January 2001

Optimization Properties for Classes of Conjunctive Regular Path Queries

Alin Deutsch
University of Pennsylvania

Val Tannen
University of Pennsylvania, val@cis.upenn.edu

Optimization Properties for Classes of Conjunctive Regular Path Queries

Abstract

We are interested in the theoretical foundations of the optimization of conjunctive regular path queries (CRPQs). The basic problem here is deciding query containment both in the absence and presence of constraints. Containment without constraints for CRPQs is EXPSPACE-complete, as opposed to only NP-complete for relational conjunctive queries. Our past experience with implementing similar algorithms suggests that staying in PSPACE might still be useful. Therefore we investigate the complexity of containment for a hierarchy of fragments of the CRPQ language. The classifying principle of the fragments is the expressivity of the regular path expressions allowed in the query atoms. For most of these fragments, we give matching lower and upper bounds for containment in the absence of constraints. We also introduce for every fragment a naturally corresponding class of constraints in whose presence we show both decidability and undecidability results for containment in various fragments. Finally, we apply our results to give a complete algorithm for rewriting with views in the presence of constraints for a fragment that contains Kleene-star and disjunction.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-01-20.

Optimization Properties for Classes of Conjunctive Regular Path Queries

Alin Deutsch Val Tannen

University of Pennsylvania

Abstract

We are interested in the theoretical foundations of the optimization of conjunctive regular path queries (CRPQs). The basic problem here is deciding query containment both in the absence and presence of constraints. Containment without constraints for CRPQs is EXPSPACE-complete, as opposed to only NP-complete for relational conjunctive queries. Our past experience with implementing similar algorithms suggests that staying in PSPACE might still be useful. Therefore we investigate the complexity of containment for a hierarchy of fragments of the CRPQ language. The classifying principle of the fragments is the expressivity of the regular path expressions allowed in the query atoms. For most of these fragments, we give matching lower and upper bounds for containment in the absence of constraints. We also introduce for every fragment a naturally corresponding class of constraints in whose presence we show both decidability and undecidability results for containment in various fragments. Finally, we apply our results to give a complete algorithm for rewriting with views in the presence of constraints for a fragment that contains Kleene-star and disjunction.

1 Introduction

Semistructured data models and query languages [1] have become a very active area of interesting research in databases In this paper we are interested in semistructured query languages, more precisely in theoretical foundations of query optimization for such languages. We concentrate on two computational problems:

- The problem of query equivalence (more generally, query containment), with or without integrity constraints.
- The problem of rewriting queries to make (some) use of views, again with or without integrity constraints.

For queries on relational, complex values, dictionary and OO data, these problems can be solved nicely and uniformly with a strengthening of the classical ideas on tableaux and chase. (See the chase & backchase technique in [9] and some of its theoretical foundation in [22].) Although the problems have theoretically intractable lower bounds, these bounds are in terms of query and constraint size. It turns out that these techniques are in fact practical for practical-size queries and constraints [21]. Our experience with implementing them suggests that a necessary condition for practicality is the ability to decide containment in polynomial space. Can this be done for semistructured languages?

At the theoretical core of such languages lie the *conjunctive regular path queries (CRPQs)* of [11, 6]. Here is an example:

$$Q(Y,Z) \leftarrow \texttt{start}\ (a^*|b).c\ X\ ,\ X\ a.b^*\ Y\ ,\ X\ c^*\ Z$$

This is interpreted in a graph whose edge labels are taken from a set containing a, b, c while `start` is a constant node. The query returns the set of pairs (Y, Z) of nodes such that for some node X there are paths $\texttt{start} \to X$, $X \to Y$, $X \to Z$ whose labels belong to the regular languages $(a^*|b).c$, $a.b^*$, c^* respectively.

However, containment of general CRPQs is EXPSPACE-complete [6, 11]! Therefore, in this paper we pay attention to restricted fragments of CRPQs. This a an approach validated by practice: typical users exploit only a fraction of the expressive power of regular expressions. This is based on the experiences of users of the semistructured query language StruQL [10], but also of the XML query language XML-QL [8], and it is supported by the restrictions on path expressions imposed by the XPath standard [26]. Here is a very simple example of query optimization in such a fragment. Consider the query

$$A(N) \leftarrow \texttt{start} * X\ ,\ X\ name.John\ Y,\ X\ *.tells.name\ N$$

which returns the names of persons who find out a secret from somebody connected directly or indirectly to John. Assume that the following view is materialized

$$Divulge(V,W) \leftarrow\ \texttt{start}\ *\ U,\ U\ name\ V\ ,\ U\ tells\ W$$

and the following integrity constraint holds

$$\text{(tellAll)}\ \ \forall X,Y\ [\ X\ \ *.tells.*\ \ Y \to\ \ X\ tells\ Y\ ]$$

saying that our database models a society in which whenever two of its members share a secret, eventually everybody connected to them shares that secret. *Under this constraint*, the query A can be equivalently rewritten to use *Divulge*:

$$A'(N) \leftarrow Divulge(X,Y)\ ,\ X\ John\ Z,\ Y\ name\ N.$$

Depending on the storage schema, A' may be cheaper to evaluate. In appendix D we show in detail how the methods we develop in this paper succeed in finding this rewriting.

To study various fragments of CRPQs we develop a novel technique. [6, 11] use automata-theoretic techniques but here we will try something different: reductions to problems formulated in the relational setting. The fragments for which we prove upper bounds and decidability results are such that we can translate queries and dependencies into relational versions, over a special relational schema. For example, the query Q shown above translates to the following union of relational conjunctive queries:

$$\begin{array}{rcl} Q'(y,z) & = & C_1(y,z) \cup C_2(y,z) \\ C_1(y,z) & \leftarrow & a^*(\texttt{start},w_1)\ ,\ c(w_1,X)\ ,\ a(x,w_2), \\ & & b^*(w_2,y)\ ,\ c^*(x,z) \\ C_2(y,z) & \leftarrow & b(\texttt{start},w_1)\ ,\ c(w_1,X)\ ,\ a(X,w_2)\ , \\ & & b^*(w_2,y)\ ,\ c^*(x,Z) \end{array}$$

We think of C_1, C_2 as ordinary relational conjunctive queries over a schema containing a, a^*, b^*, c, c^*. A priori a and a^* etc., are independent binary relation symbols, but we interpret them only in relational instances in which certain relational constraints hold. The constraints are first-order and they say, for example, that a^* is transitive, reflexive, and includes a. Of course, transitive closure itself cannot be expressed in first-order logic. It is therefore remarkable that first-order reasoning suffices for some of the semistructured language fragments we consider in this paper. However, we also provide undecidability results that together with the aforementioned EXPSPACE lower bound [6] show some of the theoretical limits of what can be done about optimization in semistructured languages.

Organization of the remainder of this paper. In section 2 we define the classes (language fragments) of queries and dependencies under study here, as well as their translation into relational correspondents. In section 3 we summarize our results and discuss some related work. Section 4 contains our results on upper bounds for pure query containment while section 5 contains the corresponding lower bound results. Section 6 presents our results on deciding (or not!) containment of queries in the presence of dependencies. Section 7 extends the chase & backchase technique [9] to two of the fragments we study. We conclude in section 8.

We have relegated some proofs and a worked example to the appendices, to be consulted at the discretion of the reader. We have also put in an appendix the extension to unions and disjunction of the chase. Although this extension, in the form that we need, has not—apparently—been published previously, it will not suprise anyone with an understanding of the classical chase.

2 Queries and Constraints

Databases. Let $\mathcal{L}$ be a set of *labels*. For technical reasons we assume that $\mathcal{L}$ is infinite, but of course only a finite number of labels will occur in a given database, query, or constraint. A *semistructured database* is a finite directed graph whose edges are $\mathcal{L}$-labeled. Equivalently, we can be given a set N (the nodes of the graph) and a finite set of labels from $\mathcal{L}$, each interpreted as a non-empty binary relation on N.

A word about **constants** denoting nodes. The upper bound and decidability results do *not*, as stated, assume the presence of such constants. Equalities between distinct constants cause the usual problem [2] and our results can be extended straightforwardly to deal with this. For clarity of exposition we have omitted this extension. On the other hand, some of our examples and even some of the constructions used in lower bounds and undecidability results do use constants denoting nodes. Such use is in fact inessential and is made for the same reasons of clarity.

Queries: CRPQs. A *conjunctive regular path query (CRPQ)* [11, 6] has the general form

$$Q(x_1,\ldots,x_n) \leftarrow A_1\ ,\ \ldots\ ,\ A_m \qquad (1)$$

Here the *atoms* (*conjuncts*) A_i are either equalities $y = z$ or *regular path atoms* of the form $y\ R\ z$ where R is a regular expressions defined by [1]

$$\begin{array}{rcl} R & ::= & l \mid \_ \mid R^* \mid R_1.R_2 \mid (R_1|R_2) \\ * & \overset{\text{def}}{=} & \_^* \end{array} \qquad (2)$$

where l ranges over labels in $\mathcal{L}$ and _ means *any* (single) label. Of course, each *distinguished* variables x_j must also occur in the right hand side. As indicated, we follow [11] in using the *shorthand* * for _*.

If B is a semistructured database, an atom $x\ R\ y$ is *satisfied* by a valuation that maps x, y to nodes s, t in B if there is a path from s to t in B which spells out a word in the language denoted by the regular expression R.

[1]We ask the reader to distinguish between the | in regular expressions and the *meta* use of | as part of the BNF for the syntax.

We extend this definition of atom satisfaction to give semantics to whole CRPQs in the way that is usual for conjunctive queries. Query containment is also defined as usual.

Unions of CRPQs. In spite of being called "conjunctive", CRPQs contain implicit forms of disjunction, most glaringly because of the $|$ operator in regular expressions. In fact, we are naturally led to consider *unions of CRPQs* as the class of queries of interest. It is easy to see that the EXPSPACE upper bound on containment [11, 6] still holds for unions of CRPQs.

Containment and Dependencies. Much of the early relational database theory dealt with conjunctive (tableau) queries and *embedded dependencies* [2] which are logical assertions of the special form

$$\forall \vec{x}\ [C_1(\vec{x}) \rightarrow \exists \vec{y}\ C_2(\vec{x}, \vec{y})] \tag{3}$$

where C_1, C_2 are conjunctions of relational atoms or (in C_2) equalities [2]. Such dependencies are tightly related to containment assertions [27]. Given two (type-compatible) conjunctive queries Q_1, Q_2 it is easy to construct an embedded dependency that is equivalent (in each database instance) to the containment $Q_1 \subseteq Q_2$. It is equally easy to construct an equivalent containment assertion from any given embedded dependency.

In this paper we will consider several classes of queries, and for each of them we will identify a class of dependencies (constraints) that has this kind of tight correspondence with the containment of queries from the associated class.

Add Disjunction: DEDs. Generalizing from conjunctive queries to *unions of conjunctive queries*, we consider the associated class of *disjunctive embedded dependencies (DEDs)* which are logical assertions of the form

$$\forall \vec{x}\ [C_1(\vec{x}) \rightarrow \bigvee_{i=1}^{m} \exists \vec{y_i}\ C_{2,i}(\vec{x}, \vec{y_i})] \tag{4}$$

where $C_1, C_{2,i}$ are as in (3). We don't need disjunction in the premise of the implication because it is equivalent to conjunctions of DEDs. We have the following tight correspondence: the containment of two unions of conjunctive queries is equivalent to a finite number of DEDs, and a single DED is equivalent to the containment of a conjunctive query into a union of conjunctive queries.

A DED is *full* if it does not have existentially quantified variables. The *chase* [3] can be extended to DEDs, giving a decision procedure for containment of unions of conjunctive queries under a set of full DEDs (see [13, 14] for a partial treatment and appendix A for a sketch of the results we use.)

Semistructured Constraints: DERPDs. As with DEDs, we define the class of dependencies that corresponds to unions of conjunctive regular path queries (CRPQs). We call such dependencies *disjunctive embedded regular path dependencies (DERPDs)* and they are defined as assertions that have the same logical form as DEDs, see (4), but in which $C_1, C_{2,i}$ are conjunctions of regular path atoms $x\ R\ y$ or equalities. The definition for satisfaction of a given DERPD in a given semistructured database follows from the usual meaning of logical connectives and quantifiers and from the satisfaction for regular path atoms given earlier.

When the regular expressions are restricted to single labels in $\mathcal{L}$, CRPQs are equivalent to the usual conjunctive queries and DERPDs to just DEDs seen over a relational schema consisting of binary symbols from $\mathcal{L}$.

Examples. DERPDs can express a large variety of constraints on semistructured data. As we saw, they generalize most relational dependencies of interest. In addition we can express constraints similar to the ones DTDs [25] specify for XML. The first two below say that "any person has exactly one social security number". The third says that "telephone numbers can only be of two (if any) kinds, voice or fax" while the fourth is a kind of generalized join-like dependency.

$$\forall x\ [\texttt{start} * . \mathit{person}\ x\ \rightarrow\ \exists y\ x\ \mathit{ssn}\ y]$$

$$\forall x \forall y_1 \forall y_2\ [\texttt{start} * . \mathit{person}\ x\ \wedge$$
$$\wedge\ x\ \mathit{ssn}\ y_1\ \wedge\ x\ \mathit{ssn}\ y_2\ \rightarrow\ y_1 = y_2]$$

$$\forall x \forall y \forall z\ [x\ \mathit{telNo}\ y\ \wedge\ y\ \_\ z\ \rightarrow\ y\ \mathit{voice}\ z\ \vee\ y\ \mathit{fax}\ z]$$

$$\forall x \forall y \forall z\ [x\ \mathit{child}\ y\ \wedge\ y\ \mathit{child}\ z\ \rightarrow\ z\ \mathit{grandparent}\ x]$$

Fragments: F-queries and F-dependencies Since containment of CRPQs is EXPSPACE-complete [6] we study *fragments* of the language defined by restricting the regular expressions allowed in atoms (conjuncts). The simplest fragment, allowing just labels and concatenation, is equivalent to conjunctive queries over binary relations. Between these and general CRPQs we consider the following fragments

[2]The notation $\vec{x}$ abbreviates $x_1, \ldots, x_n$.

Fragment name	Regular expressions syntax
conj. queries	$R \to l \mid R_1.R_2$
$(*)$	$R \to l \mid * \mid R_1.R_2$
$(*, \|)$	$R \to l \mid * \mid R_1.R_2 \mid (R_1 \| R_2)$
$(*, \_)$	$R \to l \mid \_ \mid * \mid R_1.R_2$
(l^*)	$R \to l \mid l^* \mid R_1.R_2$
$(*, \_, l^*, \|)$	$R \to l \mid * \mid \_ \mid l^* \mid R_1.R_2 \mid (R_1 \| R_2)$
W	$R \to l \mid \_ \mid S^* \mid R_1.R_2 \mid (R_1 \| R_2)$
	$S \to l \mid \_ \mid S_1.S_2$
Z	$R \to S \mid S^*$
	$S \to l \mid S_1.S_2 \mid (S_1 \| S_2)$
CRPQs	$R \to l \mid \_ \mid R^* \mid R_1.R_2 \mid (R_1 \| R_2)$

For any fragment F, we call the corresponding queries *F-queries.* Applying the same restriction to the atoms that appear in dependencies, we define corresponding classes of DERPDs, calling the respective constraints *F-dependencies.* The correspondence discussed above, between containment assertions and dependencies, continues to hold for each fragment F. The fragments called W and Z have technical importance but their definitions did no suggest anything better than choosing these arbitrary names.

First-Order Relational Translation. At the core of our technique is a translation of semistructured queries and dependencies into first-order logic, namely into (unions of) conjunctive queries and DEDs *over a special relational schema* that includes l and l^* as well as $\_$ and $*$ as separate binary relation symbols. A priori these symbols are independent, but we will try to capture *some* of the Kleene star semantics through relational dependencies.

Our translation is designed for the $(*, \_, l^*, |)$-fragment only. It relies essentially on the fact that in this fragment concatenation and $|$ are *not* nested inside Kleene stars.

The first thing we do is **translate away** $|$. Using the equivalence $(a|b).c = (a.c)|(b.c)$ we move $|$ in the outermost position in the $(*, \_, l^*, |)$-regular expressions. Then, we note that $Q \leftarrow \ldots, x\ R_1|R_2\ y, \ldots$ is equivalent to $Q_1 \cup Q_2$ where $Q_i \leftarrow \ldots, x\ R_i\ y, \ldots$. For dependencies, we note that $x\ R_1|R_2\ y$ is equivalent to $x\ R_1\ y \lor x\ R_2\ y$ after which logical equivalences bring the disjunctions out. A disjunctions in the premise of the implication in a dependency is equivalent to a conjunction (a set) of dependencies. To summarize:

Remark 2.1 *By translating away the $|$, any $(*, \_, l^*, |)$-query becomes an equivalent union of $(*, \_, l^*)$-queries. Similarly, any $(*, \_, l^*, |)$-dependency becomes an equivalent set of $(*, \_, l^*)$-dependencies.*

Next, we translate any $(*, \_, l^*)$-queries and dependencies into (relational) conjunctive queries and DEDs over the special schema

$$\mathcal{L}\text{-}Rel \overset{\text{def}}{=} \{l \mid l \in \mathcal{L}\} \cup \{l^* \mid l \in \mathcal{L}\} \cup \{\_, *\} \cup \{N\}$$

in which all symbols are binary relations with the exception of N which is unary (the need for N is explained below).

The **translation** $\mathcal{T}(Q)$ of a $(*, \_, l^*)$-query Q is defined by translating its conjuncts according to the rules (for each binary r in $\mathcal{L}$-*Rel*)

$$\begin{aligned} \mathcal{T}(x\ r\ y) &= r(x, y) \\ \mathcal{T}(x\ R_1.R_2\ y) &= \mathcal{T}(x\ R_1\ v), \mathcal{T}(v\ R_2\ y) \end{aligned}$$

The variable v is (implicitely) existentially quantified and so it must be fresh each time its rule is applied. For example, $Q(x, y) \leftarrow x\ a.*.b\ y$, translates to $Q'(x, y) \leftarrow a(x, z),\ *(z, u),\ b(u, y)$.

The **translation** $\mathcal{T}(d)$ of a $(*, \_, l^*)$-dependency d is defined similarly. The presence of concatenation in the conclusion of the implication in d will add existentially quantified variables, while the presence of concatenation in the premise of the implication in d will add universally quantified variables.

Example of translation. Let d be the dependency

$$\forall x \forall y\ [x\ (a|b).*\ y\ \to\ \exists z\ y\ *.(a|b)\ z]$$

It translates to the following set of two DEDs

$$\begin{aligned} \forall x \forall y \forall u\ [x\ a\ u\ \land\ u\ *\ y\ \to\ & [\exists z_1 \exists v_1\ \ y\ *\ v_1\ \land\ v_1\ a\ z_1] \\ & \lor\ [\exists z_2 \exists v_2\ \ y\ *\ v_2\ \land\ v_2\ b\ z_2]] \end{aligned}$$

$$\begin{aligned} \forall x \forall y \forall u\ [x\ b\ u\ \land\ u\ *\ y\ \to\ & [\exists z_1 \exists v_1\ \ y\ *\ v_1\ \land\ v_1\ a\ z_1] \\ & \lor\ [\exists z_2 \exists v_2\ \ y\ *\ v_2\ \land\ v_2\ b\ z_2]] \end{aligned}$$

Now, $\mathcal{T}(Q)$ is a relational query and $\mathcal{T}(d)$ is a relational dependency, both over the schema $\mathcal{L}$-*Rel.* However, we will use them not over arbitrary instances of $\mathcal{L}$-*Rel* but only over instances that satisfy specific sets of relational dependencies. To deal with the various fragments, we consider two such sets

The Σ_* dependencies:

$$\begin{aligned} (\text{node}_l) &\quad \forall x \forall y\ [l(x, y) \to N(x) \land N(y)] \\ (\text{node}_*) &\quad \forall x \forall y\ [*(x, y) \to N(x) \land N(y)] \\ (\text{base}) &\quad \forall x \forall y\ [l(x, y) \to *(x, y)] \\ (\text{refl}_*) &\quad \forall x\ [N(x) \to *(x, x)] \\ (\text{trans}_*) &\quad \forall x \forall y \forall z\ [*(x, y) \land *(y, z) \to *(x, z)] \end{aligned}$$

where l ranges over $\mathcal{L}$. (This is an infinite set of dependencies but of course only finitely many matter for

Containment of	Upper bound
conjunctive queries	NP [7]
$(*)$-queries	NP [11] or corollary 4.3
unions of conj. queries	Π_2^p [23]
unions of $(*, \mid)$-queries	$\uparrow$
unions of $(*, \_)$-queries	$\uparrow$
unions of (l^*)-queries	$\uparrow$
unions of $(*, \_, l^*, \mid)$-queries	Π_2^p theorem 4.8
unions of W queries	?
unions of Z queries	$\uparrow$
unions of CRPQs	EXPSPACE [11, 6]

Containment of	Lower bound
conjunctive queries	NP [7]
$(*)$-queries	$\downarrow$
unions of conj. queries	Π_2^p [23]
$(*, \mid)$-queries	Π_2^p remark 3.1
$(*, \_)$-queries	Π_2^p theorem 5.1
(l^*)-queries	Π_2^p theorem 5.1
$(*, \_, l^*, \mid)$-queries	$\downarrow$ $\downarrow$ $\downarrow$
W queries	PSPACE theorem 5.2
Z queries	EXPSPACE [6] and remark 3.2
CRPQs	$\downarrow$

Figure 1: Upper and lower bounds for containment

a given database, query, or dependency.) Here we see how we use N: we want the chase with (refl$_*$) to apply only to variables x that are already present.

The Σ_{l^*} dependencies: are obtained by replacing $*$ with l^* in Σ_* above.

The intention behind these dependencies is to narrow the gap between the semistructured meaning of the Kleene star and the arbitary interpretation that could be given to the relational schema $\mathcal{L}$-*Rel*. We can associate directly to each semistructured database a relational $\mathcal{L}$-*Rel*-instance that satisfies $\Sigma_* \cup \Sigma_{l^*}$ (call it a $\Sigma_* \cup \Sigma_{l^*}$-instance). But this will not cover $\Sigma_* \cup \Sigma_{l^*}$-instances containing pairs of distinct nodes which are not connected by any path with labels from $\mathcal{L}$. Of course, it is not possible to close the gap this way, since transitive closure is not first-order definable. It is therefore remarkable that first-order reasoning suffices for some of the semistructured language fragments we consider in this paper.

Full Dependencies. Relational dependencies (3) and DEDs (4) are called *full* when they do not have existentially quantified variables. In the case of DERPDs fullnes must be more complicated because concatenation in regular expressions introduces an implicit existential. Here we take a very simple approach.

Let d be an $(*, \_, l^*, |)$-dependency and let $\mathcal{T}(d)$ be the set of DEDs into which d translates. We say that d is a *full dependency* if each DED in $\mathcal{T}(d)$ is full.

3 Summary of Results

Containment for F-queries. We summarize in figure 1 our new results on the complexity of deciding containment for queries in the various fragments, putting them in the context of known results.

The upper bounds are for containment of unions of F-queries, with the remarkable exception of the $(*)$-fragment for which containment of $(*)$-queries is in NP, just like containment of conjunctive queries. This was already shown in [11]. Motivated by the study of containment *under dependencies*, the new technique introduced here reproves, along the way, this NP bound, see corollary 4.3. [3]

Our new upper bound result is that containment of unions of $(*, \_, l^*, |)$-queries is in Π_2^p (theorem 4.8). It can be seen from the lower bounds table that this is a tight bound.

We have tried to state our lower bound results in their strongest form, for F-queries rather than unions of F-queries. It is not suprising to see a Π_2^p lower bound in the presence of $|$. This does not follow directly from [23] but we have

Remark 3.1 *The Π_2^p-hardness proof in [23] for the lower bound on containment of unions of conjunctive queries can be adapted to containment of F-queries provided that F includes $|$.*

It is suprising however what happens in the absence of $|$. While containment of $(*)$-queries is in NP, we show in theorem 5.1 that containment of (l^*)-queries is Π_2^p-hard. Moreover a simple variation of the same proof applies to the $(*, \_)$-fragment. Therefore, we find that the increase in complexity does not stem from the mere presence of the Kleene star in the query, but from the *interaction* between l and l^* or between _ and _*.

A more liberal nesting of regular expressions within the Kleene star increases complexity. If we allow concatenation inside the Kleene star, we get the W-fragment,

[3]Although we do not consider it explicitely here, the fragment obtained by adding just _ to labels and concatenation is easily seen to yield no suprises: containment is still in NP.

Containment of	Under what constraints	Decidable?
conjunctive queries	full relational dependencies	YES ([3])
unions of conjunctive queries	full DEDs	YES (theorem A.3)
unions of $(*, \|)$-queries	full $(*, \|)$-deps.	YES (theorem 6.1)
unions of $(*, \_)$-queries	full V-deps.	YES (theorem 6.2)
$(*, \_)$-query in union of $(*, \_)$-queries	full DEDs over special models	NO (theorem 6.3)
(l^*)-query in union of (l^*)-queries	full DEDs	NO (theorem 6.4)

Figure 2: Results for containment under dependencies

for which we show in theorem 5.2 a PSPACE lower bound on containment. We don't know (mainly because of difficulties with a relational translation) if this bound is tight, which is why we put a question mark in the corresponding upper bound entry. If in addition we allow disjunction within the Kleene star, we obtain the Z-fragment which is as bad as general CPQRs:

Remark 3.2 *The EXPSPACE-hardness proof in [6] applies to containment of Z-queries.*

Containment under dependencies. The chase technique in classical relational theory gives us the decidability of containment of conjunctive queries under full dependencies [2]. Decidability extends straightforwardly to containment of unions of conjunctive queries under full DEDs (theorem A.2).

This nice situation for relational languages contrasts with the situation for semistructured languages, as summarized in figure 3. The general problem studied is containment of unions of F-queries under full F-dependencies. It turns out that even containment of (l^*)-queries under just full DEDs is undecidable (theorem 6.4).

There is some good news, as our technique carries through in theorem 6.1 to prove decidability for the $(*, |)$-fragment.

We leave open the general problem corresponding to the $(*, \_)$, but we have two partial results that suggest that the problem might be complicated. We show decidability in the case of a restricted class of $(*, \_)$-dependencies, that we call V-*dependencies* (definition in section 6). And we show *un*decidability with just DEDs in the case of a class of *special models* (also defined in section 6).

In our two undecidability proofs, just like in the proof of theorem 5.1, we make essential use of of the *interaction* between l and l^* or between $\_$ and $\_^*$.

Rewriting with Views under Dependencies. Given a set V of views, a set D of dependencies expressing integrity constraints, and a query Q, we are interested in finding "rewritings" Q' which mention some of the views (but may still contain labels from Q) and are *exactly* equivalent to Q.

We do not study this problem in its full generality, but rather we look at extending to some of the F-fragments the *chase&backchase (C&B)* algorithm that we introduced in [9]. This algorithm relies on the chase with dependencies. In view of the undecidability results we have obtained for other F-fragments, we have have looked at rewriting with views only for the $(*)$- and $(*, |)$-fragments.

In theorem 7.1 we show that (essentially) the C&B algorithm is complete for the $(*)$-fragment, in the sense that it finds all rewritings that are *minimal* in a precise sense.

For the $(*, |)$-fragment we extend the original C&B algorithm to account for disjunction, and we prove that this extended version is also complete.

Related work. Perhaps the closest in spirit is [4], which gives an EXPTIME-complete decision procedure for containment of queries and constraints expressed in a different fragment of CRPQs, which corresponds to description logics. This fragment allows unrestricted regular expressions in the conjuncts, but restricts the shape of the query graph (thus being incompatible with our classification principle for query fragments). The corresponding dependencies allow unrestricted regular path expressions and even cardinality constraints, but have restricted shape and in particular cannot express functional dependencies. As a matter of fact, [12] shows that, when adding functional dependencies to a generalization of description logics called the Guarded Fragment of first order logic, satisfiability (and hence containment) becomes undecidable. None of our query fragments is contained in description logics.

The class of $(*)$-queries was introduced in [11] (under the name of "simple StruQL$_0$ queries") as a class of semistructured queries using transitive closure and whose containment problem is in NP. The decision procedure was based on an automata-theoretic argument which was applicable to CRPQs with arbitrary regular path expressions.

[18, 19] study the expressivity and satisfiability of queries over tree structures, in formalisms that are equivalent to MSO. Classes of tree structures are given as grammars, which can be viewed as constraints on their structure in a broader sense.

[5] gives a complete algorithm for finding rewritings of regular path queries (i.e. single-conjunct CRPQs) with views defined by regular path queries. The path expressions allowed in the conjunct are unrestricted, but no

constraints are taken into account, and only complete rewritings are obtained (that is, rewritings mentioning only views). [16] addresses the problem of finding arbitrary rewritings of reqular path queries, and [15] gives an algorithm for the related problem of *answering* regular path queries using incomplete views.

4 Upper Bounds

$(*)$-queries. Recall that a $(*)$-query is a CRPQ whose atoms allow only regular expressions built from labels, $*$, and their concatenation. For example, $Q(x,y) \leftarrow x\ a.*.b\ y,\ y\ c\ x$ is a $(*)$-query, as opposed to $Q'(y) \leftarrow x\ a.b^*.c\ y$ (because of b^*) and $Q''(y) \leftarrow x\ a|b\ y$ (because of $|$).

We have shown in section 2 how to translate any $(*)$-query into a conjunctive query $\mathcal{T}(Q)$ over the schema $\mathcal{L}$-*Rel.* While not obvious, it turns out that reasoning about $\mathcal{T}(Q)$ under the set of dependencies Σ_* introduced in section 2 suffices:

Proposition 4.1 *Let Q_1, Q_2 be two $(*)$-queries. The containment $Q_1 \subseteq Q_2$ is valid if and only if $\Sigma_* \models \mathcal{T}(Q_1) \subseteq \mathcal{T}(Q_2)$.*

Proof: It suffices to show that for any Σ_*-instance I there exists a semistructured database B such that $Q_i(B) = \mathcal{T}(Q_i)(I)\ \ i = 1, 2$.

Indeed, pick a label $l \in \mathcal{L}$ which does not occur in either Q_i (infinitely many labels qualify, since the Q_is are finite and $\mathcal{L}$ is not). To obtain B, start by reducing I to the schema $\mathcal{L}$-*Rel* Now keep extending B by adding to l's interpretation every pair $(s,t) \in *$ for which there is no path from s to t in B. It is easy to see that that $Q_i(B) = \mathcal{T}(Q_i)(I)$, since the Q_is don't even mention l, and thus cannot derive answers that rely on the length or label of the path from s to t, for any s,t as above. •

Next, we observe that the dependencies in Σ_* are full hence the chase with them terminates, giving a decision procedure for $\Sigma_* \models \mathcal{T}(Q_1) \subseteq \mathcal{T}(Q_2)$ [3, 2] We denote with $chase_{\Sigma_*}(Q)$ the result of chasing the query Q with the dependencies in Σ_*.

Theorem 4.2 *The $(*)$-query Q_1 is contained in the $(*)$-query Q_2 if and only if there exists a containment mapping (see [2]) from $\mathcal{T}(Q_2)$ into $chase_{\Sigma_*}(\mathcal{T}(Q_1))$.*

Corollary 4.3 [11]
$(*)$-query containment is NP-complete.

Proof: First notice that the size of $\mathcal{T}(Q)$ is linear in that of Q. The chase takes time polynomial in the size of the queries, but exponential in the maximum size of a dependency and the maximum arity of the relations in the schema [3]. However, the dependencies in Σ_* have fixed size and the maximum arity of a relation in the schema is 2. The upper bound follows noting that the containment mapping can be found in NP. For the lower bound, it can be seen that the proof of NP-hardness for containment of conjunctive queries in [7] can be adapted to require binary relations only. •

Example. Consider $Q_1(x_1,x_3) \leftarrow x_1\ a\ x_2,\ x_2\ b.c\ x_3$ and $Q_2(y_1,y_2) \leftarrow y_1\ \ *.a.*\ \ y_2$. It is easy to see that Q_1 is contained in Q_2. We show how we infer this using theorem 4.2. The translation to conjunctive queries yields $TQ_1 = \mathcal{T}(Q_1)$ and $TQ_2 = \mathcal{T}(Q_2)$, with $TQ_1(x_1,x_3) \leftarrow a(x_1,x_2), b(x_2,u_1), c(u_1,x_3)$ and $TQ_2(y_1,y_2) \leftarrow *(y_1,v_1), a(v_1,v_2), *(v_2,y_2)$. Note that there is no containment mapping from TQ_2 to TQ_1 as the latter contains no $*$-atoms to serve as image for the former's $*$-atoms. But by chasing TQ_1 with (node$_a$) and then with (refl$_*$), we obtain $Q'(x_1,x_3) \leftarrow N(x_1), N(x_2), *(x_1,x_1), a(x_1,x_2), b(x_2,u_1), c(u_1,x_3)$ thus creating an image for TQ_2's conjunct $*(y_1,v_1)$. We continue chasing with (base$_*^b$), then (base$_*^c$) and (trans$_*$), obtaining $Q"(x_1,x_3) \leftarrow N(x_1), N(x_2), *(x_1,x_1), a(x_1,x_2), b(x_2,u_1), c(u_1,x_3), *(x_2,u_1), *(u_1,x_3), *(x_2,x_3)$. Now $\{y_1 \mapsto x_1, y_2 \mapsto x_3, v_1 \mapsto x_1, v_2 \mapsto x_2\}$ is a containment mapping from $\mathcal{T}(Q_2)$ into $Q"$. There are further applicable chase steps, omitted here as they only add new atoms and hence do not affect the existence of the containment mapping. •

Unions of $(*, \_, l^*, |)$-queries. The idea we have just used to handle $(*)$-queries is easily extended to $(*, |)$-queries (giving a Π_2^p procedure), but how about other fragments? Can we deal with (l^*)-queries using their relational translation and the set Σ_{l^*} of dependencies defined in section 2? The answer is negative, which is surprising given the syntactic similarity of the $(*)$- with the (l^*)-fragment.

Proposition 4.4 *There exist (l^*)-queries Q, Q' such that $Q \subseteq Q'$ but $\Sigma_{l^*} \not\models \mathcal{T}(Q) \subseteq \mathcal{T}(Q')$.*

Proof: Here are the queries (see figure 4 for possibly helpful graph representations of Q, Q'):

$$\begin{aligned} Q(x,y) &\leftarrow x\ a\ u_1,\ x\ a\ u_2,\ u_1\ c\ u_3,\ u_1\ b\ u_4,\ u_2\ b\ u_5, \\ &\qquad u_2\ c\ u_4,\ u_3\ l.l\ y,\ u_4\ l.l^*\ y,\ u_5\ l\ y \\ Q'(x,y) &\leftarrow x\ a\ v_1,\ v_1\ b\ v_2,\ v_1\ c\ v_3,\ v_2\ l\ y,\ v_3\ l.l.l^*\ y \end{aligned}$$

To see that Q is contained in Q', observe that $ll^* = l \cup lll^*$ and Q is equivalent to the union of queries $Q_1 \cup Q_2$ where Q_1, Q_2 are obtained by replacing the conjunct $u_4\ ll^*\ y$ with $u_4\ l\ y$, respectively $u_4\ lll^*\ y$ in Q. But both Q_1, Q_2 are contained in Q', as witnessed by the containment mappings $\{v_1 \mapsto u_1, v_2 \mapsto u_4, v_3 \mapsto u_3\}$ and $\{v_1 \mapsto u_2, v_2 \mapsto u_5, v_3 \mapsto u_4\}$. Intuitively, for any instance I, and any mapping from Q to I, depending on whether $u_4\ l.l^*\ y$ in Q is satisfied by a path of length 1 or at least 2, $v_1\ c\ v_3$ in Q' is satisfied by the same path which satisfies either $u_1\ c\ u_3$ or $u_2\ c\ u_4$, respectively.

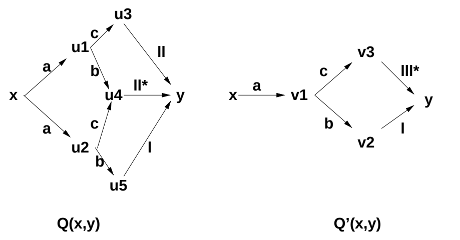

Figure 3: Counterexample queries for proposition 4.4

On the other hand, according to the chase theorem [2], $\mathcal{T}(Q)$ is not contained in $\mathcal{T}(Q')$ under Σ_{l*} because there is no containment mapping from $\mathcal{T}(Q')$ into $chase_{\Sigma_{l*}}(\mathcal{T}(Q))$. (Intuitively, what Σ_{l*} does not capture is the *minimality* of l^*: it only states that l^* contains the reflexive transitive closure of l, but it doesn't rule out pairs of nodes that aren't reachable via a path of l-edges. Instances containing such a pair (s,t) are counterexamples for the containment: conjunct $u_4\ ll^*\ y$ in Q is satisfied by the endpoints of the path $r \xrightarrow{l} s \xrightarrow{l^*} t \xrightarrow{l} q$ even if s has no outgoing l-edge, while $v_3\ lll^*\ y$ in Q' is not.) $\bullet$

A simple variation of the counterexample above applies to $(*, \_)$-queries. In any case, if the same idea would have applied it would have given us NP algorithms, and we show in theorem 5.1 that containment for both the (l^*)- and the $(*, \_)$-fragment is Π_2^p-hard! Therefore, we will take another route towards a containment test.

We start from the observation that $\Sigma_* \cup \Sigma_{l*}$ is sufficient in deciding containment of Q_1 in Q_2 in the restricted case in which Q_1 contains no Kleene star (no $*$ or l^*), and Q_2 is a $(*, \_, l^*)$-query. We call Q_1 *star-free.*

Proposition 4.5 *The star-free query Q_1 is contained in the $(*, \_, l^*)$-query Q_2 if and only if there is a containment mapping from $\mathcal{T}(Q_2)$ to $chase_{\Sigma_* \cup \Sigma_{l*}}(\mathcal{T}(Q_1))$.*

A proof sketch is given in appendix B.1. Next we show how to use proposition 4.5 to decide containment even if Q_1 is a proper $(*, \_, l^*)$-query.

In the rest of this section l will denote either a label in $\mathcal{L}$ or the symbol _. Observe that for any $l \in \mathcal{L} \cup \{\_\}$, $l^* = \bigcup_{0 \le p} l^p$, where l^p is short for the concatenation of p successive l's. More generally, let $Q(l_1^*, \ldots, l_n^*)$ be a $(*, \_, l^*)$-query in which $(l_1^*, \ldots, l_n^*)$ are all the *occurrences* of starred symbols (the l_i's are not necessarily distinct). Such a query is equivalent to an infinite union of star-free queries:

$$Q(l_1^*, \ldots, l_n^*) \; = \bigcup_{0 \le p_1, \ldots, 0 \le p_n} Q(l_1^{p_1}, \ldots, l_n^{p_n})$$

The key to our containment test is that this infinite union can be replaced with a finite one. For any $(*, \_, l^*)$-query Q let *sfs*(Q) be the *star-free size* of Q, defined as the count of all occurrences of non-Kleene-starred labels in Q. For example, for $Q(x,y) \leftarrow x\ a.b^*\ y\ ,\ y\ *.c\ z$ we have *sfs*$(Q) = 2$.

Proposition 4.6 *Let Q_1, Q_2 be two $(*, \_, l^*)$-queries and let $k \overset{\text{def}}{=}$ sfs$(Q_2) + 1$. Then, $Q_1 \subseteq Q_2$ if and only if*

$$\bigcup_{0 \le p_1 \le k, \ldots, 0 \le p_n \le k} Q_1(l_1^{p_1}, \ldots, l_n^{p_n}) \; \subseteq \; Q_2$$

The proof is given in appendix B.2. We can now give our decision procedure for containment of unions of $(*, \_, l^*, |)$-queries, which has four steps:

Step 1: We first translate away the $|$, obtaining finite unions U_1, U_2 of $(*, \_, l^*)$-queries.

Step 2: Next we use proposition 4.6 to obtain from U_1 a finite union of star-free queries SF_1, which must be checked for containment in U_2.

Step 3: Containment of SF_1 in U_2 is decided using the following easy result:

Proposition 4.7 *The union of star-free queries $\bigcup_{i=1}^n Q_i$ is contained in the union of $(*, \_, l^*)$-queries $\bigcup_{j=1}^m Q'_j$ if and only if for every $1 \le i \le n$ there is a $1 \le j \le m$ such that $Q_i \subseteq Q'_j$.*

Step 4: Finally, checking each star-free Q_i for containment in Q'_j is done using proposition 4.5.

The upper bound for this algorithm is straightforward, and its proof is in appendix B.3:

Theorem 4.8 *Containment of unions of $(*, \_, l^*, |)$-queries is in Π_2^p.*

5 Lower Bounds

(l^*)-queries, $(*, \_)$-queries. The $|$ operator corresponds to the union and containment for unions of conjunctive queries is Π_2^p-complete [23]. But it turns out that even in the absence of $|$ we have Π_2^p-hardness results, with completely different proofs:

Theorem 5.1 *Containment of (l^*)-queries is Π_2^p-hard. Containment of $(*, \_)$-queries is Π_2^p-hard.*

Proof: Essentially the same proof works for both fragments. We show the proof for the (l^*)-fragment (the proof for the $(*, \_)$-fragment is a straightforward modification).

The proof is by reduction from the Π_2^p-complete $\forall\exists 3-SAT$ problem [20]: the instances of this problem are first-order sentences ϕ of general form
$\forall x_1 \ldots \forall x_n \exists y_1 \ldots \exists y_m \ \bigwedge_{i=1}^{l} C_i$,
where each clause C_i is a disjunction of three literals which are any of the variables $x_1, \ldots, x_n, y_1, \ldots, y_m$ or their complements. The literals are said to be *negative*

in the latter case and *positive* in the former. ϕ is a "yes" instance if and only if it is valid.

For every instance ϕ, we construct the instance $Q_1 \subseteq Q_2$, where ϕ's variables appear as labels, and Q_1 contains an occurrence of x_i^* for every $1 \leq i \leq n$. We use the notation $Q_1(x_1^*, \ldots, x_n^*)$ introduced for proposition 4.6. The containment holds if and only if $Q_1(x_1^{p_1}, \ldots, x_n^{p_n}) \subseteq Q_2$ for all $0 \leq p_i$. We *claim* that the reduction is defined such that the latter holds if and only if ϕ has a satisfying assignment which makes x_i false if $p_i = 0$, and true if $p_i > 0$. This makes ϕ valid if and only if $Q_1 \subseteq Q_2$. The claim is proved after we give the construction.

Both Q_1, Q_2 are boolean-valued queries, i.e. they have no distinguished variables. Q_1's body is constructed using copies of a *universal gadget*. The universal gadget (defined shortly) is denoted $U(v_{in}, v_0, v_1, l)$ and it consists of a set of conjuncts containing the variables v_{in}, v_0, v_1 (among others) and an occurrence of l^* for some label l. For every x_i, Q_1 contains a copy of U, in which v_{in}, v_0, v_1 are substituted with the variables $v_{in}^{x_i}, v_0^{x_i}, v_1^{x_i}$, l is substituted with x_i, and the remaining variables are freshly renamed. We denote this copy with $U(v_{in}^{x_i}, v_0^{x_i}, v_1^{x_i}, x_i)$. Q_1 also contains a conjunct $r\ x_i\ v_{in}^{x_i}$ for every x_i.

For every y_j, we add the conjuncts $r\ y_j\ v_{in}^{y_j}$, $v_{in}^{y_j}\ y_j\ v_0^{y_j}$, $v_{in}^{y_j}\ y_j\ v_1^{y_j}$ to Q_1.

Every clause C_i is satisfied by 7 distinct truth assignments $a_1, \ldots, a_7$ to the variables in its literals. For every a_j, we add a conjunct $s_j\ C_i\ t_j$ to Q_1. For every literal of C_i, let u be its variable. If $a_j(u) = true$, add a conjunct $v_1^u\ a\ s_j$ to Q_1, otherwise add the conjunct $v_0^u\ a\ s_j$ instead.

This completes the construction of Q_1, up to the specification of the universal gadget. First we show the construction of Q_2. Q_2's body contains copies of a *satisfaction gadget* (defined shortly). The satisfaction gadget is denoted $S(w_{in}, w_{out}, l)$ and it consists of a set of conjuncts containing the variables w_{in}, w_{out} (among others) and an occurrence of l^* for the same label l as in the universal gadget. For every x_i, Q_2 contains a copy $S(w_{in}^{x_i}, w_{out}^{x_i}, x_i)$ (in the same sense as copies of U). Q_2 also contains a conjunct $r\ x_i\ w_{in}^{x_i}$. For every y_j in ϕ, Q_2 contains the conjuncts $r\ y_j\ w_{in}^{y_j}, w_{in}^{y_j}\ y_j\ w_{out}^{y_j}$. For every clause C_i, Q_2 contains the conjunct $s_i\ C_i\ t_i$, and for every variable u corresponding to one of C_i's literals, Q_2 contains the conjunct $w_{out}^u\ a\ s_i$. The construction so far is exemplified in figure 5, on a $\forall\exists 2 - SAT$ instance for simplicity sake.

We now specify the universal and satisfaction gadgets. Recalling the counterexample in proposition 4.4, $U(v_{in}, v_0, v_1, l)$ is a copy of the body of Q, with x, u_3, u_4, l acting as v_{in}, v_0, v_1, l, respectively. $S(w_{in}, w_{out}, l)$ is a copy of the body of Q', with x, y, l acting as w_{in}, w_{out}, l.

We still have to prove the claim. Note that Since

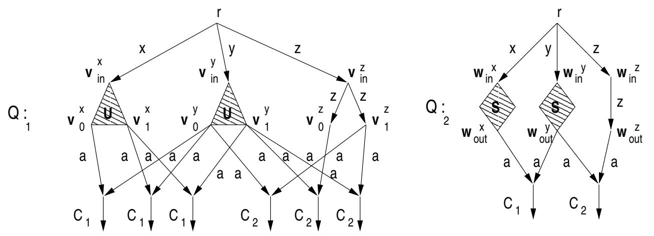

Figure 4: Example reduction for $\phi = \forall x \forall y \exists z\ \underbrace{(x \vee \overline{y})}_{C_1} \wedge \underbrace{(y \vee z)}_{C_2}$

$Q_1(x_1^{p_1}, \ldots, x_n^{p_n})$ contains no Kleene star, $\mathcal{T}(Q_1(x_1^{p_1}, \ldots, x_n^{p_n}))$ can be regarded as a semistructured instance, whose nodes are the variables. Hence $Q_1(x_1^{p_1}, \ldots, x_n^{p_n}) \subseteq Q_2$ if and only if there is a mapping m from the variables of Q_2 to those of $\mathcal{T}(Q_1(x_1^{p_1}, \ldots, x_n^{p_n}))$ which maps distinguished variables to distinguished variables, such that any conjunct $x\ R\ y$ of Q_2 is satisfied by $(m(x), m(y))$. Also note that, by construction, $m(w_{out}^u) = v_0^u$ or $m(w_{out}^u) = v_1^u$ for every such m and every variable u in ϕ, so m determines a truth assignment (assigns *true* to u if and only if $m(w_{out}^u) = v_1^u$). Moreover, every conjunct $s_i\ C_i\ t_i$ in Q_2 is satisfied by $(m(s_i), m(t_i))$ if and only if ϕ's clause C_i is satisfied under the truth assignment corresponding to m. So every m corresponds to a truth assignment satisfying all clauses of ϕ. But from the discussion in the counterexample, it follows that for every universally quantified variable x_i of ϕ, $m(w_{out}^{x_i}) = v_0^{x_i}$ if and only if $p_i = 0$, and $m(w_{out}^{x_i}) = v_1^{x_i}$ if and only if $p_i > 0$, in other words x_i is assigned *true* if and only if $p_i > 0$. Since containment holds for all values of p_i (including 0), this means that $Q_1 \subseteq Q_2$ if and only if every truth assignment to the x_is has an extension to the y_js that satisfies all clauses of ϕ (or, equivalently, if and only if ϕ is valid). •

As we pointed out in figure 1, the Π_2^p lower bound for containment of $(*, \_, l^*, |)$-queries follows (independently) from three sources: the two lower bounds in the previous theorem and the one in remark 3.1.

W-queries. The following result shows that a more liberal nesting of regular path expressions withing the Kleene star is problematic in terms of complexity of containment. If we allow concatenations of labels withing the Kleene star, we obtain the W-fragment, whose lower bound for containment is PSPACE (a proof sketch is in appendix C):

Theorem 5.2 *Containment of W-queries is PSPACE-hard.*

As pointed out in remark 3.2, a bit more nesting than that yields EXPSPACE-hardness!

6 Containment Under Dependencies

The $(*, |)$-fragment. This is where our technique of relational translation is most effective. First recall that by translating $|$ away, any union of $(*, |)$-queries is equivalent to a union of $(*)$-queries. Recall also that any set C of $(*, |)$-dependencies is translated into a set $\mathcal{T}(C)$ of DEDs. By definition, "the dependencies in C are full" means that the DEDs in $\mathcal{T}(C)$ are full.

Since the DEDs in Σ_* are all full, the fact that containment of unions of $(*, |)$-queries under full $(*, |)$-dependencies is decidable follows from theorem A.2 and the following result:

Theorem 6.1 *Let C be a set of full $(*, |)$-dependencies, and U_1, U_2 two unions of $(*, |)$-queries. Let the equivalent unions of $(*)$-queries be $\bigcup_{i=1}^{n} Q_i$, respectively $\bigcup_{j=1}^{m} Q'_j$. Then U_1 is contained in U_2 under C if and only if for every $1 \leq i \leq n$ there exists $1 \leq j \leq m$ such that $\mathcal{T}(Q_i)$ is contained in $\mathcal{T}(Q'_j)$ under $\Sigma_* \cup \mathcal{T}(C)$.*

The proof exploits the work we already did in section 4 and is omitted.

The $(*, \_)$-fragment. As stated, this problem is open. However, we have two variations of it, one decidable, the other one, surprisingly, not.

Variation 1: V-dependencies. Consider a subclass of full $(*, \_)$-depenencies, which disallows

- occurences of the wildcard $\_$ in the premise of the implication, and
- occurrences of $*$ in the conclusion of the implication (see formula (4)).

We call them V-dependencies.

Theorem 6.2 *Containment of unions of $(*, \_)$-queries under full V-dependencies is decidable.*

The proof is omitted. The decision procedure is basically the same as the one for deciding containment of unions of $(*, \_, l^*)$-queries without dependencies: consider only a finite union of star-free queries, and check containment chasing with Σ_* and (as only difference from that case) with the translation of the V-dependencies.

Variation 2: Attributed models. Suppose now that we restrict the full $(*, \_)$-dependencies even more, forcing their atoms to be star-free. We obtain precisely the full DEDs. But assume that we allow a special class of semistructured databases, in which the data graph can be "adorned" by attaching attributes to its nodes. More precisely, attributed models have schema $\mathcal{L}$-*Rel* $\cup$ $\mathcal{A}$), where $\mathcal{A}$ is a set of binary relations names, called *attributes*, who are disjoint from $\mathcal{L}$. The only difference between an attribute and a label is that the former is not included in the interpretation of $\_$, while the latter is [4].

Theorem 6.3 *Containment of a $(*, \_)$-query in a union of $(*, \_)$-queries under full DEDs, but over attributed models, is undecidable.*

The proof is omitted, but very similar to that of theorem 6.4.

The (l^*)-fragment. Surprisingly, this problem is undecidable, despite the syntactic similarity of the (l^*) and $(*)$-fragments. We show a stronger undecidability result, which holds even if the dependencies are star-free, thus corresponding to purely relational full DEDs.

Theorem 6.4 *The containment of an l^*-query in a union of l^*-queries in the presence of full DEDs is undecidable.*

Proof: By reduction from the following undecidable problem: Given context-free grammar $G = (\Sigma, N, S, P)$ where Σ is the set of terminals (containing at least two symbols), N the nonterminals, $S \in N$ the start symbol, $P \subseteq N \times (\Sigma \cup N)^*$ the productions, and $L(G)$ the language generated by G, the question whether $L(G) = \Sigma^*$ is undecidable [17].

The reduction. Given context-free grammar $G = (\Sigma, N, S, P)$, we construct an instance of containment

$$Q \subseteq_D Q_S \cup Q_{cyc} \cup \bigcup_{\sigma_1 \neq \sigma_2 \in \Sigma} Q_{\sigma_1, \sigma_2}$$

as follows:

$$\begin{aligned}
Q() &\leftarrow \mathsf{b}\ H^*\ \mathsf{e} \quad (\mathsf{b}, \mathsf{e}\ \textit{constants}, H \notin \Sigma \cup N)\\
Q_S() &\leftarrow \mathsf{b}\ S\ \mathsf{e}\\
Q_{cyc}() &\leftarrow x\ H.H^*\ x\\
Q_{\sigma_1,\sigma_2}() &\leftarrow x\ \sigma_1\ y,\ x\ \sigma_2\ y \quad (\sigma_1, \sigma_2 \in \Sigma)
\end{aligned}$$

D consists of the following full, star-free DERPDs (DEDs):

$$\begin{aligned}
(fn) &\quad \forall x, y, z\ [x\ H\ y \wedge x\ H\ z\ \rightarrow y = z]\\
(inj) &\quad \forall x, y, z\ [y\ H\ x \wedge z\ H\ x\ \rightarrow y = z]\\
(symb) &\quad \forall x, y\ [x\ H\ y \rightarrow \bigvee_{\sigma \in \Sigma} x\ \sigma\ y]\\
(d_p) &\quad \forall x_0, \ldots, x_k\ [\bigwedge_{i=1}^{k} x_{i-1}\ M_i\ x_i \rightarrow x_0\ N\ x_k]\\
&\quad (\textit{for every}\ p = N \rightarrow M_1 \ldots M_k\ \in\ P)
\end{aligned}$$

[4]This model is similar in spirit to the XML data model and XPath specification [26], where attribute nodes are not reachable by navigation along the `child` axis.

All the queries are binary, so the following holds regardless of their construction:

$$
\begin{array}{c}
Q \not\subseteq_D Q_S \cup Q_{cyc} \cup \bigcup_{\sigma_1 \neq \sigma_2 \in \Sigma} Q_{\sigma_1,\sigma_2} \\
\Leftrightarrow \\
\exists I \quad [I \models D \wedge Q(I) \neq \emptyset \wedge Q_S(I) = \emptyset \wedge Q_{cyc}(I) = \emptyset \\
\wedge \bigwedge_{\sigma_1 \neq \sigma_2 \in \Sigma} Q_{\sigma_1,\sigma_2}(I) = \emptyset]
\end{array}
\tag{5}
$$

We claim that

$$
\begin{array}{c}
\exists I \quad [I \models D \wedge Q(I) \neq \emptyset \wedge Q_S(I) = \emptyset \wedge Q_{cyc}(I) = \emptyset \\
\wedge \bigwedge_{\sigma_1 \neq \sigma_2 \in \Sigma} Q_{\sigma_1,\sigma_2}(I) = \emptyset] \\
\Leftrightarrow \\
\exists (w \in \Sigma^*) \ w \notin L(G)
\end{array}
\tag{6}
$$

which, together with (5), implies that the construction is a reduction. The intuition behind the claim is that the instance I that acts as a counterexample for containment represents a word w which witnesses the non-containment of Σ^* in $L(G)$.

Proof of claim: $\Rightarrow$: Assuming the standard interpretation of H^*, $Q(I) \neq \emptyset$ implies that there exists a path of H-edges from b to e in I. $Q_{cyc}(I) = \emptyset$ implies that all paths of H-edges are simple (no cycles). $I \models (fn) \wedge (inj)$ implies that there is a unique (simple) path of H-edges from b to e which we call the H-chain.

$I \models (symb)$ says that every H-edge has in parallel with it a σ-edge for some symbol $\sigma \in \Sigma$, and it follows from $\bigwedge Q_{\sigma_1,\sigma_2}(I) = \emptyset$ that this edge is unique. The H-chain thus corresponds to a string w in Σ^*, of length equal to that of the H-chain. Each H-edge along the chain corresponds to a position in w.

We make the following subclaim: let x, y be the source, respectively target nodes of a subchain of the H-chain, and let u be the corresponding substring of w. Let N be any nonterminal such that there exists a derivation of u in G starting from N. Then there is an N-edge from x to y in I. The subclaim is shown by induction on the length of the derivation, and it uses the fact that $I \models \bigwedge_{p \in P}(d_p)$.

Together with $Q_S(I) = \emptyset$, the subclaim implies that there is no derivation of w in G starting from the start symbol S, in other words $w \notin L(G)$.

$\Leftarrow$: Starting from w, build the minimal model I consisting of (i) an H-chain of length $|w|$, (ii) the corresponding parallel edges spelling w, and (iii) for every subchain from node x to node y corresponding to the substring u of w, and every nonterminal N from which there is a derivation of u in G, add an N-edge from x to y. (i) implies $(I \models (fn) \wedge (inj)) \wedge Q_{cyc}(I) = \emptyset \wedge Q(I) \neq \emptyset$, (ii) ensures $(I \models (symb)) \wedge \bigwedge Q_{\sigma_1,\sigma_2}(I) = \emptyset$ and (iii) guarantees $I \models \bigwedge_{p \in P}(d_p)$. $w \notin L(G)$ and the minimality of the model enforce $Q_S(I) = \emptyset$. •

7 Rewriting With Views Under Dependenciess

[9] introduces the *chase&backchase (C&B)* algorithm for rewriting queries with views under dependencies [5] Due to space constraints we can only sketch here the idea and we omit proofs. The strategy of the C&B algorithm is to reduce the problem of rewriting with views to the problem of rewriting under dependencies. If V_i is a view name and QV_i the query that defines it, we capture V_i by writing a pair of inclusion dependencies that essentially say $V_i \subseteq QV_i$ and $QV_i \subseteq V_i$. Denote the set of all such pairs of dependencies with D_V and let us also assume that we rewrite under an additional set D of dependencies.

The C&B algorithm on a query Q has two phases. First the **chase** of Q with $D \cup D_V$. The dependencies in D_V that apply are full, so if those in D are full too, the chase will terminate, with a query we call the **universal plan** UP because it explicitly mentions all views that can be used to answer Q.

The second phase is the **backchase** which considers all *subqueries* of the universal plan UP (subsets of its conjuncts, mentioning all distinguished variables). The output of the algorithm is the set of those subqueries equivalent to Q for whom the removal of any conjunct compromises this equivalence. We call such queries *minimal* rewritings of Q. [6] The subqueries of the universal plan are tested for equivalence to Q again by chasing.

The (*)-fragment. The C&B algorithm applies almost directly here. We should point out that the views may not be binary relations and therefore the rewritings we obtain will not correspond to pure $(*)$-queries, but rather may contain relational atoms with the view names. We have the following *completeness* result for the algorithm:

Theorem 7.1 *Let Q be a $(*)$-query, V be a set of $(*)$-views and D be a set of full $(*)$-dependencies. Let $E = \Sigma_* \cup \mathcal{T}(D) \cup \mathcal{T}(D_V)$ and let $UP = chase_E(\mathcal{T}(Q))$ (the chase terminates).*

Then, for any minimal rewriting Q' of Q with V, $\mathcal{T}(Q')$ is a subquery of UP.

The $(*,|)$-fragment. In this case the query and views translate to unions of conjunctive queries and the $(*,|)$-dependencies translate to DEDs.

If we plug into the C&B method the extended chase with DEDs (see appendix A), we obtain a *union* of universal plans $U_1, \ldots, U_n$ afer the chase phase. Each U_i

[5] This is done in [9] for *path-conjunctive* queries and dependencies, which generalize the relational setting to a data model with dictionaries and complex values that also captures the OO setting.

[6] Under a *monotonic cost assumption* minimal queries are cheaper.

plan is backchased yielding a set of minimal subqueries S_i. Every entry in the cartesian product $S_1 \times \ldots \times S_n$ corresponds to a set of queries whose union is a rewriting of $\mathcal{T}(Q)$. We call this extension of the C&B algorithm the *disjunctive C&B* algorithm.

We say that a union of queries is *reduced* if all members are minimal and none of them is contained in another member. The following result implies that the disjunctive C&B algorithm is *complete* for the $(*, |)$-fragment.

Theorem 7.2 *Given a $(*, |)$-query Q, and one of its reduced rewritings $Q' = Q'_1 \cup \ldots \cup Q'_m$, for every $1 \leq j \leq m$, there is some $1 \leq i \leq n$ such that $\mathcal{T}(Q'_j)$ is a subquery of U_i.*

8 Conclusions

In this work, we propose a classification of conjunctive regular queries (CRPQ) and the associated constraint languages by the expressivity of the regular path expressions allowed in the conjuncts. We have studied the complexity of containment, with or without integrity constraints for the various fragments proposed. For certain fragments we have also studied the completeness of a specific kind of algorithm (chase & backchase) for rewriting with views under constraints.

A subtle observation that can be made based on the results we have obtained is that $\_$ is "more" than the union (the $|$ actually) of the labels that occur in a given context. Indeed, one might attempt to contradict the decidability for the $(*, |)$-fragment by reducing $(*, \_)$-queries and -dependencies to $(*, |)$-queries and -dependencies, using a translation like $\_ = l_1 | \ldots | l_n | f$ where $l_1, \ldots, l_n$ are all the labels mentioned in the queries and dependencies and f is a fresh label. This attempt fails because it does not capture the equivalence $* = \bigcup_{n \geq 0} \_^n$, which in turn is essential for the undecidability result. Of course, the correct translation an infinite disjunction of labels takes us out of the languages considered here.

We conclude that as a query language feature regular expressions are suprisingly "naughty", in the sense that adding supposedly inocuous operators to some fragments causes surprising increases in complexity. (For example, adding either $*$ or $\_$ to the fragment of conjunctive queries does not affect complexity of containment– still NP–, but adding *both* raises the complexity to Π_2^p.)

We are leaving some interesting problems open. One is the upper bound on containment in the W fragment. Another open problem is the decidability of containment under constraints in the $(*, \_)$-fragment. The reader can see that several open questions can be formulated about rewriting with views in certain fragments.

One interesting direction of future work is the application of our results to conjunctive queries over XML documents with XPath [26] expressions in their conjuncts. (We have submitted a couple of preliminary results on this to a separate conference.)

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufman, 1999.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[3] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4), 1984.

[4] D. Calvanese, G. De Giacomo, and M. Lenzerini. Queries and constraints on semi-structured data. In *CAiSE*, 1999.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Answering regular path queries using views. In *ICDE*, 2000.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

[7] Ashok Chandra and Philip Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[8] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A Query Language for XML. In *Proc. of 8th International WWWW Conference*, 1999.

[9] Alin Deutsch, Lucian Popa, and Val Tannen. Physical Data Independence, Constraints and Optimization with Universal Plans. In *VLDB*,1999.

[10] Mary F. Fernandez, Daniela Florescu, Jaewoo Kang, Alon Y. Levy, and Dan Suciu. Strudel: A web-site management system. In *SIGMOD*, 1997.

[11] Daniela Florescu, Alon Y. Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *PODS*, 1998.

[12] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64, 1999.

[13] Gösta Grahne. Dependency satisfaction in databases with incomplete information. In *VLDB*, 1984.

[14] Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT*, 1999.

[15] Gösta Grahne and Alex Thomo. An optimization technique for answering regular path queries. In *WebDB*, 2000.

[16] Gösta Grahne and Alex Thomo. Algebraic rewritings for optimizing regular path queries. *ICDT*, 2001.

[17] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, 1979.

[18] Frank Neven and Thomas Schwentick. Query automata. In *PODS*, 1999.

[19] Frank Neven and Thomas Schwentick. Expressive and efficient pattern languages for tree-structured data. In *PODS*, 2000.

[20] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[21] Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A Chase Too Far? In *SIGMOD*, May 2000.

[22] Lucian Popa and Val Tannen. An equational chase for path-conjunctive queries, constraints, and views. In *ICDT*, 1999.

[23] Yehoushua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27, 1980.

[24] P. van Emde Boas. The convenience of tilings. In A. Sorbi(Ed.) *Complexity, Logic, and Recursion Theory*, pp. 331–363, 2000.

[25] W3C. Extensible Markup Language (XML) 1.0. W3C Recommendation 10-February-1998. Available from `http://www.w3.org/TR/1998/REC-xml-19980210`.

[26] W3C. XML Path Language (XPath) 1.0. W3C Recommendation 16 November 1999. Available from `http://www.w3.org/TR/xpath`.

[27] Mihalis Yannakakis and Christos Papadimitriou. Algebraic dependencies. *JCSS*, 25, 1982.

A Chasing with Disjunctive Embedded Dependencies

Recall the *disjunctive embedded dependencies (DEDs)* defined in section 2 (see formula (4)). *Full* DEDs are restrictions of general form (4) to disallow existentially quantified variables on the right hand side of the implication ($k_i = 0$ for every i). Satisfaction of DEDs is defined as expected.

Chase with DEDs. Let d be a DED of general form 4, Q be a conjunctive query and let h be a homomorphism from ϕ into Q. We say that the *chase step* of Q with d using h is *applicable*, if h allows no extension which is a homomorphism from $\phi \wedge \psi_i$ into Q for any $1 \leq i \leq l$. In this case, the *result* of applying this chase step is the union of queries $\bigcup_{i=1}^{l} Q_i$, where each Q_i is defined as $Q \wedge \psi_i(h(x_1), \ldots, h(x_n), f_1, \ldots, f_{k_i})$, where the f_j's are fresh variables.

For example, chasing $Q(x,y) \leftarrow a(x,y)$ with $\forall u \forall v\ [a(u,v) \rightarrow b(u,v) \vee c(u,v)]$ results in $Q_b \cup Q_c$ with $Q_b(x,y) \leftarrow a(x,y), b(x,y)$ and $Q_c(x,y) \leftarrow a(x,y), c(x,y)$.

If we continue applying chase steps to each Q_i (with DEDs from a set D), we build a *chase tree* rooted at Q, whose subtrees are the chase trees rooted at the Q_i's. The leaves of the chase tree are conjunctive queries to which no chase step with any DED from D applies. In general, the chase may diverge, but when it terminates, we call its *result* the set of leaves of the chase tree, denoted $chase_D(Q)$. If we fix the dependencies and the schema arities, the finite tree produced by a terminating chase has depth polynomial in the size of Q.

Theorem A.1 *For a fixed set of full DEDs D, given conjunctive query Q, the chase tree obtained by chasing Q with D is* finite. *Moreover, for all instances satisfying the DEDs in D, Q is equivalent to $\bigcup_{j=1}^{m} Q_m$, where $Q_1, \ldots, Q_m$ are the leaves of the chase tree.*

Theorem A.2 *Given conjunctive queries Q_1, Q_2 and the set D of full DEDs, Q_1 is contained in Q_2 under D if and only if for every leaf QL of the chase tree obtained by chasing Q with D, there is a containment mapping from Q_2 into QL.*

This extension was already suggested by Beeri and Vardi in [3], and for a restricted kind of disjunctive dependencies, it was performed by Grahne [13], and by Grahne and Mendelzon [14], who generalize chase -sequences to chase-trees. Containment of unions is reduced to containment of single queries:

Theorem A.3 *Let D be a set of full DEDs and let $U_1 = \bigcup_{i=1}^{n} Q_i$ and $U_2 = \bigcup_{j=1}^{m} Q'_j$ be two unions of conjunctive queries. Then, U_1 is contained in U_2 under D if and only if for every $1 \leq i \leq n$ there exists a $1 \leq j \leq m$ such that Q_i is contained in Q'_j under D.*

B Upper Bound Proofs

B.1 Proof of proposition 4.5

Proof sketch: The "if" direction is trivial, because of the one-to-one correspondence from semistructured databases to Σ_{l^*}-instances. For "only if", regard $chase_{\Sigma_* \cup \Sigma_{l^*}}(\mathcal{T}(Q_1))$ as a $\Sigma_* \cup \Sigma_{l^*}$-instance whose nodes are $\mathcal{T}(Q_1)$'s variables and whose conjuncts $a(x,y)$ determine that (x,y) belongs to the interpretation of label a. The crucial observation is that this instance is minimal, in the sense that pairs of nodes not connected by a path of l-edges are ruled out from l^*, for any label $l \in \mathcal{L} \cup \{\_\}$. This follows from the absence of l^* from $\mathcal{T}(Q_1)$ to begin with, and from the fact that the chase with $\Sigma_* \cup \Sigma_{l^*}$ only introduces l^* between variables that are connected by l-edges. But minimal $\Sigma_* \cup \Sigma_{l^*}$-instances are in one-to-one correspondence with semistructured databases, which together with the hypothesis implies the existence of a containment mapping from $\mathcal{T}(Q_2)$ into $chase_{\Sigma_* \cup \Sigma_{l^*}}(\mathcal{T}(Q_1))$. •

B.2 Proof of proposition 4.6

The "only if" part is trivial. For the "if" part, observe that each $\mathcal{T}(Q_1(l_1^{p_1}, \ldots, l_n^{p_n}))$ can be regarded as a semistructured database, whose nodes are the variables. This database contains a simple path $simple_i$ of l_i-edges and length p_i for every $1 \leq i \leq n$. The only difference between these databases lies in the lengths of the simple paths $simple_i$.

Denote with B the database corresponding to $\mathcal{T}(Q_1(l_1^k, \ldots, l_n^k))$. It is easy to see that the tuple o consisting of Q_1's distinguished variables is in the answer of Q_1 when applied to B. By hypothesis and by definition of the answer of a CRPQ, there exists a mapping m from the variables of Q_2 to the nodes of B such that the distinguished variables of Q_2 map to o and every conjunct $x\ R\ y$ of Q_2 is satisfied by $(m(x), m(y))$. Recall that this means that there exists a path $path_R$ from $m(x)$ to $m(y)$, whose labels spell out a word from $L(R)$.

We claim that m determines a mapping from Q_2 into the databases corresponding to $\mathcal{T}(Q_1(l_1^{p_1}, \ldots, l_n^{p_n}))$ (for $p_1, \ldots, p_n > k$), such that every conjunct $x\ R\ y$ of Q_2 is satisfied by $(m(x), m(y))$. This implies the proposition, because for every database I, each tuple t in $\mathcal{T}(Q_1(l_1^{p_1}, \ldots, l_n^{p_n}))(I)$ is the image of Q_1's distinguished variables under some homomorphism h, and hence t is the image of Q_2's distinguished variables under $h \circ m$. It is easy to check that $h \circ m$ satisfies the conjuncts of Q_2. Therefore t belongs also to the answer of Q_2. In order to prove our claim, we distinguish two cases.

Case 1: If every conjunct $x\ R\ y$ in Q_2 is satisfied by a path $path_R$ from $m(x)$ to $m(y)$ which has no edge in common with $simple_i$ for all $1 \leq i \leq n$, then m is a mapping into all databases $\mathcal{T}(Q_1(l_1^{p_1}, \ldots, l_n^{p_n}))$, where m satisfies all of Q_2's conjuncts.

Case 2. Assume now that, given m from Q_2 to B, Q_2 has a conjunct $x\ R\ y$ such that for any choice of $path_R$ from $m(x)$ to $m(y)$, there is a $1 \leq j \leq n$ for which $simple_j$ has at least one edge in common with $path_R$. Let $s_1, \ldots, s_\alpha$ be the list of nodes from $simple_j$ (ordered by distance from $simple_j$'s source) which are either its endpoints or are in the image of m (by assumption, $\alpha \geq 3$). For every $1 \leq r \leq \alpha - 1$, define the *rth interval* to be the section of $simple_j$ starting at s_r and ending at s_{r+1}.

We make the *subclaim* that there exists a $1 \leq q \leq \alpha - 1$ such that for all conjuncts $x\ R\ y$ in Q_2, either there is a path $path_R$ from $m(x)$ to $m(y)$ satisfying the conjunct and not including the qth interval, or R contains l_j^* as a subexpression. Indeed, assume the contrary towards a contradiction: then every interval is included in a path that satisfies some conjunct $x\ R\ y$ where R is star-free. This means that every one of $simple_j$'s edges corresponds to at least one occurrence of the label l_j in Q_2, whence it follows that p_j, the length of $simple_j$, is less than or equal to $sfs(Q_2) = k - 1$. This contradicts the fact that for B, $p_j = k$, which proves our subclaim.

By the subclaim, we can extend the qth interval with arbitrarily many l_j-edges without compromising m's property of satisfying every conjunct in Q_2. This proves our claim and concludes the proof of the proposition. •

B.3 Proof of theorem 4.8

We prove equivalently that non-containment is in Σ_2^p, in fact we show that it is decidable by an NP machine with an NP oracle. By proposition 4.6, it is enough if the machine exhibits a star-free query in the finite union which is not contained in U_2. To this end, the machine guesses Q_1 in SF_1 and $Q_2 \in U_2$, guesses $0 \leq p_1, \ldots, p_n \leq sfs(Q_2)$, constructs $T = \mathcal{T}(Q_1(l_1^{p_1}, \ldots, l_n^{p_n}))$ (in PTIME) computes $chase_{\Sigma_{l^*}}(T)$ (in PTIME, because all dependencies in $\Sigma_* \cup \Sigma_{l^*}$ are full), asks the oracle if there is a containment mapping from $\mathcal{T}(Q_2)$ into the chase result, and answers "yes" if and only if the oracle answers "no". •

C Lower Bound Proofs

Proof of theorem 5.2.

Proof sketch: Along the lines of [6], we reduce containment to the PSPACE-complete "corridor tiling problem" (CTP) ([24]).

An instance of CTP is a set Δ of tile types, along with two binary relations V, H over Δ, which specify the vertical, resp. horizontal compatibility between tile types. We are also given two special tile types t_s, t_f, and an integer n. A solution to the CTP is a tiling of the corridor of width n in the plane using tiles of types in Δ, starting in the top left corner with a tile of type t_s, ending in the bottom right corner with a tile of type t_f, such that the compatibility relations are satisfied.

An instance of the CTP is reduced to an instance of containment of W-queries as follows:

$$\begin{aligned} AllTilings(x,y) &= x\ \ t_s \_^{n-1}(\_^{n})^* t_f \_^{n-1}\ \ y \\ IllegalTilings(x,y) &= x\ \ (\bigcup_{k=0}^{n-2} \bigcup_{(t_1,t_2) \notin H} (\_^{n})^* \_^{k} t_1 t_2 \_^{n-k-2} (\_^{n})^*) \\ &\quad \cup (\bigcup_{k=0}^{n-1} \bigcup_{(t_1,t_2) \notin V} (\_^{n})^* \_^{k} t_1 \_^{n-1} t_2 \_^{n-k-2} (\_^{n})^*)\ y \end{aligned}$$

We answer "yes" if and only if

$$AllTilings(x,y) \not\subseteq IllegalTilings(x,y)$$

.

D Rewriting with views: an example

Recalling the motivating example from section 1 and denoting with Div the relational translation $\mathcal{T}(Divulge)$, we have further translate as follows:

$$Div(v,w) \leftarrow *(\texttt{start}, u),\ name(u,v),\ tells(u,w)$$

and the inclusion dependencies capturing it are

$$\begin{aligned} (c_{Div})\ \ \forall u,v,w\ [\ \ & *(\texttt{start},u) \wedge name(u,v) \\ & \wedge tells(u,w)\ \rightarrow\ Div(v,w)\ ] \\ (b_{Div})\ \ \ \forall v,w\ [\ \ \ & Div(v,w)\ \rightarrow\ \exists u\ *(\texttt{start},u) \\ & \wedge name(u,v) \wedge tells(u,w)\ ] \end{aligned}$$

Furthermore, we have $TA = \mathcal{T}(A)$, $TA' = \mathcal{T}(A')$, tell= $\mathcal{T}(tellAll)$, where:

$$\begin{aligned} TA &\leftarrow *(\texttt{start}, x),\ name(x, z_1),\ John(z_1, y),\ *(x, z_2), \\ &\quad tells(z_2, z_3),\ name(z_3, n) \\ \text{tell} &= \forall x, y, w_1, w_2\ [\ *(x, w_1),\ tells(w_1, w_2), \\ &\quad *(w_2, y) \to\ tells(x, y)\ ] \\ TA' &\leftarrow Div(x, y),\ John(x, z),\ name(y, n)\ \bullet \end{aligned}$$

Phase 1: Chase. $\mathcal{T}(A)$ chases with (base$_{tells}$), (refl$_*$), then ($\mathcal{T}$(tellAll)) to:

$$\begin{aligned} A_1 \leftarrow\ & *(\texttt{start}, x),\ name(x, z_1),\ John(z_1, y),\ *(x, z_2), \\ & tells(z_2, z_3),\ N(z_2),\ N(z_3),\ *(z_3, z_3), \\ & tells(x, z_3),\ name(z_3, n) \end{aligned}$$

By bringing the conjunct $tells(x, z_3)$ into A_1, the chase step with (tell) enables a further chase step of A_1 with c_{Div}, whose effect is to explicitly "chase-in" the view Div, adding the goal $Div(z_1, z_3)$ to A_1. Continuing the chase until it terminates, we obtain the universal plan UP whose body is defined by

$$\begin{aligned} & *(\texttt{start}, x),\ name(x, z_1),\ John(z_1, y),\ *(x, z_2), \\ & tells(z_2, z_3),\ tells(x, z_3),\ name(z_3, n),\ Div(z_1, z_3), \\ & *(\texttt{start}, \texttt{start}),\ *(x, x),\ *(z_1, z_1),\ *(z_2, z_2), \\ & *(z_3, z_3),\ *(y, y),\ *(n, n), \\ & *(x, z_1),\ *(z_1, y),\ *(z_2, z_3),\ *(x, z_3),\ *(z_3, n), \\ & *(\texttt{start}, z_1),\ *(\texttt{start}, z_2),\ *(\texttt{start}, z_3),\ *(\texttt{start}, y), \\ & *(x, y),\ *(x, n),\ *(z_2, n), \\ & N(\texttt{start}),\ N(x),\ N(y),\ N(n),\ N(z_1),\ N(z_2),\ N(z_3) \end{aligned}$$

where the conjuncts in the third and fourth lines were introduced by chasing with (refl$_l$), those in lines 5 to 7 by chasing with (trans$_l$), and in line 8 by chasing with (base$_l$), for appropriate instantiations of l.

Phase 2: Backchase. Note that, up to renaming of variables, $\mathcal{T}(A')$ is a subquery of UP. It is easy to check that $\mathcal{T}(A')$ is minimal, and equivalent to $\mathcal{T}(A)$. This is done by chasing $\mathcal{T}(A)$ with (b_{Div}). We can straightforwardly retrieve A' from $\mathcal{T}(A')$, thus obtaining a rewriting of A using $Divulge$.