predicate in a new schema that integrates the two mapped schemas and the mapping.

Finally, we have described the results of some experiments we performed using an implementation of our CQC Method for validating the four properties in different scenarios.

As a future work, it would be interesting to find new desirable properties of mappings capturing validation information not covered by the four properties discussed here and that were helpful for mapping designers. Moreover, further work on optimizing the implementation of the CQC Method could also be interesting. We also envisage extending our approach for mappings beyond the relational-to-relational setting, that is, considering other classes of schemas in addition to the relational one (e.g. XML, object-oriented, etc.).

7. REFERENCES

[1] Bogdan Alexe, Laura Chiticariu, Wang Chiew Tan: SPIDER: a Schema mapPIng DEbuggeR. VLDB 2006: 1179-1182

[2] Denilson Barbosa, Juliana Freire, Alberto O. Mendelzon: Information Preservation in XML-to-Relational Mappings. XSym 2004: 66-81

[3] Philip A. Bernstein: Applying Model Management to Classical Meta Data Problems. CIDR 2003

[4] Philip A. Bernstein, Todd J. Green, Sergey Melnik, Alan Nash: Implementing Mapping Composition. VLDB 2006: 55-66

[5] Microsoft BizTalk Server 2004. BizTalk Mapper. http://msdn.microsoft.com/library/en-us/introduction7/html/ebiz_intro_story_jgtg.asp.

[6] Angela Bonifati, Elaine Qing Chang, Terence Ho, Laks V. S. Lakshmanan, Rachel Pottinger: HePToX: Marrying XML and Heterogeneity in Your P2P Databases. VLDB 2005: 1267-1270

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Y. Vardi: Lossless Regular Views. PODS 2002: 247-258

[8] Laura Chiticariu, Wang Chiew Tan: Debugging Schema Mappings with Routes. VLDB 2006: 79-90

[9] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, Pedro Domingos: iMAP: Discovering Complex Matches between Database Schemas. SIGMOD Conference 2004: 383-394

[10] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, Lucian Popa: Data exchange: semantics and query answering. Theor. Comput. Sci. 336(1): 89-124 (2005)

[11] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, Wang Chiew Tan: Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst. 30(4): 994-1055 (2005)

[12] Carles Farré, Ernest Teniente, Toni Urpí: Checking query containment with the CQC method. Data Knowl. Eng. 53(2): 163-223 (2005)

[13] Hendrik Decker, Ernest Teniente, Toni Urpí: How to Tackle Schema Validation by View Updating. EDBT 1996: 535-549

[14] Marc Friedman, Alon Y. Levy, Todd D. Millstein: Navigational Plans For Data Integration. AAAI/IAAI 1999: 67-73

[15] Ariel Fuxman, Mauricio A. Hernández, C. T. Howard Ho, Renée J. Miller, Paolo Papotti, Lucian Popa: Nested Mappings: Schema Mapping Reloaded. VLDB 2006: 67-78

[16] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, Mary Roth: Clio grows up: from research prototype to industrial tool. SIGMOD Conference 2005: 805-810

[17] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, Igor Tatarinov: The Piazza Peer Data Management System. IEEE Trans. Knowl. Data Eng. 16(7): 787-798 (2004)

[18] Alon Y. Halevy, Inderpal Singh Mumick, Yehoshua Sagiv, Oded Shmueli: Static analysis in datalog extensions. J. ACM 48(5): 971-1012 (2001)

[19] Phokion G. Kolaitis: Schema mappings, data exchange, and metadata management. PODS 2005: 61-75

[20] Maurizio Lenzerini: Data Integration: A Theoretical Perspective. PODS 2002: 233-246

[21] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, Alon Y. Halevy: Corpus-based Schema Matching. ICDE 2005: 57-68

[22] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, Alon Y. Halevy: Representing and Reasoning about Mappings between Domain Models. AAAI/IAAI 2002: 80-86

[23] Jayant Madhavan, Alon Y. Halevy: Composing Mappings Among Data Sources. VLDB 2003: 572-583

[24] Altova MapForce. http://www.altova.com.

[25] Sergey Melnik, Philip A. Bernstein, Alon Y. Halevy, Erhard Rahm: Supporting Executable Mappings in Model Management. SIGMOD Conference 2005: 167-178

[26] The Mondial database http://www.dbis.informatik.uni-goettingen.de/Mondial/.

[27] Erhard Rahm, Philip A. Bernstein: A survey of approaches to automatic schema matching. VLDB J. 10(4): 334-350 (2001)

[28] Luc Segoufin, Victor Vianu: Views and queries: determinacy and rewriting. PODS 2005: 49-60

[29] Stylus Studio. http://www.stylusstudio.com.

[30] Ernest Teniente, Carles Farré, Toni Urpí, Carlos Beltrán, David Gañán: SVT: Schema Validation Tool for Microsoft SQL-Server. VLDB 2004: 1349-1352

[31] Ling-Ling Yan, Renée J. Miller, Laura M. Haas, Ronald Fagin: Data-Driven Understanding and Refinement of Schema Mappings. SIGMOD Conference 2001: 485-496

[32] Xubo Zhang, Z. Meral Özsoyoglu: Implication and Referential Constraints: A New Formal Reasoning. IEEE Trans. Knowl. Data Eng. 9(6): 894-910 (1997)