View Synthesis from Schema Mappings

Diego Calvanese
Faculty of Computer Science
Free Univ. of Bozen-Bolzano
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

Giuseppe De Giacomo
Maurizio Lenzerini
Dip. di Inf. e Sist.
Univ. di Roma “La Sapienza”
I-00198 Roma, Italy
lastname@dis.uniroma1.it

Moshe Y. Vardi
Dep. of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892,
U.S.A.
vardi@cs.rice.edu

ABSTRACT
In data management, and in particular in data integration, data exchange, query optimization, and data privacy, the notion of view plays a central role. In several contexts, such as data integration, data mashups, and data warehousing, the need arises of designing views starting from a set of known correspondences between queries over different schemas. In this paper we deal with the issue of automating such a design process. We call this novel problem “view synthesis from schema mappings”: given a set of schema mappings, each relating a query over a source schema to a query over a target schema, automatically synthesize for each source a view over the target schema in such a way that for each mapping, the query over the source is a rewriting of the query over the target wrt the synthesized views. We study view synthesis from schema mappings both in the relational setting, where queries and views are (unions of) conjunctive queries, and in the semistructured data setting, where queries and views are (two-way) regular path queries, as well as unions of conjunctions thereof. We provide techniques and complexity upper bounds for each of these cases.

1. INTRODUCTION
A view is essentially a (virtual or materialized) data set that is known to be the result of executing a specific query over an underlying database. There are several data management tasks where the notion of view plays an important role [21].

- In database design, following the well-known principle of data independence, views may be used to provide a logical description of the storage schema (cf., [33]). In this setting, since queries are expressed at the logical level, computing a query plan over the physical storage involves deciding how to use the views in the query answering process.

- In query optimization [13], the computation of the answer set to a query may take advantage of materialized views, because part of the data needed for the computation may be already available in the view extensions.

- In data privacy, authorization views associated with a user represent the data that such user is allowed to access [35]. When the system computes the result of a query posed to a specific user, only those answers deriving from the content of the corresponding authorization views are provided to the user.

- In data integration, data warehousing, and data exchange, a target schema represents the information model used to either accessing, or materializing the data residing in a set of sources [24, 25]. In these contexts, views are used to provide a characterization of the semantics of the data sources in terms of the elements of the target schema, and answering target queries amounts to suitable accessing the views.

The above discussion points out that techniques for using the available views when computing the answers to query are needed in a variety of data management scenarios. Query processing using views is defined as the problem of computing the answer to a query by relying on the knowledge about a set of views, where by “knowledge” we mean both view definitions and view extensions [22].

View-based query processing. Not surprisingly, the recent database literature witnesses a proliferation of methods, algorithms and complexity characterizations for this problem. Two approaches have emerged, namely, query rewriting and query answering. In the former approach, the goal is to reformulate the query into an expression that refers to the views (or only to the views), and provides the answer to the query when evaluated over the view extension. In the latter approach, one aims at computing the so-called certain answers, i.e., the tuples satisfying the query in all databases consistent with the views.

Query rewriting has been studied in relational databases for the case of conjunctive queries, and many of their variants, both with and without integrity constraints (see a survey in [22]). A comprehensive framework for view-based query answering in relational databases, as well as several interesting complexity results for different query languages, are presented in [2, 19].

View-based query processing has also been addressed in the context of semi-structured databases. In the case of graph-based models, the problem has been studied for the class of regular path queries and its extensions (see, for example, [9, 20]). In the case of XML-based model, results on both view-based query rewriting and view-based query
answering are reported in for several variants of the XPath query language (see, for example, \[11\], \[12\]).

Where do the views come from? All the above works assume that the set of views to be used during query processing is available. Therefore, a natural question arises: where do these views come from? Some recent papers address this issue from different points of views. In \[14\], the authors introduce the so-called “view definition problem”: given a database instance and a corresponding view instance, find the most succinct and accurate view definition, for a specific view definition language. Algorithms and complexity results are reported for several families of languages. (Note that the problem dealt with in \[32\] can be seen as a variant of the view definition problem.)

In the context of both query optimization and data warehousing, there has been a lot of interest in the so-called “view-selection problem” \[14\], that is the problem of choosing a set of views to materialize over a database schema, such that the cost of evaluating a set of workload queries is minimized and such that the views fit into a pre-specified storage constraint. Note that the input to an instance of this problem includes knowledge about both a set of queries that the selected views should support, and a set of constraints on space limits for the views.

In data integration and exchange, the “mapping discovery problem” has received significant attention in the last years: find correspondences between a set of data sources and a target (or, global) schema so that queries posed to the target can be answered by exploiting such mappings, and accessing the sources accordingly. Several types of mappings have been investigated in the literature \[25\]. In particular, in the so-called LAV (Local-As-Views) approach, mappings associate to each source a view over the target schema. In other words, the LAV approach to data integration and exchange advocate the idea of modeling each source as a view.

The problem of semi-automatically discovering mappings has been addressed both by the database and AI communities \[25\], \[18\]. In \[30\], a theoretical framework is presented for discovering relationships between two database instances over distinct schemata. In particular, the problem of understanding the relationship between two instances is formalized as that of obtaining a schema mapping so that a minimum repair of this mapping provides a perfect description of the target instance. In \[16\], the iMAP system is described, which semi-automatically discovers both 1-1 and complex matches between different data schemata, where a match specifies semantic correspondences between elements of both schemas, and is therefore analogous to mappings. None of the above papers addresses the issue of automatically deriving LAV mappings. This implies that none of the methods described in those papers can be used directly to derive the view definitions associated with the data sources.

Synthesizing views from schema mappings. In this paper, we tackle the problem of deriving view definitions from a different angle. We assume that we have as input a set of schema mappings, i.e., a set of correspondences between a source schema and a target schema, where each correspondence relates a source query (i.e., a query over the sources) to a target query. The goal is to automatically synthesize one view for each source relation, in such a way that all schema mappings are captured. We use two interpretations of a “schema mapping captured by the synthesized views”. Under the former interpretation, the schema map-
Describing the semantics of such a service means to describe it as a query over a domain-specific alphabet. Once this has been done, the mashup is essentially characterized as a schema mapping from the external sources to a virtual global database. So, similarly to the above mentioned cases, view synthesis can be used to turn the set of mashups into a full-fledged data LAV data integration system, with all the advantages pointed out before.

In all the above scenarios, view-synthesis is used for deriving a set of LAV mappings starting from a set of available schema mappings. This is not surprising, since, as we said before, in the LAV approach sources are modeled as views. Nevertheless, one might wonder why deriving the LAV mappings, and not using directly the original schema mappings for data warehousing, integration and mashup. There are several reasons why one is interested in LAV mappings:

- LAV mappings allow one to exploit the algorithms and the techniques that have been developed for view-based query processing in the last years.

- Several recent papers point out that the language of LAV mappings enjoys many desirable properties. For example, in [34] it is shown that LAV mappings always admit universal solutions, allow the rewriting of unions of conjunctive queries over the target into unions of conjunctive queries over the sources, and are closed both under target homomorphism and union. Recently, LAV mappings have also been shown to be closed under composition, and to admit polynomial time recoverability checking [3].

- LAV mappings allow a characterization of the sources in the terms of the element of the target schema, and, therefore, are crucial in all the scenarios where a precise understanding, and a formal documentation of the content of the sources are needed.

Contributions of the paper. In this paper we propose a formal definition of the view-synthesis and the view-existence problems, and present the first study on such problems, both in the context of the relational model, and in the context of semistructured data.

For relational data, we address the case where queries and views are both conjunctive queries, and the case where they are unions of conjunctive queries. In the former case, we show that both view-existence and exact view-existence are in $NP$. In the latter case, we show that both problems are in $PSPACE$.

In the context of semistructured data, we refer to a graph-based data model, as opposed to the popular XML-based model. The reason is that in many interesting scenarios, including the ones where XML data are used with refids, semistructured data form a graph rather than a tree. For graph-based semistructured data, we first study view synthesis and view existence in the cases where queries and views are regular path queries. We first present a techniques for view-existence based on automata on infinite trees [34], and provide an $PSPACE$ upper bound for the problem. We then illustrate an alternative technique based on the characterization of regular languages by means of left-right congruence classes. Such a characterization allows us to prove an $ExpSpace$ upper bound for the exact view-existence problem. Finally, by exploiting a language-theoretic characterization for containment of regular path queries with inverse (called two-way regular path queries) provided in [10], we extend the congruence class-based technique to the case where queries and views are two-way regular path queries, as well as conjunctive two-way regular path queries, and unions of such queries.

Organization of the paper. The paper is organized as follows. In Section 2, we recall some preliminary notions. In Section 3, we formally define the problem of view-synthesis from schema mappings, and the problem of view-existence. In Section 4, we study the problem in the case where queries and views are conjunctive queries, and unions thereof. In Section 5 and Section 6, we illustrate the techniques for the view synthesis problem in the case of RPQs over semistructured data. Section 7 extends the technique to two-way RPQs, and to (unions of) conjunctive two-way RPQs, respectively. Section 8 concludes the paper.

2. PRELIMINARIES

In this work we deal with two data models, the standard relational model [8], and the graph-based semistructured data model [9].

Given a (relational) alphabet $\Sigma$, a database $D$ over $\Sigma$, and a query $q$ over $\Sigma$, we denote with $q^D$ the set of tuples resulting from evaluating $q$ in $D$. A query $q$ over $\Sigma$ is 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 contained in $q_2$, denoted $q_1 \subseteq q_2$, if $q_1^D \subseteq q_2^D$ for every database $D$ over $\Sigma$. The queries $q_1$ and $q_2$ are equivalent, denoted $q_1 \equiv q_2$, if both $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$.

We assume familiarity with the relational model and with (unions of) conjunctive queries, (U)CQs, over a relational database. Below we recall the basic notions regarding the graph-based semistructured data model and regular path queries.

A semistructured database is a finite graph whose nodes represent objects and whose edges are labeled by elements from an alphabet of binary relational symbols $\{0, 1, 10\}$. 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 regular-path query (RPQ) over an alphabet $\Sigma$ of binary relation symbols is expressed as a regular expression or a noneterministic finite word automaton (NWA) over $\Sigma$. When evaluated on a (semistructured) 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 ($q$) defined by $q$. Containment between RPQs can be characterized in terms of containment between the corresponding regular languages: given two RPQs $q_1$ and $q_2$, we have that $q_1 \subseteq q_2$ iff $(q_1) \subseteq (q_2)$ [9].

We consider also two-way regular-path queries (2RPQs) [8, 10], which extend RPQs with the inverse operator. Formally, let $\Sigma^\pm = \Sigma \cup \{r^- | r \in \Sigma\}$ 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 $p^-$ is $r^-$, and if $p$ is $r^-$, then $p^-$ is $r$. 2RPQs are expressed by means of an NWA 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.
A semipath in \(\mathcal{D}\) 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 \(p_i \in \Sigma^p\), and, if \(p_i = r\) then \((y_{i-1}, y_i) \in r^p\), and if \(p_i = r^\prime\) then \((y_{i-1}, y_i) \in r^{p^\prime}\). We say that a semipath \((y_0, p_1, \ldots, p_n, y_n)\) conforms to \(q\) if \(p_1 \ldots p_n \in (q)\).

We will also consider conjunctions of 2RPQs and their unions, abbreviated (U)2RPQs, which are (unions of) conjunctive queries constituted only by binary atoms whose predicate is a 2RPQ. Specifically, a 2RPQ \(q\) of arity \(n\) is written in the form

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

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 \(q_i\) is a 2RPQ. When evaluated over a database \(\mathcal{D}\) over \(\Sigma\), the 2RPQ \(q\) computes the set of tuples \((a_1, \ldots, a_n)\) of objects such that there is a total mapping \(\varphi\) from \(\{z_1, \ldots, z_k\}\) to the objects in \(\mathcal{D}\) with \(\varphi(x_i) = a_i\), for \(i \in \{1, \ldots, n\}\), and \((\varphi(y_{2j-1}), \varphi(y_{2j})) \in q^p_j\), for \(j \in \{1, \ldots, m\}\).

Containment between 2RPQs and (U)2RPQs can also be characterized in terms of containment between regular languages. We elaborate on this in Section 7. We conclude by observing that (U)CQs, 2RPQs, 2RPQs, and (U)2RPQs are monotone.

### 3. The View-Synthesis Problem

The view-synthesis and the view-existence problems refer to a scenario where one source schema, one target schema, and a set of schema mappings between the two, where the goal is to synthesize one view for each source.

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 model the queries used in both the mappings and the views, we use three query languages, namely, the source language \(Q_s\) over \(\Sigma_s \cup \Sigma_t\), the target language \(Q_t\) over \(\Sigma_s\), and the view language \(Q_v\) over \(\Sigma_s\). Notice that queries expressed in the language \(Q_v\) may use symbols in the target alphabet.

A schema mapping, or simply a mapping, between the source and the target is a statement of the form \(q_s \leadsto q_t\), with \(q_s \in Q_s\) and \(q_t \in Q_t\). Intuitively, a mapping of this type specifies that all answers computed by executing the source query \(q_s\) are answers to the target query \(q_t\). This means that \(q_s \in Q_s\) is actually a rewriting of \(q_t\). This explains why we allow \(Q_s\) to use symbols in the target alphabet: in general, the rewriting of a target query may use symbols both in the source alphabet, and in the target alphabet [20].

The problem we consider aims at defining one view for each source, in such a way that all input schema mappings are captured. The views \(V\) over \(\Sigma_t\) to be synthesized are modeled as a (not necessarily total) function \(V: \Sigma_s \rightarrow Q_v\) that associates to each source symbol \(a \in \Sigma_s\) a query \(V(a) \in Q_v\) over the target alphabet \(\Sigma_t\). As we said before, our notion of “views capturing a set of mappings” relies on view-based query rewriting, whose definition we now recall.

In the following, given a source database \(D_s\), and a target database \(D_t\), we say that \(V\) is coherent with \(D_s\) and \(D_t\) if for each element \(a\) in the source alphabet, the extension of this element in \(D_t\) is contained in the result of evaluating \(V(a)\) over the database \(D_t\) (where \(V(a)\) is the query that \(V\) associates to \(a\)). Formally, \(V\) is coherent with \(D_s\) and \(D_t\) if for each \(a \in \Sigma_s\), \(a^{D_s} \subseteq V(a)^{D_t}\).

Following [11], we say that a query \(q_s \in Q_s\) is a sound rewriting, or simply a rewriting, of a query \(q_t \in Q_t\), wrt views \(V\), if for every source database \(D_s\) and for every target database \(D_t\) such that \(V\) is coherent with \(D_s\) and \(D_t\), we have that \(q_s^{D_s} \subseteq q_t^{D_t}\). If \(q_s^{D_s} = q_t^{D_t}\), the rewriting is said to be exact. Further, we say that \(q_s\) is empty wrt \(V\) if for every source database \(D_s\) and for every target database \(D_t\) such that \(V\) is coherent with \(D_s\) and \(D_t\), we have that \(q_s^{D_s} = \emptyset\). Notice that, if all views in \(V\) are empty (i.e., for each \(a \in \Sigma_s\), \(V(a)\) is the empty query), then trivially \(q_s\) is empty wrt \(V\). However, \(q_s\) may be empty wrt \(V\) even in the case in which all (or some) views are non-empty.

We observe that, when \(Q_s\) and \(Q_t\) are monotonic query languages, the above definitions of sound and exact rewritings are equivalent to the ones where the notion of “\(V\) being coherent with \(D_s\) and \(D_t\)” is replaced by the condition: for each \(a \in \Sigma_s\), \(a^{D_s} = V(a)^{D_t}\) (see [11]). It is easy to see that, under this monotonic assumption, \(q_s\) is a rewriting of \(q_t\) wrt views \(V\) if \(q_s[V] \subseteq q_t\), where here and in the following we use \(q[V]\) to denote the query over \(\Sigma_t\) obtained from \(q_s\) by substituting each source symbol \(a \in \Sigma_s\) with the query \(V(a)\). Further, \(q_s\) is empty wrt \(V\) if \(q_s[V] \equiv \emptyset\), and it is an exact rewriting wrt \(V\) if \(q_s[V] \equiv q_t\). Note that in all the settings considered in the next sections, the languages \(Q_s\) and \(Q_t\) are monotonic.

We are now ready to come back to the notion of “views capturing a set of mappings”. We say that views \(V\) capture mappings \(M\) if for each \(q_s \leadsto q_t \in M\), the query \(q_s\) is a rewriting of \(q_t\) wrt \(V\) and is non-empty wrt \(V\). Anologically, we say that \(V\) exactly capture \(M\) if for each mapping \(q_s \leadsto q_t \in M\), the query \(q_s\) is an exact rewriting of \(q_t\) wrt \(V\) and is non-empty wrt \(V\).

We are now ready to introduce the (exact) view-synthesis and the (exact) view-existence problems formally.

**Definition 1.** The (exact) view-synthesis problem is defined as follows: given a set \(M\) of mappings, find views \(V\) (exactly) capturing \(M\).

The (exact) view-existence problem is defined as follows: given a set \(M\) of mappings, decide whether there exist views \(V\) (exactly) capturing \(M\).

Finally, we also consider maximal views capturing mappings \(M\), which are views \(V\) such that there is no view \(V'\) capturing \(M\) such that \((i) V(a) \subseteq V'(a) \) for every \(a \in \Sigma_s\), and \((ii) V(a) \neq V'(a)\) for some \(a \in \Sigma_s\).

### 4. View Synthesis for (U)CQs

We start our investigations by tackling the case of view-synthesis and view-existence for conjunctive queries (CQs) and their unions (UCQs).

We start with the case where \(Q_s\), \(Q_t\), and \(Q_v\) are CQs, and establish the following upper bounds.

**Theorem 2.** In the case where \(Q_s\), \(Q_t\), and \(Q_v\) are CQs, the view-existence and the exact view-existence problems are in NP.

**Proof.** Consider a mapping \(q_s \leadsto q_t\), where \(q_t\) contains \(\ell\) atoms, and views \(V\) such that \(q_s[V] \subseteq q_t\). Then, there exists a containment mapping from \(q_s\) to \(q_t[V]\), and at most \(\ell\) atoms of \(q_s[V]\) will be in the image of this containment mapping. Hence, for each symbol \(a \in \Sigma_s\) occurring in \(q_s\) only at most \(\ell\) atoms in query \(V(a)\) are needed to satisfy
the containment mapping. In general, for a set \( M \) of mappings, in order to satisfy all containment mappings from \( q_i \) to \( q_j[V] \), for each \( q_i \sim q_j \in M \), we need in the query \( V(a) \) at most \( \ell_M = \sum_{a \sim q_j \in M} \ell_{q_j} \) atoms, where \( \ell_{q_j} \) is the number of atoms in \( q_j \). Hence, in order to synthesize the views \( V \), it suffices to guess, for each symbol \( a \in \Sigma_t \) appearing in one of the mappings in \( M \), a CQ \( V(a) \) over \( \Sigma_t \) of size at most \( \ell_M \), and check that \( q_j[V] \subseteq q_i \) (and \( q_i \subseteq q_j[V] \) for the exact variant), for each \( q_i \sim q_j \in M \). This gives us immediately an NP upper bound for the (exact) view-existence problem.

In the case where \( q_s \) and \( q_t \) are UCQs, we can generalize the above argument by considering containment between UCQs instead of containment between CQs.

**Theorem 3.** In the case where \( q_s \) and \( q_t \) are UCQs and \( q_s \) is CQs, the view-existence and the exact view-existence problems are in NP.

**Proof.** Consider a mapping \( q_s \sim q_t \) and views \( V \) such that \( q_t[V] \subseteq q_s \). We have that \( q_t[V] \subseteq q_s \) if for each CQ \( q_s \) in the UCQ \( q_t[V] \) there is a CQ \( q_s \) in the UCQ \( q_t \) such that \( q_s \subseteq q_t \). For a set \( M \) of mappings, in order to satisfy all containment mappings from \( q_s \) to \( q_t \), we need in the query \( V(a) \) at most \( \ell_M = \sum_{a \sim q_s \in M} \ell_{q_s} \) atoms, where \( \ell_{q_s} \) is the number of atoms in each of the UCQs in \( q_s \). Hence, in order to synthesize the views \( V \), it suffices to guess, for each symbol \( a \in \Sigma_t \) appearing in one of the mappings in \( M \), a CQ \( V(a) \) over \( \Sigma_t \) of size at most \( \ell_M \), and check that \( q_s[V] \subseteq q_t \) (and \( q_t \subseteq q_s[V] \) for the exact variant), for each \( q_s \sim q_t \in M \). The last case we consider is the one where, in addition to \( q_s \) and \( q_t \), also \( q_s \) is UCQs. As for view-existence, we observe that the problem admits a solutions for UCQ views iff it admits a solution for CQ views.

**Lemma 4.** An instance of the view-existence problem admits a solution in the case where \( q_s \), \( q_t \), and \( q_s \) are UCQs and \( q_t \) is CQs if and only if it admits a solution in the case where \( q_s \), \( q_t \), and \( q_s \) are UCQs and \( q_t \) is CQs.

**Proof.** Indeed, let \( V \) be a set of UCQ views such that \( q_s[V] \subseteq q_t \) for each mapping \( q_s \sim q_t \in M \). For such a mapping, \( q_s[V] \) is a nonempty positive query. Consider the views \( V' \) obtained from \( V \) by choosing, for each symbol \( a \in \Sigma_t \), as \( V'(a) \) one of the UCQs in \( V(a) \). Then, each nonempty UCQ in \( q_s[V'] \) is contained in \( q_t[V] \), and hence in \( q_t \). It follows that also views \( V' \) provide a solution to the view-synthesis problem.

Hence by the above lemma, we trivially get:

**Theorem 5.** In the case where \( q_s \), \( q_t \), and \( q_s \) are UCQs, the view-existence problem is in NP.

As for exact view-existence, allowing for views that are UCQs changes indeed the problem.

**Theorem 6.** In the case where \( q_s \), \( q_t \), and \( q_s \) are UCQs, the exact view-existence problem is in \( \Pi^2_2 \).

**Proof.** Let \( V \) be a set of UCQ views such that \( q_i[V] = q_i \) for each mapping \( q_s \sim q_i \in M \). Let us first consider one such mapping \( q_s \sim q_i \) and let \( m_{q_i} \) be the number of CQs in \( q_i \), and \( r_{q_i} \) the maximum number of atoms in each of the CQs in \( q_i \). Since \( q_j[V] \subseteq q_i \), there is a containment mapping from each of the \( m_{q_i} \) CQs in \( q_i \) to some CQ in the UCQ \( q_j[V] \), where \( q_j[V] \) is obtained from \( q_j[V] \) by distributing, for each atom \( a_i \) in \( q_i \), the unions in the UCQ \( q_j[V] \) over the conjunctions of each CQ of \( q_i \). Hence, for each symbol \( a \in \Sigma_t \) occurring in \( q_i \), only at most \( m_{q_i} \) CQs of at most \( r_{q_i} \) atoms in query \( V(a) \) are needed to satisfy the containment mappings. It follows that, to check the existence of UCQs views \( V \) and of such a containment mapping, it suffices to guess for each \( a \) a UCQ over \( \Sigma_t \) consisting of at most \( m_{q_i} \), CQs, each with at most \( r_{q_i} \) atoms. When considering all mappings \( q_s \sim q_i \in M \), similar to the case above, we have to use instead of \( m_{q_i} \) and \( r_{q_i} \), the sum of these parameters over all mappings in \( M \). To check whether these views satisfy \( q_i[V] \subseteq q_j[V] \), it suffices to check for the existence of a containment mapping from \( q_i[V] \) to each of the CQs in \( q_i \), which can be done in NP in the size of \( q_i \). To check whether these views satisfy \( q_i[V] \subseteq q_j[V] \), we have to check whether for each CQ \( q_i[V] \) obtained by selecting one of the CQs \( q_i'[V] \) and then substituting each atom \( a_i \) in \( q_i'[V] \) with one of the CQs in \( q_j[V] \), there is a containment mapping from some CQ in \( q_j \) to \( q_i \). We can do so by a coNP computation that makes use of an NP oracle to check for existence of a containment mapping. This gives us the \( \Pi^2_2 \) upper bound.

5. TREE-BASED SOLUTION FOR RPQs

We address now the view-synthesis problem when \( q_s \), \( q_t \), and \( q_r \) are RPQs, and present a techniques based on tree automata on infinite trees [34]. Specifically, we consider automata running over complete labeled \( \Sigma \)-trees (i.e., trees in which the set of nodes is the set of all strings in \( \Sigma^* \)).

First, we observe that every language \( L \) over an alphabet \( \Sigma \) can be represented as a function \( L : \Sigma^* \rightarrow \{0, 1\} \), which, in turn, can be considered as a \( \{0, 1\} \)-labeling of the complete \( \Sigma \)-tree. Consider a source alphabet \( \Sigma_s = \{a_1, \ldots, a_n\} \) and the target alphabet \( \Sigma_t \). We can represent the views defined on \( \Sigma_s \) by the \( \{0, 1\} \)-labeled \( \Sigma_t \)-tree \( T_v \) (i.e., a \( \Sigma_t \)-tree in which each node is labeled with an \( n \)-tuple of elements of \( \{0, 1\} \) in which the nodes representing the words in \( V(a_i) \) are exactly those whose label has 1 in the \( i \)-th component. We call such trees view trees. Note that views defined by view trees assign an arbitrary languages on \( \Sigma_t \) to each source relation; these languages need not, a priori, be regular. We return to this point later.

Given a mapping \( m = q_s \sim q_t \), we construct now a tree automaton \( A_m \) accepting all view trees representing views \( V \) capturing \( m \). Concerning the check that \( q_s \) is not empty wrt \( V \), we observe that, if there is a word \( w = c_1 \cdots c_k \) in \( (A_s) \) such that for all the letters \( a_1, \ldots, a_k \) appearing in \( w \), there are nodes in the tree where the \( i \)-th component of the label is 1. The tree automaton has to guess a set of letters in \( \Sigma_s \) that cover a word accepted by \( A_s \) (we can ignore the letters in \( \Sigma_s \)), and then check the above condition.

We assume that \( q_r \) is represented as an NWA \( A_s = (S_s, \Sigma_s \cup \Sigma_t, p_0, \delta_s, F_s) \) and \( q_t \) is represented as an NWA \( A_t = (S_t, \Sigma_t, p_1, \delta_t, F_t) \) an annotation for a view tree \( T_v \)

\( ^3 \)Transition functions of NWAs can be extended to sets of states and words in a standard way.
is a ternary relation $\alpha \subseteq S^2 \times \Sigma$. An annotation $\alpha$ is correct for $T_V$ if the following holds: $(p, p', a_i) \in \alpha$ iff there is a word $w \in \Sigma^*$ such that $T_V(w)[i] = 1$ and $p' \in h(p, w)$. Intuitively, $\alpha$ describes the transitions that $T_V$ can induce on $A_t$.

We say that an annotation $\alpha$ captures $q_i \sim q_j$ if for every word $w = c_1 \cdots c_k$ in $(A_t)$ there is a sequence $p_0, \ldots, p_{k+1}$ of states of $A_t$ such that $p_0 = p_0^0$, $p_{k+1} \in F_t$, and for $i \in \{0, \ldots, k\}$, if $c_i \in \Sigma_t$ then $p_{i+1} = \delta_t(p_i, c_i)$, and if $c_i = a_j \in \Sigma_s$, then $(p_i, p_{i+1}, a_j) \in \alpha$.

The significance of an annotation capturing a mapping comes from the following lemma.

**Lemma 7.** $V$ captures $q_i \sim q_j$ iff there is an annotation $\alpha$ that is correct for $T_V$ and captures $q_i \sim q_j$.

We now characterize when $\alpha$ captures $q_i \sim q_j$.

**Lemma 8.** $\alpha$ does not capture $q_i \sim q_j$ if there is a word $w = c_1 \cdots c_k$ in $(A_t)$ and a sequence $p_0, \ldots, p_{k+1}$ of sets of states of $A_t$, such that $P_0 = \{p_0^0\}$, $p_{k+1} \cap F_t = \emptyset$, and for $i \in \{0, \ldots, k\}$, if $c_i \in \Sigma_t$ then $p_{i+1} = \delta_t(p_i, w_i)$, and if $c_i = a_j \in \Sigma_s$, then $(p_i, p_{i+1}, a_j) \notin \alpha$.

Thus, checking that $\alpha$ does not capture $q_i \sim q_j$ can be done by guessing the word $w$ and the sequence $P_0, \ldots, P_{k+1}$ of sets of states of $A_t$ and checking the conditions. This can be done in space logarithmic in $A_s$ and polynomial in $A_t$. It follows that we can check that an annotation $\alpha$ captures $q_i \sim q_j$ in time that is polynomial in $A_s$ and exponential in $A_t$.

We now describe a tree automaton $A_m$ that accepts precisely the view trees $T_V$, where $V$ captures $m = q_i \sim q_j$. By Lemma 7 all $A_m$ has to do is guess an annotation $\alpha$ that captures $m$ and check that it is correct for $T_V$.

**Lemma 9.** Given $A_s$ and $A_t$, we can construct a tree automaton $A_m$ that accepts all view trees that capture $m = q_i \sim q_j$. The size of $A_m$ is exponential in the sizes of $A_s$ and $A_t$.

**Proof.** We construct $A_m = (S_m, \Sigma_m, m_0, \delta_m, F_m)$ as a Büchi automaton on infinite trees [34]. Recall that $\Sigma_m = \{0, 1\}^\kappa$. The state set $S_m$ is $(\Sigma^i \times \Sigma^j)^2 \times \Sigma^i$. That is, each state is a triple consisting of a pair of annotations and a binary relation on $\Sigma_t$. The initial state set $S_m$ consists of all triples $\beta = (\alpha, \alpha, R_m)$, where $\alpha$ captures $m$ and $R_m = \{(p, p, a) \mid p \in S_t\}$. Intuitively, an initial state is a guess of an annotation. The automaton $A_m$ now has to check its correctness; the second and third component of the state are used for "bookkeeping."

Let $\Sigma_e = \{b_1, \ldots, b_k\}$. Then $(\beta_1, \ldots, \beta_k) = \delta_m(\beta, c)$, where $c = (c_1, \ldots, c_n)$, $\beta = (\alpha^1, \alpha^2, \alpha^3)$, and $\beta_j = (\alpha^1_j, \alpha^2_j, \alpha^3_j)$ for $j \in \{1, \ldots, k\}$, if the following hold:

1. If $(p_1, p_2) \in \alpha^3$ and $c_i = 1$, then $(p_1, p_2, a_i) \in \alpha^1$.
2. $\alpha^1 = \alpha^1$; that is, the first component does not change.
3. $\alpha^3 = \{(p_1, p_2) \mid (p_1, p_2) \in \alpha^3 \text{ and } p_2' \in \delta(p_2, b_j)\}$; that is, the third component remembers paths between states of $A_t$.
4. If $(p_1, p_2, a_i) \in \alpha^2$, then either $p_1 = p_2$ and $c_i = 1$, or, for some $j \in \{1, \ldots, m\}$ and $p_i' \in \delta(p_1, b_j)$, we have that $(p_i', p_2, a_i) \in \alpha^2$.

Thus, the second component of the state helps to check that all the paths in $A_t$ predicted by the guessed annotation are fulfilled in the tree, while the third component helps to check that all the paths that do occur in the tree are predicted by the guessed annotation. This means that the second component must ultimately become empty. Note that once it becomes empty, it can stay empty. Thus the set $F_m$ of accepting states consists of all triples of the form $(\alpha, \emptyset, R)$.

Note that the number of states of $A_m$ is exponential in the number of states of $A_t$ and exponential in the alphabet of $A_s$. The alphabet of $A_m$ is exponential in the size of the alphabet of $A_s$.

**Theorem 10.** In the case where $Q_s$, $Q_t$, and $Q_o$ are RPQs, the view existence problem is EXP-Time.

**Proof.** We showed how to construct, with an exponential blowup a Büchi tree automaton that accept all view trees that capture $m = q_i \sim q_j$. Note that computing the set of initial states, requires applying Lemma 8 and takes exponential time. To handle a set $M$ of mappings, we simply take the product of these automata (see product construction in [34]). To check that the views are nonempty, we take the product with a very simple automaton that checks that one of the labels of the tree is not identically 0. We thus obtain a Büchi tree automaton $A_M$ that accepts all view trees that represents nonempty views that capture $M$.

We now check the nonemptiness of $A_M$ in quadratic time [34]. If $(A_M) = \emptyset$, then the answer to the view-existence problem is negative. If $(A_M) \neq \emptyset$, then the nonemptiness algorithm returns a witness in the form of a transducer $A = (S, \Sigma_t, \Sigma_m, p_0, \delta, \gamma)$, where $S$ is a set of states (which is a subset of the state set of the tree automaton), $\Sigma_t$ is the input alphabet, $\Sigma_m = \{0, 1\}^\kappa$ is the output alphabet, $p_0$ is a start state, $\delta : S \times \Sigma_t \rightarrow S$ is a deterministic transition function, and $\gamma : S \rightarrow \Sigma_m$ is the output function. From this transducer we can obtain an RPQ for each letter $a_i \in \Sigma_s$, represented by the DWA $A = (S, \Sigma_t, p_0, \delta, F_t)$, where $F_t = \{p \mid p \in S \text{ and } \gamma(p)[i] = 1\}$.

Note that the proof of Theorem 10 implies that, wrt the view-existence problem, considering views that are RPQs (as opposed to general, possibly non-regular, path languages) is not a restriction, since the existence of general views implies the existence of regular ones. In fact, a similar result holds also for the exact view-existence problem, as follows from the results in the next section. This is also in line with a similar observation holding for the existence of rewritings of RPQs wrt RPQ views [9].

A final comment regarding maximal views. A view tree $T_V$ is maximal with respect to a set $M$ of mappings if $V$ captures $M$, but flipping in one of the labels a single 0 to 1 would destroy that property. Our tree-automata techniques can be extended to produce maximal views, by quantifying over all such flipings. This, however, would imply an additional exponential increase in the complexity of the algorithm.

### 6. CONGRUENCE CLASS BASED SOLUTION FOR RPQs

We present now an alternative technique for view-synthesis for RPQs that will allow us also to extend our results to more expressive forms of queries. Our solution is
based on the characterization of regular languages by means of congruence classes \[27\].

We start by showing that we can reduce the (exact) view-synthesis problem with a set of mappings \( M \) to an (exact) view-synthesis problem with a single mapping \( m \).

**Theorem 11.** Given a set \( M \) of RPQ mappings, there is a single RPQ mapping \( m \) such that, for every set \( V \) of RPQ views, \( V \) (exactly) captures \( M \) if and only if \( V \) (exactly) captures \( m \).

**Proof.** Let \( M = \{q_{0,1} \sim q_{0,1}, \ldots, q_{0,n} \sim q_{0,n}\} \) be the set of mappings from \( \Sigma_1 \cup \Sigma_2 \) to \( \Sigma_3 \), and let \( \Sigma_1' = \Sigma_1 \cup \{\#\} \), where \( \# \) is a fresh target symbol not occurring in \( \Sigma_1 \) and \( \Sigma_3 \). We define a mapping \( m = q_{m,1} \sim q_{m,1} \) from \( \Sigma_1' \cup \Sigma_3' \) to \( \Sigma_3' \), by setting \( q_{m,1} = q_{0,1} \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \cdot \# \). Intuitively, the fresh symbol \( \# \) acts as a separator between the different parts of \( q_{m,1} \) and \( q_{m,1} \).

It is easy to verify that \( q_{m,i}[V] \subseteq q_{m,i} \), for \( i \in \{1, \ldots, h\} \).

Hence, w.l.o.g., in the following we will consider only the case where there is a single mapping \( q \sim q \).

Let \( A_1 = (\Sigma_1, \Sigma_2, \delta_1, \Sigma_3) \) be an NWA for \( q \). Then \( A_1 \) defines a set of (left-right) congruence classes partitioning \( \Sigma_1' \).

Note that the standard treatment of congruence classes is done with deterministic automata \[27\], but we do it here with NWAs to avoid an exponential blow-up. For a word \( w \in \Sigma_1' \), we denote with \( [w]_{A_1} \) the congruence class to which \( w \) belongs. Each congruence class is characterized by a binary relation \( R \subseteq \Sigma_1' \times \Sigma_1' \), where \( \Sigma_1' \) is the set of congruence classes.

**Lemma 12.** Let \( q \) be an RPQ over \( \Sigma_1 \cup \Sigma_2 \), and \( q \) an RPQ over \( \Sigma_1 \). If there exist RPQ views \( V \) capturing \( q \), then there exist views \( V' \) capturing \( q \) such that each view in \( V' \) is either a single word over \( \Sigma_1 \) or empty.

**Proof.** Since \( q_a[V] \not\in \emptyset \) and \( q_b[V] \subseteq q_b \), there exists a word \( a_1 \cdot a_2 \in \{q_a\} \) and a word \( b_1 \cdot b_2 \in \{q_b[V]\} \) and hence in \( \{A_1\} \), where \( \emptyset = (V(a_1)) \). To define new views \( V' \), we consider for each \( a \in \Sigma_1 \) appearing in \( a_1 \cdot a_2 \) a word \( w^c \in V(a) \) and set \( V'(a) = w^c \). Instead, for each \( a \in \Sigma_1 \), not appearing in \( a_1 \cdot a_2 \), we set \( V'(a) = \emptyset \). Now, \( q_b[V'] \) is nonempty by construction, and since \( V'(a) \subseteq V(a) \) for every \( a \in \Sigma_1 \), we have that \( q_b[V'] \subseteq q_b[V] \).

The next lemma shows that one can close views under congruence.

**Lemma 13.** Let \( q \) be an RPQ over \( \Sigma_1 \cup \Sigma_2 \), and \( q \) an RPQ over \( \Sigma_1 \) expressed through an NWA \( A_1 \), and \( V \) singleton views capturing \( q \). Then \( V' \) defined such that

\[
(V'(a)) = \begin{cases} 
[w^a]_{A_1}, & \text{if } V(a) = w^a \\
\emptyset, & \text{if } V(a) = \emptyset.
\end{cases}
\]

captures \( q \sim q' \).

**Proof.** Consider a word \( a_1 \cdot a_2 \in \{q_a\} \). If there is one of the \( a_i \) such that \( V(a_i) = \emptyset \), then \( V(a_i) = \emptyset \). Otherwise, we have that \( (V(a_i)) = \{w^a_i\} \), for \( i \in \{1, \ldots, h\} \), and since \( w^a_1 \cdot w^a_2 \in \{q_b[V]\} \), there is a sequence \( p_0, p_1, \ldots, p_n \) of states of \( A_1 \) such that \( p_0 = p_0', p_n = F_1 \), and \( p_i \in \delta_1(p_{i-1}, w^a_i) \), for \( i \in \{1, \ldots, n\} \). Consider now, for each \( i \in \{1, \ldots, h\} \), a word \( w_i' \in (V'(a_i)) = \{w^a_i\}_{A_1} \).

Making use of the characterization of \( [w^a_i]_{A_1} \) in terms of a binary relation over \( \Sigma_1 \), we have for each word \( w^a_i \) in \( [w^a_i]_{A_1} \), and in particular for \( w_i' \), that \( p_i \in \delta_1(p_{i-1}, w_i') \). Hence, \( p_i \in \delta_1(p_{i-1}, w_i') \) for \( i \in \{1, \ldots, h\} \).

From these two lemmas we get that, when searching for views capturing the mappings, we can restrict the attention to views that are congruence classes for \( A_1 \).

**Lemma 14.** Let \( q \) be an RPQ over \( \Sigma_1 \cup \Sigma_2 \), and \( q \) an RPQ over \( \Sigma_1 \) expressed through an NWA \( A_1 \). If there exist RPQ views \( V \) over \( \Sigma_1 \) capturing \( q \sim q' \), then there exist views \( V' \) capturing \( q \sim q' \) such that each view in \( V' \) is a congruence class for \( A_1 \).

**Proof.** If there exist RPQ views \( V \) over \( \Sigma_1 \) capturing \( q \sim q' \), then by Lemma \[12\] w.l.o.g., we can assume that \( V \) are singleton views. Then, the claim follows from Lemma \[13\].

From the above lemma, we can immediately derive an \( \text{EXPTime} \) procedure for view existence, which gives us an alternative proof of Theorem \[10\]. We first observe that, for an NWA \( A \) with \( m \) states, each view defined by a congruence class \( C_R \) for \( A \) can be represented by the NWA \( A_R \), which has at most \( 2^m \) states. For a set \( S \) of views that are congruence classes, we can test whether \( q[S] \not\in \emptyset \) and \( q[S] \subseteq q \), by

- substituting each \( a \)-transition in the NWA \( A \) for \( q_a \) with the NWA \( A_{q_a} \), where \( C_{A_{q_a}} = (V(a)) \), thus obtaining an NWA \( A_{q_a} \)
• complementing $A_t$, obtaining an NWA $\overline{A}_t$; and

• checking the nonemptiness of $A_{s,V}$ and the emptiness of $A_{s,V} \times \overline{A}_t$.

Such a test can be done in time polynomial in the size of $A_s$ and exponential in the size of $A_t$.

Considering that the number of distinct congruence classes is at most $2^m q_s$, the number of possible assignments of congruence classes to $n$ view symbols occurring in $q_s$ is at most $2^m q_s^2$. For each such assignment defining views $V$, we need to test whether $q_s[V] \neq \emptyset$ and $q_s[V] \subseteq q_t$. Hence the overall check for the view-existence problem requires time exponential in the size of $A_t$, polynomial in the size of $A_s$, and exponential in the number of source symbols occurring in $q_s$.

The technique presented here based on congruence classes can be adapted to address also the exact view existence problem. The difference wrt to (non-exact) view existence is that in this case we need to consider also views that are unions of congruence classes. Indeed, congruence classes (and hence rewritings) are not closed under union, as shown by the following example.

Let $q_s = a_1 \cdot a_2$ and $q_t = 00 + 01 + 10$. Then the following two sets of incomparable views maximally capture $q_s \leadsto q_t$:

\[
V_1(a_1) = 0, \quad V_1(a_2) = 1, \quad V_2(a_1) = 0 + 1, \quad V_2(a_2) = 0.
\]

Notice that views $V$, where $V(a_1) = V_1(a_1) + V_2(a_1)$, for $i \in \{1, 2\}$, does not capture $q_s \leadsto q_t$, since $q_s[V]$ includes 1.

On the other hand, we can show that considering views that are unions of congruence classes is sufficient to obtain maximal unfoldings. We first generalize Lemma 13 to non-singleton views.

**Lemma 15.** Let $q_s$ be an RPQ over $\Sigma_s \cup \Sigma_t$, $q_t$ an RPQ over $\Sigma_t$ expressed through an NWA $A_t$, and $V$ a set of views capturing $q_s \leadsto q_t$. Then $V'$ defined such that

\[
(V'(a)) = \begin{cases} 
\bigcup_{w \in (V(a))}[\ell_{t,A_t}], & \text{if } V(a) \neq \emptyset \\
\emptyset, & \text{if } V(a) = \emptyset.
\end{cases}
\]

captures $q_s \leadsto q_t$.

**Proof.** Consider a word $a_1 \cdots a_h$ in $(q_s)$. If there is one of the $a_i$ such that $V(a) = \emptyset$, then $(V(a_1))\cdots(V(a_h)) = \emptyset \subseteq (q_t)$. Otherwise, we have that, for $i \in \{1, \ldots, h\}$, for some $w^{a_i} \in (V(a))$, the word $w^{a_i} \cdots w^{a_h} \in (q_s[V]) \subseteq (A_t)$. We show that, for each $i \in \{1, \ldots, h\}$, we also have that $w^{a_1} \cdots w^{a_{i-1}} \cdot w^i \cdot w^{a_{i+1}} \cdots w^{a_h} \in (A_t)$, for each $w^i \in \bigcup_{w \in (V(a))}[\ell_{t,A_t}]$. First, by definition of rewriting, if $w^{a_1} \cdots w^{a_h} \in (q_s[V]) \subseteq (A_t)$, then, for each word $w \in (V(a))$, we also have that $w^{a_1} \cdots w^{a_{i-1}} \cdot w^i \cdot w^{a_{i+1}} \cdots w^{a_h} \in (q_s[V]) \subseteq (A_t)$. Then there is a sequence $p_0, p_1, \ldots, p_h$ of states of $A_t$ such that $p_0 = p^0, p_h \in F_t, p_i \in \delta(p_{i-1}, w^{a_i})$, for $j \in \{1, \ldots, i-1, i+1, \ldots, h\}$, and $p_i \in \delta(p_{i-1}, w^i)$. Then, by the definition of congruence class, for each word $w^i \in [\ell_{t,A_t}]$, we have that $p_i \in \delta(p_{i-1}, w^i)$, and hence $w^{a_1} \cdots w^{a_{i-1}} \cdot w^i \cdot w^{a_{i+1}} \cdots w^{a_h} \in (A_t)$.

The above lemma implies that, when searching for views that maximally capture the mappings, we can restrict the attention to views that are unions of congruence classes.

**Lemma 16.** Given a mapping $m = q_s \leadsto q_t$, where $q_t$ is defined by an NWA $A_t$, every set of views $V$ that maximally captures $m$ is such that each view in $V$ is a union of congruence classes for $A_t$.

**Proof.** Consider a set of views $V$ that maximally captures $m$, and assume that for some $a \in \Sigma_s$, $V(a)$ is not a union of congruence classes for $A_t$. Then there is some word $w \in (V(a))$ and some word $w^i \notin [w]_{A_t}$, such that $w^i \notin (V(a))$. By Lemma 15 the set of views $V'$ with $(V'(a)) = (V(a)) \cup (w^i)$ also captures $m$, thus contradicting the maximality of $V$.

We get the following upper bound for the exact view existence problem.

**Theorem 17.** In the case where $Q_s$, $Q_t$, and $Q_v$ are RPQs, the exact view existence problem is in $\text{ExpSpace}$.

**Proof.** By Lemma 16 we can nondeterministically choose views $V$ that are unions of congruence classes and then test whether $q_s \models q_t[V]$ (we assume that $q_s \not\models \emptyset$, otherwise the problem trivializes). To do so, we build an NWA $A_{s,V}$ accepting $(q_s[V])$ as follows. We start by observing that for each union $U$ of congruence classes, we can build the automaton $A_U = (R, p_U, a_U, \delta_U, U)$ accepting the words in $U$, which incidentally, is deterministic. Hence, by substituting each a-transition in the NWA $A_s$ for $q_s$, with the NWA $A_{s,V}$, where $V(a) = U_a$, we obtain an NWA $A_{s,V}$. Note that, even when $A_s$ is deterministic $A_{s,V}$ may be non-deterministic.

To test $q_s[V] \subseteq q_t$, we complement $A_t$, obtaining the NWA $\overline{A}_t$, and check the NWA $A_{s,V} \times \overline{A}_t$ for emptiness. The size of $A_{s,V} \times \overline{A}_t$ is polynomial in the size of $A_s$ and exponential in the size of $A_t$. Checking for emptiness can be done in exponential time, and considering the initial nondeterministic guess, we get a NEXPTime upper bound.

To test $q_s[V] \subseteq q_t$, we complement $A_{s,V}$, obtaining the NWA $\overline{A}_{s,V}$, and check $A_t \cap \overline{A}_{s,V}$ for emptiness. Since $\overline{A}_{s,V}$ is nondeterministic, complementation is exponential. However, we observe that such a complementation can be done on the fly in $\text{ExpSpace}$, while checking for emptiness and intersecting with $A_t$. As a consequence, considering the initial nondeterministic guess, exact view existence can be decided in $\text{NExpSpace}$, which is equivalent to $\text{ExpSpace}$.

7. Extensions

In this section we sketch the extension of the results of the previous section to more expressive classes of queries: 2RPQs, CRPQs, UCRPQs, and UC2RPQs.

7.1 2RPQs

Consider now the view-synthesis problem for the case where $Q_s$, $Q_t$, and $Q_v$ are 2RPQs, expressed by means of NWAs over the alphabets $\Sigma^\pm$ and $\Delta^\pm$.

A key concept for 2RPQs is that of folding. Let $u, v \in \Sigma^\pm$. We say that $v$ folds onto $u$, denoted $v \sim u$, if $v$ can be "folded" on $u$, e.g., $abb'bc \sim abc$. 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 $v_{j+1} = u_{j+1}$, or $i_{j+1} = i_j - 1$ and $v_{j+1} = u'_{j+1}$.
Let $L$ be a language over $\Sigma^\pm$. We define $\text{fold}(L) = \{ u : v \sim u, v \in L \}$.

A language-theoretic characterization for containment of 2RPQs was provided in [10]:

**Lemma 18.** Let $q_1$ and $q_2$ be 2RPQs. Then $q_1 \subseteq q_2$ if and only if $\text{fold}(q_1) \subseteq \text{fold}(q_2)$.

Furthermore, it is shown in [10] that if $A$ is an $n$-state NWA over $\Sigma^\pm$, then there is a 2NWA for $\text{fold}((A))$ with $n \cdot (|\Sigma^\pm| + 1)$ states. (We use 2NWA to refer to two-way automata on words.)

In the view-existence problem, we are given queries $q_1$ and $q_2$, expressed as NWAs $A_1$ and $A_2$, and we are asked whether there exist nonempty 2RPQ views $V$ such that $q_1[V] \nsubseteq q_2$ and such that $q_1[V] \neq \emptyset$. We can use Lemma 18 for the tree-automata solution. A labeled tree $V : (\Sigma^\pm) \rightarrow \{0, 1\}^n$ represents candidate views. To check that $q_1[V] \nsubseteq q_2$, we check that $(A_1[V]) \nsubseteq \text{fold}((A_2))$. We now proceed as in Section 5 using the 2NWA for $\text{fold}((A_1))$ instead of $A_1$. This requires first converting the 2NWA for $A_1$ to an NWA with an exponential blow-up [29], increasing the overall complexity to $2\text{ExpTime}$.

We can also use Lemma 18 for the congruence-based solution. Here also a simplistic approach would be to convert the 2NWA for $\text{fold}((A_1))$ into an NWA, with an exponential blow-up, and proceed as in Section 6. To avoid this exponential blow-up, we need an exponential bound on the number of congruence classes. For an NWA, 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 2NWA $A$, a congruence class can be defined in terms of four binary relations over the set $S_1$ of states of $A$:

1. $R_0$: a pair $(p_1, p_2) \in R_0$ means that there is a word $w$ that leads $A$ from $p_1$ to $p_2$, where $w$ is entered on the left and exited on the right.

2. $R_1$: a pair $(p_1, p_2) \in R_1$ means that there is a word $w$ that leads $A$ from $p_1$ to $p_2$, where $w$ is entered on the right and exited on the left.

3. $R_2$: a pair $(p_1, p_2) \in R_2$ means that there is a word $w$ that leads $A$ from $p_1$ to $p_2$, where $w$ is entered on the left and exited on the right.

4. $R_3$: a pair $(p_1, p_2) \in R_3$ means that there is a word $w$ that leads $A$ from $p_1$ to $p_2$, where $w$ is entered on the right and exited on the right.

Thus, the number of congruence classes when $A$ 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 5 with essentially the same complexity bounds.

**Theorem 19.** In the case where $Q_1$, $Q_2$, and $Q_v$ are CRPQs, the view-existence problem is ExpTime and the exact view-existence problem is in ExpSpace.

**7.2 CRPQs**

Consider now the view-synthesis problem for the case where $Q_v$ and $Q_2$ are CRPQs, where the constituent RPQs are expressed by means of NWAs. Here the views have to be RPQs, rather than CRPQs, since CRPQs are not closed under substitutions. Thus, we can still represent views in terms of a labeled tree $V : \Sigma^\pm \rightarrow \{0, 1\}^n$. 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 [7].

Let $q_h$, for $h \in \{1, 2\}$, be in the form

$q_h(x_1, \ldots, x_m) \leftarrow q_{h,1}(y_{h,1}, y_{h,2}) \land \cdots \land q_{h,m}(y_{h,2m_h-1}, y_{h,2m_h})$

and let $V_1, V_2$ be the sets of variables of $q_1$ and $q_2$ respectively. It is shown in [7] that the containment $q_1 \subseteq q_2$ can be reduced to the containment $(A_1) \subseteq (A_2)$ of two word automata $A_1$ and $A_2$. $A_1$ is an NWA, whose size is exponential in $q_1$ and it accepts certain words of the form

$d_1w_1d_2d_3w_2d_4 \cdots d_{2m_h-1}w_{m_1}d_{2m_2}$

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 semistructured databases that are canonical for $q_1$ in some sense. $A_2$ is a 2NWA, 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 $(A_1) \subseteq (A_2)$ is shown in [7].

We can now adapt the tree-automata technique of Section 5. From $q_h$ and $q_2$ we can construct word automata $A_1$ and $A_2$ as in [7]. We now ask if we have nonempty RPQ views $V$ such that $(A_1[V]) \subseteq (A_2)$. This can be done as in Section 6, after converting $A_2$ to an NWA.

The ability to reduce containment of CRPQs to containment of word automata means that we can also apply the congruence-class technique of Section 5. Suppose that we have nonempty RPQ views $V$ such that $(A_1[V]) \subseteq (A_2)$. Then we can again assume that the views are closed with respect to the congruence classes of $A_1$. Thus, the techniques of Section 5 can be applied.

**Theorem 20.** In the case where $Q_v$, $Q_1$, and $Q_2$ are CRPQs, the view-existence problem is in $2\text{ExpTime}$, and the exact view-existence problem is in $2\text{ExpSpace}$.

**7.3 UC2RPQs**

Here we allow both C2RPQs and unions. Since UC2RPQs are not closed under substitutions, we consider here 2RPQ views. The extension to handle unions is straightforward. To handle UC2RPQs, we need to combine the techniques of Sections 7.1 and 7.2. The key idea is the reduction of query containment to containment of word automata. The resulting upper bounds are identical to those we obtained for CRPQs.

**8. CONCLUSIONS**

In this paper we have addressed the issue of synthesizing a set of views starting from a collection of mappings relating a source schema to a target schema.

We have argued that the problem is relevant in several scenarios, especially data warehousing, data integration and mashup, and data exchange. We have provided a formalization of the problem based on query rewriting, and we have presented techniques and complexity upper bounds for two cases, namely, relational data, and graph-based semistructured data. We concentrated on the basic problems of view-existence, and we have shown that in both cases the problem is decidable, with different complexity upper bounds depending on the types of query languages used in the mappings and the views, and on the variant (sound or exact rewritting) of the problem.
We plan to continue investigating the view-synthesis problem along different directions. First, we aim at deriving lower complexity bounds for the view-existence problem. Secondly, we are interested in studying view-synthesis for tree-based (e.g., XML) semistructured data. Finally, while in this paper we have based the notion of view-synthesis on query rewriting, it would be interesting to explore a variant of this notion, based on query answering using views. In this variant, views \( V \) capture a mapping of the form \( q_\ell \leadsto q_\ell' \) if, for each source database \( D_s \), the query \( q_\ell \) computes the certain answers to \( q_\ell \) wrt \( V \) and \( D_s \). 

9. REFERENCES

[1] S. Abiteboul. Querying semi-structured data. In Proc. of the 6th Int. Conf. on Database Theory (ICDT’97), pages 1–18, 1997.

[2] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS’98), pages 254–265, 1998.

[3] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison Wesley Publ., Co., 1995.

[4] F. N. Afrati, R. Chirkova, M. Gergatsoulis, B. Kimelfeld, V. Pavlaki, and Y. Sagiv. On rewriting XPath queries using views. In Proc. of the 12th Int. Conf. on Extending Database Technology (EDBT 2009), pages 168–179, 2009.

[5] P. C. Arocena, A. Fuxman, and R. J. Miller. Composing local-as-view mappings: Closure and applications. In Proc. of the 13th Int. Conf. on Database Theory (ICDT 2010), 2010. To appear.

[6] P. Buneman. Semistructured data. In Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS’97), pages 117–121, 1997.

[7] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000), pages 176–185, 2000.

[8] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing using views for regular path queries with inverse. In Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000), pages 58–66, 2000.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. J. of Computer and System Sciences, 64(3):443–465, 2002.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. SIGMOD Record, 32(4):83–92, 2003.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. Theoretical Computer Science, 371(3):169–182, 2007.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. An automata-theoretic approach to Regular XPath. In Proc. of the 12th Int. Symp. on Database Programming Languages (DBPL 2009), volume 5708 of Lecture Notes in Computer Science, pages 18–35. Springer, 2009.

[13] D. Chen, R. Chirkova, and F. Sadri. Query optimization using restricted views: Theory and experiments. Information Systems, 34(3):353–370, 2009.

[14] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. In Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2001), pages 59–68, 2001.

[15] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In Proc. of the 13th Int. Conf. on Database Theory (ICDT 2010), 2010. To appear.

[16] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos. iMAP: Discovering complex mappings between database schemas. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 383–394, 2004.

[17] G. Di Lorenzo, H. Hacid, H.-Y. Paik, and B. Benatallah. Data integration in mashups. SIGMOD Record, 38(1):59–66, 2009.

[18] F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. J. on Data Semantics, 9:1–38, 2007.

[19] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In Proc. of the 7th Int. Conf. on Database Theory (ICDT’99), volume 1540 of Lecture Notes in Computer Science, pages 332–347. Springer, 1999.

[20] G. Grahne and A. Thono. Algebraic rewritings for optimizing regular path queries. Theoretical Computer Science, 296(3):453–471, 2003.

[21] A. Y. Halevy. Theory of answering queries using views. SIGMOD Record, 29(4):40–47, 2000.

[22] A. Y. Halevy. Answering queries using views: A survey. Very Large Database J., 10(4):270–294, 2001.

[23] W. H. Inmon. Building the Data Warehouse. John Wiley & Sons, second edition, 1996.

[24] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In Proc. of the 24rd ACM SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005), pages 61–75, 2005.

[25] M. Lenzerini. Data integration: A theoretical perspective. In Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), pages 233–246, 2002.

[26] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS’95), pages 95–104, 1995.

[27] T. E. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Language Theory, volume 1, chapter 10, pages 679–746. Springer, 1997.

[28] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. Very Large Database J., 10(4):334–350, 2001.

[29] W. Sakoda and M. Sipser. Nondeterminism and the size of two way finite automata. In Proc. of the 10th
[30] P. Senellart and G. Gottlob. On the complexity of deriving schema mappings from database instances. In Proc. of the 27th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2008), pages 23–32, 2008.

[31] B. ten Cate and P. G. Kolaitis. Structural characterizations of schema-mapping languages. In Proc. of the 12th Int. Conf. on Database Theory (ICDT 2009), pages 63–72, 2009.

[32] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 535–548, 2009.

[33] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. Very Large Database J., 5(2):101–118, 1996.

[34] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. J. of Computer and System Sciences, 32:183–221, 1986.

[35] Z. Zhang and A. Mendelzon. Authorization views and conditional query containment. In Proc. of the 10th Int. Conf. on Database Theory (ICDT 2005), volume 3363 of Lecture Notes in Computer Science, pages 259–273. Springer, 2005.