On simplification of schema mappings

Diego Calvanese\textsuperscript{a, *}, Giuseppe De Giacomo\textsuperscript{b}, Maurizio Lenzerini\textsuperscript{b}, Moshe Y. Vardi\textsuperscript{c}

\textsuperscript{a} KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Piazza Domenicani 3, I-39100 Bolzano, Italy
\textsuperscript{b} Dipartimento di Informatica e Sistemistica, Sapienza Università di Roma, Via Ariosto 25, I-00185 Roma, Italy
\textsuperscript{c} Dep. of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77251-1892, USA

\textbf{A R T I C L E  I N F O}

Article history:
Received 3 October 2011
Received in revised form 3 July 2012
Accepted 18 January 2013
Available online 21 January 2013

Keywords:
Schema mappings
Mapping simplification
LAV
GAV
GLAV
Graph databases
Regular path queries

\textbf{A B S T R A C T}

A schema mapping is a formal specification of the relationship holding between the databases conforming to two given schemas, called source and target, respectively. While in the general case a schema mapping is specified in terms of assertions relating two queries in some given language, various simplified forms of mappings, in particular LAV and GAV, have been considered, based on desirable properties that these forms enjoy. Recent works propose methods for transforming schema mappings to logically equivalent ones of a simplified form. In many cases, this transformation is impossible, and one might be interested in finding simplifications based on a weaker notion, namely logical implication, rather than equivalence. More precisely, given a schema mapping $M$, find a simplified (LAV, or GAV) schema mapping $M'$ such that $M'$ logically implies $M$. In this paper we formally introduce this problem, and study it in a variety of cases, providing techniques and complexity bounds. The various cases we consider depend on three parameters: the simplified form to achieve (LAV, or GAV), the type of schema mapping considered (sound, or exact), and the query language used in the schema mapping specification (conjunctive queries and variants over relational databases, or regular path queries and variants over graph databases). Notably, this is the first work on comparing schema mappings for graph databases.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

A schema mapping is a formal specification of the relationship holding between the databases conforming to two given schemas. Many papers in the last decade point out the importance of schema mappings in several data management tasks, especially those requiring inter-operability between different information systems, such as data integration \cite{1,2}, data exchange \cite{3,4}, and model management \cite{5}.

In data integration, schema mappings are established between the source schema and the global schema, also called mediated schema. In this context, schema mappings are used by the query processor when planning the accesses to the source data for answering queries posed in terms of the global schema. In data exchange, schema mappings are specified in terms of a source schema and a target schema, and determine how the source data should be transferred to the target in order to populate a database conforming to the target schema. Schema mappings are also the main objects of interest...
in model management, whose goal is to support the creation, compilation, reuse, evolution, and execution of mappings between schemas, expressed in a wide range of models.

A schema mapping is constituted by two schemas and a set of mapping assertions between the two. We follow the data exchange terminology, and call the two schemas source and target, respectively. As usual, we assume that each assertion relates a query \( q_s \) expressed over the source schema to a query \( q_t \) expressed over the target schema, and specifies a correspondence between the tuples computed by \( q_s \) on the source databases and those computed by \( q_t \) on the target database.

In the following, we consider two types of schema mappings, called sound and exact, respectively. The correspondence specified by assertions in a sound mapping is inclusion, whereas the correspondence specified by assertions in an exact mapping is equality. Semantically, a schema mapping \( M \) specified on source schema \( S \) and target schema \( T \) is characterized by the set of pairs \((D_s, D_t)\) such that \( D_s \) is a source database, i.e., a database conforming to \( S \), \( D_t \) is a target database, i.e., a database conforming to \( T \), and \( D_s \) and \( D_t \) satisfy the correspondences sanctioned by the assertions in \( M \), denoted as \((D_s, D_t) \models M\).

Although mappings of general form are specified using arbitrary queries, various restricted forms of mappings have been considered in this investigation. Notable cases of restricted forms of mappings are “Local-As-Views” (\( lav \)) and “Global-As-Views” (\( gav \)) mappings [1]. In \( lav \) mappings, the source queries in the assertions are constituted by one atom, and exactly one assertion appears for each relation symbol in the source schema. In other words, a \( lav \) mapping associates to each element of the source schema one view over the target schema. Conversely, a \( gav \) mapping associates to each element of the target schema one view over the source schema. Extending the above terminology, the term \( glav \) is often used to refer to unrestricted forms of schema mappings.

Since the pioneering work in data integration [6], schema mappings have been widely investigated in the last years. Recently, several research works have been focused on principles and tools for comparing both schema mapping languages, and schema mappings expressed in a certain language [7–12]. In [10], schema mapping optimization is studied, based on logical equivalence. A set of optimality criteria are proposed for an important class of relational schema mappings, and rewriting rules for transforming a schema mapping into an equivalent optimal one are presented. Notably, \( lav \) and \( gav \) enjoy many of the optimality criteria mentioned in the paper. It follows that the proposed rewriting rules often lead to transforming an input schema mapping into one of the two simplified forms.

Most of the work on optimization and simplification of schema mappings has concentrated so far on transformations preserving logical equivalence. However, there are cases where equivalence preserving simplification is not possible, as demonstrated for \( lav \) by Example 1.1 below. To address such cases, we argue that simplification should be based on a weaker notion, namely logical implication, rather than equivalence. A mapping \( M \) logically implies a mapping \( M' \), denoted \( M \models M' \), if whenever \((D_s, D_t) \models M\), we also have that \((D_s, D_t) \models M'\).

**Example 1.1.** We restrict our attention to schema mappings constituted by mapping assertions using positive queries. Consider a source schema \( S \) consisting of the relations \( r_1/3 \) and \( r_2/2 \) and a target schema \( T \) consisting of the relations \( t_1/3 \) and \( t_2/2 \). Notice that a \( lav \) mapping from \( S \) to \( T \) has the form

\[
\begin{align*}
\{ (x, y, z) \mid r_1(x, y, z) \} & \sim q_1, \\
\{ (x, y) \mid r_2(x, y) \} & \sim q_2
\end{align*}
\]

where \( q_1 \) and \( q_2 \) are two queries over \( T \) with arity three and two respectively. Consider now a schema mapping \( M \) consisting of the following mapping assertion

\[
\{ (x, w, z) \mid r_1(x, w, y) \land r_2(y, z) \} \sim \{ (x, w, z) \mid t_1(x, w, v) \land t_1(v, w, e) \land t_2(e, z) \}.
\]

Suppose that \( M \) is interpreted under the sound semantics. Now, let \( D_s \) be a database such that \( r_1^{D_s} \neq \emptyset \), and \( r_2^{D_s} = \emptyset \), and let \( D_t \) be the empty database. Clearly, \((D_s, D_t) \models M\). On the other hand, for any sound \( lav \) mapping \( M' \) on the same alphabet as \( M \), it holds that \((D_s, D_t) \not\models M'\), because \( r_2^{D_s} = \emptyset \), while the query \( M'(r_1) \) that \( M' \) associates to \( r_1 \) is such that \( M'(r_1)^{D_s} = \emptyset \), and therefore the assertion \( r_1 \sim M'(r_1) \) in \( M' \) cannot be satisfied by \((D_s, D_t)\). It follows that no \( lav \) mapping \( M' \) exists such that \( M \models M' \), and therefore equivalence preserving \( lav \) simplification of \( M \) is impossible to achieve.

**Example 1.2.** Refer again to the schema mapping \( M \) of Example 1.1, and consider the sound \( lav \) mapping \( M'' \) constituted by the following two mapping assertions:

\[
\begin{align*}
\{ (x, w, y) \mid r_1(x, w, y) \} & \sim \{ (x, w, y) \mid t_1(x, w, v) \land t_1(v, w, y) \}, \\
\{ (x, y) \mid t_2(x, y) \} & \sim \{ (x, y) \mid t_2(x, y) \}.
\end{align*}
\]

It is not difficult to see that \( M'' \models M \). Therefore, if we are happy with \( lav \) simplification based on logical implication rather than logical equivalence, \( M'' \) represents an acceptable simplification of \( M \).

Mapping simplification based on logical implication is the subject of this paper. The basic problem we consider can be stated as follows: given a schema mapping \( M \), check whether a simplified (\( lav \), or \( gav \)) schema mapping \( M' \) exists such that \( M' \) logically implies \( M \) (and, if it exists, find one). We formally introduce this problem, and study it in a variety of cases, depending on three parameters:
1. the simplified form to achieve, namely, \( LAV \) or \( GAV \),
2. the type of schema mapping considered, namely, sound or exact, and
3. the data model and the query language used in the schema mapping specification.

As for the first parameter, we essentially concentrate on \( LAV \) in this paper. We discuss \( GAV \) only briefly, pointing out that \( GAV \) simplification is an open problem in several cases.

As for the type of mapping, although the sound semantics is the most popular one in data exchange [3], the importance of considering exact schema mappings is widely recognized for both data exchange [13], and data integration [14].

As for the data model and the query language used in schema mappings, we consider both the relational data model with conjunctive queries and unions thereof, and the graph database model with regular path queries and their extensions. Note that, while schema mappings have been extensively studied for relational data, and, to some extent, for XML data [15], this is the first paper on comparing schema mappings for graph databases. Graph databases [16] were introduced in the 80’s, and are regaining wide attention recently [17–19], for their relevance in areas such as semi-structured data, biological data management, social networks, and the semantic web.

The notion of simplification put forward in our work can be seen as a form of approximation: Given a schema mapping \( M \) (of a general form), find a \( LAV \) schema mapping \( M' \) “approximating” \( M \). It is well known that there are at least two ways of approximating a logical theory \( T \): one way is through a theory \( T_1 \) that is stronger than \( T \), i.e., such that \( T_1 \) logically implies \( T \), and the other way is through a theory \( T_2 \) that is weaker than \( T \), i.e., such that \( T \) logically implies \( T_2 \). For example, in the work on Horn approximation of propositional theories, both kinds of approximations have been considered (see, [20,21]). In this paper, we study one of the two approximations, namely the one where we look for a simplified \( LAV/GAV \) schema mapping that is stronger (or, more precisely, not weaker) than the given schema mapping. Note that all inferences obtained by means of the original mapping are also obtained by means of the stronger mapping. At the same time, with this form of simplified mapping we can infer more than with the original mapping. On the other hand, with a weak approximation, we would get the opposite: all inferences obtained by means of the simplified mapping are also valid inferences with respect to the original mapping. The weak forms of approximation are also of interest, but are not investigated here and are left for future study.

The results we present in this paper can be summarized as follows. We first illustrate our ideas with relational mappings, where the results follow fairly easily from the characterization of containment for conjunctive queries and unions thereof. We show that \( LAV \) simplification is \( \Pi^3 \) \( \text{PSPACE} \)-complete in the case of both sound and exact schema mappings based on conjunctive queries. In the case of unions of conjunctive queries, the problem is still in \( \text{NP} \) for sound mappings, while it is in \( \Pi^3 \) \( \text{PSPACE} \) for exact ones.

For graph database schema mappings based on regular path queries, we prove that \( LAV \) simplification is \( \Pi^3 \) \( \text{PSPACE} \)-complete under the sound semantics, and in \( \text{ExpSPACE} \) in the case of exact schema mappings. By exploiting a language-theoretic characterization for containment of regular path queries with inverse (called two-way regular path queries) provided in [22], we also extend the results to the case where queries in schema mappings are two-way regular path queries, as well as conjunctive two-way regular path queries, and unions of such queries.

Note that a regular path query returns the set of node pairs in the graph database connected by a path conforming to the query, and therefore can be seen as the regular language constituted by all the words labeling the paths denoted by the query. Indeed, the simplification problem addressed in this paper has a language theoretic interpretation in terms of language equations. Specifically, we address systems of language constraints of the form

\[ e_1 \subseteq e_2, \quad \text{and} \quad e_1 = e_2. \]

Here, while \( e_2 \) is an ordinary regular expression, \( e_1 \) is a regular expression that may contain variables as additional alphabet symbols, to be substituted by an ordinary regular expression. The key idea of our approach is that we can prove that solutions of the above equations are closed under congruence, which enables us to represent languages as graphs over the finite-state automaton for \( e_2 \).

The paper is organized as follows. After a discussion on related work in Section 2, we recall some preliminary notions in Section 3. In Section 4, we formally define the problem of schema mapping simplification based on logical implication. In Section 5, we study the problem in the case where queries and views are conjunctive queries, and unions thereof, including a brief discussion on \( GAV \) simplification. In Section 6 we illustrate the techniques for the case of regular path queries over graph databases. In Section 7 we extend them to two-way regular path queries, and discuss the case of (unions of) conjunctions thereof. Finally, Section 8 concludes the paper.

2. Related work

As we said in the Introduction, the issue of representing schema mappings, and reasoning on them has been widely investigated in the last years. In [23–27] the emphasis is on providing foundations for data exchange systems based on schema mappings. Other works deal with answering queries posed to the target schema on the basis of both the data at the sources, and a set of source-to-target mapping assertions (see, for instance, [28,25,29] and the surveys in [6,30,2]). A large body of work has been devoted to studying operators on schema mappings relevant to model management, notably, composition, merge, and inverse (see, for example [31–34,11,35,36,12,37]).
Recently, there has been a growing interest in principles and tools for comparing both schema mapping languages, and schema mappings expressed in a certain language. Comparing schema mapping languages aims at characterizing such languages in terms of both expressive power, and complexity of mapping-based computational tasks [7,8]. In particular, [7] studies various relational schema mapping languages with the goal of characterizing them in terms of structural properties possessed by the schema mappings specified in these languages.

Methods for comparing schema mappings have been recently proposed in [9–12]. In [11,12], schema mappings are compared with respect to their ability to transfer source data and avoid redundancy in the target databases, as well as their ability to cover target data. More relevant to the present paper is the work in [9], which introduces three notions of equivalence. The first one is the usual notion based on logic: two schema mappings are logically equivalent if they are indistinguishable by the semantics, i.e., if they are satisfied by the same set of database pairs. The other two notions, called data exchange and conjunctive query equivalence, respectively, are relaxations of logical equivalence, capturing indistinguishability for different purposes.

The above discussion shows that the work on optimization and simplification of schema mappings has concentrated so far on equivalence preserving transformations. As we said in the Introduction, we follow a different approach, and address the issue of mapping simplification based on logical implication of schema mappings, rather than logical equivalence.

As we have already observed, the simplification problem in the context of graph databases has a language theoretic interpretation in terms of language equations. In general, solving systems of equations of the form \( e = e' \), where \( e \) and \( e' \) are regular expressions over an alphabet of constants and variables is undecidable, because it is easy to express the universality problem for Context Free Grammars in this way. In [38], the authors study linear equations of the form

\[
e_0 + e_1 \cdot x_1 + \cdots + e_n \cdot x_n = e'_0 + e'_1 \cdot x_1 + \cdots + e'_n \cdot x_n
\]

where \( e_0, e'_0, \ldots, e_n, e'_n \) are regular expressions, and \( x_1, \ldots, x_n \) are variables. The authors prove that solving these equations is \textsc{ExpTime}-complete. In contrast, we prove the solvability of another class of problems, namely, systems of language constraints of the form \( e_1 \sqsubseteq e_2 \) and \( e_1 = e_2 \) where both \( e_1 \) and \( e_2 \) are regular expressions, and \( e_1 \) contains variables, whereas \( e_2 \) has no variables.

Taking into account the language-theoretic view, our work has also connections with [39,40], which also study language constraints of the forms \( e_1 \sqsubseteq e_2 \), and \( e_1 = e_2 \). However, in these works, \( e_1 \) is restricted to be a single word on both the source and the target alphabets.

3. Preliminaries

In this work we deal with two data models, the standard relational model [41], and the graph database model [42].

Given a (relational) alphabet \( \Sigma \), a database over \( \Sigma \) is a finite structure over \( \Sigma \). We denote a query of arity \( n \) over \( \Sigma \) as \( [(x_1, \ldots, x_n) | \alpha] \), where \( x_1, \ldots, x_n \) are pairwise distinct variables, called the \emph{distinguished variables} of the query, and \( \alpha \) is an expression over \( \Sigma \) containing \( x_1, \ldots, x_n \), to be evaluated on databases over \( \Sigma \) in order to compute the assignment to such variables. For a query \( q \) over \( \Sigma \), we denote by \( q^D \) the set of tuples resulting from evaluating \( q \) in a database \( D \) over \( \Sigma \). A query \( q \) over \( \Sigma \) is \emph{empty} if for each database \( D \) over \( \Sigma \) we have \( q^D = \emptyset \). Given two queries \( q_1 \) and \( q_2 \) over \( \Sigma \), we say that \( q_1 \) is \emph{contained in} \( q_2 \), denoted \( q_1 \sqsubseteq q_2 \), if \( q_1^D \subseteq q_2^D \) for every database \( D \) over \( \Sigma \). The queries \( q_1 \) and \( q_2 \) are \emph{equivalent}, denoted \( q_1 \equiv q_2 \), if both \( q_1 \sqsubseteq q_2 \) and \( q_2 \sqsubseteq q_1 \).

We assume familiarity with (unions of) conjunctive queries, (U)CQs, over relational databases. In particular, a CQ over \( \Sigma \) is an expression of the form \( [(x_1, \ldots, x_n) | \alpha] \), where \( \alpha \) is a conjunction of atoms over \( \Sigma \) that may include constants, distinguished variables, and additional variables that are implicitly existentially quantified. We consider such queries as interpreted under the active domain semantics [41] modified, for technical reasons (see Definition 4.3), by assuming that the active domain contains an additional special constant that does not appear in any relation. For any arity \( n \), we consider two special queries: the \emph{universal query} denoted \( \text{true}/n \), returning the \( n \)-cartesian product of the active domain, and the \emph{empty query} denoted \( \text{false}/n \), returning the empty set. We may omit \( n \) when it is clear from the context. We recall that containment between (U)CQs can be characterized in terms of homomorphisms (also called \emph{containment mappings}) [43]: For two CQs \( q_1 \) and \( q_2 \), we have that \( q_1 \sqsubseteq q_2 \) iff there is a homomorphism from \( q_2 \) to \( q_1 \), i.e., a mapping \( h \) from the variables and constants of \( q_2 \) to those of \( q_1 \) that is the identity on distinguished variables and constants and such that, if \( r(x_1, \ldots, x_k) \) is an atom of \( q_2 \), then \( r(h(x_1), \ldots, h(x_k)) \) is an atom of \( q_1 \). For two UCQ \( q_1 \) and \( q_2 \), we have that \( q_1 \sqsubseteq q_2 \) iff for each UCQ \( q'_1 \) in \( q_1 \) there is a UCQ \( q'_2 \) in \( q_2 \) such that \( q'_1 \sqsubseteq q'_2 \) [44].

We recall the basic notions regarding graph databases and regular path queries. A \emph{graph database} is a finite graph whose nodes represent objects and whose edges are labeled by elements from an alphabet of binary relational symbols [16,45,46,22]. An edge \( (o_1, r, o_2) \) from object \( o_1 \) to object \( o_2 \) labeled by \( r \) represents the fact that relation \( r \) holds between \( o_1 \) and \( o_2 \).

A \emph{regular-path query} (RPQ) over an alphabet \( \Sigma \) of binary relation symbols is a binary query characterized by a regular language. We denote an RPQ \( \{x, y \mid e(x, y)\} \), where \( e \) is a regular expression or a \emph{nondeterministic finite state automaton (NFA)} over \( \Sigma \), simply by \( e \). When evaluated on a graph database \( D \) over \( \Sigma \), an RPQ \( q \) computes the set \( q^D \) of pairs of objects connected in \( D \) by a path in the regular language \( L(q) \) defined by \( q \). We consider also two-way \emph{regular-path queries} (2RPQs) [47], which extend RPQs with the inverse operator. Formally, let \( \Sigma^\pm = \Sigma \cup \{r^- \mid r \in \Sigma\} \) be the alphabet including a new symbol \( r^- \) for each \( r \) in \( \Sigma \). Intuitively, \( r^- \) denotes the inverse of the binary relation \( r \). If \( p \in \Sigma^\pm \), then we use \( p^- \) to...
mean the inverse of p, i.e., if p is r, then \( p^\sim \) is \( r^\sim \), and if p is \( r^\sim \), then \( p^\sim \) is r. 2RPQs are expressed by means of a 1NFA over \( \Sigma^\pm \). When evaluated on a database \( D \) over \( \Sigma \), a 2RPQ \( q \) computes the set \( q^D \) of pairs of objects connected in \( D \) by a semipath that conforms to the regular language \( L(q) \). A semipath in \( D \) from x to y (labeled with \( p_1\ldots p_n \)) is a sequence of the form \((y_0, p_1, y_1, \ldots , y_{n-1}, p_n, y_n)\), where \( n \geq 0 \), \( y_0 = x \), \( y_n = y \), and for each \( y_{i-1}, p_i, y_i \), we have that \( p_i \in \Sigma^\pm \), and, if \( p_i = r \) then \((y_{i-1}, y_i) \in r^D \), and if \( p_i = r^\sim \) then \((y_{i-1}, y_i) \in r^D \). We say that a semipath \((y_0, p_1, \ldots , p_n, y_n) \) conforms to \( q \) if \( p_1\ldots p_n \in L(q) \).

We conclude by observing that (U)CQs, RPQs, 2RPQs, and (U)2RPQs are monotone, where a query \( q \) is monotone if, whenever \( D_1 \subseteq D_2 \) (i.e., \( r^{D_1} \subseteq r^{D_2} \) for each relation \( r \)) we have that \( q^{D_1} \subseteq q^{D_2} \).

4. Schema mapping simplification

We refer to a scenario with one source schema, one target schema, and a schema mapping between the two. To model the source and the target schemas we refer to two finite alphabets, the source alphabet \( \Sigma_s \) and the target alphabet \( \Sigma_t \), and to specify the mapping, we use a correspondence between queries expressed in a given query language.

**Definition 4.1.** Given a query language \( Q \), a \( Q \)-based schema mapping assertion from \( \Sigma_s \) to \( \Sigma_t \) is a statement of the form \( q_s \sim q_t \), where \( q_s \) and \( q_t \) are two queries in \( Q \) with the same arity, respectively over \( \Sigma_s \) and over \( \Sigma_t \). A \( Q \)-based schema mapping from \( \Sigma_s \) to \( \Sigma_t \) is a set of mapping assertions from \( \Sigma_s \) to \( \Sigma_t \).

In the following, we specify explicitly \( Q, \Sigma_s, \) and \( \Sigma_t \) only when they are required or not clear from the context.

We consider two types of schema mappings, called sound and exact. Intuitively, in a sound mapping, the correspondence between the tuples computed by \( q_s \) and those computed by \( q_t \) is set containment, while in an exact mapping the correspondence is set equality. Formally, given a source database \( D_s \) and a target database \( D_t \), we say that a sound mapping \( M \) is satisfied by \((D_s, D_t)\), denoted \((D_s, D_t) \models M \), if for each mapping assertion \( q_s \sim q_t \) in \( M \), we have that \( q_t^{D_t} \subseteq q_s^{D_s} \). Similarly, an exact mapping \( M \) is satisfied by \((D_s, D_t)\) if for each \( q_s \sim q_t \) in \( M \), we have that \( q_t^{D_t} = q_s^{D_s} \).

A fundamental notion in our setting is that of logical implication between mappings.

**Definition 4.2.** A mapping \( M_1 \) logically implies a mapping \( M_2 \), denoted \( M_1 \models M_2 \), if for every pair \((D_s, D_t)\) such that \((D_s, D_t) \models M_1 \), we also have that \((D_s, D_t) \models M_2 \).

We consider two simplified forms of mappings called \( \text{lav} \) (local-as-view) and \( \text{gav} \) (global-as-view), respectively. In a \text{lav} assertion \( q_s \sim q_t \), the query \( q_t \) is constituted simply by an atom whose predicate symbol belongs to \( \Sigma_t \) and whose arguments are pairwise distinct distinguished variables,\(^1\) while \( q_t \) is an arbitrary query.\(^2\) Conversely, in a \text{gav} assertion, \( q_s \) is an arbitrary query while \( q_t \) is constituted simply by an atom whose predicate symbol belongs to \( \Sigma_t \) and whose arguments are pairwise distinct distinguished variables. A \text{lav} mapping is a set of \text{lav} assertions with one assertion for each symbol in \( \Sigma_s \). If \( M_L \) is a \text{lav} mapping and \( a \in \Sigma_s \), we denote by \( M_L(a) \) the target query to which \( a \) is mapped by \( M_L \). Conversely, a \text{gav} mapping is a set of \text{gav} assertions with one assertion for each symbol in \( \Sigma_s \). Analogously to the set of \text{lav} mappings, \( M_G(a) \) denotes the source query to which the symbol \( a \in \Sigma_t \) is mapped by the \text{gav} mapping \( M_G \). Note that some of the queries in a \text{lav} (resp., \text{gav}) mapping may be the empty query.

The problem we consider aims at checking whether a simplified mapping exists that logically implies a given mapping \( M \). We consider the case where \( M \) is constituted by a single assertion \( q_s \sim q_t \), and we aim at simplifying \( M \) by means of a \text{lav} schema mapping \( M' \). One trivial way to do this is to define \( M' \) in such a way that, whenever a pair \((D_s, D_t)\) satisfies \( M' \), the query \( q_t \) is empty when evaluated over \( D_s \). Similarly, a trivial \text{gav} simplification \( M' \) would be one that enforces that the query \( q_t \) returns the whole active domain, when evaluated over \( D_t \). We would like to rule out these meaningless simplifications, and therefore we formally introduce the notion of triviality w.r.t. a mapping assertion, distinguishing the two cases of \text{lav} and \text{gav}.

**Definition 4.3.** A \text{lav} (resp., \text{gav}) mapping \( M' \) is said to be trivial w.r.t. a mapping assertion \( q_s \sim q_t \), if for each pair \((D_s, D_t)\) with \((D_s, D_t) \models M' \), we have that \( q_t^{D_t} = \text{false}^{D_t} \) (resp., \( q_t^{D_t} = \text{true}^{D_t} \)). A \text{lav} or \text{gav} mapping \( M' \) is said to be trivial w.r.t. a mapping \( M \), denoted \( M' \models_{\text{triv}} M \), if there is a mapping assertion \( q_s \sim q_t \in M \) such that \( M' \) is trivial w.r.t. \( q_s \sim q_t \).

We observe that our assumption of having an additional constant in the active domain plays a role here. Indeed, without such an assumption, if the target signature consists of a single unary predicate, then the denotation of that predicate in a target instance would necessarily coincide with the active domain of that instance, and therefore, by definition, \( M' \models_{\text{triv}} M \) would hold, regardless of what \( M \) and \( M' \) look like.

\(^1\) Note that the assumption of having pairwise distinct variables is in line with the data integration framework, where all source tuples should be mapped to the target.

\(^2\) For RPQs and 2RPQs, where query variables are not represented explicitly, we consider an atom to be simply a binary predicate symbol.
As an example of Definition 4.3, consider a mapping $M$ constituted by the two assertions

$$\{(x) \mid a_1(x) \land a_2(x)\} \sim \{(x) \mid b_1(x) \land b_2(x)\},$$
$$\{(x) \mid a_1(x)\} \sim \{(x) \mid b_3(x)\}.$$ 

Then, the following LAV mapping $M'$ is trivial w.r.t. $M$, since for every pair $(D_1, D_2)$ satisfying $M'$, we have that $\{(x) \mid a_1(x) \land a_2(x)\}$ evaluates to $\emptyset$ over $D_1$:

$$\{(x) \mid a_1(x)\} \sim \text{true}/1,$$
$$\{(x) \mid a_2(x)\} \sim \text{false}/1.$$

**Definition 4.4.** Let $tp$ be one of SOUND or EXACT, $fm$ one of LAV or GAV, and $Q_1$ and $Q_2$ two query languages. Mapping simplification, denoted

$$\text{MSIMP}[tp, fm, Q_1, Q_2],$$

is the following decision problem: given a $Q_1$-based schema mapping $M$ of type $tp$, check whether there exists a $Q_2$-based schema mapping $M'$ of type $tp$ and form $fm$ such that $M' \models M$, and $M' \not\succsim M$.

To rule out (uninteresting) cases where all mappings that imply a given mapping $M$ are trivial w.r.t. $M$, in the following we require that for each mapping assertion $q_1 \sim q_t$ in a LAV mapping $M$, $q_t$ is different from false. Analogously, we require that for each mapping assertion $q_1 \sim q_t$ in a GAV mapping $M$, $q_t$ is different from true.

If a simplified mapping exists, it is also of interest to actually compute one. Therefore, we consider the corresponding synthesis problem.

**Definition 4.5.** Let $tp$ be one of SOUND or EXACT, $fm$ one of LAV or GAV, and $Q_1$ and $Q_2$ two query languages. Mapping synthesis, denoted

$$\text{MSYNT}[tp, fm, Q_1, Q_2],$$

is the following problem: given a $Q_1$-based schema mapping $M$ of type $tp$, find a $Q_2$-based schema mapping $M'$ of type $tp$ and form $fm$ such that $M' \models M$, and $M' \not\succsim M$.

In general we may want to synthesize simplified mappings with specific interesting properties. As an example, in this paper we are interested in the tightest simplifications of a mapping $M$, i.e., the simplifications that best approximate $M$.

**Definition 4.6.** Let $tp$ be one of SOUND or EXACT, $fm$ one of LAV or GAV, and $Q_1$ and $Q_2$ two query languages. Maximal mapping synthesis, denoted

$$\text{MAXMSYNT}[tp, fm, Q_1, Q_2],$$

is the following problem: given a $Q_1$-based schema mapping $M$ of type $tp$, find a solution $M'$ of $\text{MSYNT}[tp, fm, Q_1, Q_2]$ such that no solution $M''$ of $\text{MSYNT}[tp, fm, Q_1, Q_2]$ satisfies $M' \models M''$ and $M'' \not\models M'$.

Observe that, in general, we might have more than one maximal solution for the mapping synthesis problem. In the LAV case, a maximal solution $M'$ is such that there is no solution $M''$ such that: (1) $M'(a) \subseteq M''(a)$ for every source symbol $a$, and (2) it is not the case that $M''(b) \subseteq M'(b)$ for every source symbol $b$, i.e., the queries $M'(a)$ are maximal. Conversely, in the GAV case, a maximal solution $M'$ is such that the queries $M'(b)$ are minimal, for each target symbol $b$.

In this paper we study the above problems for a variety of cases, where $Q_1$ and $Q_2$ range over (U)CQs and variants of queries over graph databases.

We start by observing that we can characterize mapping implication, and hence mapping simplification, in terms of query unfolding w.r.t. a set of mappings. We make use of such a characterization in the technical development in the subsequent sections. The notion of query unfolding is formally defined as follows: let $q_s$ be a source query and $M_L$ a LAV mapping. The unfolding of $q_s$ w.r.t. $M_L$, denoted $q_s[M_L]$, is the target query obtained by replacing each atom $\alpha$ in $q_s$, whose predicate symbol is $\alpha$ with $M_L(\alpha)$. An analogous definition holds for the unfolding $q_s[M_G]$ of a target query $q_t$ w.r.t. a GAV mapping $M_G$.

**Proposition 4.1.** Let $Q$ be a monotone query language.

1. Let $M_L$ be a LAV mapping and $M$ a mapping, both $Q$-based and of type SOUND (resp., EXACT). Then $M_L \models M$ iff for each assertion $q_s \sim q_t$ in $M$, we have that $q_s[M_L] \subseteq q_t$ (resp., $q_s[M_L] \equiv q_t$).
2. Let $M_G$ be a GAV mapping and $M$ a mapping, both $Q$-based and of type SOUND (resp., EXACT). Then $M_G \models M$ iff for each assertion $q_s \sim q_t$ in $M$, we have that $q_s \subseteq q_t[M_G]$ (resp., $q_s \equiv q_t[M_G]$).
Proof. We provide the proof for (1) for the case of sound mappings. The other cases can be proved analogously. We start with the following observation: if $M_L$ is a $\lambda$-mapping, and $D_S$ is the source database obtained from a target database $D_T$ by letting $r_{D_T}^{M_L} = M_L(r)_{D_T}$ for every $r \in \Sigma$, then for every source query $q$, we have that $q_{D_T}^{M_L} = q[M_L]_{D_T}$.

$\Rightarrow$ Now, assume that there is an assertion $q \sim q_L$ in $M$ such that $q_L[M_L] \not\subseteq q$, and let $D_S$ be such that for some tuple $d$ we have $d \in q_L[M_L]_{D_S}$, and $d \not\in q_{D_S}$.

Let $D_S$ be the source database obtained from $D_T$ by letting $r_{D_T}^{M_L} = M_L(r)_{D_T}$ for every $r \in \Sigma$. Clearly, $(D_S, D_T) \models M_L$. By the above observation, we have that $q_{D_T}^{M_L} = q_L[M_L]_{D_T}$, and therefore $d \in q_{D_T}^{M_L}$. It follows that $(D_S, D_T) \not\models q \sim q_L$, and $M_L \not\models M$. A contradiction.

$\Leftarrow$ Assume that $M_L \models M$, i.e., there are an assertion $q \sim q_L$ in $M$, and a pair $(D_S, D_T)$ such that $(D_S, D_T) \models M_L$, and $q_{D_T}^{M_S} \not\subseteq q_{D_S}$, which means that there is $d$ such that $d \in q_{D_T}^{M_S}$ but $d \not\in q_{D_S}$. Let $D'_S$ be such that $r_{D'_S}^{M_S} = M_L(r)_{D'_S}$ for every $r \in \Sigma$. Clearly, $D'_S \supseteq D_S$, and $(D'_S, D_T) \models M_L$. Since $q$ is monotone, we have that $d \in q_{D'_S}^{M_S} = q_L[M_L]_{D'_S}$, and therefore $q_L[M_L] \not\subseteq q$. A contradiction. $\square$

Proposition 4.2. Let $Q$ be a monotone query language, and let $M$ be a mapping and $M'$ a $\lambda$-mapping or a $\lambda$-mapping, both $Q$-based and of type sound or exact. Then $M' \models_{inv} M$ iff for some mapping assertion $q \sim q_L \in M$ we have that $q_L[M'] \not\subseteq false$ (resp., true $\subseteq q_L[M']$).

Proof. We first provide the proof for the case of (sound or exact) $\lambda$-mappings.

$\Rightarrow$ Since $M' \models_{inv} M$, there is a mapping assertion $q \sim q_L \in M$ such that for each pair $(D_S, D_T)$ with $(D_S, D_T) \models M_L$, we have that $q_{D_T}^{M_S} = false_{D_T}^{M_S}$. Assume that $q_L[M'] \not\subseteq false$, i.e., that $q_L[M']_{D_T} \not\subseteq \emptyset$ for some target database $D_T'$. But then, for the source database $D_S'$ obtained from $D_S'$ by letting $r_{D_S'}^{M_S} = M_L(r)_{D_S'}$ for every $r \in \Sigma$, we have that $(D_S', D_T') \models M'$ (both when $M'$ is of type sound and of type exact) and $q_{D_T'}^{M_S} \neq false_{D_T'}^{M_S}$. A contradiction.

$\Leftarrow$ Assume that for some mapping assertion $q \sim q_L \in M$ we have that $q_L[M'] \not\subseteq false$ and that $M' \not\models_{inv} M$. Then there exists a pair $(D_S, D_T)$ such that $(D_S, D_T) \models M'$ and $q_{D_T}^{M_S} \neq false_{D_T}^{M_S}$, i.e., $q_{D_T}^{M_S} \not\subseteq \emptyset$. Consider the source database $D'_S$ obtained from $D_S$ by letting $r_{D'_S}^{M_S} = M_L(r)_{D'_S}$ for every $r \in \Sigma$. Since $(D'_S, D_T') \models M'$, we have that $r_{D_T'}^{M_S} \subseteq (M_L(r)_{D_T'})$ (actually, $r_{D_T'}^{M_S} = M_L(r)_{D_T'}$ when $M'$ is exact), hence by monotonicity of $q$, we have that $q_{D_T'}^{M_S} \subseteq q_{D_T'}^{M_S} = q_L[M']_{D_T'} = false_{D_T'}^{M_S} = \emptyset$. A contradiction.

As for the case of $\lambda$-mappings, the proof is analogous, but makes use of the fact that the active domain contains an extra constant to deal with the special case where the target alphabet consists of a single unary relation. $\square$

For many of the results in the next sections, we make use of the above characterization, without further mentioning Propositions 4.1 and 4.2.

5. Simplification and synthesis for (U)CQs

In this section, we consider the case of mappings based on conjunctive queries (CQs) and their unions (UCQs), and study the problem of simplifying a given mapping $M$ in terms of a $\lambda$-mapping or a CQ mapping. The techniques we adopt for establishing our upper bounds are based on determining a polynomial bound on the length of the queries to consider when searching for the simplified mapping logically implying $M$, and are reminiscent of those in [48].

5.1. The case of $\lambda$-mapping

We now refer to a $\lambda$-mapping logically implying a given mapping, we implicitly assume that implication is non-trivial. We start with the problem of simplifying a CQ-based mapping in terms of a CQ-based $\lambda$-mapping.

Theorem 5.1. Both MSIMP[<code>sound</code>,<code>exact</code>,<code>lambda</code>,<code>CQ,cq</code>] and MSIMP[<code>exact</code>,<code>exact</code>,<code>lambda</code>,<code>CQ,cq</code>] are in NP.

Proof. Consider a sound CQ-based mapping consisting of a single assertion $q \sim q_L$, where $q_L$ contains $\ell_q$ atoms, and a sound CQ-based $\lambda$-mapping $M_L$, such that $q_L[M_L] \subseteq q$. Then, there exists a homomorphism from $q_L$ to $q_L[M_L]$, and at most $\ell_q$ atoms of $q_L[M_L]$ are in the image of this homomorphism. Hence, for each symbol $a \in \Sigma$, occurring in $q_L$, at most $\ell_q$ atoms in query $M_L(a)$ are needed for the homomorphism. In the general case where the mapping $M$ consists of several assertions, for each $a \in \Sigma$, we need at most $\ell_M = \sum_{q \sim q_L} \ell_q$ atoms in the query $M_L(a)$, in order to guarantee the existence of the homomorphisms for all the assertions in $M$. Hence, in order to check for the existence of an appropriate $\lambda$-mapping $M_L$, it suffices to guess, for each symbol $a \in \Sigma$, appearing in one of the mapping assertions in $M$, a CQ $M_L(a)$ over $\Sigma_L$ of size at most $\ell_M$, and check that $q_L[M_L(a)] \subseteq q_L[a]$ for each $a \sim q_L \in M$. In doing so, we rule out the guess of mappings that are trivial w.r.t. $M$. This gives us immediately an NP upper bound for MSIMP[<code>sound</code>,<code>exact</code>,<code>lambda</code>,<code>CQ,cq</code>].

For MSIMP[<code>exact</code>,<code>exact</code>,<code>lambda</code>,<code>CQ,cq</code>], in addition to checking that $q_L[M_L] \subseteq q_L$, we need also to check that $q_L \subseteq q_L[M_L]$. We observe that the bound on the number of atoms in $M_L(a)$ derived for the sound case is still valid, since if $q_L \subseteq q_L[M_L]$ for a $\lambda$-mapping $M_L$, then also $q_L \subseteq q_L[M_L]$ for every $\lambda$-mapping $M'_L$ such that $M'_L(a)$ is constituted by a subset of the atoms of $M_L(a)$. Therefore, the overall complexity does not change. $\square$
The proof is by a reduction from 3-colorability. Given a graph $G = (N, E)$, with $N = \{n_1, \ldots, n_k\}$, we define the corresponding instance of MSIMP[\textsc{sound, lav, ucq, cq}] as follows.

Theorem 5.5. MSIMP[\textsc{exact, lav, ucq, cq}] is in $\Pi_2^P$.

Proof. Consider an exact UCQ-based mapping $M$ consisting of a single assertion $q_s \rightsquigarrow q_t$ and an exact UCQ-based LAV mapping $M_L$ such that $M_L \models M$. Let $m_{q_s}$ be the number of CQs in $q_s$, and $\ell_{q_s}$ the number of atoms in the longest CQ in $q_s$. Let $q'_s, M'_L$ be the UCQ obtained from $q_s, M_L$ by distributing, for each atom $\alpha$ of $q_s$, the unions in the UCQ $\alpha[M_L]$ over the conjunctions of each CQ of $q_s$. Since $q_s[M_L] \subseteq q_t$, for each CQ in $q'_s, M'_L$, there is a homomorphism from some CQ in $q_t$ to it. Hence, for each symbol $a$ in $\Sigma_1$ occurring in $q_s$, for each CQ in the UCQ $M(a)$, we need at most $\ell_{q_t}$ atoms for the homomorphisms from all CQs in $q_s$. To derive a bound for the number of such CQs, we observe that the inclusion $q_t \subseteq q_s[M_L]$ must also hold. To satisfy this inclusion, it suffices to have in $M_L(a)$ one CQ for each occurrence of $a$ in $q_t$, i.e., at most $m_{q_t} \cdot \ell_{q_t}$ CQs. It follows that, to check for the existence of the LAV mapping $M_L$ and the corresponding homomorphism, it suffices to guess for each $a \in \Sigma_1$ a UCQ over $\Sigma_1$ consisting of at most $m_{q_t} \cdot \ell_{q_t}$ CQs, each with at most $\ell_{q_t}$ atoms. In the general case where the mapping $M$ consists of several assertions $q_s \rightsquigarrow q_t$, we can proceed analogously to the case above, using instead of $m_{q_t}$ and $\ell_{q_t}$ the sum of these parameters over all mapping assertions in $M$. To check whether $q_t \subseteq q_s[M_L]$, for each of the CQs in $q_t$, it suffices to guess (i) a CQ $q'_t$ in $q_t$, (ii) a CQ in $M_L(a)$ for each occurrence of $a$ in $q'_t$, and (iii) a mapping $h$ from the query $q''_t$ obtained by unfolding $q'_t$ with the selected CQs for each occurrence of $a$, and check whether $h$ is a homomorphism. This can be done in NP. To check whether $q_t \subseteq q_s[M_L]$, we have to check whether for each CQ $q''_t$ obtained by selecting one of the CQs $q''_t$ in $q_t$, and then substituting each atom $\alpha$ in $q''_t$ with one of the CQs in $\alpha[M_L]$, there is a homomorphism from some CQ in $q_t$ to $q''_t$. We can do so by a coNP computation that makes use of an NP oracle to check for the existence of a homomorphism. This gives us the $\Pi_2^P$ upper bound.

We now show that the upper bounds for the sound cases established in Theorems 5.1, 5.2, and 5.4 are tight.

Theorem 5.6. MSIMP[\textsc{sound, lav, ucq, cq}] is NP-hard.

Proof. The proof is by a reduction from 3-colorability. Given a graph $G = (N, E)$, with $N = \{n_1, \ldots, n_k\}$, we define the corresponding instance of MSIMP[\textsc{sound, lav, ucq, cq}] as follows.
As for the source alphabet, we consider $\Sigma_3 = \{a_t/2, a_s/2, a_f/2\}$, where intuitively $a_t$ is a relation denoting graph edges, $a_s$ is a relation between (starting) elements and graph nodes, and $a_f$ is a relation between graph nodes and (final) elements. Similarly, for the target alphabet, we consider $\Sigma_2 = \{b_t/2, b_s/2, b_f/2\}$. On $\Sigma_3$ and $\Sigma_2$ we define mapping $M$ as follows:

$$q_T \sim q_G,$$

(1)

$$\{(x, y) \mid a_t(x, y)\} \sim \{(x, y) \mid b_t(x, y)\},$$

(2)

$$\{(x, y) \mid a_s(x, y)\} \sim \{(x, y) \mid b_s(x, y)\},$$

(3)

$$\{(x, y) \mid a_f(x, y)\} \sim \{(x, y) \mid b_f(x, y)\},$$

(4)

where

$$q_T = \{(s, f) \mid a_t(s, r) \land a_s(s, g) \land a_s(s, b) \land a_t(r, g) \land a_s(g, r) \land a_t(r, b) \land a_s(b, g) \land a_f(r, f) \land a_f(g, f) \land a_f(b, f)\}.$$

$$q_G = \{(s, f) \mid b_t(s, x_1) \land \cdots b_t(s, x_k) \land b_f(x_1, f) \land \cdots b_f(x_k, f)\}.$$

Intuitively, assertion (1) maps a triangle, whose three vertices are connected by $a_t$ and $a_f$ to the distinguished variables $s$ and $f$ respectively, to the graph $G$, whose nodes are connected by $b_t$ and $b_f$ to the distinguished variables $s$ and $f$ respectively.

We show that $G$ is 3-colorable iff MSIMP[$\text{SOUND, LAV, UCQ, CQ}$] with input $M$ admits a solution. For the “only-if” part, consider the LAV mapping $M_1$ consisting of the mapping assertions (2), (3), and (4). If $G$ is 3-colorable, a coloring of the nodes of $G$ with the three colors $r, g, b$ gives us immediately a homomorphism from $q_G[M_1]$ to $q_T$ in which each variable $x_i$ of $q_G[M_1]$ is mapped to the variable of $q_T$ corresponding to the color assigned to node $n_i$. Hence we have that $q_T \subseteq q_G[M_1]$. For the “if-part”, consider a LAV mapping $M_L$ such that $q_L[M_L] \subseteq q_T$ for each mapping assertion $q_L \sim q_T$ in $M$. By the mapping assertions (2), (3), and (4), we have that the queries $M_L(a_t), M_L(a_s), M_L(a_f)$, which we assume to have $(x, y)$ as distinguished variables, must include respectively the atoms $b_t(s, y), b_t(x, y)$, and $b_f(x, y)$, plus possibly additional atoms containing existentially quantified variables. Note that these existential variables appear in the unfolding $q_T[M_L]$. Now, consider a homomorphism $h$ from $q_G(s, f)$ to $q_T[M_L](s, f)$. Since $s$ and $f$ are distinguished variables, we have that $h(s) = s$ and $h(f) = f$. Suppose that for some variable $x_i \in \{x_1, \ldots, x_k\}$ of $q_G$, we have that $h(x_i)$ is an existential variable $y$ in an additional atom in $q_T[M_L]$. Then, since $q_G$ contains the atoms $b_t(s, x_i)$ and $b_f(x_i, f)$, we must have that $q_T[M_L]$ contains the atoms $b_t(s, y)$ and $b_f(y, f)$. This is impossible, since $y$ is an existential variable introduced by the unfolding of $q_T$ with $M_L$, and hence can appear in the unfolding of just one atom of $q_T$. But there is no atom of $q_T$ that contains both $s$ and $f$, and that could generate both $b_t(s, y)$ and $b_f(y, f)$. So, the only possibility for a homomorphism from $q_G[M_L]$ to $q_T$ is to map each $x_i$ of $q_G[M_L]$ to one of the variables $r, g, b$. The existence of such a homomorphism implies that $G$ is 3-colorable. $\square$

**Corollary 5.7.** MSIMP[$\text{SOUND, LAV, UCQ, CQ}$] and MSIMP[$\text{SOUND, LAV, UCQ, UCQ}$] are NP-hard.

**Proof.** From Theorem 5.6 we trivially get the result for MSIMP[$\text{SOUND, LAV, UCQ, CQ}$], and by considering Lemma 5.3, we get the result also for MSIMP[$\text{SOUND, LAV, UCQ, UCQ}$]. $\square$

It is easy to see that the proof of Theorem 5.6 shows also NP-hardness of simplification for exact mappings.

**Theorem 5.8.** MSIMP[$\text{EXACT, LAV, UCQ, CQ}$] and hence MSIMP[$\text{EXACT, LAV, UCQ, UCQ}$] are NP-hard.

We conjecture that the $P^3$ upper bound for MSIMP[$\text{EXACT, LAV, UCQ, UCQ}$] is also tight.

Next, we turn to maximal solutions to LAV simplification. We observe that all our upper bound results are based on deriving a bound on the number of atoms that may constitute the queries in a possible solution to the simplification problem, and on the possibility to guess (or enumerate) all solutions within the derived bound. Hence, it becomes possible to check also further properties of the guessed (or enumerated) solutions. Specifically, we can obtain a CQ-based LAV-mapping that is a maximal solution to simplification of ucq-based mappings by guessing a candidate mapping and checking that it is a solution; then, to check that it is maximal, we generate all other mappings and check that, if they are solutions, they are contained in our candidate solution. This immediately results in a technique to solve maximal mapping synthesis in $\Sigma^3_2$. 
Theorem 5.9. A solution to \textup{MaxMSynt}\{lav,sound,cq,cq\} and to \textup{MaxMSynt}\{lav,sound,ucq,cq\} can be computed in \(\Sigma^p_2\).

Unfortunately, we are not able to use the same argument for deriving a solution to \textup{MaxMSynt}\{lav,sound,ucq,ucq\} in \(\Sigma^p_2\), since we have no guarantee that a maximal ucq-based solution is of polynomial size.

5.2. The case of GAV

Next, we consider the case of simplifying (U)CQ-based mappings in terms of GAV mappings. Note that for exact mappings, GAV simplification is the same problem as Lav simplification, so we focus here on sound mappings. We start by considering sound cq-based mappings and show the following upper bound.

Theorem 5.10. MSimp\{sound,gav,cq,cq\} is in NP.

Proof. Consider a sound cq-based mapping \(M\) consisting of a single assertion \(q_i \rightarrow q_t\) and assume there exists some sound cq-based gav mapping \(M'_G\) such that \(q_i \subseteq q_t[\Sigma_G]\), witnessed by a homomorphism \(h\), and \(q_t[\Sigma_G] \neq \text{true}\). We show that there is a sound cq-based gav mapping \(M'_G\) of bounded size such that \(q_i \subseteq q_t[\Sigma_G]\). Indeed, since \(q_t[\Sigma_G] \neq \text{true}\), there must be one distinguished variable \(x\) and one symbol \(b \in \Sigma_t\), such that \(x\) occurs in an atomic formula with the symbol \(b\) in \(\Sigma_G\). We now obtain \(M'_G\) from \(M_G\) by selecting in \(\Sigma_G\) such an atom. For all other symbols \(b' \in \Sigma_t\), we take \(M'_G\) to be true. By constructing \(M'_G\) in this way, we have that the atoms in \(q_t[\Sigma_G]\) are a subset of the atoms in \(q_t[\Sigma_G]\), and hence the projection of \(h\) on such atoms is still a homomorphism to \(q_t\).

In the general case where the mapping \(M\) consists of \(k\) assertions, we can apply the above argument for each of the assertions in \(M\). This shows that, if there exists some sound gav mapping \(M'_G\) such that \(M'_G \models M\), then there is also a gav mapping \(M'_G\) such that \(M'_G\) has at most \(k\) atoms and \(M'_G \models M\). Hence, in order to check the existence of an appropriate gav mapping \(M'_G\), it suffices to guess (avoiding trivial mappings), for each symbol \(b \in \Sigma_t\) appearing in \(M\), a CQ \(M_G(b)\) over \(\Sigma_G\) of size at most \(k\), and check that \(q_i \subseteq q_t[\Sigma_G]\), for each \(q_i \sim q_t \in M\).

This result extends immediately to ucq-based mappings, by checking containment between UCQs, instead of CQs.

Theorem 5.11. MSimp\{sound,gav,ucq,ucq\} is in NP.

For UCQ-based mappings, we get the same upper bound, with a somewhat subtler argument.

Theorem 5.12. MSimp\{sound,gav,ucq,ucq\} is in NP.

Proof. We consider first the case of a sound ucq-based mapping \(M\) consisting of a single assertion \(q_i \rightarrow q_t\). Assume there exists some sound ucq-based gav mapping \(M_G\) such that \(q_i \subseteq q_t[\Sigma_G]\) and \(q_t[\Sigma_G] \neq \text{true}\). First note that \(q_i \subseteq q_t[\Sigma_G]\), if for each CQ \(q'_t\) of \(q_t\), we have that \(q'_i \subseteq q_t[\Sigma_G]\). This means that there are a CQ \(q'_t\) of \(q_t\) and a CQ \(q'_i\) of \(q_i\) such that \(q'_i \subseteq q'_t[\Sigma_G]\), and \(\Sigma_G\) is a subset of \(\Sigma_t\). What is left is to bound the size of these CQs.

Let \(m\) be the number of CQs in \(q_i\). Since \(q_t[\Sigma_G] \neq \text{true}\), for each CQ \(q'_t\) of \(q_t\), there must be one distinguished variable \(x\) and one symbol \(b \in \Sigma_t\) such that \(x\) occurs in some atomic formula with the symbol \(b\) in \(M_G(b)\). Define \(M'_G(b)\) to contain all such atomic formulas, and only such atomic formulas. \(M'_G(b)\) contains at most \(k\) such CQs. If we have \(k\) mappings, then \(M'_G(b)\) needs to contain only \(klm\) such CQs. To check the existence of a simplifying gav mapping it suffices to guess a mapping \(M'_G\) under such a size bound and check that \(q_i \subseteq q_t[\Sigma_G]\).

We conjecture that the above upper bounds are tight.

We observe that the arguments we have used above to derive bounds on the size of solutions to GAV simplification, cannot be used to derive bounds for maximal GAV solutions, i.e., solutions in which the source queries in the mapping assertions are as small as possible. Indeed, the problem of synthesizing maximal (u)cq-based GAV mappings is still open.

Our results on simplification for (U)CQs are summarized in Table 1.

6. LAV simplification and synthesis for RPQs

In this section, we consider the case of RPQs over graph databases, and study the problem of simplifying an rpq-based mapping in terms of an rpq-based LAV mapping. For our results, we exploit a straightforward language theoretic characterization of containment between RPQs. We observe that GAV simplification is wide open for rpq-based mappings.

Theorem 6.1. (See [42].) Let \(q_1, q_2\) be two RPQs, and \(L(q_1), L(q_2)\) the corresponding regular languages. Then \(q_1 \subseteq q_2 \iff L(q_1) \subseteq L(q_2)\).
In the following, we identify an RPQ $q$ over an alphabet $\Sigma$ with the language over $\Sigma$ accepted by the regular expression or 1NFA representing $q$. Considering the language-theoretic characterization above, it follows from Propositions 4.1 and 4.2 that, if $M_L$ is a 1NFA mapping and $M$ a mapping, both of type sound (resp., exact), then $M_L \models M$ and $M_L \not\equiv_{gl} M$ iff for each assertion $q_i \sim q_l$ in $M$, we have that $q_i[M_L] \subseteq q_l$ (resp., $q_i[M_L] = q_l$) and $q_i[M_L] \neq \emptyset$. Here, the unfolding $q_i[M_L]$ of $q_i$ w.r.t. $M_L$ denotes the language over $\Sigma$ obtained from $q_i$ by expanding in each word in $q_i$ each symbol $a \in \Sigma$ with the language $M_L(a)$. Obviously, such a language can be represented by a regular expression or a 1NFA of size linear in the product of the sizes of the (representations of) $q_i$ and $M_L$.

We start by showing that we can characterize mapping implication $M_L \models M$ between a 1NFA mapping $M_L$ and a mapping $M$ in terms of a single language containment (for sound mappings) or language equality (for exact mappings). For this, we extend the notion of unfolding of $q_i$ w.r.t. $M_L$ to the case where $q_i$ may contain additional symbols w.r.t. those in $\Sigma$.

In particular, the additional symbols are left unchanged by the unfolding.

**Proposition 6.2.** Let $M$ be an rpq-based mapping of type sound (resp., exact) from $\Sigma_1$ to $\Sigma_2$, and let $\#$ be a symbol not in $\Sigma_1 \cup \Sigma_2$. Then there are RPQs $q_{M_L}$ over $\Sigma_1 \cup \{\#\}$ and $q_M$ over $\Sigma_1 \cup \{\#\}$, both of size linear in $M$, such that an rpq-based 1NFA mapping $M$ of type sound (resp., exact) is a solution to MSYNT[SOUND,LAV,RPQ,RPQ] (resp., MSYNT[EXACT,LAV,RPQ,RPQ]) with input $M$ iff $q_{M_L}[M_L] \subseteq q_M[M]$ (resp., $q_{M_L}[M_L] = q_M[M]$) and that $q_{M_L}[M_L] \neq \emptyset$.

**Proof.** Let $M = \{q_1, \ldots, q_k\}$. We set $q_{M_L} = q_1, \ldots, q_k,$ and $q_M = \emptyset$. Clearly, the fresh symbol $\#$ acts as a separator for the different parts of $q_{M_L}$ and $q_M$. It is easy to verify that, for every 1NFA mapping $M_L$, we have that $q_i[M_L] \subseteq q_l$ (resp., $q_i[M_L] = q_l$) for $i \in \{1, \ldots, k\}$ iff $q_{M_L}[M_L] \subseteq q_M[M]$ (resp., $q_{M_L}[M_L] = q_M[M]$), and that $q_{M_L}[M_L] \neq \emptyset$ for $i \in \{1, \ldots, k\}$ iff $q_{M_L}[M_L] \neq \emptyset$. □

In the following, let $\Sigma_n$ be an non-empty alphabet of new symbols disjoint from $\Sigma_1$ and $\Sigma_2$, and let $\Sigma'_n = \Sigma_1 \cup \Sigma_n$ and $\Sigma''_n = \Sigma_1 \cup \Sigma_n \cup \Sigma_n$. By Proposition 6.2, the problem MSYNT[SOUND,LAV,RPQ,RPQ] (or MSYNT[EXACT,LAV,RPQ,RPQ]) can be polynomially reduced to the problem of checking, whether for languages $q_i$ over $\Sigma''_n$ and $q_i$ over $\Sigma''_n$ there is a 1NFA mapping $M_L$ such that $q_i[M_L] \subseteq q_l$ (resp., $q_i[M_L] = q_l$) and $q_i[M_L] \neq \emptyset$.

Our technique for mapping simplification exploits a characterization of regular languages by means of congruence classes [49,50,40]. Recall that two words $u, v \in \Sigma^*$ are congruent with respect to a language $L \subseteq \Sigma^*$ if, for all words $x, y \in \Sigma^*$ we have that $xuy \in L$ iff $xyv \in L$. Then $A_L = (\Sigma_1 \cup \Sigma_2, \delta_L, s_0, q_1)$ be a 1NFA for $q_1$. Then $A_L$ can be characterized as a set of congruence classes partitioning $\Sigma''_n$. Each congruence class is characterized by a binary relation $G \subseteq \Sigma_1 \times \Sigma_2$ (i.e., a directed graph over $\Sigma_1$), and we define the congruence class associated with $G$ as

$$L(G) = \{w \in \Sigma''_n^* \mid \text{for all } s_1, s_2 \in \Sigma_1, s_2 \in \delta_L(s_1, w) \text{ iff } (s_1, s_2) \in G\}.$$ 

Intuitively, each word $w \in L(G)$ connects $s_1$ to $s_2$ in $A_L$, for each pair $(s_1, s_2) \in G$. For a word $w \in \Sigma''_n^*$, we denote by $[w]_{A_L}$ the congruence class to which $w$ belongs.

It follows immediately from the characterization of congruence classes in terms of binary relations over the states of $A_L$ that the set of congruence classes is closed under concatenation. Specifically, for two congruence classes $L(G_1)$ and $L(G_2)$, respectively with associated relations $G_1$ and $G_2$, the binary relation associated with $L(G_1) \cdot L(G_2)$ is $G_1 \circ G_2$.

We introduce some notation that we use here, and later in this section:

- $G = 2^{\Sigma_1 \times \Sigma_1}$ denotes the set of binary relations associated with the congruence classes for $A_1$.
- $G_1 = \{(s, s) \mid s \in \Sigma_1\}$, and
- $G_1^b = \{(s_1, s_2) \mid s_2 \in \delta_L(s_1, b), \text{ for each } b \in \Sigma_1\}$.

Then, for each $G \in \mathcal{G}_1$, we can characterize the congruence class $L(G)$ associated with $G$ in terms of a DFA.

**Lemma 6.3.** (See [49]) The language $L(G)$ is accepted by the DFA $A_G = (\Sigma_1 \cup \Sigma_2, \delta_G, \delta_A, F_G)$, where $F_G = \{G\}$ and $\delta_A(R, b) = R \circ G_1^b$, for each $R \subseteq \Sigma_1 \times \Sigma_1$ and $b \in \Sigma_1$.

Notice that, if $A_1$ has $m$ states, then $A_G$ has $2m^2$ states.

We observe next that we need to allow for the presence of empty queries in the 1NFA mapping we are looking for. Consider, e.g., $q_1 = (a_1 + a_3) \cdot (a_2 + a_3)$ and $q_2 = b_1 \cdot b_2$. It is easy to see that for all 1NFA mappings $M_L$ such that $q_i[M_L] \subseteq q_i$, we have that $M_L(a_1) = \emptyset$. Thus, for any 1NFA mapping obtained from simplification we have that no nodes are connected through an edge with label $a_3$. One such 1NFA mapping $M_L$ is

$$M_L(a_1) = b_1, \quad M_L(a_2) = b_2, \quad M_L(a_3) = \emptyset.$$ 

Observe also that $[b_1]_{A_1} = [b_1]$ and $[b_2]_{A_2} = [b_2]$, where $A_i$ is the obvious 1NFA for $b_1 \cdot b_2$.

---

1. We use $\cdot$ to denote concatenation between languages, and $G_1 \circ G_2$ to denote composition of binary relations.
6.1. Upper bounds for sound mapping simplification

We first deal with the case of sound mappings, and prove two preliminary results. The first lemma states that w.l.o.g. we can restrict the attention to LA mapping in which the queries are singletons, i.e., queries that are either empty or constituted by a single word.

**Lemma 6.4.** Let \( q_t \) be an RPQ over \( \Sigma_t' \), and \( q_i \) an RPQ over \( \Sigma_i' \). If there exists an RPQ-based LA mapping \( M_L \) such that \( q_i[M_L] \subseteq q_t \), then there exists an RPQ-based LA mapping \( M_L' \) such that \( q_i[M_L'] \subseteq q_t \) in which each query is either a single word over \( \Sigma_t \) or empty.

**Proof.** If \( q_t = \epsilon \), we can simply set \( M_L'(a) = \epsilon \), for each \( a \in \Sigma_t \). Otherwise, since \( q_i[M_L] \neq \emptyset \) and \( q_i[M_L] \subseteq q_t \), there exist a nonempty word \( a_1 \cdots a_k \in q_i \) and a word \( w_1 \cdots w_k \in q_i[M_L] \) and hence in \( q_t \), where \( w_j \in L_t(a_j) \). To define the new LA mapping \( M_L' \), we consider each \( a \in \Sigma_t \) appearing in \( a_1 \cdots a_k \). Notice that \( a \) might appear in \( a_1 \cdots a_k \) multiple times, and suppose the occurrences of \( a \) are \( a_{i_1}, \ldots, a_{i_j} \), corresponding to \( w_{i_1}, \ldots, w_{i_j} \). We chose arbitrarily one \( w_{i_j} \) and set \( M_L'(a) = w_{i_j} \). Instead, for each \( a \in \Sigma_t \) not appearing in \( a_1 \cdots a_k \), we set \( M_L'(a) = \emptyset \). Now, \( q_i[M_L'] \neq \emptyset \) by construction, and since \( M_L'(a) \subseteq M_L(a) \) for every \( a \in \Sigma_t \), we have that \( q_i[M_L'] \subseteq q_i[M_L] \subseteq q_t \). \( \square \)

The next lemma shows that one can close queries in LA mappings under congruence.

**Lemma 6.5.** Let \( q_i \) be an RPQ over \( \Sigma_i' \), and \( q_t \) an RPQ over \( \Sigma_t' \) expressed through a 1NFA \( A_t \), and \( M_L \) a singleton mapping such that \( q_i[M_L] \subseteq q_t \). Then, for \( M_L' \) defined such that
\[
M_L'(a) = \begin{cases} 
[w^a]_{A_t} & \text{if } M_L(a) = w^a, \\
\emptyset & \text{if } M_L(a) = \emptyset,
\end{cases}
\]
we have that \( q_i[M_L'] \subseteq q_t \).

**Proof.** Let \( A_t = (\Sigma_t, S_t, s_0^t, \delta_t, F_t) \). Consider a word \( a_1 \cdots a_h \in q_i \). If there is one of the \( a_i \) such that \( M_L(a_i) = \emptyset \), then \( M_L(a_1) \cdots M_L(a_h) = \emptyset \subseteq q_t \). Otherwise, we have that \( M_L(a_i) = [w_i^a]_{A_t} \), for \( i \in [1, \ldots, h] \), and since \( w_i^a \cdots w_h^a \in q_i[M_L] \subseteq q_t \), there is a sequence \( s_0, s_1, \ldots, s_h \) of states of \( A_t \) such that \( s_0 = s_0^t, s_h \in F_t, \) and \( s_i \in \delta_t(s_{i-1}, w_i^a) \), for \( i \in [1, \ldots, h] \). Consider now, for each \( i \in [1, \ldots, h] \), a word \( w_i^t \in M_L'(a_i) = [w_i^a]_{A_t} \). Making use of the characterization of \( [w_i^a]_{A_t} \) in terms of a binary relation over \( S_t \), we have for each word in \( [w_i^a]_{A_t} \), and in particular for \( w_i^t \), that \( s_i \in \delta_t(s_{i-1}, w_i^t) \). Hence, \( s_h \in \delta_t(s_0^t, w_1^t \cdots w_h^t) \) and \( w_1^t \cdots w_h^t \in q_t \). \( \square \)

From these two lemmas we get that, when searching for a LA mapping \( M_L \) satisfying \( q_i[M_L] \subseteq q_t \), we can restrict the attention to queries that are congruence classes for \( A_t \).

**Lemma 6.6.** Let \( q_i \) be an RPQ over \( \Sigma_i' \), and \( q_t \) an RPQ over \( \Sigma_t' \) expressed through a 1NFA \( A_t \). If there exists an RPQ-based LA mapping \( M_L \) such that \( q_i[M_L] \subseteq q_t \), then there exists an RPQ-based LA mapping \( M_L' \) such that \( q_i[M_L'] \subseteq q_t \), and such that \( M_L'(a) \) is a congruence class for \( A_t \), for each \( a \in \Sigma_t \).

**Proof.** If there exists an RPQ-based LA mapping \( M_L \) such that \( q_i[M_L] \subseteq q_t \), then by Lemma 6.4, w.l.o.g., we can assume that \( M_L \) consists of singleton queries. Then, the claim follows from Lemma 6.5. \( \square \)

We derive now a procedure that, given an RPQ \( q_i \) over \( \Sigma_i' \), and an RPQ \( q_t \) over \( \Sigma_t' \) expressed respectively through 1NFAs \( A_i = (\Sigma_i', S_i, s_0^i, \delta_i, F_i) \) and \( A_t = (\Sigma_t', S_t, s_0^t, \delta_t, F_t) \), checks for the existence of a sound RPQ-based LA mapping \( M_L \) such that

1. \( q_i[M_L] \neq \emptyset \), and
2. \( q_i[M_L] \subseteq q_t \).

Specifically, by Lemma 6.6, it is sufficient to consider LA mappings in which each query is constituted by a single congruence class, which can be represented by a binary relation over the state set \( S_t \) of \( A_t \). Hence, for each \( a \in \Sigma_t \), we guess such a binary relation \( G_a \) and verify that for the LA mapping \( M_L \) defined by \( M_L(a) = L(G_a) \), conditions (1) and (2) are satisfied. In doing so, we exploit Lemma 6.3, which provides a characterization of \( L(G_a) \) in terms of a DFA \( A_{G_a} \).

To check condition (1), we proceed as follows:

1.1 for each \( a \in \Sigma_t \), we check whether \( a \) is a bad symbol, i.e., whether \( L(G_a) = \emptyset \);
1.2 we delete from \( A_t \) each transition labeled by a bad symbol; and
1.3 we check whether the resulting 1NFA accepts a non-empty language.

To check condition (2), we proceed as follows:
2.1 we construct a 1NFA $A_{M_1}$ accepting $q_i[M_1]$;
2.2 we construct the 1NFA $A_G = A_{M_1} \times A_t$ as the product NFA of $A_{M_1}$ and the 1NFA $A_t$ accepting $\Sigma'_t \setminus q_t$;
2.3 we check $A_G$ for emptiness.

To construct $A_{M_1}$, we observe that a word $w$ is in $q_i[M_1]$ if there are a word $a_1 \cdots a_n \in q_i$, and for $i \in \{1, \ldots, n\}$, words $w_i \in L(G_{a_i})$ such that $w = w_1 \cdots w_n$. Hence, $A_{M_1}$ simulates $A_t$ while accepting words in $\Sigma'_t$ that are concatenations of words in the various languages $L(G_{a_i})$. Specifically, $A_{M_1} = (\Sigma'_t, S_{M_1}, s_{M_1}, \delta_{M_1}, F_{M_1})$, where

- $S_{M_1} = \Sigma'_t \times S_t \times \Gamma_t$;
- $s_{M_1} = (\Sigma'_t, S_t, \Gamma_t)$;
- $F_{M_1} = \{(a, s, G_{a}) \mid s \in F_S, a \in \Sigma_S \} \cup \{(b, s, G_{b}) \mid s \in F_S, b \in \Sigma_S \}$;
- and for each $a \in \Sigma'_t, s \in S_t, R \in \Gamma_t$, and $b \in \Sigma_t$,

$$
\delta_{M_1}(a, s, R, b) = \begin{cases} 
\{(a, s, R \circ G_b)\}, & \text{if } a \in \Sigma_S, R \neq G_a; \\
\{(a, s, R \circ G_b)\} \cup \bigcup_{a' \in \Sigma'_t, s' \in \delta_t(s, a)}\{(a', s', G_{b'})\}, & \text{if } a \in \Sigma_S, R = G_a; \\
\emptyset, & \text{otherwise.}
\end{cases}
$$

**Theorem 6.7.** $MSIMP[\text{sound},LAV,RPQ,RPQ]$ is in PSPACE.

**Proof.** By Lemma 6.6, to check whether $MSIMP[\text{sound},LAV,RPQ,RPQ]$ admits a solution, it suffices to guess for each symbol $a \in \Sigma_S$ a binary relation $G_a$ over the state set $S_t$ of $A_t$, and check whether for the resulting $\text{RPQ}$-based $LAV$ mapping $M_1$ conditions (1) and (2) hold. When checking condition (1), the emptiness test in item (11) can be done for each $a \in \Sigma_S$ in $\text{NLocSpace}$ in $|A_G|$, and since the number of states of $A_G$ is exponential in $|A_t|$, in $\text{PSpace}$ in $|A_t|$. Knowing the set of bad symbols, the transformation in item (12) is linear in $|A_t|$. Finally, the non-emptiness test in item (13) can be done in $\text{NLocSpace}$ in $|A_t|$. When checking condition (2), we do not need to construct $A_{M_1}$, $A_t$, and $A_G$ explicitly, but can check the nonemptiness of $A_G$ on the fly while constructing $A_{M_1}$ and complementing $A_t$. Hence, since the number of states of $A_{M_1}$ is in $|A_S|$ and exponential in $|A_t|$, we get that condition (2) can be checked in $\text{PSpace}$ in $|A_t|$ and in $\text{NLocSpace}$ in $|A_t|$. □

**6.2. Upper bounds for exact mapping simplification and for maximal mapping synthesis**

The method based on congruence classes can be adapted to address also $LAV$ simplification for exact mappings. The difference w.r.t. sound mappings is that in this case we need to consider also $LAV$ mappings in which the queries are unions of congruence classes. Indeed, congruence classes (and hence solutions to the $LAV$ mapping synthesis problem) are not closed under union, as shown by the following example.

Let $q_i = a_1 \cdot a_2$ and $q_i = 00 + 01 + 10$. Then the following two incomparable mappings are solutions to $\text{MAXM-SYNT}[\text{sound},LAV,RPQ,RPQ]$ when the input mapping is $\{q_i \rightsquigarrow q_i\}$:

$$
M_1^1(a_1) = 0, \quad M_1^2(a_1) = 0 + 1, \\
M_1^1(a_2) = 0 + 1, \quad M_1^2(a_2) = 0.
$$

Notice that the mapping $M_1$, where $M_1(a_i) = M_1^1(a_i) + M_1^2(a_i)$, for $i \in \{1, 2\}$, is not a solution, since $q_i[M_1]$ includes 11.

On the other hand, we can show that considering mapping in which the queries are unions of congruence classes is sufficient to obtain maximal unfoldings. We first generalize Lemma 6.5 to non-singleton queries.

**Lemma 6.8.** Let $q_i$ be an $\text{RPQ}$ over $\Sigma'_t$, $q_i$ an $\text{RPQ}$ over $\Sigma'_t$ expressed through a 1NFA $A_t$, and $M_1$ a $LAV$ mapping such that $q_i[M_1] \subseteq q_i$. Then for $M'_i$ with

$$
M'_i(a) = \begin{cases} 
\bigcup_{w \in M_1(a)}[w]_{A_t}, & \text{if } M_1(a) \neq \emptyset, \\
\emptyset, & \text{if } M_1(a) = \emptyset,
\end{cases}
$$

we have that $q_i[M'_i] \subseteq q_i$.

**Proof.** Consider a word $a_1 \cdots a_h \in q_i$. If there is one of the $a_i$ such that $M_1(a_i) = \emptyset$, then $M'_i(a_1) \cdots M'_i(a_h) = \emptyset \subseteq q_i$. Otherwise, we have that, for $i \in \{1, \ldots, h\}$, for some $w_i \in M_1(a_i)$, the word $w_i \cdots w_h \in q_i[M_1] \subseteq L(A_t)$. We show that, for each $i \in \{1, \ldots, h\}$, we also have that $w_i \cdots w_{i-1} \cdot w_i \cdot w_{i+1} \cdots w_h \in L(A_t)$, for each $w_i \in \bigcup_{w \in M_1(a_i)}[w]_{A_t}$. First, since $q_i[M_1] \subseteq q_i$, if $w_i \cdots w_h \in q_i[M_1] \subseteq L(A_t)$, then, for each $w \in M_1(a_i)$, we also have that $w_i \cdots w_{i-1} \cdot w \cdot w_{i+1} \cdots w_h \in q_i[M_1] \subseteq L(A_t)$. Then there is a sequence $s_0, s_1, \ldots, s_h$ of states of $A_t$ such that $s_0 = s'_i, s_h \in F_t, s_j \in \delta_t(s_{j-1}, w_j)$, for
Given a mapping \( M = \{ q_i \sim q_j \} \), where \( q_i \) is defined by a 1NFA \( A_i \), every solution \( M_l \) to MaxMSynt\( [\text{sound,lav,rpq,rpq}] \) with input \( M \) is such that each query in \( M_l \) is a union of congruence classes for \( A_i \).

**Proof.** Consider a solution \( M_l \) to MaxMSynt\( [\text{sound,lav,rpq,rpq}] \) with input \( M \), and assume that for some \( a \in \Sigma \), \( M_l(a) \) is not a union of congruence classes for \( A_i \). Then there are some word \( w \in M_l(a) \) and some word \( w' \in \{ w \}_{\Sigma} \) such that \( w' \not\in M_l(a) \). By Lemma 6.8, the mapping \( M_l' \) with \( M_l'(a) = M_l(a) \cup \{ w' \} \) is also a solution to MSynt\( [\text{sound,lav,rpq,rpq}] \) with input \( M \), thus contradicting the maximality of \( M_l \). \( \square \)

By observing that every solution to LAV simplification for the exact case has to be a maximal LAV mapping that implies the given mapping, we get the following upper bound for LAV simplification in the exact case.

**Theorem 6.10.** MSimp\( [\text{exact,lav,rpq,rpq}] \) is in ExpSpace.

**Proof.** By Lemma 6.9, we can nondeterministically choose mappings \( M_i \) in which the queries are unions of congruence classes and then test whether \( q_i = q_i[M_i] \). To do so, we build a 1NFA \( A_3,M_i \) accepting \( q_i[M_i] \) as follows. We start by observing that for each union \( U \) of congruence classes, we can build the automaton \( A_U = (\Sigma, q_U, R_U, \delta_U, U) \) accepting the words in \( U \), which incidentally, is deterministic. Hence, by substituting each \( \alpha \)-transition in the 1NFA \( A_i \) for \( q_i \) with the 1NFA \( A_{U_i} \) where \( M_i(a) = U_0 \), we obtain a 1NFA \( A_3,M_i \). Note that, even when \( A_i \) is deterministic, \( A_3,M_i \) may be nondeterministic.

To test \( q_i[M_i] \subseteq q_i \), we complement \( A_i \), obtaining the 1NFA \( A_i \), and check the 1NFA \( A_3,M_i \times A_i \) for emptiness. The size of \( A_3,M_i \times A_i \) is polynomial in the size of \( A_i \) and exponential in the size of \( A_i \). Checking for emptiness can be done in exponential time, and considering the initial nondeterministic guess, we get a NExpTime upper bound.

To test \( q_i \subseteq q_i[M_i] \), we complement \( A_3,M_i \), obtaining the 1NFA \( \overline{A_3,M_i} \), and check \( A_i \times \overline{A_3,M_i} \) for emptiness. Since \( A_3,M_i \) is nondeterministic, complementation is exponential. However, we observe again that such a complementation can be done on the fly in ExpSpace, while checking for emptiness and intersecting with \( A_i \). As a consequence, considering the initial nondeterministic guess, MSimp\( [\text{exact,lav,rpq,rpq}] \) can be decided in NExpSpace, which is equivalent to ExpSpace. \( \square \)

Note that the proofs of Theorems 6.7 and 6.10 imply that, w.r.t. LAV mapping simplification, considering queries that are RPQs (as opposed to general, possibly non-regular, path languages) is not a restriction, since the existence of general LAV mappings implies the existence of regular ones. This is also in line with a similar observation holding for the existence of rewrites of RPQs w.r.t. RPQ views [42].

Finally, we observe that using the machinery based on unions of congruence classes, we can also solve the maximal mapping synthesis problem. We guess a mapping and check that it is a solution to mapping synthesis. To check that it is a maximal solution, we generate all other mappings and check that, if they are solutions, they are contained in our candidate solution.

**Theorem 6.11.** A solution to MaxMSynt\( [\text{sound,lav,rpq,rpq}] \) and to MaxMSynt\( [\text{exact,lav,rpq,rpq}] \) can be computed in ExpSpace.

**6.3. Lower bounds for mapping simplification**

It turns out that the upper bound established for the sound case is tight:

**Theorem 6.12.** MSimp\( [\text{sound,lav,rpq,rpq}] \) is PSpace-hard.

**Proof.** The proof is by a reduction from the universality problem for REs. Given an RE \( e \) over the alphabet \( \Sigma_l = \{ a_1, \ldots, a_n \} \), let \( \Sigma_s = \{ b_1, \ldots, b_n \} \), and let \( M_e \) be the mapping constituted by the following assertions:

\[
\Sigma^* \sim e, \quad a_1 \sim b_1 \quad \cdots \quad a_n \sim b_n. \tag{5}
\]

\[
\Sigma^* \sim b_1 \quad \cdots \quad a_n \sim b_n. \tag{6}
\]

We show that \( e \) is universal iff MSimp\( [\text{sound,lav,rpq,rpq}] \) with input \( M_e \) admits a solution. For the “only-if” part, assume that \( e \) is universal and consider the LAV mapping \( M_l \) consisting of the mapping assertions (6), we have that \( \Sigma^*[M_l] = \Sigma^* \), and since \( e \) is universal, also \( \Sigma^*[M_l] \subseteq e \). For the “if” part, consider a LAV mapping \( M_l \) such that \( q_i[M_l] \subseteq q_i \) and \( q_i[M_l] \neq \emptyset \), for each mapping assertion \( q_i \sim q_i \) in \( M_e \). By the mapping assertions (6), we have that \( M_l(a_i) \neq \emptyset \) (since \( a_i \) is the left-hand side of a mapping assertion) and that \( M_l(a_i) \) must include \( b_i \), for \( i \in \{ 1, \ldots, n \} \), and hence \( \Sigma^*[M_l] = \Sigma^* \). Since \( \Sigma^*[M_l] \subseteq e \), we have that \( e \) is universal. \( \square \)

It is easy to see that the above proof shows also PSpace-hardness of simplification for exact mappings.
Corollary 6.13. MSimp[exact, lAV, rpq, rpq] is PSPACE-hard.

We do not have a tight upper bound for MSimp[exact, lAV, rpq, rpq]. We do, however, show a tight lower bound for a generalization of MSimp[exact, lAV, rpq, rpq] in which we allow the input mapping to contain both sound and exact mapping assertions. We do so by showing an ExpSpace lower bound for a problem that is closely related to the mapping simplification problem.

Consider a finite alphabet Σ and a finite set V of variables. A language constraint is a statement of the form e1 ⊑ e2, where e1 and e2 are regular expressions over Σ ∪ V. A language-constraint problem P is a finite set of language constraints. A solution to P is an assignment σ : V → 2Σ+, assigning a language over Σ to each variable in V such that L(e1[σ]) ⊑ L(e2[σ]). It is easy to express the universality problem for context-free grammars as a language-constraint problem, which implies that the latter is undecidable. Here we consider left-handed language constraint problems, where we allow constraints of the form e1 ⊑ e2 and e3 = e4, but require that variables appear only in the left-hand side of the constraint. The technique for the exact version of the lAV mapping-simplification problem can be used to show an ExpSpace upper bound for solving left-handed language constraint problems. We now show a matching lower bound.

To prove the result we exploit a reduction from tiling problems [51,52]. A tile is a unit square of one of several types and the tiling problem we consider is specified by means of a finite set Δ of tile types, two binary relations H and P over Δ, representing horizontal and vertical adjacency relations, respectively, and two distinguished tile types t5, t6 ∈ Δ. The tiling problem here consists in determining whether, for a given number n in unary, a region of the integer plane of size 2n × k, for some k > 0, can be tiled consistently with the adjacency relations H and P, and with the left bottom tile of the region of type t5 and the right upper tile of type t6. We also require that the last tile of a row and the first tile of the next row are consistent with H. Using a reduction from acceptance of ExpSpace Turing machines analogous to the one in [51], it can be shown that this tiling problem is ExpSpace-complete.


Proof. Let T = (Δ, H, P, t5, t6) be an instance of the ExpSpace-complete tiling problem above and n a number in unary. The alphabet is Σ = Δ ∪ {0, 1}3 ∪ {#}. Intuitively, the letters in Δ denote tiles, symbols in {0, 1}3 denote address bits, and # denotes a separation marker. The idea is to encode each tile cell by a word of length n + 2 of the form # · ((0, 1)3)n · Δ, consisting of a marker, an n-bit address, and a tile symbol. We use an element in {0, 1}3 for each address bit to make it easy to check that two n-bit addresses are consecutive; we use n-bits for the current address, n bits for the carry, and n bits for the next address. Thus, each tiling can be described by a word in (# · ((0, 1)3)n · Δ)+, obtained by encoding each cell as described above, and then concatenating the symbols, first column by column and then row by row.

Consider a word w ∈ Σ+. Such a word does not describe a proper tiling if one of the following errors can be found in the word:

1. The symbol # does not occur precisely in positions (n + 2)i, for i = 0, 1, . . . .
2. The symbols in Δ do not occur precisely in positions n + 1 + (n + 2)i, for i = 0, 1, . . . .
3. The first address is not 00.
4. The last address is not 11.
5. There is a pair of adjacent but not successive addresses.
6. The first tile is not t5.
7. The last tile is not t6.
8. There is a pair of adjacent blocks with tiles that violate the relation H.
9. There is a pair of vertically adjacent blocks with tiles that violate the relation P.

We do need to define the notion of vertical adjacency. Two blocks are vertically adjacent if their addresses agree and either both addresses are 00 and there is no occurrence of 01 between them, or both addresses are not 00 and there is precisely one occurrence of 01 between them.

If the tiling problem has no solution, then every word in Σ* must contain an error. Conversely, if the tiling problem has a solution, then that solution can be described by a word with no errors. We now define a constraint of the form eerror = Σ*, where the “task” of eerror is to discover errors in candidate words. The expression eerror is the sum of several terms corresponding to the various errors. We now sketch how to “discover” these possible errors. In order to have the left-hand sides use only variables, we introduce a variable vα for each letter a ∈ Σ, accompanied by the constraint vα = a. We use VΣ to abbreviate \[ \sum_{\alpha \in \Sigma} v_{\alpha} \].

Most of the errors can be discovered with a single regular expression. For example, the error where the symbol # does not occur precisely in positions (n + 2)i, for i = 0, 1, . . . , is described using the expression

\[ \left( \sum_{1 \leq i \leq n + 1} v_{\Sigma}^{i} \right)^{n+2} \cdot \left( \sum_{1 \leq i \leq n + 1} v_{\Sigma}^{i} \right) \cdot # \cdot V_{\Sigma}^{n+2} \].
We encode addresses by words in \((\{0,1\}^3)^n\), which consist of three \(n\)-bit words: the first word is an \(n\)-bit address, the second word is the successor of that address, and the third word is the sequence of \(n\) carry bits. The expression \(e_{\text{error}}\) contains a term that finds errors in words of the form \((\{0,1\}^3)^n\) used in the encoding candidate solutions. To ensure that adjacent addresses are successive, \(e_{\text{error}}\) contains, for \(0 \leq i \leq n-1\), the expression

\[
(\mathcal{V}_\Sigma^{i+2})^* \cdot \cdot ((\{0,1\}^3)^i \cdot \{0,1\} \times \{0,1\} \cdot ((\{0,1\}^3)^{n-i-1} \cdot \Delta \\
\cdot \cdot ((\{0,1\}^3)^i \cdot \{1\} \times \{0,1\} \cdot ((\{0,1\}^3)^{n-i-1} \cdot \mathcal{V}_\Sigma^{i+2})^*)
\]

as well as the expression

\[
(\mathcal{V}_\Sigma^{i+2})^* \cdot \cdot ((\{0,1\}^3)^i \cdot \{0,1\} \times \{0,1\} \cdot ((\{0,1\}^3)^{n-i-1} \cdot \Delta \\
\cdot \cdot ((\{0,1\}^3)^i \cdot \{0\} \times \{0,1\} \cdot ((\{0,1\}^3)^{n-i-1} \cdot \mathcal{V}_\Sigma^{i+2})^*)
\]

These expressions compare corresponding bits in adjacent addresses; the successor of the first address has to agree with the second address.

The one error that is challenging is where there is a pair of vertically adjacent blocks with tiles that violate the relation \(P\). Discovering this error is more difficult and cannot be done by one regular expression; rather, several additional constraints are needed. For simplicity we ignore here the fact that each address bit is encoded by three bits rather than one.

Let \(e_{\text{nzava}}\) be a regular expression that describes non-zero addresses: \(\sum_{0 \leq i \leq n-1} (\{0,1\}^i \cdot \{0,1\}^{n-i-1})\).

We add to \(e_{\text{error}}\) the following term, which discovers non-matching tiles at zero-addressed vertically adjacent tiles:

\[
\sum_{(t,t') \in P} \left((\mathcal{V}_\Sigma^* \cdot \# \cdot 0^n \cdot t \cdot (\# \cdot e_{\text{nzava}} \cdot \Delta)^* \cdot \# \cdot 0^n \cdot t' \cdot \mathcal{V}_\Sigma^*\right)
\]

We need to deal with non-zero-addressed vertically adjacent blocks. For this we use several constraints. First:

\[
V_{\text{nzava}} \subseteq \sum_{(t,t') \in P} \left((\# \cdot e_{\text{nzava}} \cdot t \cdot (\# \cdot e_{\text{nzava}} \cdot \Delta)^* \cdot \# \cdot 0^n \cdot \Delta \cdot (\# \cdot e_{\text{nzava}} \cdot \Delta)^* \cdot \# \cdot e_{\text{nzava}} \cdot t')\right).
\]

This says that \(V_{\text{nzava}}\) describes sequences of blocks that start and end with a pair of non-matching non-zero-addressed blocks, with a single zero-addressed block in between. We still have to impose the constraint that the first and last block have equal addresses. We do this with \(n\) constraints, one for each of the address. That is for each \(i\), \(0 \leq i \leq n-1\), we add the constraint:

\[
V_{\text{nzava}} \subseteq \left((\# \cdot [0,1]^i \cdot 0 \cdot [0,1]^{n-i-1} \cdot \Delta \cdot (\# \cdot [0,1]^n \cdot \Delta)^* \cdot \# \cdot [0,1]^i \cdot 0 \cdot [0,1]^{n-i-1} \cdot \Delta \right)
\]

\[
+ \left((\# \cdot [0,1]^i \cdot 1 \cdot [0,1]^{n-i-1} \cdot \Delta \cdot (\# \cdot [0,1]^n \cdot \Delta)^* \cdot \# \cdot [0,1]^i \cdot 1 \cdot [0,1]^{n-i-1} \cdot \Delta \right).
\]

This constraint says that the \(i\)-th bits of the first and last addresses are either both 0 or both 1.

Now we can add to \(e_{\text{error}}\) the term \(V_{\Sigma}^* \cdot V_{\text{nzava}} \cdot V_{\Sigma}^*\), which discovers all errors due to not-matching, non-zero-addressed vertically adjacent blocks.

Note that the constraint system constructed is of size quadratic in the size of the tiling system. If the tiling problem has no solution, then every word in \(\Sigma^*\) contains an error and the constraint problem constructed is satisfiable. If the tiling problem has a solution, then a word describing a proper tiling has no error, and for no assignment \(\sigma : \mathcal{V} \rightarrow 2^{\Sigma^*}\) we have \(\mathcal{L}(e_{\text{error}}(\sigma)) = \Sigma^*\), since \(e_{\text{error}}\) captures only errors. \(\square\)

Let \(\text{MSimp}[\text{mixed}, \text{lav}, \text{rpq}, \text{rpq}]\) be the following decision problem: given an \(\text{rpq}\)-based schema mapping \(M\) consisting both of \text{sound} and of \text{exact} mapping assertions, check whether there exists an \(\text{rpq}\)-based \(\text{lav}\) schema mapping \(M'\) of type \text{exact} such that \(M' \models M\) and \(M' \not\models_{\text{lav}} M\).

As a corollary of Theorem 6.14, we get the following result.

**Corollary 6.15.** \(\text{MSimp}[\text{mixed}, \text{lav}, \text{rpq}, \text{rpq}]\) is ExpSpace-complete.

We conjecture that \(\text{MSimp}[\text{exact}, \text{lav}, \text{rpq}, \text{rpq}]\) is also ExpSpace-complete.

Our results on simplification for \(\text{rpqs}\) are summarized in Table 1.

### 7. Extensions

In this section we sketch the extension of the results of the previous section on simplification in terms of \(\text{lav}\) mappings to more expressive classes of queries: \(2\text{rpqs}\) and their conjunctions.
7.1. 2RPQs

Consider now simplification for mappings based on 2RPQs, expressed by means of 1NFAs over the alphabets \( \Sigma^+ \) and \( \Sigma^\pm \).

A key concept for 2RPQs is that of folding of a language [22], which intuitively denotes the set of words that are the result of repeatedly canceling out adjacent occurrences of a symbol and its inverse. Let \( u, v \in \Sigma^\pm \). We say that \( v \) folds onto \( u \), denoted \( v \rightsquigarrow u \), if \( v \) can be “folded” on \( u \), e.g., \( abb \cdot bc \rightsquigarrow abc \). Formally, we say that \( v = v_1 \cdots v_m \) folds onto \( u = u_1 \cdots u_n \) if there is a sequence \( i_0, \ldots, i_m \) of positive integers between 0 and \(|u|\) such that:

- \( i_0 = 0 \) and \( i_m = n \), and
- for \( j \in [0, \ldots, m] \), either \( i_{j+1} = i_j + 1 \) and \( v_{j+1} = u_{i_{j+1}} \), or \( i_{j+1} = i_j - 1 \) and \( v_{j+1} = u_{i_j}^{-1} \).

Let \( L \) be a language over \( \Sigma^\pm \). We define \( \text{fold}(L) = \{ u | v \rightsquigarrow u, \ v \in L \} \).

A language-theoretic characterization for containment of 2RPQs was provided in [22]:

**Lemma 7.1.** Let \( q_1 \) and \( q_2 \) be 2RPQs. Then \( q_1 \subseteq q_2 \) iff \( \mathcal{L}(q_1) \subseteq \text{fold}(\mathcal{L}(q_2)) \).

Furthermore, it is shown in [22] that if \( A \) is an \( n \)-state 1NFA over \( \Sigma^\pm \), then there is a 2NFA \( A^f \) over \( \Sigma \) for \( \text{fold}(\mathcal{L}(A)) \) with \( n \cdot (|\Sigma| + 1) \) states. (We use 2NFA to refer to two-way automata.)

In the mapping simplification problem, we are given 2RPQs \( q_1 \) and \( q_2 \), expressed as 1NFAs \( A_1 \) and \( A_2 \), respectively, and we are asked whether there exists a 2RPQ-based \( L \) mapping \( M_L \) such that \( q_1[M_L] \subseteq q_1 \) or \( q_2[M_L] = q_2 \), and also \( q_2[V] \neq \emptyset \).

The approach using congruence classes described above applies also to 2RPQs. A simplistic approach would be to convert the 2NFA for \( \text{fold}(\mathcal{L}(A_1)) \) into a 1NFA, with an exponential blow-up, and proceed as in Section 6. To avoid this exponential blowup, we need an exponential bound on the number of congruence classes. For a 1NFA, we saw that each congruence class can be defined in terms of a binary relation over its set of states. It turns out that for a 2NFA \( A \), a congruence class can be defined in terms of four binary relations over the set \( S_1 \) of states of \( A \):

1. \( R_{lf} \): a pair \((s_1, s_2) \in R_{lf} \) means that there is a word \( w \) that leads \( A \) from \( s_1 \) to \( s_2 \), where \( w \) is entered on the left and exited on the right.
2. \( R_{ll} \): a pair \((s_1, s_2) \in R_{ll} \) means that there is a word \( w \) that leads \( A \) from \( s_1 \) to \( s_2 \), where \( w \) is entered on the left and exited on the left.
3. \( R_{rr} \): a pair \((s_1, s_2) \in R_{rr} \) means that there is a word \( w \) that leads \( A \) from \( s_1 \) to \( s_2 \), where \( w \) is entered on the right and exited on the right.
4. \( R_{lr} \): a pair \((s_1, s_2) \in R_{lr} \) means that there is a word \( w \) that leads \( A \) from \( s_1 \) to \( s_2 \), where \( w \) is entered on the right and exited on the left.

That is, every choice of four binary relations over the state space of \( A^f \) corresponds to a congruence class with respect to the language of \( A^f \). Thus, the number of congruence classes when \( A^f \) has \( m \) states is \( 2^{4m^2} \) rather than \( 2^{m^2} \), which is still an exponential. This enables us to adapt the technique of Section 6 with essentially the same complexity bounds.

**Theorem 7.2.** \( \text{MSimp}[\text{sound,LAV},2\text{RPQ},2\text{RPQ}] \) is PSpace-complete.

\( \text{MSimp}[\text{exact,LAV},2\text{RPQ},2\text{RPQ}] \) is in ExpSpace.

7.2. Conjunctions of 2RPQs and their unions

Finally, we consider the mapping simplification problem for conjunctions of 2RPQs and their unions, abbreviated (U)\( \cup \)2RPQs [53], which are (unions of) conjunctive queries constructed from binary atoms whose predicate is a 2RPQ. Specifically, a 2CRPQ \( q \) of arity \( n \) is written in the form:

\[ \{ (x_1, \ldots, x_n) \mid q_1(y_1, y_2) \land \cdots \land q_m(y_{2m-1}, y_{2m}) \} \]

where \( x_1, \ldots, x_n, y_1, \ldots, y_{2m} \) range over a set \( \{ z_1, \ldots, z_k \} \) of variables, \( \{ x_1, \ldots, x_n \} \subseteq \{ y_1, \ldots, y_{2m} \} \), and each \( q_j \) is a 2RPQ. When evaluated over a database \( D \) over \( \Sigma \), the 2CRPQ \( q \) computes the set of tuples \((o_1, \ldots, o_n)\) of objects such that there is a total mapping \( \psi \) from \( \{ z_1, \ldots, z_k \} \) to the objects in \( D \) with \( \psi(x_i) = o_i \), for \( i \in \{1, \ldots, n\} \), and \( (\psi(y_{2j-1}), \psi(y_{2j})) \in q_j^D \), for \( j \in \{1, \ldots, m\} \).

Consider first the mapping simplification problem for the case where the input mapping is expressed in terms of CRPQs, where the constituent RPQs are expressed by means of 1NFAs, over the alphabets \( \Sigma^+ \) and \( \Sigma^\pm \). Here the LAV mappings have to be in terms of RPQs, rather than CRPQs, since CRPQs are not closed under substitutions. The crux of our approach is to reduce containment of two CRPQs, \( q_1 \) and \( q_2 \) to containment of standard languages. This was done in [53]. Let \( q_h \), for \( h = \{1, 2\} \), be in the form:

\[ q_h = \{ (x_1, \ldots, x_n) \mid q_h,1(y_h,1, y_h,2) \land \cdots \land q_h,m_h(y_h,2m_h-1, y_h,2m_h) \} \]
and let $V_1$, $V_2$ be the sets of variables of $q_1$ and $q_2$ respectively. It is shown in [53] that the containment $q_1 \subseteq q_2$ can be reduced to the containment $L(A_1) \subseteq L(A_2)$ of two word automata $A_1$ and $A_2$. $A_1$ is a 1NFA, whose size is exponential in $q_1$ and it accepts certain words of the form

$$\$d_1 w_1 d_2 d_3 w_2 d_4 \cdots d_{2m_1} w_{m_1} d_{2m_1} \$$$

where each $d_i$ is a subset of $V_1$ and the words $w_i$ are over the alphabet of $A_1$. Such words constitute a linear representation of certain graph databases that are canonical for $q_1$ in some sense. $A_2$ is a 2NFA, whose size is exponential in the size of $q_2$, and it accepts words of the above form if there is an appropriate mapping from $q_2$ to the database represented by these words. The reduction of the containment $q_1 \subseteq q_2$ to $L(A_1) \subseteq L(A_2)$ is shown in [53].

The ability to reduce containment of CRPQs to containment of word automata means that we can also apply the congruence-class technique of Section 6. Suppose that we have an RPQ-based LAV mapping $M_1$ such that $L(A_1[M_1]) \subseteq L(A_2)$. Then we can again assume that the queries in the LAV mapping are closed with respect to the congruence classes of $A_1$. Thus, the techniques of Section 6 can be applied, and one can show that $MSimp[sound, LAV, CRPQ, RPQ]$ is in ExpSpace and $MSimpa[exact, LAV, CRPQ, RPQ]$ is in 2ExpSpace.

Finally, dealing with UC2RPQ queries, which combine UCQs and 2RPQs, requires combining the techniques developed for RPQs, 2RPQs, and CRPQs. The key idea is the reduction of query containment to containment of word automata. The resulting upper bounds are identical to those for CRPQs.

8. Conclusions

We have introduced the problem of simplifying schema mappings based on logical implication. The problem comes in different forms, depending on the type of simplification to achieve, on whether the mappings are sound or exact, and on the types of queries used in the mappings. We have provided a formalization of the problem, and we have presented techniques and complexity bounds for both relational and graph databases. We have concentrated on LAV simplification, and we have discussed the GAV case only for relational schema mappings. Our results are summarized in Table 1. There are a number of results left open by our investigation. In particular, GAV simplification and synthesis for RPQs is largely open. Moreover, in LAV tight lower-bounds are missing for the cases of UCQs and RPQs. We observe that the techniques we have presented in Sections 5, 6, and 7 to prove our upper bounds for LAV simplification in the sound and exact cases are based on performing suitable guesses and containment checks independently for each input mapping assertion. Hence, they extend easily to the mixed case, providing the same upper bounds as for the exact case, which justifies the entries in the last row of the table.

In the future, we plan to continue investigating schema mapping simplification along different directions. In particular, we aim at addressing GAV simplification for graph databases, and we plan to study schema mapping simplification for tree-based (e.g., XML) semi-structured data.

Acknowledgments

The work is partially supported by the EU under the project ACSI (Artifact-Centric Service Interoperation), grant No. FP7-257593, by Regione Lazio under the project “Integrazione semantica di dati e servizi per le aziende in rete”, and by NSF grants CCF-0613889, CCF-0728882, and CNS 1049862.

References

834


