Query Containment for Conjunctive Queries With Regular Expressions

Daniela Florescu* Alon Levy* Dan Suciu*

Abstract

All query languages proposed for semistructured data share as common characteristic the ability to traverse arbitrary long path in the data in the form of regular path expressions. The expressive power of these languages lies in between that of the relational calculus, and that of query languages with recursion, like datalog. We consider the problems of query containment and query equivalence for a certain subset of the *StruQL* language implemented in the *Strudel* web-site management system, consisting of *conjunctive queries with regular expressions*. It was previously known that containment and equivalence are NP-complete for the conjunctive fragment of the relational calculus, and undeciable for datalog. We show that these problems are decidable for conjunctive queries with regular path expressions. Both are PSPACE hard: the complexity of our decision algorithm however is higher, leaving a gap between the lower and upper complexity bounds. For a restricted class of conjunctive queries with regular path exrepssions we show that containment and equivalence are NP-complete. This offers, to our knowledge, the first example of a query language with recursion in which containment and equivalence have the same complexity as that of conjunctive queries in the relational calculus.

1 Introduction

The management of *semistructured* data has recently received significant attention because of the need of several applications to model and query large volumes of irregular data [1, 5]. For example, researchers in biology store their data in structured files in various data exchange formats. Similarly, large volumes of online documentation, document collections and program libraries are available in structured files.

Several characteristics distinguish semistructured data from relational and object-oriented data. Unlike traditional data that fits a pre-existing and fixed schema, semistructured data is irregular: attributes may be missing, the type and cardinality of an attribute may not be known or may vary from object to another, and the set of attributes may not be known in advance. Furthermore, the schema of semistructured data, even if it exists, is often unknown in advance. Because of these characteristics, models of semistructured data have been shown to be very valuable for data integration [24, 1].

The focus of the research on semistructured data has been on formulating appropriate models for such data, and designing appropriate query languages (e.g., [10, 3, 6]). The data model that has been generally adopted is based on labeled directed graphs, where nodes correspond to objects, and the labels on the edges correspond to attributes. Although the query languages proposed for semistructured data are based on different paradigms, all of them share the following key feature. As a consequence of the lack of schema (or lack of knowledge about the schema), users need the ability to express queries navigating irregular or unpredictable structures. This is done by allowing the queries to include *regular path expressions* over the attributes, and express queries about the schema.

This paper considers the problem of query containment for a query language over semistructured data that contains the essential feature explained above. We consider the language STRUQL$_0$, a subset of the STRUQL language implemented in the STRUDEL web-site management system [13, 14]. The language STRUQL$_0$ allows expressing regular path expressions over attributes in a graph and permits *arc variables* that range over attribute names. Ignoring the restructuring capabilities of languages for querying semistructured data, STRUQL$_0$ is more expressive than UnQL [6][1], and is equivalent to a certain fragment of Lorel [3]. Considering

*AT&T Laboratories, Florham Park, NJ. {dana,levy,suciu}@research.att.com

[1] UnQL does not handle oid equalities.

the restructuring capabilities, the full STRUQL language is more expressive than both UnQL and Lorel: however, we do not discuss the restructuring aspects in this paper. Furthermore, STRUQL$_0$ is a subset of datalog with a limited yet interesting form of recursion. Importantly, the containment result for datalog do not yield any interesting results for STRUQL$_0$. In particular, STRUQL$_0$ identifies a subset of datalog for which containment is decidable.

Algorithms for query containment are important in several contexts. Originally, algorithms for containment have been developed in the context of query optimization [7, 25, 4]. For example, query containment can be used to find redundant subgoals in a query and to test whether two formulations of a query are equivalent. Also, query containment has been used to determine when queries are independent of updates to the database [21], rewriting queries using views [8, 19], and maintenance of integrity constraints [17]. More recently, query containment, applied in the context of rewriting queries using views, have been used as a tool in data integration [20, 16, 30].

An important motivation for our work is that solving the query containment problem, and its related query rewriting problem, is essential for supporting the separation between the logical view of the data and its storage. This particular usage of query rewriting algorithms has been pioneered in [29]. In that work, the storage pattern of data is abstracted by a query over the logical view of data. Hence, to translate a query expressed over the logical view of the data into a query execution plan tailored to the physical storage of the data, the query optimizer needs to solve the query rewriting problem which, in turn, reduces to a query containment problem [19]. In the context of semistructured data, the graph data model is often introduced *artificially* for modeling purposes, in order to support a wide range of data stored in existing formats. Hence, the separation between the logical view of the data and the physical view becomes even more important, and is crucial for query optimization [15].

We make the following contributions. First, we give a semantic criteria for STRUQL$_0$ query containment: we show that it suffices to check containment on only finitely many *canonical* databases, hence query containment is decidable: the resulting algorithms has a high complexity however. Second we give a syntactic criteria for query containment, based on a notion of *query mappings*, which extends containment mappings for conjunctive queries. This results in a second algorithm, with exponential space complexity. Third, we consider a certain fragment of STRUQL$_0$, obtained by imposing restrictions on the regular path expressions. We show that query containment for this fragment of STRUQL$_0$ is NP complete. This is a surprising result, since it offers the first example (to the best of our knowledge) of a query language with recursion for which containment checking is no harder than for conjunctive queries.

Previous work on query containment has considered queries in the relational algebra [7, 25, 4] and datalog [27, 26, 9]. Several works have considered the extension of containment algorithms for queries involving order [18, 31, 21, 32, 17], and queries over complex objects [22]. In [11], Courcelle characterized a large class of decidable problems involving Datalog rules: our results however do not fit in that framework. Except for the undecidability result for query containment for datalog [27] none of these works has considered query languages with regular path expressions or arc variables.

This paper is organized as follows. We describe the data model and query language in Section 2, and define the query containment problem. We give the semantic criteria equivalent to query containment in Section 3, and show that containment is decidable. Using that, we give the syntactic criteria in Section 4, and prove that it is equivalent to query containment: this results in an exponential space algorithm for checking containment. In Section 5 we describe a fragment of our query language for which the containment is NP-complete, then conclude in Section 6.

2 Preliminaries

In this section we describe our data model and query language, and then define the problems considered in the paper.

Data Model and Query Language We model a database of semistructured data as a labeled directed graph. Nodes in the graph correspond to objects in the domain, and the labels on the edges correspond to attribute names. Intuitively, this model can be viewed as an object-oriented data model without type constraints.

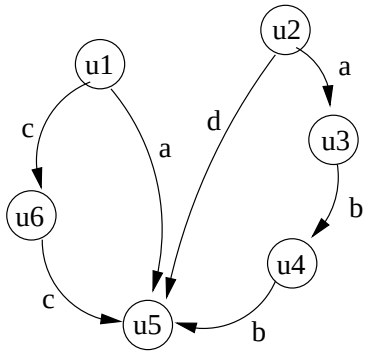


Figure 1: An example of a graph database.

Formally, we assume we have a universe of constants $\mathcal{D}$, which is disjoint from a universe of object identifiers $\mathcal{I}$. A database DB is a pair (V, E), where $V \subseteq \mathcal{I}$ is a set of nodes and $E \subseteq V \times \mathcal{D} \times V$ is a set of directed edges, labeled with constants from $\mathcal{D}$. Figure 1 contains an example of a graph database.

In this paper we consider a subset STRUQL$_0$ of the STRUQL language [14]. Informally, we consider conjunctive queries with two distinct features. First, some of the conjuncts may describe regular path expressions over the edge labels in the graph. Second, some of the variables are *arc variables,* and range over the labels of edges in the graph.

Formally, in our discussion we distinguish arc variables from normal variables. We denote arc variables by the letter L, possibly with various subscripts. Other variables are denoted by capital letters from the end of the alphabet. A regular path expression is defined by the following grammar (R, R_1 and R_2 denote regular path expressions, and a denotes a constant in $\mathcal{D}$):

$$R := \epsilon \mid a \mid \_ \mid L \mid (R_1.R_2) \mid (R_1 \mid R_2) \mid R*.$$

We abbreviate $R.(R*)$ with $R+$. A query in STRUQL$_0$ is an expression of the form

$$Q : q(\bar{X}) : -Y_1 R_1 Z_1, \ldots, Y_n R_n Z_n.$$

Here $nvar(Q) \stackrel{\text{def}}{=} \{Y_1, \ldots, Y_n, Z_1, \ldots, Z_n\}$ are the query's node variables (they need not be distinct), and $R_1, \ldots, R_n$ are regular path expressions. We denote with $avar(Q)$ the set of arc variables occurring in $R_1, \ldots, R_n$, and with $var(Q) \stackrel{\text{def}}{=} nvar(Q) \cup avar(Q)$ all the variables in Q. $\bar{X} \subseteq nvar(Q)$ are Q's head variables. Finally, $atoms(Q)$ denotes the set of all constants occurring in $R_1, \ldots, R_n$.

The semantics of such a query is an extension of the semantics of conjunctive queries. Define a *substitution* to be a function $\varphi : var(Q) \to \mathcal{I} \cup \mathcal{D}$, s.t. φ maps node variables to $\mathcal{I}$ and arc variables to $\mathcal{D}$. Consider all the possible substitutions, φ, such that φ satisfies the following constraint:

- for every i, $1 \leq i \leq n$, there is a path P in the database between $\varphi(X_i)$ and $\varphi(Y_i)$, such that P satisfies the regular path expression $\varphi(R_i)$, where $\varphi(R_i)$ denotes the application of the substitution of φ to the regular path expression R_i (i.e., replacing the arc variables with their value under φ: hence $\varphi(R_i)$ is a regular expression without arc variables).

Each substitution φ defines a tuple in a relation R_Q, whose arity is the number of variables in Q. The answer to Q is the projection of R_Q on the variables in $\bar{X}$. We denote the result of applying Q to a database DB by $Q(DB)$.

Example 2.1 Consider the following query:
$Q_1 : q1(X, Z) : -XL+Z, YaZ, X(a+ \mid (a.b*))Z.$

The relation $R_Q(X, Y, Z, L)$ has arity 4, and contains 4 tuples, when Q is applied to the database in Figure 1: $\{(u_1, u_1, u_5, c), (u_1, u_1, u_5, a), (u_2, u_1, u_5, d), (u_2, u_2, u_3, a)\}$: $Q(DB)$ is it's projection on X and Z, $\{(u_1, u_5), (u_2, u_5), (u_2, u_3)\}$.

The full STRUQL language contains several additional features not discussed here. First, the STRUQL allows some of the conjuncts to be arbitrary predicates or membership conditions in a predefined set of collection names. Since the containment problem in the presence of these additional features is a straightforward extension of the algorithms we present, we omit them from the discussion. Finally, the view definition mechanism of STRUQL allows definition of new graphs using the **Create, Link** and **Collect** clauses. In this paper we do not consider the restructuring capabilities of STRUQL.

It is important to emphasize that STRUQL$_0$ has two features essential for querying semistructured data, and in turn introduce the novel difficulties to the problems we consider. These features are the presence of regular path expressions in the query and the ability to query the schema via the arc variables. The queries we consider in this paper can be translated into datalog. The problem of query containment for datalog is known to be undecidable [27], and none of the restricted cases for containment that have been considered in the literature apply to STRUQL$_0$. However, STRUQL$_0$ is an interesting subset of datalog with a limited form of recursion for which containment is decidable.

Containment The problem of query containment is defined as follows:

Definition 2.2 *A query Q_1 is* contained *in a query Q_2, written $Q_1 \subseteq Q_2$, if for all databases DB, $Q_1(DB) \subseteq Q_2(DB)$. The queries Q_1 and Q_2 are* equivalent, *written $Q_1 \equiv Q_2$, if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.*

Example 2.3 The query Q_2 below is contained in Q_1:
$Q_2 : q2(X, Z) : -Xa^+Z$.

For example, on the database in Figure 1, Q_2 returns $\{(u_1, u_5), (u_2, u_3)\}$.

3 Query Containment and Canonical Databases

We describe in this section a semantic condition for query containment. Namely we show that $Q \subseteq Q'$ iff $Q(DB) \subseteq Q'(DB)$ for all *canonical* databases, DB, that is, databases with a certain shape, imposed by Q.

A *canonical database* DB for Q is easy to explain intuitively. Namely DB will have one distinguished node for each variable in Q — called a *bifurcation node* — and one distinguished path for each conjunct in Q — called an *internal path*, and its nodes *internal nodes*. The internal path associated to the conjunct YRZ in DB must be between the bifurcation nodes associated to Y and Z, and its labels must satisfy the regular expression R. We give the formal definition next. Let Q be a STRUQL$_0$ query, and recall that $avar(Q) = \{L_1, \ldots, L_p\}$ denotes the set of all arc variables in Q. We will assume that $avar(Q)$ is disjoint from $\mathcal{D}$, our universe of constants.

Definition 3.1 *A* canonical database *for Q is a pair (DB, ξ), where $DB = (V, E)$ is a graph databases with constants in $\mathcal{D}' \stackrel{\text{def}}{=} \mathcal{D} \cup avar(Q)$, and $\xi : var(Q) \to \mathcal{I} \cup \mathcal{D}'$ is a substitution, such that the conditions below hold. DB's nodes are partitioned into* bifurcation nodes, *denoted V_B, and* internal nodes, *denoted V_I: $V = V_B \cup V_I$. We call a path $b \to u_1 \to u_2 \to \ldots \to u_n \to b'$ with b, b' bifurcation nodes and $u_1, \ldots, u_n$ internal nodes an* internal path. *The conditions are:*

1. *Each internal node belongs to an internal path, and has exactly one incoming edge and one outgoing edge.*
2. *The substitution ξ (1) maps all node variables in Q to bifurcation nodes, s.t. the restriction of ξ to $nvar(Q) \to V_B$ is surjective, and (2) maps each arc variable $L \in avar(Q)$ to itself.*
3. *There exists a 1 to 1 correspondence between the conjuncts in Q and the internal paths in DB, s.t. for every conjunct $Y_iR_iZ_i$, the sequence of labels on the corresponding internal path satisfies the regular expression $\xi(R_i)$.*

Note that distinct node variables may correspond to the same bifurcation node, i.e. we may have $\xi(Y) = \xi(Z)$. In that case the internal path from $\xi(Y)$ to $\xi(Z)$ associated to some conjunct YRZ may be empty (then, of course, $\epsilon \in R$). If at least one of the conjuncts in Q has a Kleene closure $*$, then there are infinitely many canonical databases. This is because the internal path corresponding to that conjunct can

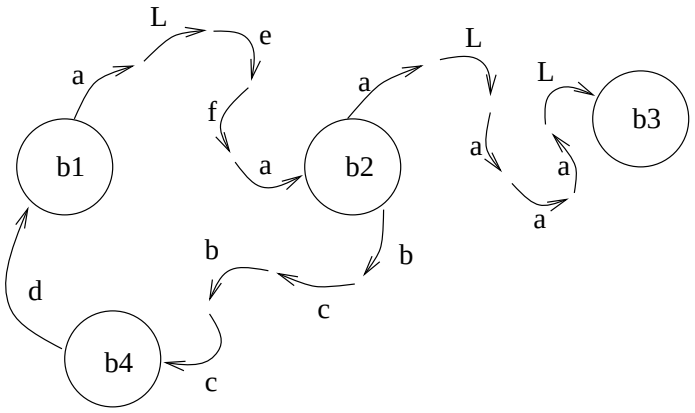


Figure 2: A canonical database. The bifurcation nodes are encircled.

be made arbitrarily large. Compare this to the classical case of conjunctive queries[2], where each query determines a unique, canonical database, given by its body. As an example, a canonical database for the query $Q : q(X_1, X_2) : -X_1(a.L.(\_*))X_2, X_2(b.c)*Y, X_2(a|L)*Z, Y(c|d)X_1$ is illustrated in Figure 2. Note that $\_$ can be replaced by any atomic value in $\mathcal{D} \cup \mathit{avar}(Q)$, with or without repetitions. In this case we still have inifinitely many canonical databases even if Q doesn't have a Kleen closure.

Given a query Q with head variables $X_1, \ldots, X_k$, and a canonical databases (DB, ξ) for Q, we call $(\xi_(X_1), \ldots, \xi(X_k))$ the *canonical tuple* for DB. Obviously, the canonical tuple belongs to $Q(DB)$. As for the case of conjunctive queries, in order to check containment of two queries $Q \subseteq Q'$ it suffices to test whether the canonical tuple is in the answer of Q': this time however we have to do that for *all* canonical databases.

Proposition 3.2 *Given two queries Q, Q', we have $Q \subseteq Q'$ iff for any canonical database (DB, ξ) for Q, it's canonical tuple is in the answer of Q'.*

Proof: For the "only if" direction we note that STRUQL$_0$ queries are *generic*, hence if $Q \subseteq Q'$ for databases over the universe $\mathcal{D}$, then $Q \subseteq Q'$ for databases over universe $\mathcal{D}' \stackrel{\text{def}}{=} \mathcal{D} \cup \mathit{avar}(Q)$: we defer the rather straightforward details to the final version of the paper. Consider now the "if" implication. Assume the contrary, i.e. $Q \not\subseteq Q'$. Then there exists some database DB and some tuple of nodes and/or arc values $\bar{u} = (u_1, \ldots, u_k)$ in DB such that $\bar{u} \in Q(DB)$, but $\bar{u} \notin Q'(DB)$. We construct from that a canonical database which contradicts the assumption. Since $\bar{u} \in Q(DB)$, there exists a substitution φ, such that for each conjunct YRZ in Q there exists some path from $\varphi(Y)$ to $\varphi(Z)$ in DB whose sequence of labels satisfies R. We start the construction of the canonical database (DB_0, ξ) by picking the bifurcation nodes to be the image under φ of the node variables in Q: that is $V_B \stackrel{\text{def}}{=} \{\varphi(X) \mid X \in \mathit{nvar}(Q)\}$, and we will define $\xi(X) \stackrel{\text{def}}{=} \varphi(X)$ for every $X \in \mathit{nvar}(Q)$. Next consider each conjunct YRZ in Q. There exists a path in DB from $\varphi(Y)$ to $\varphi(Z)$ whose labels match the regular expression R. This path is not necessarily simple (i.e. it may have loops), and may go through nodes which have been designated bifurcation node: we introduce fresh internal node in DB_0 for every occurence of a node in that path, thus creating a simple path from $\varphi(Y)$ to $\varphi(Z)$ with the same labels. We will make this the internal path corresponding to the conjunct YRZ, but first we replace some of its labels, as follows. Let A be some nondeterministic automaton equivalent to R, where the arc variables $L_1, L_2, \ldots$ are viewed as constants. By definition, the sequence of labels on the internal path from $\xi(Y)$ to $\xi(Z)$ is accepted by $\varphi(A)$. We replace each label a causing a transition in $\varphi(A)$ corresponding to some arc variable $L \in \mathit{avar}(DB)$ with L: hence, the resulting path has a sequence of labels from $\mathcal{D}' \stackrel{\text{def}}{=} \mathcal{D} \cup \mathit{avar}(DB)$ which is accepted by A. Obviously, the resulting DB_0 is a canonical database. Moreover, we have a graph morphism $\psi : DB_0 \to DB$, sending bifurcation nodes to themselves, internal nodes back to their originating nodes, and each arc variable L to $\varphi(L)$. Now we use the assumption on Q', and argue that the canonical tuple in DB_0 must be in $Q'(DB)$. This gives us a substitution φ' from Q' to

[2] A *conjunctive query* is a First Order Logic formula which is conjunction of positive atomic literals, preceeded by some existentail quantifier. See [2].

DB_0. We compose it with $\psi : DB_0 \to DB$, and get a substitution $\psi \circ \varphi'$ from Q' to DB, which implies that $\bar{u}$ is in the answer of Q' too: this contradicts our assumption. □

Since in general there are infinitely many canonical databases, this does not give us a decision procedure for testing containment. The main result in this section consists in showing that it suffices to check only those canonical databases whose internal paths are of lengths which are bounded by some number depending only on Q and Q'. From that we derive a decision procedure.

Theorem 3.3 *Let Q, Q' be two queries. Then there exists a number N, which depends only on Q and Q' s.t. $Q \subseteq Q'$ iff for every canonical database (DB, ξ) for Q, whose internal paths are of length $\leq N$, its canonical tuple is in $Q'(DB)$.*

The proof is given in the Appendix. Note that this still does not imply decidability, because there are still infinitely many canonical databases with bounded length. For example, consider $Q : q(X, Y) : -X \_ Y$. The canonical databases for Q are all databases of the form $\xi(X) \xrightarrow{a} \xi(Y)$ with $a \in \mathcal{D}$. However, we can prove that it suffices to restrict to those having constants from a set of $n \times N$ atomic values, where n is the number of conjuncts in Q. More precisely, let $D_0 \subseteq \mathcal{D}$ be any set of cardinality $n \times N$ (this is the maximum number of atomic constants in the databases in Theorem 3.3), disjoint from $atoms(Q)$, $atoms(Q')$, and let $D_{Q,Q'} \stackrel{\text{def}}{=} atoms(Q) \cup atoms(Q') \cup D_0$. We prove in the full version of the paper that it suffices to check only the canonical databases with constants in $D_{Q,Q'} \cup avar(Q)$. Hence, we have:

Corollary 3.4 *Query containment for* StruQL$_0$ *is decidable.*

The complexity of the algorithm following from the proof is high: triple exponential space. We will describe an exponential space algorithm in the next section.

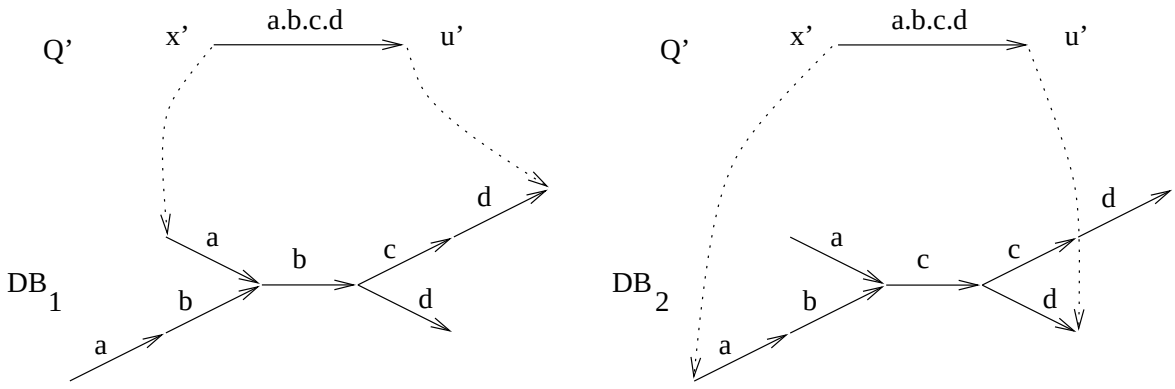
4 Query Containment and Query Mappings

We give in this section a syntactic criteria for query containment, similar in spirit to query mappings for conjunctive queries. This is an alternative to the semantic one of the previous section. Before we proceed, we note two differences from the classical case of conjunctive queries. First, in the case of conjunctive queries, there exists a one-to-one correspondence between the syntactic components of Q (its variables and conjucts) and the items in the canonical database DB of Q: thus, a substitution $\varphi : var(Q') \to DB$ immediately yields a containment mapping. In our setting, the syntactic components of Q are its variables, conjuncts, and automata states, and the correspondence to the nodes of a canonical database is one-to-many: thus, a substitution does not yield immediately a query mapping. Second, in our setting we have several canonical databases for a query Q. Since they all have the same shape, one may hope that they will all induce the same query mapping. But these hopes are ruined by the example in Figure 3, showing two boolean queries (i.e. without output variables) where Q has exactly two canonical databases, DB_1 and DB_2, and the two substitutions from Q' to DB_1 and DB_2 have totally different shapes. Thus, we are forced to consider *sets* of query mappings: we will show that $Q \subseteq Q'$ iff a certain condition holds on *all* query mappings from Q' to Q.

We start with some definitions.

Definition 4.1 *Let Q be a query with n conjuncts, $Q : q(\bar{X}) : -Y_1 R_1 Z_1, \ldots, Y_n R_n Z_n$, and let $nvar(Q) = \{Y_1, Z_1, \ldots, Y_n, Z_n\}$ be its set of node variables. We fix some nondeterministic automaton A_i for each regular expression R_i. We define a* point *in Q to be (1) either a node variable, or (2) some automata/state pair, i.e. (A_i, s), with s a state in A_i: we call the first kind a* variable-point*, the second an* automaton-point*. We denote with $points(Q)$ the set of points in Q.*

The intuition behind this definition is that any bifurcation node in a canonical database corresponds to variable-point, and any internal node to an automaton-point. The correspondence is many-to-one however, since several internal nodes may correspond to the same automaton-point.



$Q :- xay, x_1(a.b)y, y(b|c)z, z(c.d)u, zdu_1$
$Q' :- x'(a.b.c.d)u'$

Figure 3: Two substitutions with different shapes.

Definition 4.2 *Given a query Q, a* path of points *is a sequence $p_1, \ldots, p_n$, $n \geq 2$, s.t. (1) $p_2, \ldots, p_{n-1}$ are all variable-points (while p_1, p_n may be either variable- or automaton-points), and (2) any two adjacent points p_j, p_{j+1} are "connected" in Q, in the following way:*

- *If both p_j, p_{j+1} are variable points, then there exists a conjuct $Y_i R_i Z_i$ in Q, with $p_j = Y_i$, $p_{j+1} = Z_i$.*
- *If p_1 is an automaton-point (A_i, s) and p_2 is a variable point, then there exists a conjunct $Y_i R_i Z_i$ in Q, s.t. A_i is the automaton associated to R_i, and $p_2 = Z_i$.*
- *Similarly, if p_n is an automaton-point (A_i, s) and p_{n-1} is a variable point, then there exists a conjunct $Y_i R_i Z_i$ in Q, s.t. A_i is the automaton associated to R_i, and $p_{n-1} = Y_i$.*
- *Finally, if $n = 2$ and both p_1, p_2 are automaton-points, then they refer to the same automaton.*

Again, the intuition comes from canonical databases. There, a path can be fully specified by its endpoints (which can be either bifurcation nodes or internal nodes), and by its intermediate bifurcation nodes: that is, all intermediate internal nodes are redundant. Formally, let (DB, ξ) be a canonical database for Q. Since every internal node u is on a unique simple path corresponding to some conjuct $Y_i R_i Z_i$, and whose sequence of labels is accepted by A_i, we can associate to each internal node u a point (A_i, s). Given any path $U = (u_1, u_2, \ldots, u_m)$ in DB, we say that it *corresponds* to a path of points $p_1, \ldots, p_n$ in Q iff (1) $\xi(p_2), \ldots, \xi(p_{n-1})$ are precisely the bifurcation nodes on the path U, and (2) p_1 is either a variable-point and $\xi(p_1) = u_1$, or it is an automaton-point which is associated to u_1, (3) similarly for p_n and u_m.

Note that, when $n = 2$, the definition allows a path of points to be of the form $(A_i, s), (A_i, s')$, even if there is no path from s to s' in the automaton A_i.

Consider now some other query Q', and fix some nondeterministic automaton A'_i for each regular expression R'_i in Q'. We define below a query mapping, $f : Q' \to Q$. Recall that we defined at the end of Section 3 a finite set of constants $D_{Q,Q'} \subseteq \mathcal{D}$ and showed that it suffices to consider only those canonical databases for Q whose edges are labeled with constants in $D_{Q,Q'} \cup avar(Q)$. Let $\bar{X}$, $\bar{X}'$ be the head variables in Q, Q' respectively.

Definition 4.3 *A* query mapping *$f : Q' \to Q$ consists of the following.*

1. *A function $f : var(Q') \to points(Q) \cup (D_{Q,Q'} \cup avar(Q))$, sending $nvar(Q')$ to $points(Q)$ and $avar(Q)$ to $D_{Q,Q'} \cup avar(Q)$, such that $f(\bar{X}') = \bar{X}$.*
2. *For each goal $Y'_i R'_i Z'_i$ in Q', a path of points $p_1, p_2, \ldots, p_n$, in Q, of length $n \leq | \; nvar(Q) \; | \times | \; states(A_i) \; | + 2$, s.t. $p_1 = f(Y'_i)$, $p_n = f(Z'_i)$.*

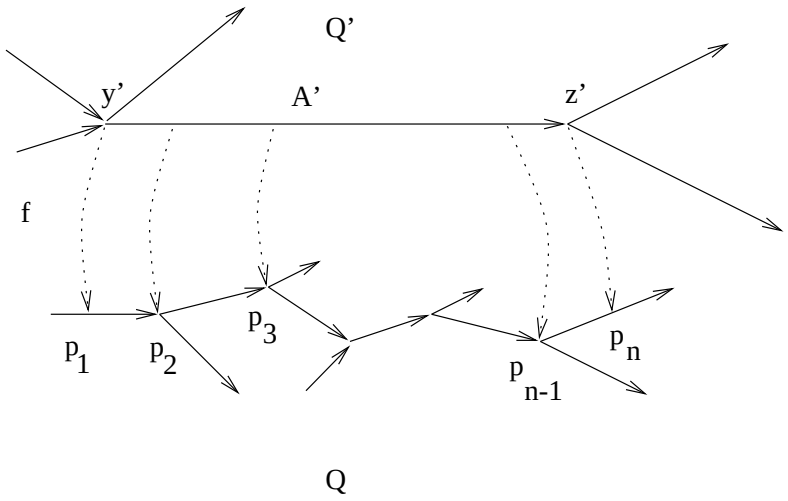


Figure 4: Illustration of a containment mapping from Q' to Q.

3. *For each goal $Y_iR_iZ_i$ in Q, a total preorder*[3] *on those variables $Z' \in nvar(Q')$ for which $f(Z')$ is an automaton-point corresponding to A_i. The preorder is required to satisfy the following: whenever $X' \preceq Y'$ and $Y' \preceq X'$ then $f(X') = f(Y')$ (i.e. they are mapped to the same automaton-point).*

The intuition is that such a query mapping rephrases a substitution φ from Q' to DB, for some canonical database (DB, ξ), in terms of the syntactic elements in Q. Conditions 1 and 2 are illustrated in Figure 4, which shows how the goal $Y'R'Z'$ (with A' being the associated automaton) is mapped to the path of points $p_1, \ldots, p_n$. The intuition is that this corresponds to a substitution $\varphi : Q' \rightarrow DB$ sending the goal $Y'R'Z'$ to some path in the canonical database DB. The path of points $p_1, \ldots, p_n$ may have cycles. However, its length is bounded as in item 2 of the definition above, because of the following argument. Consider some substutution $\varphi : Q' \rightarrow DB$, for some canonical database for Q. For the goal $Y'R'Z'$ in Q', there exists a path in DB from $\varphi(Y')$ to $\varphi(Z')$, possible with cycles, whose labels satisfy R. We decorate the nodes on path with states in the automaton A', starting at the initial state and ending at the terminal state. We argue that, w.l.o.g it suffices to assume that each node will be decorated by the same state at most once: otherwise we may simply make the path shorter, by cutting a cycle. Hence, the upper bound imposed in item 2 on the length of the path, n. Condition 3 is also easy to relate intuitively to substitutions. To see that, recall that any preorder $\preceq$ on a set induces (1) an equivalence relation $X' \equiv Y'$ iff $X' \preceq Y'$ and $Y' \preceq X'$, and (2) a total order on these equivalence classes. For example, consider the set $\{X', Y', Z', U', V', W'\}$. Then a total preorder can be concisely denoted as in $X' < Y' = Z' = U' < V' = W'$, meaning $X' \preceq Y'$, $Y' \preceq Z'$, $Z' \preceq Y'$, etc. Then, the intuition of condition 3 is that the query mapping imposes such an order on all variables sent by f to points on the same automaton $(A, s_1), (A, s_2), (A, s_3), \ldots$.

Note that so far we have no semantic conditions on a query mapping: there always exists query mappings between two queries Q' and Q. By contrast, in the case of conjunctive queries, the existence of a containment mapping implies immediately query containment. However note that, for given Q, Q', there exists only finitely many query mappings $f : Q' \rightarrow Q$. In fact each of them can be encoded in space which is polynomial in the size of Q and Q'. We will show that $Q \subseteq Q'$ is equivalent to a certain condition on the collection of *all* query mappings.

To derive that condition, consider some canonical database (DB, ξ) for Q. Assume first that ξ is injective on the node variables in Q. Then, every a substitution $\varphi : Q' \rightarrow DB$ uniquely induces a query mapping $f : Q' \rightarrow Q$. Namely to obtain the path of points $p_1, \ldots, p_n$ corresponding to some conjunct $Y_i', R_i'Z_i'$ we only need to follow that path from $\varphi(Y_i')$ to $\varphi(Z_i')$ in DB, whose labels satisfy the regular expression R_i. We say in this case that f *covers* that canonical database. We give below the general definition:

[3] A *preorder* $\preceq$ on a set S is a reflexive and transitive relation on S. It is called *total* if for every X', Y' in the set, at least one of the following holds: $X' \preceq Y'$ or $Y' \preceq X'$.

Definition 4.4 *Let $f : Q' \to Q$ be a query mapping, and (DB, ξ) be a canonical database for Q. We say that f* covers *DB if there exists some substitution $\varphi : var(Q') \to \mathcal{I} \cup \mathcal{D}'$ (where $\mathcal{D}' = \mathcal{D} \cup avar(Q)$), mapping the head variables in Q' to DB's canonical tuple, s.t. for any conjunct $Y_i' R_i' Z_i'$ in Q' mapped by f into the path of points $p_1, \ldots, p_n$ in Q, there exists a path from $\varphi(Y_i')$ to $\varphi(Z_i')$ in DB, corresponding to $p_1, \ldots, p_n$, and whose sequence of labels satifies the regular expression R_i'. In particular, whenever f covers (DB, ξ), then the canonical tuple of DB is in $Q'(DB)$.*

Note that some query mappings don't cover *any* canonical database. However, it is obvious that $Q \subseteq Q'$ iff, together, all query mappings cover all canonical databases. We will translate this condition into a statement about certain regular languages.

Let \$ be a new symbol, not present in the universe of constants $\mathcal{D}$. Given the query Q with n conjucts, fix some arbitrary order on the conjuncts. For any canonical database (DB, ξ) for Q, we define its *encoding* to be the following word over the alphabet $\mathcal{D} \cup avar(Q) \cup \{\$\}$: $w_{DB} \stackrel{\text{def}}{=} w_1.\$.w_2.\$.w_3 \ldots \$.w_n$, where w_i is the sequence of labels on the internal path of DB corresponding to the goal i. For example the canonical database DB in Figure 2 is encoded as $w_{DB} = a.L.e.f.a.\$.b.c.b.c.\$.a.L.a.a.a.L.\$.d.$ A set of canonical databases will then be encoded by a language. When this set is the set of all canonical databases for Q, then this encoding is given by $W_Q \stackrel{\text{def}}{=} R_1 \$ R_2 \$ \ldots \$ R_n$, where R_i is the regular language generated by the expression R_i. Obviously, W_Q is a regular language.

Given a query mapping f, we consider the language containing exactly the words w_{DB} for (DB, ξ) a canonical databases covered by f. We will prove that this language is equal to a certain regular expression W_f, to be defined below. Before giving the general definition, we illustrate with two examples the main idea behind the definition of W_f. First we introduce some notations. For some automaton A and states s, s', denote with $A(s, s')$ the same automaton, but with s designated as input state, and s' as output state. Similarly, $A(s, \_)$ ($A(\_, s)$) denotes the same automaton, with only the input state (output states) redefined as s, while the output states (input state) are unchanged.

Example 4.5 Let $Q : q(X_1, X_2) : -X_1 R_1 Y, Y R_2 X_2$ and $Q' : q'(X_1', X_2') : -X_1' R' X_2'$, where R_1, R_2, R' are arbitrary regular expressions, and let A_1, A_2, A' be any nondeterministic automaton associated to R_1, R_2, R' respectively. There exists a unique query mapping $f : Q' \to Q$, mapping the conjunct $X_1' R' X_2'$ into the path of points X_1, Y, X_2. Let S be the set of states of A'. For each $s \in S$, define $W^1(s) \stackrel{\text{def}}{=} A_1 \cap A'(\_, s)$ and $W^2(s) \stackrel{\text{def}}{=} A_2 \cap A'(s, \_)$. Here we define:

$$W_f \stackrel{\text{def}}{=} \bigcup_{s \in S} W^1(s).\$.W^2(s)$$

We briefly argue why W_f is the encoding of all canonical databases covered by f. Let (DB, ξ) be some canonical database with encoding $w_{DB} = w_1.\$.w_2$, and assume $w_{DB} \in W_f$. We will show that there exists a substitution φ from Q' to DB mapping (X_1', X_2') to the canonical tuple $(\xi(X_1), \xi(X_2))$ in DB. It suffices to prove that the word $w_1.w_2$ on the unique path $\xi(X_1) \to \xi(X_2)$ in DB is accepted by the automatom A'. This indeed follows from the fact that there exists $s \in S$ s.t. $w_1 \in A'(\_, s)$ and $w_2 \in A'(s, \_)$, and the property $A'(\_, s).A'(s, \_) \subseteq A'$

Example 4.6 Let $Q : q(X_1, X_2) : -X_1 R X_2$ and $Q' : q'(X_1', X_2') : -X_1' R_1' Y', X_1' R_2' Z', Y' R_3' X_2', Z' R_4' X_2'$. Denote with $A, A_1', A_2', A_3', A_4'$ some nondeterministic automata equivalent to the given regular expressions. Consider the query mapping $f : Q' \to Q$ sending X_1', X_2' to X_1, X_2 respectively, Y', Z' to $(A, s_1), (A, s_2)$, and with the order $Y' < Z'$. The mapping is illustrated in Figure 5. Let S_2', S_3' be the sets of states of the automata A_2' and A_3' respectively. For every $s \in S_2'$ and $t \in S_3'$ define:

$$\begin{aligned} W_1(s,t) &\stackrel{\text{def}}{=} (A(\_, s_1) \cap A_1' \cap A_2(\_, s)) \\ W_2(s,t) &\stackrel{\text{def}}{=} (A(s_1, s_2) \cap A_3(\_, t) \cap A_2(s, \_)) \\ W_3(s,t) &\stackrel{\text{def}}{=} (A(s_2, \_) \cap A_3(t, \_) \cap A_4) \end{aligned}$$

Then $W_f = \bigcup_{s \in S_2', t \in S_3'} W_1(s,t).W_2(s,t).W_3(s,t)$. It is easy to check that f indeed covers all canonical databases DB for which $w_{DB} \in W_f$.

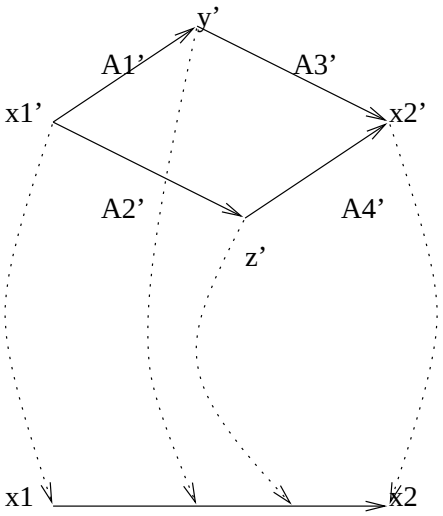


Figure 5: Illustration for Example 4.6.

We briefly sketch the general definition for W_f. Let $f : Q' \to Q$ be a query mapping. For each conjunct $Y_i R_i Z_i$ in Q, consider the sequence of points $q_{i0} = Y_i, q_{1i}, q_{2i}, \ldots, q_{(m-1)i}, q_{mi} = Z_i$, where the intermediate points $q_{1i}, \ldots, q_{(m-1)i}$ are all automaton-points in the image of f, and in the order imposed by f[4]. We call these the *points of interest* in Q. Consider now a conjunct, $Y'_j R'_j Z'_j$, in Q' which is mapped to the path of points $p_1, \ldots, p_n$. We refine this path by including all intermediate points of interest, to obtain a longer sequence of points of interest associated to that conjunct, $p_1 = r_{1j}, r_{2j}, \ldots, r_{s_j j} = p_n$: we emphasize that its length, s_j, depends on j. Let $P \stackrel{\text{def}}{=} \{(j,k) \mid 1 \le j \le \text{nr. of conjuncts in } Q', 1 \le k \le s_j\}$, and define Σ to be the set of all mappings $\sigma : P \to \bigcup_j (states(A_j) \cup \{\_\})$, s.t. for all $(j,k) \in P$, if $k > 1$, $k < s_j$ then $\sigma(j,k) \in states(A'_j)$, otherwise $\sigma(j,k) = \_$. Now we change perspectives again, and consider a conjunct $Y_i R_i Z_i$ in Q, with its sequence of points of interest $q_{i0} = Y_i, q_{1i}, q_{2i}, \ldots, q_{(m-1)i}, q_{mi} = Z_i$. Recall that its intermediate points are automaton-points of the form $(A_i, s_1), (A_i, s_2), \ldots, (A_i, s_{m-1})$, where $s_1, \ldots, s_{m-1}$ are states in A_i: we further define $s_0 \stackrel{\text{def}}{=} \_$ and $s_m \stackrel{\text{def}}{=} \_$. For a given σ and for each $l = 0, m-1$, define $W^i_l(\sigma)$ to be the intersection of all languages $A'_j(\sigma(j,k), \sigma(j,k+1))$ for which $r_{kj} = q_{li}$ and $r_{(k+1)j} = q_{(l+1)i}$, further intersected with $A_i(s_k, s_{k+1})$: this intersection contains at most as many factors as states in all automata in Q', plus one. Now define $W^i(\sigma) \stackrel{\text{def}}{=} W^i_1(\sigma).W^i_2(\sigma) \ldots W^i_m(\sigma)$, and $V_f(\sigma) \stackrel{\text{def}}{=} W^1(\sigma).\$.W^2(\sigma).\$ \ldots \$.W^n(\sigma)$. Finally, recall that Q has n conjuncts, and define:

$$W_f \stackrel{\text{def}}{=} \bigcup_{\sigma \in \Sigma} V_f(\sigma) \tag{1}$$

We prove in the full version of the paper:

Proposition 4.7 *Let $f : Q' \to Q$ be some query mapping and (DB, ξ) some canonical database for Q. Then f covers Q iff $w_{DB} \in W_f$.*

This implies the main result of this section:

Theorem 4.8 *Let Q, Q' be two queries, and F be the set of all query mappings $f : Q' \to Q$. Also let W_Q be the regular language encoding all canonical databases for Q. Then $Q \subseteq Q'$ iff $W_Q \subseteq \bigcup_{f \in F} W_f$.*

Finally, we comment on the complexity of checking StruQL$_0$ query containment. It is known that containment of regular expressions is PSPACE complete [28], hence StruQL$_0$ query containment is PSPACE

[4]That is, for every $k = 1, m-2$ there exists variables $X', Y' \in nvar(Q')$ s.t. $f(X') = q_{ki}$, $f(Y') = q_{(k+1)i}$, $X' \preceq Y'$, and $Y' \not\preceq X'$

hard. The algorithm resulting from Theorem 4.8 has exponential space complexity however. Indeed after combining $\bigcup_{f \in F} W_f$ with Equation (1), all we need to check is:

$$W_Q \subseteq \bigcup_{f \in F, \sigma \in \Sigma} V_f(\sigma) \tag{2}$$

where each $V_f(\sigma)$ can encoded using space which is polynomial in the size of Q and Q'. Hence there are only exponentially many distinct such expressions, which shows that the algorithm is in exponential space. The question whether STRUQL$_0$ query containment is in PSPACE remains open.

5 Query Containment for Simple STRUQL$_0$ Queries

We consider in this section a fragment of STRUQL$_0$, by imposing certain restrictions on the regular expressions used in queries. We show that query containment for this fragment is NP complete. This offers, to the best of our knowledge, the first example of a query language with recursion for which query containment has the same complexity as for conjunctive queries. The restricted form described here actually captures a class of queries very frequently used in pracice. Indeed, in the experience we had so far with the Strudel system [12], all queries had regular expressions conforming to these restriction.

Before giving the definition, we make the following convention: we abbreviate the regular expression $\_*$ as $*$.

Definition 5.1 *A* simple *regular expression is a regular expression of the form $r_1.r_2 \ldots r_n$, $n \geq 0$, where each r_i is either $*$, or some label constant from $\mathcal{D}$. A* simple STRUQL$_0$ *query is a* STRUQL$_0$ *query in which all regular expressions are simple.*

For example $a.*.b.*$ and $*.*.a.a.*$ are simple regular expressions, while $a*.b$ or $\_.\_$ are not. We normalize a simple regular expression, by replacing every $*.*$ with $*$. A simple regular expressions of length n and with $m \leq n$ constants, has a canonical nondeterministic automaton associated to it, consisting of $m+1$ states arranged in a chain, of which $n-m$ have loops labeled $\_$.

Simple regular expressions have the following properties.

Proposition 5.2

1. *If R, R' are two simple regular expressions of lengths n, n' respectively, then $R \cap R'$ can be expressed as $R_1 \cup R_2 \cup \ldots \cup R_k$, where each R_i is a simple regular expression of length $\leq n + n'$. Here k may be exponentially large in n, n'.*

2. *Let A be the canonical automaton for a simple regular expression R, and s, s' two states in A. Then the regular languages accepted by the automata $A(s, s')$, $A(s, \_)$, $A(\_, s')$ can each be expressed as a simple regular expression.*

3. *If the alphabet $\mathcal{D}$ is infinite, then whenever $R \subseteq R_1 \cup \ldots \cup R_k$, with $R, R_1, \ldots, R_k$ simple regular expressions, then there exists some i s.t. $R \subseteq R_i$.*

4. *Given two simple regular expressions R, R', one can check in PTIME whether $R \subseteq R'$ [23].*

Proof: (Sketch) To prove 1, let A, A' be the canonical automata for R, R', and let $\{s_1, \ldots, s_n\}$, $\{s'_1, \ldots, s'_{n'}\}$ be their sets of states. $R \cap R'$ is equivalent to the product automaton. Its states are pairs (s_i, s'_j), and its transitions are (a) either *successor* transitions $(s_i, s'_j) \rightarrow (s_{i+1}, s'_j)$, or $(s_i, s'_j) \rightarrow (s_i, s'_{j+1})$, which can be either labeled with a constant from $\mathcal{D}$, or (b) *loops*, labeled $\_$. Thus, if we ignore the loops, it is shaped like a dag. We unfold it, by taking all possible paths in the dag from the initial state (s_1, s'_1) to some terminal state (s_n, s'_j) or $(s_i, s'_{n'})$: we obtain $\binom{n+n'}{n}$ paths, all of length $n + n'$: hence, the claim follows. Item 2 is easy to show by a straightforward inspection on the shape of A. To prove 3, assume w.l.g. that $R_i \subseteq R$ for every $i = 1, k$: otherwise replace R_i with $R \cap R_i$, and apply item 1. Assume the contrary, i.e. $R_i \subset R$, for each $i = 1, k$. Let $\mathcal{D}_0$ be the set of all constants mentioned in $R, R_1, \ldots, R_k$. Since $\mathcal{D}$ is infinite, there exists some constant $c \in \mathcal{D} - \mathcal{D}_0$. Let w be the word obtained from R by replacing each $*$ with c. We will show

that, for every $i = 1, k$, $w \notin R_i$. Indeed, since $R_i \subseteq R$, all the constants appearing in R must also appear in R in the same order. But we also have $R_i \neq R$. There are two cases: (a) R_i has more constants than R. Since none of the additional constants is c, $w \notin R_i$. (b) R_i has exactly the same constants, but has some $*$'s replaced with ϵ. Let a_j and a_{j+1} be the constants preceding, respectively following that $*$ in R (the cases when $*$ is at the beginning or the end are treated similarly): that is, R has a substring of the form $a_j.*.a_{j+1}$, while R_i has $a_j.a_{j+1}$ instead. But then w contains one more c between a_j and a_{j+1} than R_i allows, hence $c \notin R_i$. □

Note that item 3 fails if we relax the definition of simple expressions. For example, if we allow _, then we have $\_.* \subseteq \_ \cup (\_.\_.*)$. Worse, the expression $*.a_1.*.a_2 \ldots *.a_n$ of length $2n$ is contained in the union of all 2^n expressions of length $\leq 4n$ obtained by replacing each $*$ with either ϵ or $\_.*$, but is not contained in the union of any subset of these expressions.

These two properties allow us to prove the following.

Theorem 5.3 *Let Q, Q' be two simple* StruQL$_0$ *queries. Recall that W_Q is the regular language encoding all canonical databases for Q. Then:*

- *$Q \subseteq Q'$ iff there exists some query mapping f which covers all canonical databases for Q: formally, $W_Q \subseteq W_f$).*
- *The problem of testing $Q \subseteq Q'$ is NP-complete.*

Proof: Consider Equation (2). Each regular expression $V_f(\sigma)$ is obtained expressed as polynomially many intersections of simple expressions. Hence, by Proposition 5.2, item 1, the large union in Equation (2) is equivalent to a (even larger) union of simple regular expressions, each of size which is polynomial in that of Q and Q'. Hence, from item 3, W_Q must be included in one of those simple regular expressions: the latter can be checked in PTIME [23]. Finally, to prove NP-hardness, we reduce the containment problem for simple StruQL$_0$ queries to that of conjunctive queries, for which query containment is NP-complete [7]. □

6 Conclusions

We have discussed query containment for the query language StruQL$_0$, consisting of conjunctive queries with regular path expressions, from two angles: a *semantic* angle, where we showed that query containment is equivalent to containment on certain canonical databases, and a *syntactic* angle, where we showed that query containment can be rephrased in terms of a certain condition on the set of all query mappings. We used the results from the semantic characterization in an essential way to derive the syntactic one. Query containment for StruQL$_0$ is known to be PSPACE hard, while the complexity of our algorithm is exponential space, hence leaving a gap between the upper bound and lower bound. Finally, we have considered a certain restriction of StruQL$_0$ to *simple* queries, and shown that containment for this fragment is NP complete.

References

[1] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the ICDT*, 1997.

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

[3] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet Wiener. The Lorel query language for semistructured data, 1996. Manuscript available from `http://www-db.stanford.edu/lore/`.

[4] Alfred Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, (8)2:218–246, 1979.

[5] Peter Buneman. Semistructured data. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona*, pages 117–121, 1997.

[6] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of SIGMOD–96*, pages 505–516, 1996.

[7] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[8] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of International Conference on Data Engineering*, 1995.

[9] Surajit Chaudhuri and Moshe Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, pages 55–66, 1992.

[10] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogenous information sources. In proceedings of IPSJ, Tokyo, Japan, October 1994.

[11] B. Courcelle. Recursive queries and context-free graph grammars. *Theoretical Computer Science*, 78:217–244, 1991.

[12] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with strudel: experience with a web-site management system. Submitted for publication, 1997.

[13] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. System demonstration - strudel: A web-site management system. In *ACM SIGMOD Conference on Management of Data*, 1997.

[14] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, September 1997.

[15] Daniela Florescu, Alon Levy, and Dan Suciu. A query optimization algorithm for semistructured data. Submitted for publication, 1997.

[16] Daniela Florescu, Louiqa Rashid, and Patrick Valduriez. Answering queries using OQL view expressions. In *Workshop on Materialized Views, in cooperation with ACM SIGMOD, Montreal, Canada*, 1996.

[17] Ashish Gupta, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. Constraint checking with partial information. In *Proceedings of the Thirteenth Symposium on Principles of Database Systems (PODS)*, pages 45–55, 1994.

[18] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.

[19] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[20] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd VLDB Conference, Bombay, India.*, 1996.

[21] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[22] Alon Y. Levy and Dan Suciu. Deciding containment for queries with complex objects and aggregations. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona.*, 1997.

[23] Tova Milo and Dan Suciu. Index structures for path expressions. Submitted for publication, 1997.

[24] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the 22nd VLDB Conference, Bombay, India.*, 1996.

[25] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.

[26] Yehoshua Sagiv. Optimizing datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA, 1988.

[27] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[28] L. J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *5th STOC*, pages 1–9. ACM, 1973.

[29] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[30] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the International Conference on Database Theory*, 1997.

[31] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, pages 331–345, 1992.

[32] X. Zhang and M. Z. Ozsoyoglu. On efficient reasoning with implication constraints. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, pages 236–252, 1993.

A Appendix

Proof of Theorem 3.3 To illustrate the idea, we start by taking a closer look at a substitution φ from Q' to some canonical database (DB, ξ): then φ maps the node variables in Q' to bifurcation and/or internal nodes in DB. Consider each conjunct $Y_i' R_i' Z_i'$ in Q', and let A_i' be some automaton (could be nondeterministic) for R_i'. Assume that there exists a path from $\varphi(Y_i')$ to $\varphi(Z_i')$ in DB whose sequence of labels satisfies R_i'. This path may contain loops, of course. Since A_i' accepts the sequence of labels on that path, we can associate states s from A_i' to the nodes on the path, as we traverse it. Several states may be associated to the same node, because our path may have loops. However we may assume w.l.g. that the paths does not go twice through the same node *and* in the same state (because otherwise we can cut out a portion of the path). Summarizing, φ associates to each node u in DB a set S_u consisting of node variables in Q' and automaton/state pairs.

Now we change perspectives and look at fixed internal path in DB. We pick some segment $[b, u]$ of that path starting at the bifurcation node b, and ending at some internal node u. The action of φ on this particular segment in isolation can be described by the following. (1) The set S_b associated to the node b, (2) the set S_u associated to the node u, (3) a set of variables $W \subseteq nvar(Q')$, which are mapped by φ to the interior of the segment, and which we call *middle variables*, and (4) a DAG, $C = (N, E)$, over the nodes $N \stackrel{\text{def}}{=} S_b \cup W \cup S_u$, s.t. all nodes in S_b are initial nodes and all nodes in S_u are terminal nodes, whose edges are labeled by automata in Q'. See Figure 6 for an illustration, where $S_b = \{(A_5', s_9), (A_5', s_3), (A_7', s_2), X_5'\}$, $S_u = \{(A_5', s_2), (A_1', s_8), x_1', (A_2', s_8)\}$, $W = \{X_4', X_3', X_7', W_6', X_9'\}$, and the dag is illustrated in the figure. This suggests the following definition.

Definition A.1 *Let (DB, ξ) be a canonical database for Q, and let Q' be some other query. We define a* configuration, *C, to be a labeled dag $C = (N, E)$ where N consists of variables from Q' and/or automata/state pairs from Q', and is partitioned into three sets*[5] *$N = S_1 \cup W \cup S_2$ with W containing only variables, and with all nodes in S_1 being initial (i.e. without incoming edges) and all nodes in S_2 terminal. The edges E are labeled by automata from Q', such that the following condition holds. If some edge $n \to n'$ is labeled A_i', then: (1) either n is of the form (A_i', s) (i.e. has the same automaton), or n is a variable y' and there exists some conjunct $y' R_i' z'$ in Q' (i.e. referring to the same automaton A_i'), and (2) either n' is of the form (A_i', s), or n' is a variable z and there exists a conjunct $y R_i' z$ in Q'.*

[5] A pair (A_i', s) may occur twice in N: once in S_1 and once in S_2. A variable, however, may occur only once in $S_1 \cup W \cup S_2$.

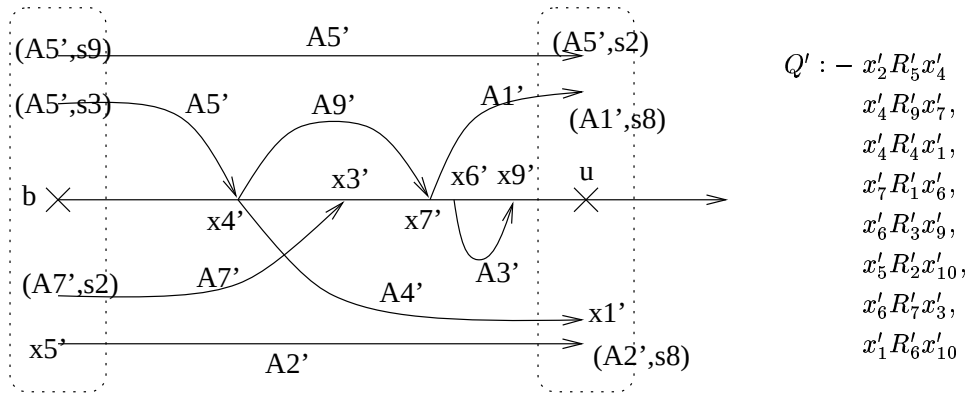


Figure 6: Illustration of a configuration for nodes b and u.

We use the definitions in Section 4, where $A(s, s')$ denotes the automaton A with the initial state redefined to be s, and the terminal states to be s': also, we use the notations $A(s, \_)$ and $A(\_, s)$, where only the terminal state, or only the initial states have been redefined to be s. Then, we associate a certain regular language to each edge $n \to n'$ in C, as follows. Let A'_i be the automaton labeling that edge. If $n = (A'_i, s)$, then define $\sigma \stackrel{\text{def}}{=} s$; otherwise, $\sigma \stackrel{\text{def}}{=} \_$. Similarly, if $n' = (A'_i, s')$, define $\sigma' \stackrel{\text{def}}{=} s'$; otherwise, $\sigma' \stackrel{\text{def}}{=} \_$. Then we define the regular language associated to the edge $n \to n'$ to be $A'_i(\sigma, \sigma')$. For example for the top edge in Figure 6, the regular language is $A'_5(s_9, s_2)$. Given a segment $[b, u]$ and a configuration C, one can think of C as a piece of information about the substitution φ: namely it says that all variables in S_1 are mapped to b, those in S_2 to u, and the middle variables W somewhere inside the segment $[b, u]$. We say that a configuration is *satisfiable* at the segment $[b, u]$ if there exists a mapping of W into nodes inside the segment $[b, u]$, such that every edge in C is mapped to a path whose sequence of labels belongs to the regular language associated to that edge. For example, the configuration in Figure 6, one of the requirements would be that the variable x'_7 be mapped to some node u', s.t. the labels between u' and u belong to the language $A'_1(\_, s_8)$.

Now we can show that in Proposition 3.2 it suffices to check only canonical databases with "short" internal paths. Consider some canonical database (DB, ξ) for Q, and some query Q'. Assume that the canonical tuple is not in the answer of Q' on DB. Choose some "long" internal path in DB (of length to be specified shortly), starting at the bifurcation node b. We will construct a new canonical database DB_0 which is identical to DB except that it has that internal path replaced by a shorter one, such that the canonical tuple is still not in the answer of Q': repeating this argument, we obtain a canonical database in which all internal paths are short, which proves our claim. For each internal node u on that path, let $\mathcal{C}_u$ be the set of all configurations satisfiable on $[b, u]$. As we vary u from the beginning of the internal path to its end, the set $\mathcal{C}_u$ varies: however, the number of possible configurations is bounded by some number $c(Q')$ depending only on the query Q', hence there will be at most $2^{c(Q')}$ different distinct values for $\mathcal{C}_u$. Now we recall that DB is a canonical database for Q, hence the internal path under consideration corresponds to some conjunct $y_i R_i z_i$ of Q. Choose some (nondeterministic) automata A_i for R_i, then we can label the nodes u along the path with states s_u of A_i, starting with the initial state and ending in a terminal state. In conclusion we have labeled each node u with a pair $(s_u, \mathcal{C}_u)$, and there exists at most $n_i \times 2^{c(Q')}$ distinct such labels (where n_i is the number of states in A_i). If the internal path is longer than $n_i \times 2^{c(Q')}$, then pick any two nodes u and u' with the same label, and cut the path between them, i.e. collapse u and u' into a single node called u and delete everything between them. We argue that the resulting database DB_0 (1) is still a canonical database, and (2) the answer of Q' on DB_0 still doesn't contain the canonical tuple. Is easy to check (1), since DB_0 has the right shape, and since we made sure that the nodes we collapsed during the cut were labeled by the same state of A_i. We prove (2). Assume the contrary, i.e. there exists some substitution φ_0 from Q' to DB_0 mapping the head variables of Q' to the canonical tuple. This gives us a configuration C with is satisfiable at $[b, u]$ in DB_0. The segment $[b, u]$ is the same in DB and DB_0, hence C is satisfiable at $[b, u]$ in DB too. But then C is a satisfiable at $[b, u']$ too, because we chose u and u' s.t. they have the same sets of satisfiable configurations. Hence we can construct from φ_0 a substitution $\varphi : Q' \to DB$, by redirecting the middle variables W of C from the segment $[b, u]$ to $[b, u']$: but this would imply that the

canonical tuple is in the answer of Q' on DB, which is a contradiction. To summarize, we have proven:

Proposition A.2 *Let Q, Q' be two queries, and let $c(Q')$ be the number of configurations for Q'. Also, let n_i be the number of states of the automaton A_i, for each conjunct $y_i R_i z_i$ in Q. Then $Q \subseteq Q'$ iff for any canonical database whose internal path associated to the conjunct $y_i R_i z_i$ has length $\leq n_i 2^{c(Q')}$, its canonical tuple is in the answer of Q'. As a consequence, the problem of checking whether $Q \subseteq Q'$ is decidable.*

This implies Theorem 3.3. We conclude with some remarks on the complexity. Let $m \stackrel{\text{def}}{=} \mid nvar(Q') \mid$ be total number of node variables in Q', and n be total number of states in all (nondeterministic) automata A'_i of Q'. An upper bound for the number of configurations can be quickly obtained as follows. There are $M \stackrel{\text{def}}{=} 2^m \times 2^{2n}$ choices for the set of nodes in a configuration (any set of variables, and each pair (A'_i, s) may occur independently in S_1 and S_2), and we can build 2^{M^2} graphs (which includes all dags): hence $c(Q') \leq 2^{2^{2m} \times 2^{4n}}$. Thus, the complexity of checking query containment this way is triple exponential space. By contrast, the algorithm resulting from Theorem 4.8 has an exponential space complexity.