Query Containment for Conjunctive Queries With Regular Expressions

Daniela Florescu*
INRIA, France
Daniela.Florescu@inria.fr

Alon Levy*
Univ. of Washington, USA
alon@cs.washington.edu

Dan Suciu
AT&T Labs, USA
suciu@research.att.com

Abstract

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. This paper considers the problem of query containment for a query language over semistructured data, STRUQL_0, that contains the essential feature common to all such languages, namely the ability to specify regular path expressions over the data. We show here that containment of STRUQL_0 queries is decidable. First, we give a semantic criterion for STRUQL_0 query containment: we show that it suffices to check containment on only finitely many *canonical* databases. Second, we give a syntactic criteria for query containment, based on a notion of *query mappings*, which extends containment mappings for conjunctive queries. Third, we consider a certain fragment of STRUQL_0, obtained by imposing restrictions on the regular path expressions, and show that query containment for this fragment of STRUQL_0 is NP complete.

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 [28, 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., [12, 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 [15, 17]. The fragment STRUQL_0 can be briefly described as the union-free, negation-free subset of STRUQL and therefore plays a similar role for STRUQL as conjunctive queries for the relational calculus [2]. 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 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 results known 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

*This work was done while the author was at AT&T Labs.

[1]UnQL does not handle oid equalities.

PODS '98 Seattle WA USA

have been developed in the context of query optimization [8, 29, 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 [25], rewriting queries using views [9, 23], and maintenance of integrity constraints [21]. More recently, query containment, applied in the context of rewriting queries using views, have been used as a tool in data integration [24, 18, 33, 19].

We show here that containment of STRUQL_0 queries is decidable. Specifically, 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 algorithm has triple exponential space 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 of a query language with recursion for which checking containment of a pair of recursive queries is no harder than for a pair of conjunctive queries.

Query containment for first order conjunctive queries is decidable [8, 29, 4]. Several works have considered the extension of containment algorithms for queries involving order [22, 34, 25, 37, 21], and queries over complex objects [26]. Queries in STRUQL (and, hence, in STRUQL_0) can be translated into datalog. Hence, our containment result for STRUQL_0 is related to the problem of testing containment of datalog programs and, indirectly, to the more general problem of checking properties of datalog programs. Shmueli [31] showed that containment of datalog programs is undecidable. Sagiv [30] shows that containment is decidable for the weaker condition of *uniform containment*. All positive results for containment so far are restricted to the particular case when one of the datalog programs is nonrecursive. Namely, Chaudhuri and Vardi [10] show that the equivalence of a recursive and a non-recursive datalog program is decidable in triple exponential time, and improve the complexity bounds for certain particular cases in [11]. A related problem is the *boundedness problem*, asking whether a recursive datalog program is equivalent to *some* non-recursive one: undecidability is shown by Gaifman et al. [20] for general datalog programs, and by Vardi [35] for *linear* datalog programs, while decidability is shown by Cosmadakis et al. [13] for *unary* datalog programs, and by Wang [36] for other particular cases. Containment of bounded queries is of course decidable, but, in the context of STRUQL_0, we are mostly interested in unbounded queries.

A remarkable positive result was shown by Courcelle [14], who proved that any property expressible in monadic second order logic on the syntactic expansions of a datalog program is decidable. Chaudhuri and Vardi [10] show how to apply this result to prove that containment of a datalog program in a non-recursive datalog program is decidable. Their technique can be extended to show decidability of STRUQL_0 query containment (see Remark 3.5): however, the resulting algorithm has non-elementary time complexity. Finally, we note that decidability for STRUQL_0 query containment also follows from a result proven independently in [7].

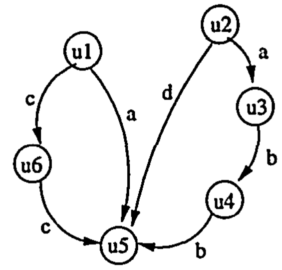


Figure 1: An example of a graph database.

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.

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 [17]. 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 $\mathcal{D}$):

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

Here ϵ is the empty string, a a label constant, _ denotes any label and L is a label variable. We abbreviate $R.(R^*)$ with R^+. A query in StruQL$_0$ is an expression of the form

$Q : q(\bar{X}) : -Y_1R_1Z_1, \ldots, Y_nR_nZ_n$.

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 call each condition $Y_iR_iZ_i$ a *conjunct*, $i = 1, n$. 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 $\mathcal{D}$ such that:

- for every i, $1 \leq i \leq n$, there is a path P in the database between $\varphi(Y_i)$ and $\varphi(Z_i)$, 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).

We denote a substitution with $\varphi : Q \to DB$, and denote with $\varphi(Y_iR_iZ_i)$ the path in DB corresponding to the conjunct $Y_iR_iZ_i$. 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. When Q is applied to the database in Figure 1, the relation R_Q contains the 4 tuples: $\{(u_1, u_1, u_5, c), (u_1, u_1, u_5, a), (u_2, u_1, u_5, d), (u_2, u_2, u_3, a)\}$. The result $Q(DB)$ is the projection of R_Q 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 [31], 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 we show that 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. Specifically, 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. A database 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*. The nodes on an internal path are called *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 database with constants in $\mathcal{D}' \stackrel{\text{def}}{=} \mathcal{D} \cup avar(Q)$, and $\xi : Q \to DB$ 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. *For each conjunct $Y_iR_iZ_i$, the path $\xi(Y_iR_iZ_i)$ in DB is an internal path. Moreover, this mapping from conjuncts to internal paths is one to one.*

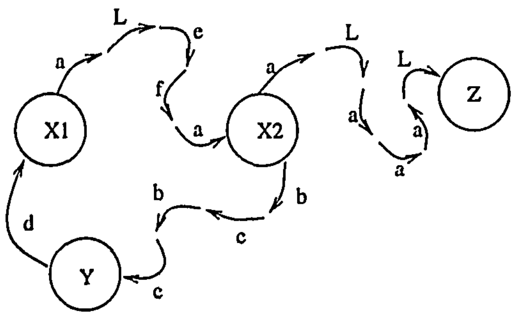


Figure 2: A canonical database. The bifurcation nodes are encircled.

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, it must be the case that $\epsilon \in R$). 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 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

$$Q : q(X_1, X_2) \quad :- \quad X_1(a.L.(\_^*))X_2, X_2(b.c)^*Y, \\ X_2(a|L)^*Z, Y(c|d)X_1$$

is illustrated in Figure 2 (here $\xi(X) = X$ for every $X \in var(Q)$). Note that _ can be replaced by any atomic value in $\mathcal{D} \cup avar(Q)$, with or without repetitions. In the presence of _, we still have infinitely many canonical databases even if the query doesn't have a Kleene 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', the containment $Q \subseteq Q'$ holds iff for any canonical database (DB, ξ) for Q, its 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 $\mathcal{D}' \stackrel{\text{def}}{=} \mathcal{D} \cup avar(Q)$. We omit the straightforward details of this direction here. 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 label constants $\bar{u} = (u_1, \ldots, u_k)$ 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 $\varphi : Q \to DB$, s.t. $\varphi(\bar{X}) = \bar{u}$. We start the construction of the canonical database (DB_0, ξ) by picking the bifurcation nodes to be $V_B \stackrel{\text{def}}{=} \{\varphi(X) \mid X \in nvar(Q)\}$, and we will define $\xi(X) \stackrel{\text{def}}{=} \varphi(X)$ for every $X \in nvar(Q)$. Next, for each conjunct YRZ in Q, we consider the path $\varphi(YRZ)$ in DB. It 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 occurrence of a node in that path, thus creating a simple path from $\varphi(Y)$ to $\varphi(Z)$ with the same labels, denoted $\xi(YRZ)$. Now 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 $\xi(YRZ)$ is accepted by $\varphi(A)$. We replace each label a causing a transition in $\varphi(A)$ corresponding to some arc variable $L \in avar(DB)$ with L. After this change, the path $\xi(YRZ)$ has labels from $\mathcal{D}' \stackrel{\text{def}}{=} \mathcal{D} \cup avar(DB)$, and 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 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 full version of the paper. 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:

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

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.

Remark 3.5 The (possible infinite) set of canonical databases for some query Q can be described by a context free graph grammar [14]. Moreover, one can show that any StruQL_0 query Q' can be expressed in monadic second order logic. Then, the containment $Q \subseteq Q'$ is equivalent to checking a certain formula in monadic second order logic on all graphs generated by a graph grammar: Courcelle [14] showed that this problem is decidable. This implies Corollary 3.4. However the resulting algorithm has non-elementary complexity.

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 a major difference from the classical case of conjunctive query containment mappings. In our setting we have several (possibly infinitely many) canonical databases for Q, as opposed to only one for conjunctive queries. Since all have a rather similar shape, one may hope that a single query mapping from Q' to Q could witness the evaluation of Q' on all of them. 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.

In the first part of this section we rephrase the three notions "a node in DB", "a path in DB", and "a substitution $\varphi : Q' \to DB$", where DB is a canonical database for Q, in terms of the syntactic elements of query Q. In the second part we express containment $Q \subseteq Q'$ in terms of these rephrasings.

Rephrasing canonical databases. We give the three definitions below.

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.*

Nodes in a canonical database DB for Q correspond to points in Q in an obvious way. The correspondence is many-to-many however: several internal nodes in DB may correspond to the same automaton-point, while bifurcation nodes in DB correspond both to variable-points and to automaton-points of the form (A_i, s) with s an initial or a terminal state.

Definition 4.2 *Given a query Q, a* path of points *in Q 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 conjunct $Y_i R_i Z_i$ 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.*

Let $U = u_1, u_2, \ldots, u_m$ a path in DB be a path in a canonical database (DB, ξ) for Q. Suppose we drop all internal nodes from $u_2, \ldots, u_{m-1}$ (i.e. keep only bifurcation nodes and the end nodes): let $u_{i_1} = u_1, u_{i_2}, u_{i_3}, \ldots, u_{i_{n-1}}, u_{i_n} = u_m$ be the resulting subsequence (it still fully determines U). We say that U *corresponds* to the path of points $p_1, \ldots, p_n$ iff each u_{i_k} corresponds to p_k, for $k = 1, n$. Of course, the correspondence is many-to-many; nevertheless, the intuition is that paths of points rephrase paths in canonical databases.

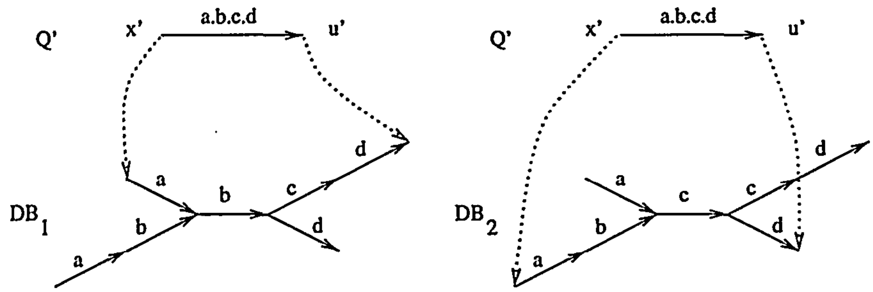
Note that, when $n = 2$, the definition allows a path of points to be of the form $(A_i, s), (A_i, s')$, even if there is no path from s to s' in the automaton A_i.

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. *Two mappings $f : nvar(Q') \to points(Q)$, and $f : avar(Q') \to D_{Q,Q'} \cup avar(Q)$, s.t. $f(\bar{X}') = \bar{X}$.*
2. *A mapping from conjuncts $Y'_i R'_i Z'_i$ in Q' to paths of points in Q, $f(Y'_i R'_i Z'_i) = p_1, p_2, \ldots, p_n$, s.t. $n \leq |nvar(Q)| \times |states(A_i)| + 2$ and $p_1 = f(Y'_i)$, $p_n = f(Z'_i)$.*
3. *For each conjunct $Y_i R_i Z_i$ in Q, a total preorder[3] on those variables $Z' \in nvar(Q')$ for which $f(Z')$*

[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 $Y' \preceq X'$.



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

Figure 3: Two substitutions with different shapes.

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).

For some canonical database (DB, ξ) for DB, call a substitutions $\varphi : Q' \to DB$ *canonical* if $\varphi(\bar{X}')$ is the canonical tuple in DB. Query mappings rephrase canonical substitutions. Conditions 1 and 2 are illustrated in Figure 4, which shows how the conjunct $Y'R'Z'$ (with A' being the associated automaton) is mapped to the path of points $P \stackrel{\text{def}}{=} p_1, \ldots, p_n$. This rephrases a substitution $\varphi : Q' \to DB$ sending the conjunct $Y'R'Z'$ to some path in the canonical database DB which corresponds to P. 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 substitution $\varphi : Q' \to DB$ for which $\varphi(Y'R'Z')$ is a "long" path in DB: of course, it may have cycles. However if some node occurs more than $|\,states(A')\,|$ times, then we can cut $\varphi(Y'R'Z')$ to a shorter path which still satisfies R': hence, the upper bound imposed in item 2. Finally, we explain Condition 3. Any preorder $\preceq$ on a set induces (1) an equivalence relation $X' \equiv Y'$ defined as $X' \preceq Y'$ 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 $(A, s_1), (A, s_2), (A, s_3), \ldots$.

Formally we define a correspondence between canonical substitutions $\varphi : Q' \to DB$ and query mappings $f : Q' \to Q$: namely φ *corresponds* to f if (1) for each conjunct $Y'R'Z'$ in Q' the path $\varphi(Y'R'Z')$ corresponds to the path of points $f(Y'R'Z')$, and (2) for any internal path in DB corresponding to some conjunct YRZ, the preorder on all variables mapped by φ onto that path coincides with the preorder given by f.

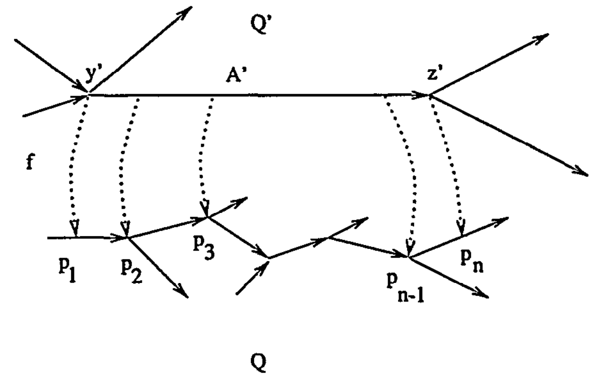


Figure 4: Illustration of a containment mapping from Q' to Q.

Containment condition. Now we focus on finding conditions on query mapping(s) which guarantee containment $Q \subseteq Q'$. 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 $f : Q' \to Q$. 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. We need a definition:

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 canonical substitution $\varphi : Q' \to DB$ s.t. φ corresponds to f.*

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. In the remainder of this section we will translate this condition into a containment problem of two regular languages.

Let \$ be a new symbol, not present in the universe of constants $\mathcal{D}$. Given the query Q with n conjuncts, 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 conjunct 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 show that this language is equal to a certain regular expression W_f, to be defined below. Before giving the general construction for W_f, we illustrate with two examples the main ideas behind it, and hint to its correctness: the proof of the correctness of W_f is deferred to the full version of the paper. 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. With this notation, $A(\_, \_)$ is A.

Example 4.5 Let $Q : q(X_1, X_2) : -X_1R_1Y, YR_2X_2$ 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 with 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. Which canonical databases are covered by f ? Let S be the set of states of A'. For each $s \in S$, define the regular languages $W^1(s) \stackrel{\text{def}}{=} A_1 \cap A'(\_, s)$ and $W^2(s) \stackrel{\text{def}}{=} A_2 \cap A'(s, \_)$. Let:

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

We briefly argue why W_f is the encoding of all canonical databases covered by f, by showing that each canonical database (DB, ξ) with encoding $w_{DB} = w_1.\$.w_2 \in W_f$ is covered by f (the other direction is similar). We have to show that there exists a canonical substitution $\varphi : Q' \to DB$ corresponding to f. We define $\varphi(\bar{X}') \stackrel{\text{def}}{=} \xi(\bar{X})$ (hence φ is canonical): 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 automaton 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 $Q : q(X_1, X_2) : -X_1RX_2$ 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

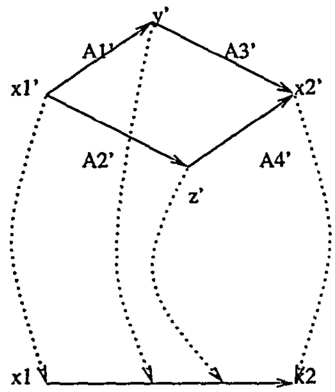


Figure 5: Illustration for Example 4.6.

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 $w_{DB} \in W_f$.

We briefly sketch now the construction of W_f in the general case, by generalizing the two examples above. Let $f : Q' \to Q$ be a query mapping. For each conjunct $Y_iR_iZ_i$ 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 with that conjunct, $p_1 = r_{1j}, r_{2j}, \ldots, r_{s_jj} = p_n$: we emphasize that its length, s_j, depends on j. Let $P \stackrel{\text{def}}{=} \{(j,k) \mid 1 \leq j \leq \text{no. of conjuncts in } Q', 1 \leq k \leq s_j\}$, 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 $Y_iR_iZ_i$ 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

[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 $Y' \not\preceq X'$

as many factors as states in all automata in Q', plus one. Now define $W^i(\sigma) \stackrel{\text{def}}{=} W_1^i(\sigma).W_2^i(\sigma)\ldots W_m^i(\sigma)$, 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 DB 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 [32], hence StruQL$_0$ query containment is PSPACE 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 for which checking containment of a pair of recursive queries is no harder than for a pair of conjunctive queries. The restricted form described here actually captures a class of queries very frequently used in practice. Indeed, in the experience we had so far with the Strudel system [16], 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 with 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 $R \subseteq R'$ [27].*

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 $(s_i, s'_j) \to (s_{i+1}, s'_j)$, or $(s_i, s'_j) \to (s_i, s'_{j+1})$, 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 $n+n'$: after placing the loops back, each of them is equivalent to a simple regular expression, 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.o.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_i 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 $\_.* \subseteq \_ \cup (\_.\_.*)$. More general, 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 [27]. 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 [8]. □

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] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *PODS*, 1998. (this volume).

[8] 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.

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

[10] 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.

[11] Surajit Chaudhuri and Moshe Vardi. On the complexity of equivalence between recursive and nonrecursive datalog programs. In *The Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 55–66, 1994.

[12] 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.

[13] Stavros Cosmadakis, Haim Gaifman, Paris Kanellakis, and Moshe Vardi. Decidable optimization problems for database logic programs. In *STOC*, pages 477–490, 1988.

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

[15] 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.

[16] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with strudel: experience with a web-site management system. In *Proceedings of ACM SIGMOD*, 1998.

[17] 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.

[18] 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.

[19] Marc Friedman and Dan Weld. Efficient execution of information gathering plans. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[20] H. Gaifman, H. Mairson, Y. Sagiv, and M. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM*, 40(3):683–713, 1993.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] Tova Milo and Dan Suciu. Index structures for path expressions. In preparation, 1997.

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

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

[30] 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.

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

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

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

[34] 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.

[35] Moshe Vardi. Decidability and undecidability results for boundedness of linear recursive queries. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 341–351, 1988.

[36] Ke Wang. Some positive results for boundedness of multiple recursive rules. In *Proceedings of the International Conference on Database Theory*, pages 383–396, 1995.

[37] 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.